Assignment of bachelor's thesis

Title:  Automatic categorization of job ads
Student:  Patricie Petríňková
Supervisor:  Ing. Karel Klouda, Ph.D.
Study program:  Informatics
Branch / specialization:  Artificial Intelligence 2021
Department:  Department of Applied Mathematics
Validity:  until the end of summer semester 2023/2024

Instructions

This work aims to design and test algorithms for automatically classifying job advertisements into categories. The English-written job ads are from the up2staff.com server, where the ads are now sorted manually.

1) Study and analyze the data provided.
2) Research methods suitable for the automatic categorization of advertisements.
3) Select at least three methods, apply them to the data provided and properly evaluate the results.
4) After a discussion with the up2staff company, implement the best method so that it can be deployed in their ad processing process.
5) Based on the data analysis and experiments, consider the possibility of changing the categories used.

__________________________________________________________________________

Electronically approved by Ing. Magda Friedjungová, Ph.D. on 14 February 2023 in Prague.
Contents

Acknowledgments vi
Declaration vii
Abstract viii
List of abbreviations ix
Introduction 1

1 Data Preparation and Analysis 3
  1.1 Dataset ........................................... 3
  1.2 Data Preprocessing ................................. 4
    1.2.1 Tokenization .................................. 4
    1.2.2 Stop Words and Non-alphabet Characters Removal ................ 4
    1.2.3 Vocabulary Normalization .......................... 5
  1.3 Data Analysis ..................................... 5

2 Algorithm Research 11
  2.1 Known Results at Job Classification ....................... 11
  2.2 Supervised Machine Learning Pipeline ...................... 11
  2.3 Feature Engineering ................................ 12
    2.3.1 Feature Creation ................................ 12
    2.3.2 Dimensionality Reduction ......................... 14
    2.3.3 Feature Scaling / Normalization .................... 14
  2.4 Supervised Machine Learning Algorithms .................. 15
    2.4.1 KNN ........................................ 15
    2.4.2 Logistic Regression ........................ .... 15
    2.4.3 Naive Bayes .................................. 15
    2.4.4 SVM ......................................... 16

3 Experiments 17
  3.1 F1 Score ........................................ 17
  3.2 Naive Bayes ...................................... 18
    3.2.1 BoW and TF-IDF ................................. 18
    3.2.2 Word2Vec .................................... 21
  3.3 KNN ................................................ 23
    3.3.1 Feature Reduction ............................... 23
    3.3.2 Title Classification ............................ 24
    3.3.3 Content Classification ........................... 26
    3.3.4 KNN Overview .................................. 26
  3.4 Logistic Regression ................................ 26
    3.4.1 Title Classification ............................ 27
    3.4.2 Content Classification ........................... 28
<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>3.5 SVM</td>
<td>28</td>
</tr>
<tr>
<td>3.5.1 Linear Kernel</td>
<td>28</td>
</tr>
<tr>
<td>3.5.2 RBF Kernel</td>
<td>30</td>
</tr>
<tr>
<td>3.6 Final Evaluation</td>
<td>31</td>
</tr>
<tr>
<td>3.6.1 Title Classification</td>
<td>31</td>
</tr>
<tr>
<td>3.6.2 Content Classification</td>
<td>31</td>
</tr>
<tr>
<td>3.6.3 Final Classification</td>
<td>34</td>
</tr>
<tr>
<td>4 Implementation</td>
<td>37</td>
</tr>
<tr>
<td>Conclusion</td>
<td>41</td>
</tr>
<tr>
<td>A Requirements for Implementation</td>
<td>43</td>
</tr>
<tr>
<td>Contents of Enclosed Archive</td>
<td>47</td>
</tr>
</tbody>
</table>
List of Figures

2.2 CBoW and Skip-Gram Word2Vec [12] ................................. 14  
3.1 Comparison of Confusion Matrices for Classification of New and Default Categories 20  
3.2 Word2Vec Validation Score For Tuning Of Hyperparameters window and vector_size 23  
3.3 Comparison for PCA and LSA Feature Reduction on TF-IDF and BoW for 5 and 10 neighbors in KNN ........................................ 24  
3.4 Impact of Logistic Regression Hyperparameter $C$ on F1-score of Title Categorization 27  
3.5 Impact of PCA Reduction for Logistic Regression on F1-score of Content Categorization ........................................ 27  
3.6 Impact of SVM Hyperparameter $C$ on F1-score of Title Categorization ........................................ 29  
3.7 Impact of PCA Reduction for SVM on F1-score of Content Categorization ........................................ 29  
3.8 True Predictions Normalized Confusion Matrix for SVM Classification of Default Categories ........................................ 31  
3.9 True Predictions Normalized Confusion Matrix for Title Categorization Using SVM 32  
3.10 True Predictions Normalized Confusion Matrix for Content Categorization Using SVM ........................................ 33  
3.11 True Predictions Normalized Confusion Matrix of Classification Model ........................................ 36  
4.1 Overview of Moduls in Implementation ........................................ 39  
A.1 Issues for Implementation of Final Categorization Script ........................................ 43

List of Tables

1.1 Category Overview ........................................ 6  
1.2 Overview of Unique and Characteristics Words ........................................ 7  
3.1 Results of Multinomial Naive Bayes Tuning with Default Categories ........................................ 19  
3.2 Results of Multinomial Naive Bayes Tuning with New Categories ........................................ 21  
3.3 Title Categorization Results for KNN ........................................ 25  
3.4 Content Categorization Results for KNN ........................................ 25  
3.5 Title Categorization Algorithm Comparison ........................................ 32  
3.6 Content Categorization Algorithm Comparison ........................................ 33  
3.7 Best Performing Algorithms for Text and Content ........................................ 35
I would like to express gratitude to my family, who always supported me on my personal and academic journey. I would also like to thank my supervisor, Ing. Karel Klouda, Ph.D., for his guidance and invaluable feedback.
Declaration

I hereby declare that the presented thesis is my own work and that I have cited all sources of information in accordance with the Guideline for adhering to ethical principles when elaborating an academic final thesis. I acknowledge that my thesis is subject to the rights and obligations stipulated by the Act No. 121/2000 Coll., the Copyright Act, as amended, in particular the fact that the Czech Technical University in Prague has the right to conclude a licence agreement on the utilization of this thesis as a school work pursuant of Section 60 (1) of the Act.

In Prague on May 10, 2023
Abstract

This thesis presents the development of the classification model for Information Technology job advertisement at webpage up2staff.com. The objective is to create a reliable classification system that reduces the time and costs associated with manual categorization of job ads. The process involves analyzing and preprocessing a dataset of job ads, researching appropriate algorithms, and experimenting with combinations of feature engineering techniques and supervised machine learning classification algorithms. The model decides the final decision based on weighted decisions from two classification algorithms; one created for the content and the other for the job ads’ title. Both classifications perform with the highest F1-score for the Support Vector Machines algorithm applied to TF-IDF features. The classification model achieves F1-score of 0.909.

Keywords  text processing, classification of job advertisement, comparison of algorithms, TF-IDF, SVM

Abstrakt

Tato práce je zaměřena na vývoj klasifičního modelu pro pracovní nabídky v oblasti informačních technologií na webové stránce up2staff.com. Cílem je vytvořit spolehlivý klasifikační systém, který sníží čas a náklady spojené s ruční kategorizací. Proces zahrnuje analýzu a zpracování souboru dat s inzeráty pracovních nabídek, výzkum vhodných algoritmů a experimentování s kombinací technik tvorby příznaků a klasifikačních algoritmů supervizovaného strojového učení. Model vyhodnotí konečné rozhodnutí o kategorii na základě vážených rozhodnutí dvou klasifikačních algoritmů, jeden pro obsah a druhý pro titulek inzerátu. Obě klasifikace jsou založeny na metodě podpírých vektorů (SVM) aplikovaného na vektory příznaků tvořené pomocí TF-IDF. Klasifikační model dosahuje F1-skóre 0,909.

Klíčová slova  zpracování textu, klasifikace pracovních inzerátů, porovnání algoritmů, TF-IDF, SVM
List of abbreviations

CAT_ID  category ID
BoW     Bag of Words
CBoW    Continuous Bag of Words
KNN     K-Nearest Neighbors
LSA     Latent Semantic Analysis
MAP     Maximum A Posteriori
MLE     Maximum Likelihood Estimate
NLP     Natural Language Processing
NB      Naive Bayes
PCA     Principal Component Analysis
POS     part-of-speech
SVM     Support Vector Machine
SVD     Singular Value Decomposition
TF-IDF  Term Frequency Inverse Document Frequency
List of abbreviations
The web page up2staff.com is a platform for job advertisements mostly in the Computer Science area, primarily focused on remote working opportunities. Currently, job ads are classified into categories manually. However, the manual classification is time-consuming, costly, and requires personnel. Therefore, the task of job classification is preferred to be outsourced to a machine-learning classification model that quickly and efficiently categorizes new job ads. The use of a classification model increases the efficiency of categorization with systematic and unbiased decisions. The classification model will consider various features and characteristics of the job ads to provide accurate classifications. Additionally, it will help job seekers efficiently find relevant job opportunities and aid employers with finding suitable candidates.

This thesis aims to find a suitable solution and develop a reliable automatic job ads classification model for up2staff.com. To achieve this objective, algorithms for feature engineering and supervised machine learning classification are studied, examined, and evaluated on the dataset of job advertisements provided by up2staff.com. The evaluation of classification algorithms’ performance is assessed using appropriate evaluation metrics. The final goal of this thesis is to implement a classification model with high reliability in categorizing job ads, and is suitable for deployment on a real-world webpage with job ads.

The thesis comprises of four chapters focused on a different classification model development aspect. The first chapter gets the reader acquainted with the dataset and provides an overview of the dataset. Furthermore, it explains the text preprocessing process and job ad analysis. Analyzing the dataset helps identify the key attributes and job ad characteristics that distinguish different categories and enable the development of an effective classification model.

The second chapter surveys methods useful for classification. This chapter includes a discussion of feature engineering methods and an overview of the classification algorithms used in the experiments’ chapter.

The third chapter of the thesis will present the experiments performed to evaluate the different feature engineering and classification techniques. The experiments evaluate the use of standardization, various feature creation algorithms, and different classification algorithms. The chapter presents each experiment’s results, including tables and graphs to illustrate the performance of each technique.

Finally, in chapter four, the best-performing and the most effective model is implemented. This model will be employed as the classification model for job ads on the up2staff.com.
Introduction
Chapter 1

Data Preparation and Analysis

The aim of this chapter is to examine and prepare text data for their use in feature extraction. The first section briefly examines the dataset and the data it contains. The next section covers various text preprocessing techniques, such as tokenization, stemming, or stop-word removal. The second section focuses on the analysis of preprocessed data, specifically the content of the data.

1.1 Dataset

The aim of this section is to explain how the data was obtained and stored in a format that can be used for the experiments. Additionally, the section discusses job ad structure.

The data used for the development of the appropriate classification model were received from page up2staff.com through a MySQL file that creates a database of job ads. I created a local database on my computer, ran the script, and exported job ads from the database into CSV file, which was used within the project. The database has 44,283 records of job ads. Most of the job ads in the dataset were manually labeled with a unique ‘CAT_ID’ (category ID), providing a relatively reliable base for model training and evaluation.

Job ads are classified into nine categories, each with its unique ‘CAT_ID’ (all listed in parentheses):

- Full-Stack Programming (27)
- Design (29)
- Product (34)
- Sales and Marketing (45)
- DevOps and Sysadmin (46)
- All Other Remote (51)
- Management and Finance (58)
- Front-End Programming (65)
- Back-End Programming (67)
All Other Remote contains uncategorized data; after removing those, 38,504 job ads were left. The dataset contains no duplicates; however, sometimes, it contained spam. One was a job ad with an email address repeated numerous times in the job ad content. Another was a job ad for a remote tarot reader, which was in the dataset 5 times, and categorized into: Full-Stack Programming, two times Sales and Marketing, Management and Finance, All Other Remote. Furthermore, some of the records were in non-English languages.

Full-Stack Programming, Front-End Programming, Back-End Programming and DevOps and Sysadmin are very similar categories that are hard to distinguish and often blend into each other. up2staff.com expressed interest in merging these categories into a single category.

Each job ad record consists of several attributes: ‘ID’, ‘CAT_ID’, ‘post_title’ and ‘post_content’. The ‘ID’ attribute contains a unique number assigned to each job ad record, while the ‘CAT_ID’ describes the category which it belongs to. ‘post_title’ and ‘post_content’ contain all information about the job ad posted on the page. Consequently, attributes ‘post_title’ and ‘post_content’ will be used in experiments for feature creation in classification.

1.2 Data Preprocessing

Data preprocessing is the first step of NLP (Natural Language Processing). Preprocessing involves preparing and cleaning text data from characters and words irrelevant for classification. By removing noise from data, a more suitable and relevant representation of text in the form of features is achieved, and subsequently, the classification algorithm achieves better performance. The goal of preprocessing is to reduce the size of the vocabulary and, consequently, decrease the computational effort of feature extraction. Furthermore, preprocessing helps to find the most informative set of features and reduces the likelihood of overfitting. [1]

1.2.1 Tokenization

Tokenization is the first step of preprocessing. Tokenization is the act of breaking down text, unstructured data, into tokens made of strings, numbers, and punctuation marks that can be used to represent the document, and hence creating structured data. All further preprocessing steps are applied to tokens. [1]

This thesis uses for preprocessing text one-word tokens split by white space or punctuation marks. For example sentence “The brown fox jumped over 2 logs.” would be transformed into an array of tokens: [“The”, “brown”, “fox”, “jumped”, “over”, “2”, “logs”, “.”].

1.2.2 Stop Words and Non-alphabet Characters Removal

Stop words are common words occurring with high frequency that carry low information about the meaning of a phrase. Non-alphabetic characters are punctuation marks or numbers. They are useful for text orientation for humans; however, they carry little information and bring noise into data. Therefore, removing tokens containing stop words and non-alphabet characters after tokenization is beneficial for relevant feature extraction. Many pre-defined stop word lists for all languages are created to identify stop words. Some examples of stop words are “a”, “they”, or “will”. [1, 2]

Given an array of tokens [“The”, “brown”, “fox”, “jumped”, “over”, “2”, “logs”, “.”], where “2” and “.” are non-alphabetic tokens and “over” and “The” are stop words, the resulting array after processing is [“brown”, “fox”, “jumped”, “logs”].
1.2.3 Vocabulary Normalization

Normalization of vocabulary reduces the number of tokens and combines tokens with similar meanings into one normalized form. Normalization techniques are lower casing words, stemming, and lemmatization. [1]

**Lower-casing**

Lower-casing converts all uppercase letters in a token to lowercase to join data with the same meaning, for instance, when the same word is at the start of the sentence and in the middle. Although, the words that capitalization differentiates are also joined. For example, words “FIT” and “fit” are combined in one token but have a different meaning.

**Stemming**

Normalization technique that reduces words to their stem\(^1\) by removing the suffix and prefix. The resulting token is not a word but the token representing words with the same stem. For example, “development” and “developer” become the same token “develop”. Nevertheless, stemming can result in the loss of information. For instance, token “better” from phrase “better benefits” becomes “bet”, completely losing its original meaning and becoming an unrelated token to the previous context. [1]

**Lemmatization**

Lemmatization maps the token to its lemma\(^2\), preserving the meaning of the original word and its part of speech. Lemmatization considers a word’s meaning by taking into consideration the word ending and, combined with POS tagging, results in joining tokens with identical meanings but different forms. POS tagging, or part-of-speech tagging, is the process of labeling each word in a text corpus with its corresponding part of speech, such as noun, verb, adjective, or adverb. For example, without POS tagging, the lemmatizer would not know whether the word “quickly” should be lemmatized as “quick” (an adjective) or “quickly” (an adverb).

Lemmatization retains more information in the token compared to stemming. The array of tokens [“brown”, “fox”, “jumped”, “logs”] is transformed to equal arrays, [“brown”, “fox”, “jump”, “log”], using either lemmatization or stemming. However, when lemmatizing the phrase “better benefits”, the word “better” would be transformed to its base form “good”, which retains the information about the original word’s meaning. [1]

1.3 Data Analysis

In the analytical part, I aim to assess the content and title for each category by observing used vocabulary, evaluating job ads’ content in each category, and exploring data patterns across categories before choosing and experimenting with classification models. Libraries Pandas\(^3\) and Streamlit\(^4\) were used for the data analysis and visualization.

As can be seen in Table 1.1 containing the number of articles and the average length of text in content and title for each category, Full-Stack Programming is the dominant category taking up 61.7 % of all the data. The second biggest is Sales and Marketing, which is 11 % of job offers, and the lowest number of job ads is Front-End Programming, with only 2 %. This is a significant imbalance that must be considered when choosing the classification metric. The average number of words in a job ad is 272 for a content and 3.5 for a title. Most categories have around 300 words per job content; however, Full-Stack Programming brings the average down with its value of 244. The title range is between 3-4 words per title for all categories.

---

\(^1\)part of a word carrying its lexical meaning

\(^2\)semantic root of a word

\(^3\)https://pandas.pydata.org/docs/

\(^4\)https://streamlit.io
<table>
<thead>
<tr>
<th>Category</th>
<th>Size of Category</th>
<th>Average Words for Content</th>
<th>Average Words for Title</th>
</tr>
</thead>
<tbody>
<tr>
<td>Full-Stack Programming</td>
<td>23755</td>
<td>244.6</td>
<td>3.5</td>
</tr>
<tr>
<td>Costumer Support</td>
<td>1902</td>
<td>310.9</td>
<td>3.9</td>
</tr>
<tr>
<td>Design</td>
<td>1936</td>
<td>311.9</td>
<td>3.4</td>
</tr>
<tr>
<td>Product</td>
<td>1152</td>
<td>347.6</td>
<td>3.3</td>
</tr>
<tr>
<td>Sales and Marketing</td>
<td>4254</td>
<td>324.3</td>
<td>3.6</td>
</tr>
<tr>
<td>DevOps and Sysadmin</td>
<td>1068</td>
<td>328.3</td>
<td>3.4</td>
</tr>
<tr>
<td>Management and Finance</td>
<td>2466</td>
<td>298.7</td>
<td>3.2</td>
</tr>
<tr>
<td>Front-End Programming</td>
<td>857</td>
<td>295.6</td>
<td>4.0</td>
</tr>
<tr>
<td>Back-End Programming</td>
<td>1115</td>
<td>300.3</td>
<td>3.8</td>
</tr>
<tr>
<td>All categories</td>
<td>38504</td>
<td>271.7</td>
<td>3.54</td>
</tr>
</tbody>
</table>
After the text was preprocessed using the methods discussed in the previous section, the resulting sentences were transformed into unigrams, bigrams, and trigrams for analysis. Unigrams are single tokens that appear in the text. They are the simplest type of n-gram, where \( n \) represents the number of tokens in a sequence. Bigrams are two-word sequences, and trigrams are three-word sequences that appear in the text. N-grams contain more information with increasing \( n \); however, this comes at the cost of increasing the dimensionality of the feature space which can lead to issues with computational efficiency and overfitting. After a thorough review, I decided to concentrate on pursuing the use of unigrams and bigrams in a vector representation of the job ads, as the information trigrams brought about was not found valuable.

To evaluate the significance of n-grams and analyze categories of job ads, I accessed tokens by calculating the token frequency distribution for a specific category as shown in the following equation:

\[
\text{Frequency of token in category corpus} = \frac{\text{Token count in documents for category}}{\text{Document count in category}} \quad (1.1)
\]

Specifically, the average number of times a token appeared in the text of the category per ad. Using these token frequencies, the dominant tokens (with the highest frequency) in the category were classified as characteristic for the category. Furthermore, unique tokens were also evaluated for each category.

| Table 1.2 Overview of Unique and Characteristics Words |
|---------------------------------|---------------------------------|-------------------------------------------------|-------------------------------|------------------------|
| Category                        | Title                           | Characteristic Words                      | Unique Words                  | Characteristic Words      |
| Full-Stack Programming          | fullstack, application          | software, stack                           | asp, iou                      | software, stack           |
| Costumer Support                | service, happiness              | costumer, support                         | zendesk, complaint            | costumer, support         |
| Design                          | designer, ux                    | designer, ux                              | typography, illustration      | design, user              |
| Product                         | owner                           | product, manager                          | groom, omnipresent            | product, company          |
| Sales and Marketing             | seo, ads                        | marketing, sale                           | sdr, tiktok                   | marketing, sale           |
| DevOps and Sysadmin             | infrastructure, administrator    | engineer, devops                          | invoca, tucow                 | infrastructure, system    |
| Management and Finance          | technology, cto                 | lead, director                            | bookkeeping, cfo              | project, manage           |
| Front-End Programming           | angular, shopify                | developer, frontend                       | semantic, flutter             | development, build        |
| Back-End Programming            | go, laravel                     | backend, ruby                             | dash, ingest                  | backend, api              |

**Full-Stack Programming**

Only five words are characteristic for the title in this category: “software”, “full”, “stack”, “web”, and “engineering”. Unigrams “full” and “stack” appear once per ten job offers, while unique words like “fullstack” or “application” are clear indicators of this category. On the other hand, “full” and “stack” also have a frequency of 0.02 in *DevOps and Sysadmin*. Other
characteristic words are also frequently mentioned in other categories. While many characteristic words in the content have a higher appearance in this category, they appear with slightly lower frequency in Back-end Programming or DevOps and Sysadmin. For instance, “stack” is the second most common characteristic word used with a frequency of 0.54, but it also appears with a frequency of 0.49 in DevOps and Sysadmin. Common technologies such as “java”, “net” (from .NET), and “node” are also mentioned, but they appear in Back-end Programming with similar frequency.

Customer support

This category is characterized by the frequent use of words related to helping people, such as “support”, “help”, “provide” or “service”. The most common word in this category “customer” appears with a frequency of 8.2 in the content and 0.6 in the title of job ads. Compared to a Product, which has “customer” with the second highest frequency of all categories with the frequency of 2.9 in the content and 0.02 in the title part of the job ad, it is a significant difference. Product contains many words with significant frequency from the vocabulary of Costumer Support. Additionally, many words used in this category have a positive connotation, such as “happiness” in titles or “love” in content.

Design

The Design category has few characteristic words but with particularly high frequency. Similar to Customer Support, this category features characteristic words such as “ux”, “user”, or “create”, which are associated with Design. Words used in this category are often associated with an artistic context, such as characteristic words “sketch”, “craft” and “beautiful” or unique words such as “illustration” or “storyboard”.

Product

The keyword for this category is “product”, which appears with a frequency of 9.0 in the content and 0.7 in the title, making it the most commonly used word per category in the entire dataset. Other top characteristic words are often not technical and are common in other categories, such as “work”, “team”, or “company”. The Product has stronger unique words than other categories with frequency values around 0.1. These words often have a long-term planning context, such as “impact”, “focus”, “vision” or “deliver”.

Sales and Marketing

Category Sales and Marketing has many strong words with high frequency such as “marketing”, “sale” or “media” for both content and title. Characteristic words often have a majorly higher frequency in comparison to other categories; nevertheless, there are still many characteristic words with similar frequency in Finance and Management.

DevOps and Sysadmin

This category has many strong characteristic words for job titles such as “security” or “cloud”, but not for content. The characteristic words in content often appear with similar but lower frequency in Back-End Programming and Front-End Programming. This category, like Product, has stronger unique words compared to other categories, with a value of around 0.1.

Management and Finance

The Management and Finance has characteristic words in the title related to management. For example, synonyms for “manager”, “lead” or “chief” are characteristic with the highest frequency of 0.1. However, “manager” is a characteristic word for the Product category with a frequency of 0.56 compared to 0.49 for Management and Finance.

In terms of content, characteristic words in this category often appear with similar frequency in Product or Sales and Marketing. Many words appear in all categories with comparable frequency, resulting in job ads for managers being a general mix of all categories without any standout characteristic words.
Front-End Programming

*Front-End Programming* has few strong characteristic words in both the job title and content. Most of the appearing words are also found with comparable frequency in other categories, particularly *Full-Stack Programming* and *Back-End Programming*. The only words that stand out are “frontend” or “front” combined with “end”, which is to be expected given the name of the category.

Back-End Programming

The characteristic words for the title have the second-highest appearance in *Full-Stack Programming*, making it difficult to distinguish between the two categories based on job titles alone. Additionally, the characteristic words for the content of this category have the lowest frequency compared to other categories, making it challenging to identify this category based on the content of job ads. The most frequently used characteristic word in the content of this category is “remote”, which has a low-frequency value of 1.41 and is a general word not specific to this category. Similarly, other characteristic words with frequencies below 1.0, such as “api” or “backend”, are strong indicators of this category but are used less frequently than characteristic words in other categories. This category also shares many characteristic words with *DevOps and Sysadmin* and *Full-Stack Programming* categories.

The title has an average length of 3.5 words. Therefore, I decided to pursue unigrams and bigrams in experiments. I expect those to absorb all information from three to four words long titles, and I do not expect improvements utilizing trigrams, only overfitting.

In the content part of the job ad, trigrams in each category have a very low frequency. There are not significantly dominant trigrams representing the category. The count is very low, creating sparse space of dimension 4,529,347 where the dominant part of a vector will be zeros as the text is on average of length 271.7.

Bigrams have, as expected, a higher frequency than trigrams, and their number is 2,834,735, which is half of the trigrams. In the title of job ads, unique bigrams appeared with higher frequency and in a higher quantity than non-unique characteristic bigrams. This could lead to overfitting when using bigrams; however, I plan to experiment with bigrams.

In the following text, bigrams will be represented by two unigrams enclosed in parentheses: (“unigram”, “unigram”). Bigrams in content closely follow unigrams trends. For example, *Product* has many variations of bigrams with “product” unigram. *Customer Support* has top characteristic bigrams with the unigram “customer” and *Product* with “product”. *Front-End Programming* contains mainly bigrams for general job descriptions such as bigram (“compensation”, “equity”), all with low frequency. Often the most common bigrams were (“full”, “time”) or (“salary”, “compensation”), which are not useful for the categorization. The characteristic bigram is (“front”, “end”) while other bigrams appear with similar frequency in *Back-End Programming* and *Full-Stack Programming*. Bigram (“open”, “source”) is a characteristic bigram for Back-End with the highest appearance frequency of 0.28, has 0.27 in *DevOps and Sysadmin*, 0.22 in *Front-End Programming* and 0.22 in *Full-Stack Programming*.

After analysis, it appears that *Customer Support*, *Design* or *Sales and Marketing* should be easy to recognize as they are more distinct and specific. On the other hand, *Management and Finance* with *Product* could be hard to classify, while *Front-End Programming*, *Full-Stack Programming*, and *Back-End Programming* with *DevOps and Sysadmin* seem to be very hard to distinguish as the text has similar characteristics. The text of job offers is technical and often with common context; therefore, classification models could be missing a level of expertise that human experienced in the field could provide.
Chapter 2

Algorithm Research

This chapter aims to introduce algorithms applied in text classification. The first section explores known results in job classification. The second section covers a sequence of steps used in supervised machine learning. The third chapter explains techniques utilized in feature engineering for feature creation and their preparation for text classification models. The last section concentrates on classifiers in supervised machine learning.

2.1 Known Results at Job Classification

The classification system in this thesis for up2staff.com should be simple to understand and edit. Consequently, the final algorithm should be simply modified to the needs of up2staff.com and have low cost and effort maintenance.

Systems for job classification were already researched and developed. However, classification focused on the field of information technologies was not attempted or documented. Therefore, the success rate cannot be determined.

One of the examples of building a job classifier is written by a team from the Department of Statistics and Quantitative Methods, University of Milan-Bicocca, where the classifier is one part of the paper [3]. This team had a million job vacancies on hand and tried to classify data into general groupings such as managers, service and sales workers, technicians, and associate professionals. They used both titles and content to classify job advertisements. They used BoW (Bag of Words) for feature extraction and evaluated several machine learning models: SVM (Support Vector Machine), Random Forest and Artificial Neural Networks. They achieved the highest F1-score using linear SVM and the lowest using Random Forest. Next, this team is currently working on applying word embedding in their research.

Another example is the study [4] by a team from the Data Science Department, The Islamic University in Palestine. This team used TF-IDF (Term Frequency Inverse Document Frequency) method for feature extraction and Multinomial Naive Bayes, Support Vector Machine, Decision Tree, KNN, and Random Forest for classification with a dataset of size 18,000 job ads. The team had the highest accuracy with Random Forest, followed by KNN and lowest with Multinomial Naive Bayes. However, all experiment results were above the accuracy of 0.95, which is highly accurate.

2.2 Supervised Machine Learning Pipeline

When building a supervised machine learning model, the creator usually progresses with several distinct critical phases as outlined in Figure 2.1. The first phase is data retrieval, the extraction
of relevant data from a dataset that will be processed and used in the supervised machine learning model. Next, relevant data goes through preparation, where raw data is transformed into features accurately representing data. This preparation includes data processing, feature extraction, and feature transformation into features suitable for the chosen model. Features are then used in a model which goes through hyperparameters tuning, and its efficiency is evaluated using chosen standard evaluation metric. This pipeline can be implemented with different features and models until the most accurate classification system is found.

2.3 Feature Engineering

Feature engineering is the process of developing the most appropriate features given the data, the model, and the task [6]. The aim of feature engineering is to improve the performance of the machine learning model. The resulting features can be transformed by scaling or have dimensionality reduced.

2.3.1 Feature Creation

The goal of feature creation is to extract information representing the data accurately. Ways of creating feature representation for text data which are going to be discussed in this section are algorithms BoW, TF-IDF or Word2Vec.

2.3.1.1 BoW

In the BoW algorithm, each document in the corpus (the set of all documents) is represented by a feature vector. The vocabulary of the corpus is then used to create features for the document. The feature is created by counting each word’s appearance from the document’s vocabulary. In BoW, every word becomes an element of the vector; therefore, if there are n-words in the vocabulary, the document becomes a point in n-dimensional space. The BoW breaks down a sentence into words; consequently, the words lose semantic meaning. [6] For example, “pet” can mean the action of stroking or a domestic animal; however, these two meanings are merged into one.

This problem is addressed by the bag-of-n-grams algorithm, an extension of BoW, which takes N-gram as a feature. The algorithm retains more from the original structure of a sentence and can detect context slightly better. For lower N, the semantic meaning can still be hard to pinpoint. [6] For example, the sentences “I have pet dogs at home.” and “I like to pet dogs.” still have the bigram “pet dogs” but with different meanings. We can solve this problem by increasing
N, however, with the increasing N, the dimension of feature space also increases and becomes more sparse [6].

2.3.1.2 TF-IDF - Term Frequency–Inverse Document Frequency

TF-IDF is an algorithm that builds on BoW. TF-IDF highlights meaningful words and measures how important a term is within the document compared to other documents in the corpus. [6]

To attain this goal TF-IDF uses an equation for each term $t$ in the document $d$ calculated as:

$$\text{TF-IDF} = \text{TF} \times \text{IDF}$$

TF, term frequency, is calculated in (2.2) and IDF, inverse document frequency, which measures how important a term is within the corpus is given by (2.3):

$$\text{TF} = \frac{\text{number of times term } t \text{ appeared in document } d}{\text{number of words present in the document } d},$$

$$\text{IDF} = \log \frac{N}{DF} = \log \frac{\text{number of documents in the corpus}}{\text{number of documents in the corpus containing term } t}.$$ (2.3)

The TF-IDF approach favors terms with high frequency in the analyzed document and low frequency in other documents. Similarly to BoW, TF-IDF does not account for the order of words in a document and can be used with N-grams.

2.3.1.3 Word2Vec

Word embedding is a feature learning technique in which each word or phrase from the vocabulary is mapped to an N-dimension vector of real numbers. This technique considers word similarity and semantic meaning. [7]

There are two Word2Vec architectures, CBoW (Continuous Bag of Words) or Skip-Gram architecture. Both models use a shallow neural network with a single hidden layer. The hidden layer is trained by backpropagation with gradient descent to minimize the loss function of weights. The model learns the semantic meaning of words by adjusting weights during the training. In the resulting weight matrix, rows represent the vocabulary and columns the neurons. The output softmax layer takes in a word embedding vector created by multiplying one-hot encoded vector of the input word with weights of the hidden layer and outputs probability distribution over the vocabulary of the target words. [8, 9]

The skip-gram approach predicts the surrounding window of the size $N$, given as a hyperparameter of the model, of words based on the target word. The input vector is a one-hot encoded vector representing vocabulary with a single 1 in place of the target word. The output is $N$ vectors containing, for every word in our vocabulary, the probability of the context word at that position given the target word. [9, 10]

CBoW predicts the target word based on surrounding context words in the window of size $N$. The $N$ input vectors have 1 in place of the context word, and the output vector is a single vector containing probability for the target word from the vocabulary based on surrounding context words. [9, 11]

The models can be seen in Figure 2.2. In the diagram, the window hyperparameter is of size 2 with target word at $w_0$ and $w_i$ for $i \neq 0$ are context words. Weights of the hidden layer are split into $W$ and $W'$, where $W$ is the input hidden-weight matrix connecting the input layer to the hidden layer, and $W'$ is the output hidden-weight matrix connecting the hidden layer to the output layer.

Skip-gram approach works well with small corpus and rare words, while CBoW is a faster and better choice for frequent words. [5]
2.3.2 Dimensionality Reduction

Dimensionality reduction is a data transformation technique where data is transformed from high-dimensional space to low-dimensional while retaining meaningful data properties. Dimensionality reduction is useful with algorithms that suffer from the curse of dimensionality such as KNN.

2.3.2.1 PCA

PCA (Principal Component Analysis) is a linear transformation that reduces the number of features by identifying the most important features, known as principal components, which capture the majority of the variation in the data. PCA achieves this by performing an orthogonal projection of the data onto a lower-dimensional subspace while maximizing the data’s variance. [13]

2.3.2.2 LSA

LSA (Latent Semantic Analysis) is dimension reduction analysis useful in combination with TF-IDF where it gathers up words into topics. It optimizes topics to maintain diversity, and while capturing semantics, it reduces the number of features. [1]

LSA creates a matrix from features and then applies SVD (Singular Value Decomposition) to create three matrixes. Matrixes are truncated with high-variance dimensions left and multiplied back together. This results in the reduction of dimensions while retaining the underlying semantic structure of the text. [1]

2.3.3 Feature Scaling / Normalization

Feature scaling is a feature engineering technique used to normalize features range typically to the interval between 0 and 1 to ensure that each feature contributes equally to the learning process.
Feature scaling is vital to algorithms that use distance calculations or variance between data, as the feature with a higher range has a disproportional influence in the algorithm. Examples of feature scaling are Min-Max Scaling and Standardization.

## 2.4 Supervised Machine Learning Algorithms

This section briefly introduces selected supervised machine learning classification algorithms that will be applied for the classification of text features.

### 2.4.1 KNN

KNN (K-Nearest Neighbors) classifier is a simple algorithm based on distance. The algorithm calculates the K-closest neighbors and assigns labels to points based on the most common label between neighbors.

### 2.4.2 Logistic Regression

Binary logistic regression predicts the probability that $Y$ has the value 1 given a feature vector $x \in \mathbb{R}^n$. This probability is represented by the number $P(Y = 1 \mid x)$ from the interval $[0, 1]$. The vector of coefficients $w$ is estimated using MLE (Maximum Likelihood Estimate) and Sigmoid function in (2.4) ensures the output value is between $[0, 1]$:

$$P(Y = 1 \mid x) = \frac{1}{1 + e^{x^\top w}}. \quad (2.4)$$

Extension of logistic regression for multiclass classification uses Softmax function instead of Sigmoid. The probability of $Y$ being from class $k$ is

$$P(Y = k \mid x) = \frac{e^{x^\top w_k}}{\sum_{j=1}^{K} e^{x^\top w_j}}. \quad (2.5)$$

where $w_j$ is a vector of coefficients for class $j$. [14]

Another way to use multinomial logistic regression is the one-vs-rest approach, where one binary logistic regression model is created for every class. Then the resulting class is chosen from the model with the highest probability.

### 2.4.3 Naive Bayes

Naive Bayes is a classification algorithm based on Bayes’ Theorem for calculating conditional probabilities. It assumes feature independence, which may not always hold in practice, but often produces good results due to the fact that the final class is chosen by MAP (Maximum A Posteriori) estimate. The formula for the Naive Bayes classifier’s MAP estimate of target variable $\hat{Y}$ from a set of possible target variable values and features $X_i$ with $i \in 1, \ldots, n$ is:

$$\hat{Y} = \arg\max_{y \in Y} \prod_{i=1}^{n} P(X_i = x_i \mid Y = y) P(Y = y). \quad (2.6)$$

From the equation is clear that the algorithm estimates the probabilities of the target variable conditioned on the features and chooses the target with the highest probability. However, if there are no occurrences of a target variable and a feature in the training data, the probability estimate for that combination will be zero due to the product in the equation. This limitation is solved by Laplace Smoothing, which adds a specified alpha value to each count of the feature. [15]
Different types of Naive Bayes algorithms are applied for classification based on the distribution of features and ways of computing conditional variables. The first commonly used Naive Bayes is Bernoulli, used for binary features. The second type is Multinomial, used for discrete values such as text features created by BoW. However, it also works well for TF-IDF features [16]. Another common Naive Bayes algorithm is Gaussian Bayes, which is useful for continuous features distributed according to a Gaussian distribution.

2.4.4 SVM

SVM is a classification algorithm that classifies data by creating a separating hyperplane between two classes. The data is in the form of vectors in n-dimensional space, where n is a number of features.

If the data are linearly separable, an infinite number of solutions for the separating hyperplane exist. In SVM, the separating hyperplane is selected using the Largest Margin Principle. This principle selects a hyperplane that has the largest distance to the nearest points of the hyperplane. These points are called support vectors.

If data are not linearly separable, two approaches are possible. First is soft margin SVM which allows misclassifications in the training data by introducing a slack variable that measures the degree of misclassification. The regularization parameter $C$ is then introduced to control the trade-off between maximizing the margin and minimizing the misclassification. [17]

The second approach involves the kernel trick. The kernel trick is the use of a kernel function:

$$ K(x, y) = \langle f(x), f(y) \rangle. \quad (2.7) $$

The kernel function accepts vectors $x$ and $y$ in a lower dimensional space and returns the dot product of the transformed vectors using function $f$ to the higher dimensional space where the data becomes linearly separable. Examples of commonly used transformations in kernel functions are linear, polynomial and Gaussian kernel. [18]

SVM for multi-class classification problems is handled by one-vs-rest or one-versus-one approach.
Chapter 3

Experiments

This chapter explores various classification algorithms and feature engineering techniques for the categorization of job ads from webpage up2staff.com. During all experiments, text for feature creation was preprocessed using tokenization, case lowering, stop-word removal and lemmatization. Evaluation of models is done by the hold-out method, where a portion of the available data is set aside as a test set to assess the performance of the model. The dataset was split to 80% of the dataset as the train set and 20% for the test set. Afterward, the train set is split further with 80% for the final train set and 20% validation set. The train set is used for model training, and the validation set is for hyperparameter tuning. The test set is set aside for determining the ultimate performance of the best-performing model on the validation set. Data is split in a stratified fashion, each set having the representative distribution of classes in the dataset.

I started with default categories as they are currently used; however, due to the similarity of categories Front-End Programming, Back-End Programming, Full-Stack Programming and DevOps and Sysadmin, and after agreement with up2staff.com, it was decided to join these categories into one called Programming. Sections in this chapter concentrate on the evaluation and analysis of the performance of Multinomial Naive Bayes and Support Vector Machine algorithms for both default and newly created categories. Furthermore, the performance of Logistic Regression and K-Nearest Neighbors algorithms is assessed specifically for the newly created categories. The F1-score is considered as an indicator of success, and hyperparameters are tuned to optimize the classification results. Additionally, for the default categories, the classification of both title and content of job ads is combined to assess the need to create a new category Programming. For the new categories, a subsection is dedicated to the evaluation of results across classification algorithms, and the best-performing algorithms for title and content are combined for the final decision for categorization. Unless specified differently, all algorithms used in this chapter come from scikit-learn\(^1\) library.

### 3.1 F1 Score

In experiments covered in this chapter we use weighted F1-score from library sklearn.metrics. The F1-score can be interpreted as a harmonic mean of precision and recall, which is especially useful for unbalanced datasets.

The equation for F1-score for binary classification is:

\[
F1 = 2 \times \frac{\text{precision} \times \text{recall}}{\text{precision} + \text{recall}},
\]

\(^{(3.1)}\)

\(^1\)https://scikit-learn.org/stable/
For multi-class classification, the value of this metric is typically calculated for each class and the final metric value is determined by weighing or arithmetically averaging these values.

3.2 Naive Bayes

In the first section, we will investigate the use of two types of NB. First is Multinomial NB evaluated at features created by TF-IDF, BoW and Word2Vec. The second NB is Gaussian NB evaluated on Word2Vec features. Algorithms are implemented in sklearn library at sklearn.naive_bayes.MultinomialNB and sklearn.naive_bayes.GaussianNB.

3.2.1 BoW and TF-IDF

To improve the classification performance of NB, the hyperparameter alpha used for Laplace Smoothing has to be adjusted. Additionally, dimensionality reduction techniques and feature selection methods should be applied to decrease the impact of noise and remove irrelevant features. To achieve this, the hyperparameter max_features available in BoW and TF-IDF methods is tuned. max_features selects top n features based on their frequency. This method filters out noise in data and finds an optimal number of features while maximizing the performance of the model. By selecting only the most relevant features, the complexity of the model is reduced. Furthermore, it can help prevent overfitting by eliminating unnecessary features.

3.2.1.1 Exploration of Classification of Default Categories

Firstly, I explored how well classification works with default categories that are currently used by up2staff.com.

The results of hyperparameters tuning of max_features for the feature creation method and alpha for Naive Bayes are in Table 3.1. BoW created features are better for the title classification with an F1-score of 0.784, and TF-IDF features for content classification with an F1-score of 0.761. The results fell short of the desired level of classification success.

Following the optimization of classification for title and content, the results with the highest F1-score were combined to obtain a final decision of the algorithm. A combination of algorithms is done by decision-level fusion. Chosen algorithms are trained separately for title and content, generating probability vectors for each job ad. Generated probability vectors for title and content are then multiplied by the corresponding weight and added together to create a final decision vector. The weights for title and content decision sum up to 1 (title_weight + content_weight = 1). The category with the highest probability in the final decision vector is selected as the job ad category. This approach is evaluated iteratively for different ratios of title_weight and content_weight, and the final ratio of weights is then chosen by the highest F1-score.

For the NB with default categories, the best-found ratio of weights is 0.5 for both content and title. Combination of decisions for content and title results in the increase of the F1-score to 0.795. The equal weights imply that content and title are equally important for the final classification decision.

To assess the results, I created the confusion matrix exhibited in Figure 3.1a to determine which categories are underperforming. The confusion matrix is normalized for true predictions meaning the numbers are percentages of how many of the given labels in data are correctly categorized. The confusion matrix shows Front-End Programming with only 5.6 % and Back-End Programming with 1.5 % are essentially not categorized. Also, DevOps and Sysadmin is underperforming with a low 16.4 %. Those three categories are massively classified as Full-Stack Programming.

After data exploration, I noticed that Full-Stack Programming has around 2k job ads with the title “Front-End Developer” or “Back-End Developer” or some different variation of this with
### Table 3.1 Results of Multinomial Naive Bayes Tuning with Default Categories

<table>
<thead>
<tr>
<th>Feature Creation Method</th>
<th>Feature Creation <code>max_features</code> and <code>NB alpha</code> Hyperparameters</th>
<th>Validation F1-Score</th>
<th>Data Source</th>
</tr>
</thead>
<tbody>
<tr>
<td>TF-IDF</td>
<td><code>max_features = 9 525</code>&lt;br&gt;<code>alpha = 0.01</code></td>
<td>0.761</td>
<td>Content</td>
</tr>
<tr>
<td>BoW</td>
<td><code>max_features = 13 950</code>&lt;br&gt;<code>alpha = 1e-06</code></td>
<td>0.757</td>
<td>Content</td>
</tr>
<tr>
<td>TF-IDF</td>
<td><code>max_features = 700</code>&lt;br&gt;<code>alpha = 0.1</code></td>
<td>0.769</td>
<td>Title</td>
</tr>
<tr>
<td>BoW</td>
<td><code>max_features = 1 700</code>&lt;br&gt;<code>alpha = 1.0</code></td>
<td>0.784</td>
<td>Title</td>
</tr>
</tbody>
</table>

different levels of Full-Stack background. As the job ads are very similar which was also found during the analysis, joining these four categories into one category `Programming` appears to be a sound decision.

`Management and Finance` with `Product` also have a lower percentage of correctly categorized job ads; nevertheless, these two cannot be clearly joined with one category as they are often classified with a mix of multiple different labels, and the context of these categories is different, meaning the `Management and Finance` job offers do not fit in. On the other hand, `Customer Support`, `Design` with `Sales and Marketing` are categorized sufficiently.

Creation of new category `Programming` reduces the number of categories from nine to six. Due to the massive classification of categories in `Programming` into `Full-Stack Programming`, improvements can be expected in the overall classification. The new category `Programming` is expected to have high classification success as the `Full-Stack Programming` is already very well categorized, and it absorbs other categories which will be part of the merged category. Other categories could have a slight increase; however, they will not be as influenced. This can be projected in an increased F1-score.

#### 3.2.1.2 New Category Programming

In this section, I evaluate classification with a newly created category `Programming`, which is the union of `Full-Stack Programming`, `Front-End Programming`, `Back-End Programming` and `DevOps and Sysadmin`.

The results from tuning Multinomial NB with TF-IDF or BoW are summarized in Table 3.2. According to the table, BoW is the most effective feature creation method for the classification using NB for newly created categories. The F1-score for the classification of the title features is 0.883 and of the content features 0.855. Comparison of Table 3.1 with the original categories mentioned in the previous section and Table 3.2 with the category `Programming` shows an increase in the F1-score by 0.1, which is summarized in last column of Table 3.2. The highest increase in F1-score is with the use of TF-IDF for the title of job ads; however, the F1-score is favoring BoW. Also, the number of features needed for classification decreased as can be seen in lower values of hyperparameter `max_features`. For the content classification, hyperparameter `alpha` decreased, while for a title it remained equal. The value of 1e-10 is the lowest value that ensures the numerical stability of the algorithm due to the use of floating-point numbers\(^2\). The content classification prefers very low `alpha`, which may be even lower than 1e-10. The lower value of `alpha`\(^2\)

Figure 3.1 Comparison of Confusion Matrices for Classification of New and Default Categories

(a) True Predictions Normalized Confusion Matrix for Multinomial NB with Default Categories

(b) True Predictions Normalized Confusion Matrix for Multinomial NB with New Categories
for classification implies that the classification is performing better with less smoothing of the probability estimates for features not present in the training data which can lead to overfitting. F1-score is 0.897 for the final classification achieved by combining the models for content and title with weights of 0.61 for the title and 0.39 for the content. The higher weight given to the classification of the title indicates that with the newly created categories, the decision based on the title plays higher importance. The F1-score increased more than the score of individual classifications by 0.102 from the original 0.795.

The confusion matrix created for the final evaluation of newly created categories is in Figure 3.1b, and shows that percentage of correctly classified categories increased for all categories except Sales and Marketing. Sales and Marketing decreased by 3.1 % because Management and Finance with Product are now more than before classified as Sales and Marketing. On the other hand, Product increased by 5.1 %, and the lowest performing category after creating Programming, Management and Finance increased by 8.4 %. All categories are also less classified into newly created categories Programming than into original ones.

Management and Finance with 54.7 % cannot be improved by merging with one or more categories as it is misclassified into all categories except Costumer Support.

3.2.2 Word2Vec

In previous chapter is explained that Word2Vec tries to capture context and word similarity when creating a feature vector. Due to this, one can expect Word2Vec to improve classification in comparison to TF-IDF and BoW.

Two models of Word2Vec are used for feature creation, differing in training approach, the skip-gram and CBoW model. Also, the selection of three pre-trained models is considered, trained with articles on Wikipedia with different vector sizes and one trained on tweets from Twitter: glove-wiki-gigaword-100, glove-wiki-gigaword-300, glove-twitter-200. Trained and pre-trained models of Word2Vec are from Gensim library, and are combined with Gaussian and Multinomial NB for the classification. The advantage of pre-trained model is its' previous training on large and diverse datasets. Due to the fact that the content has on average 271 words, and the training time is non-trivial for Word2Vec, removing the training part could significantly speed up feature creation. The Word2Vec trained by skip-grams or CBoW approaches is firstly trained on the train set and then used to generate vectors for the documents in train and validation set.

To accommodate the extensive experimentation and tuning of hyperparameters window and vector_size, I deemed it necessary to reduce the dataset size to 60 %. The training and generating

---

<table>
<thead>
<tr>
<th>Feature Creation Method</th>
<th>Feature Creation max_features and NB alpha Hyperparameters</th>
<th>Validation F1-Score</th>
<th>Data Source</th>
<th>Increase of F1-Score</th>
</tr>
</thead>
<tbody>
<tr>
<td>TF-IDF</td>
<td>max_features = 3 975, alpha = 1e-10</td>
<td>0.847</td>
<td>Content</td>
<td>0.086</td>
</tr>
<tr>
<td>BoW</td>
<td>max_features = 5 700, alpha = 1e-10</td>
<td>0.855</td>
<td>Content</td>
<td>0.098</td>
</tr>
<tr>
<td>TF-IDF</td>
<td>max_features = 350, alpha = 0.1</td>
<td>0.877</td>
<td>Title</td>
<td>0.108</td>
</tr>
<tr>
<td>BoW</td>
<td>max_features = 750, alpha = 1.0</td>
<td>0.883</td>
<td>Title</td>
<td>0.099</td>
</tr>
</tbody>
</table>

Table 3.2 Results of Multinomial Naive Bayes Tuning with New Categories

---

3https://radimrehurek.com/gensim/models/Word2Vec.html
of vectors ran in 6-8 processes by the capabilities of the computer I was using; however, training and creating vectors for the documents in the train set and validation set ran up to 200 min for 60% of data. The final vector representing a document was constructed by generating individual vectors for each word and computing a single vector with the mean of all vectors.

3.2.2.1 Gaussian Naive Bayes

The first experiment with features created by Word2Vec was classification by Gaussian NB. For the title, the skip-grams approach had a higher F1-score from trained Word2Vec, but was outperformed by pre-trained model. Gaussian NB for the classification of new categories achieved an F1-score of 0.850 for features generated by the skip-gram model compared to 0.844 for features created by CBoW. Better performing skip-gram ran with hyperparameters of vector size = 20 and window = 1. On the other hand, the F1-score for default categories dropped to 0.450 and below. The pre-trained model had an F1-score of 0.855 using glove-twitter-200, but its score steeply dropped to 0.380 for default categories.

The F1-score fell lower for the content classification. The highest F1-score of 0.713 was achieved by CBoW Word2Vec with hyperparameters vector size = 500 and window = 1. The skip-gram was regularly outperformed by 0.005-0.02. For default categories, the F1-score dropped to a mere 0.193.

The main weakness of Gaussian NB combined with Word2Vec is the drop in F1-score when using default categories. Also, the performance is inferior to any combination of Multinomial NB with BoW or TF-IDF. Due to this, Gaussian Bayes with Word2Vec is not reaching the wanted results for the categorization of job ads.

3.2.2.2 Multinomial Naive Bayes

The second classification algorithm used with Word2Vec-created features was Multinomial NB evaluated in the same fashion as Gaussian NB. For the creation of features both trained and pre-trained Word2Vec were used.

The skip-gram approach demonstrated superior performance in feature creation for the title category. The skip-grams model used for title features favored window = 4, which is more than the length of some titles as the average is 3.54 words. With the vector size of 350 the F1-score was 0.843. However, the difference in the F1-score between sizes of the window hyperparameter with values 3 and 4 is low, and they follow the same trend as can be seen in Figure 3.3a demonstrating the tuning of the model for title classification. This is the only time the Word2Vec features had better performance for a higher value of the hyperparameter window. For title features CBoW model and for the content, both trained models favored a window of size 1. For default categories had skip-gram F1-score of 0.729 for vectors of size 200. The CBoW was outperformed with a small difference in F1-score while pre-trained models exhibited poor performance, with the highest score of 0.668 by pre-trained model glove-wiki-gigaword-300 for newly created categories and approximately 0.500 and lower for default categories. Gaussian NB achieved for the title a higher F1-score than Multinomial NB; on the other hand, the drop in score for default categories is not as high.

For the content categorization, CBoW outperforms skip-gram model by 0.01, and as can be seen in Figure 3.3b CBoW has the most success at vector size = 1250 with 0.842. At vector size of 1250 is a peak in F1-score, and then the F1-score starts to decrease. For default categories, the classification results are very similar for all Word2Vec approaches. F1-score is below 0.5, often achieving only minimal improvements between a tight range of 0.469-0.471. The content classification for the pre-trained Word2Vec model had all similar results with a high using glove-twitter-200 of 0.569 for new categories and 0.471 for default categories. This classification for the content is the worst performing classification.

During the process of tuning the hyperparameter alpha for Multinomial NB, F1-score had a very low difference between 0.0001 and beyond, and the score was often equal after rounding.
Given the lack of improvement and low F1-scores for classification, I did not pursue further tuning and concentrated on finding a more suitable algorithm.

Compared to TF-IDF and BoW, using Word2Vec for feature creation brought about a decrease in F1-score. Not only F1 validation score is lower, but also, if up2staff.com decided to add categories, split categories, or edit categories in any way, it could negatively affect categorization. Due to this, Word2Vec is not going to be further pursued. The inefficiency of Word2Vec, as Word2Vec works on the basis of word similarity and captures the syntactic meaning, could be attributed to the similarity of job offers. The drop in score when using default categories can also hint at it as the merged categories are the most alike. If the job offers were not all IT-related, Word2Vec could achieve better results.

### 3.3 KNN

This section discusses the suitability of KNN for the classification of job ads by title and content. The KNN algorithm used in this section is implemented in `sklearn.neighbors.KNeighborsClassifier`. Furthermore, this section covers the use of different feature engineering techniques, such as dimension reduction or standardization. KNN returns results with higher F1-score for all experiments with hyperparameter `weights` set to ‘distance’. Due to this, the whole section defaultly uses the value ‘distance’ for this hyperparameter.

#### 3.3.1 Feature Reduction

Here we present experiments with PCA and LSA based on the fact from the research section in this thesis and book [1] that mentions that LSA performs better in combination with TF-IDF than PCA. Therefore, I wanted to compare if one reduction algorithm is more suitable for TF-IDF and BoW features.

Considered results were for 5, 10, 15, and 20 neighbors in KNN and components up to 1000 since afterward, increasing the number of components led to a decrease in F1-score. F1-score was higher for a lower number of neighbors and a lower number of components. An example of how PCA (dashed line) and LSA (solid line) develop for TF-IDF and BoW is illustrated in graphs in Figure 3.3. The legend contains used reduction algorithm and the number of neighbors used in KNN classification. Both feature reductions follow the same trend when their reduced features are used with equal hyperparameters in KNN. The graphs also show reduced TF-IDF features suitability in KNN for the content part of the job ad. Except for the sudden dip at
200 components, the TF-IDF has a higher score for both dimension reduction algorithms than BoW. However, the difference in F1-score is around 0.02, which is not groundbreaking. Both algorithms during evaluation performed better for the lower number of neighbors. For some number of components, LSA dominated PCA, and for others, PCA dominated LSA for both feature reduction methods. Thus, it cannot be said one feature reduction algorithm is better than another. As no algorithm outperforms other and both algorithms follow the same trend, I will use PCA.

### 3.3.2 Title Classification

In this subsection we compare TF-IDF and BoW in combination with reduction methods and standardization. To reduce the dimension of features is experimented with two strategies, PCA and most frequent feature selection based on hyperparameter $\text{max\_features}$ in TF-IDF and BoW. Results of tuning are summarized in Table 3.3. As can be seen from the table, the best results are achieved by using $\text{max\_features}$ 1900 and 18 neighbors without standardization. Applying further PCA reduction of features to 160 components and 13 neighbors resulted in a decrease in F1-score to 0.893.

When using standardization with a selection of features based on frequency, KNN achieved the highest performance with 88 neighbors for BoW and 72 for TF-IDF with lower F1-score results. A higher number of neighbors generally means the model is less likely to overfit. However in this case, the majority of data is in category Programming; consequently, the results could start to lean with an increasing number of neighbors to Programming. When applying standardization in combination with PCA, the F1-score falls even lower. The decrease in performance when using standardization is likely due to the fact that scaling reduces the differences between important features with high values which KNN is sensitive to. Consequently, features with high values are not emphasized in the distance and in the classification process. TF-IDF appears to be less sensitive to standardization, possibly because the feature creation process already penalizes very common features. The higher number of neighbors in combination with $\text{max\_features}$ could be KNNs’ way of compensating for high dimensionality and each feature importance having the same influence.

The table also indicates that the information from the title can be reduced by using PCA into a very low number of features effectively representing the title of the job ad with a low decrease in the resulting classification score. The BoW method preferred as little as 20 components for new features, while TF-IDF reduced it to 40. This is a significant reduction compared to the

![Figure 3.3 Comparison for PCA and LSA Feature Reduction on TF-IDF and BoW for 5 and 10 neighbors in KNN](image)
### Table 3.3 Title Categorization Results for KNN

<table>
<thead>
<tr>
<th>F1-score</th>
<th>Hyperparameters</th>
<th>F1-score</th>
<th>Hyperparameters</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.891</td>
<td>PCA - n_components = 20, KNN - n_neighbors = 17, weights = distance</td>
<td>0.889</td>
<td>PCA - n_components = 40, KNN - n_neighbors = 13, weights = distance</td>
</tr>
<tr>
<td>0.897</td>
<td>BoW - max_features = 1900, KNN - n_neighbors = 18, weights = distance</td>
<td>0.885</td>
<td>TF-IDF - max_features = 2600, KNN - n_neighbors = 16, weights = distance</td>
</tr>
<tr>
<td>0.869</td>
<td>Standarization + BoW - max_features = 550, KNN - n_neighbors = 88, weights = distance</td>
<td>0.879</td>
<td>Standarization + TF-IDF - max_features = 1175, KNN - n_neighbors = 72, weights = distance</td>
</tr>
<tr>
<td>0.849</td>
<td>Standarization + BoW - n_components = 105, KNN - n_neighbors = 10, weights = distance</td>
<td>0.858</td>
<td>Standarization + TF-IDF - n_components = 125, KNN - n_neighbors = 10, weights = distance</td>
</tr>
</tbody>
</table>

### Table 3.4 Content Categorization Results for KNN

<table>
<thead>
<tr>
<th>F1-score</th>
<th>Hyperparameters</th>
<th>F1-score</th>
<th>Hyperparameters</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.860</td>
<td>PCA - n_components = 70, KNN - n_neighbors = 10, weights = distance</td>
<td>0.876</td>
<td>PCA - n_components = 90, KNN - n_neighbors = 10, weights = distance</td>
</tr>
<tr>
<td>0.846</td>
<td>BoW - max_features = 525, KNN - n_neighbors = 5, weights = distance</td>
<td>0.853</td>
<td>TF-IDF - max_features = 955, KNN - n_neighbors = 25, weights = distance</td>
</tr>
<tr>
<td>0.818</td>
<td>Standarization + BoW - n_components = 110, KNN - n_neighbors = 6, weights = distance</td>
<td>0.840</td>
<td>Standarization + TF-IDF - n_components = 75, KNN - n_neighbors = 8, weights = distance</td>
</tr>
</tbody>
</table>
initial size of features which was 3083.

The takeaway from this subsection is KNN algorithm has positive response to feature reduction based on selecting the most frequent features created by BoW with the highest F1-score of 0.897. This score is higher by 0.014 compared to Naive Bayes method.

### 3.3.3 Content Classification

I tuned hyperparameter `max_features` for the feature creation algorithm for the selection of features or `n_components` in PCA, and afterward `n_neighbors` in KNN. The F1-score for content classification significantly decreased when using standardization as shown in Table 3.4. This decrease is similar to that observed in title classification and is likely due to the same reason, the reduction of features' importance.

Furthermore, Table 3.4 shows that KNN responds better to PCA feature reduction than tuning based on the most frequent features. This is likely on account of KNNs' sensitivity to the curse of dimensionality. Content has much more information per document compared to the title, and PCA reduces dimension while keeping information which results in a lower number of features with more information compared to simply selecting the most frequent features.

The best results were achieved with reduced TF-IDF features by PCA, resulting in a score of 0.897, a higher F1-score by 0.042 than accomplished by Naive Bayes. Both BoW and TF-IDF in combination with PCA preferred classification using 10 closest neighbors. On the contrary, tuning `max_features` resulted in widely different numbers of neighbors used for classification. TF-IDF preferred 25 neighbors and BoW 5. Since the feature reduction using PCA accomplished better classification performance compared to selecting top features based on their frequency, and standardization reduces the difference between the importance of features, I did not pursue standardization in combination with `max_features`.

### 3.3.4 KNN Overview

In addition to the findings discussed in the previous sections for title and content, there is also an interesting observation regarding the use of the most frequent features in KNN algorithm for title and content. In Table 3.3 for the title and Table 3.4 for content, it is shown that content needs fewer features to classify. There is a difference of 1900 for the title to 525 for content for BoW and 2600 to 955 features for TF-IDF. This is a major difference due to the fact that the title is on average of length 3.5 words while the content 271.7. It suggests that content does not have many representing informative features for categories which could be the reason why the title categorization performs better. Also, it should be noted that the title uses, in general, for the classification larger number of neighbors. However, this is likely due to the higher dimension of title features.

I tried feature-level fusion by combining features for title and content and applying PCA for their common reduction to get one set of features representing a full job ad. However, this resulted in F1-score of 0.83, a score lower than scores of classification using only content or title.

In conclusion, TF-IDF combined with PCA resulted in better classification performance of the job ad for content, while BoW for the 1900 most common features of the title. Classification of title and content has a higher F1-score compared to Naive Bayes.

### 3.4 Logistic Regression

Moving onto the next classification algorithm, this section discusses Logistic Regression using implementation `sklearn.linear_model.LogisticRegression`.

Firstly, I experimented with multiple solvers and penalty L1 (available with use of the Saga solver) and L2 for both title and content. L1 decreased F1-score in general, so the used penalty
in this section is L2. Solvers have equal results with similar computation time, so the rest was evaluated using Sag solver. This left for optimization hyperparameter $C$ which controls the regularization strength with lower values of $C$ corresponding to stronger regularization. Tuning of Logistic Regression classification for content was done on reduced dataset size to 60%, with the final evaluation of the selected hyperparameters done on the full dataset.

### 3.4.1 Title Classification

For Logistic Regression, I did not pursue reduction algorithms for the title. Reduction based on selecting the most prominent features or by reducing dimension using PCA did not improve F1-score. This is likely due to the fact that coefficients of features in Logistic Regression represent the importance of features, and the classification decision is influenced by features with the most positive or the most negative coefficients, effectively selecting the most important features for the given category.

However, to decrease dimension without concentrating on finding the best number of components of features, I increased hyperparameter $\text{min}\_df$ which removes from vocabulary words below this threshold for feature creation methods from default 1 to 5. It resulted in decreasing dimension to a mere 811 to reduce noise and unimportant features (which is equal to reduction using $\text{max}\_features$ with value 811). It did not reduce F1-score, but it decreased the feature
dimension and sped up the algorithm. After increase of \textit{min\_df} further reduction using PCA or \textit{max\_features} decreased F1-score.

The graph in Figure 3.4 shows the impact of the regularization hyperparameter \(C\) on the Logistic Regression classification of the title. The graph shows that BoW outperforms TF-IDF in combination with Logistic Regression. BoW features have stable results for different levels of regularization with the highest F1-score of 0.895 for \(C\) set to 1. This is higher F1-score compared to Naive Bayes but very similar to KNN, which had score of 0.897. TF-IDF had highest F1-score of 0.891 for \(C\) equal to 1 or 10. From the graph can be seen that strong regularization decreases F1-score.

### 3.4.2 Content Classification

For content classification, the feature dimension was reduced using PCA with results displayed in Figure 3.5. To reduce noise, \textit{min\_df} in feature creation algorithm was set to 5. BoW had similar results for all \(C\) values with no value of \(C\) performing remarkably, while TF-IDF unmistakably performed better with \(C\) set to 10. PCA applied to TF-IDF features had no prominent influence on the final score. TF-IDF had a stable F1-score around 0.878-0.880 with different numbers of components, while BoW fluctuated between 0.870 and 0.878. The highest F1-score of 0.879 (0.88 on 60 % of data and 0.879 for 100 %) was achieved with TF-IDF features, PCA reduction to 700 components, and regularization hyperparameter \(C\) set to 10. For the content, Logistic Regression outperforms KNN by 0.004 and Naive Bayes by 0.025.

### 3.5 SVM

Concluding our exploration of classification algorithms in this section the last classification algorithm SVM is explored. I experimented with linear, RBF (Radial Basis Function) and poly kernels. Poly kernel was continuously outperformed by RBF and linear; consequently, I abandoned its exploration and concentrated on tuning of SVM with linear and RBF kernel. RBF kernel could take up to 120 min to calculate, so I tuned hyperparameters \textit{kernel}, \textit{gamma} and \(C\) for content classification on 60 % of the data and lowered \textit{tol} from 0.001 to 0.005. \textit{tol} is hyper-parameter representing the tolerance of change in the decision boundary used as the stopping criteria for the solver. SVM implementation used in this section is \texttt{sklearn.svm.SVC}.

By making trial runs with different values of hyperparameters, I observed that reduction techniques make no significant impact on the final score. Likewise to Logistic Regression classification, I increased \textit{min\_df} from the default value of 1 to 5 to filter noise and insignificant features. The content part of the job ad utilizes features that have reduced dimension by PCA, while the application of PCA to the title features resulted in a decrease in the success of the algorithm.

#### 3.5.1 Linear Kernel

The SVM model with a linear kernel has only hyperparameter for tuning the regularization strength \(C\).

**Title Classification**

Development of F1-score based on hyperparameter \(C\) is in Figure 3.6, containing a comparison of linear (dashed line) and RBF (solid line) kernel in combination with TF-IDF or BoW features. BoW has stable results with different levels of regularization with F1-score in range 0.890-0.893. TF-IDF returns better results for the tuned algorithm with a high of 0.896. Both feature creation methods achieve highest score at \(C=1\).
I also considered PCA and a combination of PCA with BoW features made up of unigrams and bigrams. This resulted in dimension increase from 3,083 to 16,718 and slight decrease of F1-score while extending computing times.

**Content Classification**

For the content part of the job ad, linear kernel is visibly outperformed by RBF. The evolution of reduced features for different values of components and regularization hyperparameter is represented in Figure 3.7. TF-IDF features have the highest F1-score with $C$ set to 1, and BoW features with strong regularization of $C$ set to 0.01. The BoW achieves better results with fewer components, while TF-IDF appears more stable with best performance around 500. After more detailed tuning, both TF-IDF and BoW achieve equal best results at 450 with an F1-score of 0.877 for BoW and 0.876 for TF-IDF, which is the highest score for both algorithms.

I also tried, in the same way as for the title, to boost F1-score using bigrams in combination with unigrams, which increased dimension to 151,000 while only using 60% of data and became uncomputable on my computer. To compensate, I reduced dimension by selecting $n_{\text{max,features}}$ of unigrams and bigrams for BoW and TF-IDF, but it did not improve F1-score.

![Figure 3.6 Impact of SVM Hyperparameter $C$ on F1-score of Title Categorization](image)

![Figure 3.7 Impact of PCA Reduction for SVM on F1-score of Content Categorization](image)
3.5.2 RBF Kernel

SVM with RBF kernel is more flexible for non-linearly separable data than the linear kernel but can be computationally expensive for large datasets. The RBF kernel has an additional hyperparameter \( \gamma \), which determines the kernel width and is tuned in addition to the hyperparameter \( C \). After computing numerous runs for each feature creation algorithm with values of \( \gamma \) parameter: 'auto', 'scale', 0.01, 0.1, 1, and 10, I noticed the best results or, in some cases, equal results of F1-score were obtained in each run for \( \gamma \) set to 'scale'. The TF-IDF features for the title showed similar or nearly equal F1-scores for the same values of hyperparameter \( C \) when the \( \gamma \) was set to 'scale' and 1. SVM applied on BoW features also returned equal results for \( \gamma \) set to 'scale' or 0.1. BoW and TF-IDF for content had significantly higher F1-score for 'scale' over any other value of \( \gamma \). Consequently, tuning was only needed for \( C \) with \( \gamma \) set to 'scale'. The graphs in Figure 3.6 and Figure 3.7 show that RBF kernel (solid line) outperforms linear for both content and title.

Title Classification

Figure 3.6 demonstrates the effect of regularization for features and comparison of SVM classification with RBF kernel and linear kernel. RBF kernel with TF-IDF for the value of \( C \) set to 1 surpasses with validation F1-score of 0.901 all algorithms tested in this thesis and as only algorithm crosses the 0.9 threshold. However, strong regularization has more negative effect on TF-IDF than on BoW features or for linear kernel.

Content Classification

Similarly to the classification of title, TF-IDF surpasses BoW, and both outperform linear kernel with the highest F1-score of 0.886-0.887 between 475-520 components. The final F1-score done on 100 % of data is 0.892 for TF-IDF which is just as the title highest F1-score for content achieved in this thesis.

As the RBF kernel appears to be the most successful for classification of job ads, I tried to tune it further, not only concentrating on \( 10^x \) values of \( C \) but generating random whole positive numbers from the range 1-100. This did not bring about an improvement.

3.5.2.1 Classification of Default Categories

SVM with RBF kernel is algorithm with the highest F1-score for the categorization of job ads. I tuned it again, also for default categories, to find out whether the algorithm would be classifying with satisfactory results.

In a similar fashion to tuning for new categories, F1-score is higher when TF-IDF feature creation method is used and SVM hyperparameter \( \gamma \) is set to 'scale'. For the title, best value of \( C \) is 10, while for content is 100. SVM for default categories performs better with lower regularization compared to new categories. The content dimension was, again, reduced with PCA to the dimension of 500 components. The F1-score for the title is 0.791, while for content is 0.821. The ratio of weights set to probability is 0.2 for the title, and 0.8 for the content with a final F1-score of 0.825. Naive Bayes had lower F1-score by 0.030 for final classification, 0.007 for title and 0.060 for content.

From the confusion matrix in Figure 3.8, it can be seen that in comparison to Multinomial Naive Bayes, the Front-End Programming, Back-End Programming with DevOp and Sysadmin are classified significantly better. This may suggest that SVM algorithm is more effective for classifying job offers into IT-related job categories than the Naive Bayes algorithm. The percentages of correctly classified categories from merged category Programming showed significant improvement. Back-End Programming increased from 5.6 % to 33.6 %, Front-End Programming from 1.5 % to 33.6 % and DevOps and Sysadmin from 16.4 % to 26.9 %. This is steep and effective growth for all categories except Full-Stack Programming and Design; however, the categorization success rate for default categories is still inadequate and cannot be used.
3.6 Final Evaluation

Now, we compare the best-performing classification algorithms found for newly created categories, selecting the most successful and using them for the final decision on the categorization of job ads.

3.6.1 Title Classification

The tuned classification algorithms for the title are summarized in Table 3.5. The table shows the difference between Linear Regression and SVM compared to KNN and Naive Bayes is around 0.05 with SVM as the most precise classification algorithm.

The confusion matrix for SVM in Figure 3.9 shows that Customer Support, Design, Programming and Sales and Marketing are categorized correctly in more than 80% cases, and Management and Finance with Design are classified around 50% correctly. Management and Finance is misclassified into Sales and Marketing the most, while Product into Management and Finance. Both categories are also categorized as Programming absorbs most of the incorrectly classified predictions. In confusion matrix can also be seen that Design is never predicted as Costumer Support, and Costumer Support is never predicted as Product.

3.6.2 Content Classification

From the overview in Table 3.6, it is apparent that for the content classification SVM with the use of RBF kernel is unambiguously the top-performing algorithm. The confusion matrix of the algorithms is in Figure 3.10.
Table 3.5 Title Categorization Algorithm Comparison

<table>
<thead>
<tr>
<th>Feature Creation Method</th>
<th>Feature Reduction Method</th>
<th>Classification Algorithm</th>
<th>Validation F1-score</th>
</tr>
</thead>
<tbody>
<tr>
<td>BoW</td>
<td>TF-IDF</td>
<td>KNN</td>
<td>0.858</td>
</tr>
<tr>
<td>max_features = 1900</td>
<td>frequency based</td>
<td>n_neighbors = 18</td>
<td></td>
</tr>
<tr>
<td>BoW</td>
<td>-</td>
<td>Logistic Regression</td>
<td>0.895</td>
</tr>
<tr>
<td>min_df = 5</td>
<td>-</td>
<td>solver = sag</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>C = 1</td>
<td></td>
</tr>
<tr>
<td>BoW</td>
<td>BoW</td>
<td>Multinomial NB</td>
<td>0.846</td>
</tr>
<tr>
<td>max_features = 1550</td>
<td>frequency based</td>
<td>alpha = 1e-07</td>
<td></td>
</tr>
<tr>
<td>TF-IDF</td>
<td>-</td>
<td>SVM</td>
<td>0.901</td>
</tr>
<tr>
<td>min_df = 5</td>
<td></td>
<td>kernel = RBF</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>gamma = scale</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>C = 1</td>
<td></td>
</tr>
</tbody>
</table>

Figure 3.9 True Predictions Normalized Confusion Matrix for Title Categorization Using SVM
Table 3.6 Content Categorization Algorithm Comparison

<table>
<thead>
<tr>
<th>Feature Creation Method</th>
<th>Feature Reduction Method</th>
<th>Classification Algorithm</th>
<th>Validation F1-score</th>
</tr>
</thead>
<tbody>
<tr>
<td>TF-IDF min_df = 5</td>
<td>PCA n_components = 90</td>
<td>KNN n_neighbors = 10</td>
<td>0.876</td>
</tr>
<tr>
<td>TF-IDF min_df = 5</td>
<td>PCA n_components = 700</td>
<td>Logistic Regression</td>
<td>0.879</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>solver = sag</td>
</tr>
<tr>
<td>BoW max_features = 4280</td>
<td>BoW frequency based</td>
<td>Multinomial NB</td>
<td>0.846</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>alpha = 0.9</td>
</tr>
<tr>
<td>TF-IDF min_df = 5</td>
<td>PCA n_components = 510</td>
<td>SVM</td>
<td>0.892</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>kernel = RBF gamma = scale</td>
</tr>
</tbody>
</table>

Figure 3.10 True Predictions Normalized Confusion Matrix for Content Categorization Using SVM
From the confusion matrix for content, it is clear that the algorithm categorizes correctly in more than 80% Costumer Support, Sales and Marketing and Programming. Category Design achieved satisfactory results with 74% of correctly categorized job ads. Management and Finance with Product have lower success rates with less than half (49.4%) of correctly categorized job ads for Management and Finance and slightly above half for Product (54%) which is slightly higher compared to title categorizations.

From the matrix, it is clear that each category is on some level incorrectly identified as Programming category. Management and Finance is most often miscategorized as Programming with almost one-fourth of job ads (24.3%) and with 17.4% as Sales and Marketing. Both categories make up above 40% of incorrect categorizations of Management Finance. Sales and Marketing have the lowest number of incorrectly categorized job ads into Programming with only 3% of job ads. Product, the second most poorly categorized section with 54% success rate, is often placed into Programming, and Management and Finance.

Compared to the confusion matrix for the title classifications in Figure 3.10, content has lower success with Design by 8.8%. Also, Costumer Support and Sales and Marketing have a lower percentage of correct classification. On the other hand, the worst classified Management and Finance with Product have slightly higher percentages. Management and Finance is in title classification more classified as Sales and Marketing while in content classification as Programming, and joining of title and content classifications could improve this category.

### 3.6.3 Final Classification

This section discusses the combined results of the best classification models for the title and content data. Furthermore, the final F1-score and confusion matrix are presented to assess the algorithm’s performance.

The validation F1-score for the combination of the content and title classification results is 0.911. The model performed with the highest F1-score for weights of 0.45 for the title decision and 0.55 for the content decision. However, the Management and Finance category has only 50.1% of correctly categorized job ads. Due to the low classification accuracy of Management and Finance, I added an offset (a weight of 0.2 added to the content decision for Management and Finance) to improve the results. Content classifies correctly 49.4% of Management and Finance job ads while title 48.6%. When combing title and content with offset, validation F1-score increases to 0.913 with weights 0.46 for the title decision and 0.54 for the content decision, and Management and Finance has 55.7% of job ads classified correctly with low impact on other categories. Due to the increase of the F1-score and percentage of Management and Finance correctly categorized job ads, I am using offset in the final algorithm. Additionally, to accommodate the growing dataset over time, the fixed value of 5 for min_df is replaced with a relative proportion of 0.0002, which is equivalent considering the size of the dataset during training. This adjustment helps to reduce noise, improve the accuracy of the feature representation and reduce overfitting with increasing dataset size in the future.

Now we will evaluate the selected best-performing classification model in this thesis. For this part, up2staff.com supplied new data. The dataset size increased from 38 504 to 40 521. The new job ads 2 017 are approximately 0.05% of overall data and serve as a foundation for the test set. The original dataset is split 85-15 for the train-test set, and new data are added to the test set, creating a new test set with approximately 20% of data (the created train-test ratio is 0.807-0.193). The combination of applied algorithms is summarized in Table 3.7. The final classification model has the test F1-score of 0.909, and its confusion matrix normalized for true predictions is in Figure 3.11. The confusion matrix follows confusion matrix trends for validation data in Figure 3.10 and Figure 3.9. The model accurately predicts Programming, Sales and Marketing and Costumer Support. Afterward, Design has acceptable predictions. However, Product and Management and Finance are underperforming. Management and Finance especially with 60.0% of correctly predicted job ads.
Table 3.7 Best Performing Algorithms for Text and Content

<table>
<thead>
<tr>
<th>Data Source</th>
<th>Feature Engineering Methods</th>
<th>Classification Algorithm</th>
<th>Test F1-score</th>
<th>Weight of Decision</th>
</tr>
</thead>
<tbody>
<tr>
<td>Title</td>
<td>TF-IDF min_df = 0.0002</td>
<td>SVM</td>
<td>0.895</td>
<td>0.46</td>
</tr>
<tr>
<td></td>
<td></td>
<td>kernel = RBF</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>gamma = scale</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>C = 1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Content</td>
<td>TF-IDF min_df = 0.0002</td>
<td>SVM</td>
<td>0.895</td>
<td>0.54</td>
</tr>
<tr>
<td></td>
<td>PCA n_components = 510</td>
<td>kernel = RBF</td>
<td></td>
<td>+ 0.2 offset</td>
</tr>
<tr>
<td></td>
<td></td>
<td>gamma = scale</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>C = 10</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Due to the low performance of the classification model for Product and Management and Finance, I went through its incorrectly categorized data. For Management and Finance, job ads assigned to Programming are written similarly to developer position, but it is a position of manager of the developer team. Those are hard to distinguish as many words point to Programming. For Sales and Marketing, the problem with categorization is the same as with Programming. The job ad fits in Sales and Marketing, but it is a manager position. A possible solution could be to increase the weight of the feature manager in algorithms. Another solution could be to remove the category for managers altogether because the context of the predicted category fits.

Incorrectly categorized Product is sometimes related to Product but not entirely, for example, analysts and testers. Furthermore, some of the job ads fit in the predicted category better. For example, Product contains manager positions that should be in Management and Finance, and engineer or developer positions which should be in Programming. After reading through incorrectly predicted job ads, the algorithm performs for this category better than it seems from the confusion matrix.

Turning back to the analysis presented in the Data Preparation and Analysis chapter. It is clear from the confusion matrix that the categories of Customer Support and Sales and Marketing are well classified as predicted. However, the category of Design is more challenging to recognize than initially expected, with 76.5% accuracy. Categories within the Programming category are indeed difficult to separate due to their similar characteristics. They were merged into one due to meager categorization success. Moreover, as expected, it is challenging to classify the categories of Management and Finance and Product.
Figure 3.11 True Predictions Normalized Confusion Matrix of Classification Model
Chapter 4

Implementation

The final script has functionalities as required by up2staff.com. The requirements were written by the thesis supervisor and are described in issues on GitHub (see Appendix). One issue specifies the creation and tuning of the classification model, while the other use of the trained model for evaluation. The final script is made up of five modules, as described in the following paragraphs.

**database_utils**

This module provides functionality for connecting to a MySQL database, creating a new database from a given SQL file, and reading data from the created database into a Pandas DataFrame. For database operations, the following libraries are used: SQLAlchemyn, PyMySQLz, and mysqlw.

**preprocessing**

This module provides functions for preprocessing text data in ‘post_content’ and ‘post_title’ using the spaCy library and NLTK. Function preprocess_data() takes a DataFrame as input and returns a modified DataFrame with preprocessed columns. Text is tokenized, lemmatized, lower-cased, and non-words and stop words are removed. All preprocessing steps are done using spaCy; however, the stop-word list is taken from NLTK library due to the fact that it additionally contains stop words such as “ourselves”, “should”, and “would”.

**model**

Module’s functionality is to train, create, and save the final classification model for further use in evaluate_job_ad module. The classification model is built using library scikit-learn. Important functions for the creation of the final classification model are:

1. tf_idf(): Vectorizes the input text data using the TF-IDF.
2. run_pca(): Applies PCA to the input data.
3. run_svm(): Trains an SVM model using the radial basis function kernel for given hyperparameter \( C \).
4. tune_svm(): Tunes the hyperparameter \( C \) for the SVM model.
5. tune_ratio(): Tunes the ratio of weights between the probabilities from the title and content SVM models.

---

1https://www.sqlalchemy.org
2https://pymysql.readthedocs.io/en/latest/
4https://spacy.io/api
5https://www.nltk.org
6. `evaluate_model()`: Prints F1-score and creates a confusion matrix for the evaluation of the performance of the classification model.

7. `save_model()`: Trains the final classification model and saves the output SVM models for title and content together with the PCA model, the weights ratio for title and content, and the TF-IDF model using library `pickle`.

**support_functions**

This module provides functions for visualizing a confusion matrix and splitting data into training, testing, and optionally validation sets.

**evaluate_job_ad**

This module contains an evaluation function for job ad classification `evaluate()`. It preprocesses the given title and content of the job ad and loads saved SVM, TF-IDF and PCA models to calculate probabilities of the job ad’s categories.

Figure 4.1 illustrates the data flow and calls between modules to create the classification model and evaluate job ad. The creation of the model starts with loading data from the database and transforming it to Pandas DataFrame in module `database_utils`. Then text in columns ‘post_content’ and ‘post_title’ is preprocessed using module preprocessing. The preprocessed DataFrame is then passed to the module `model`, where the final classification model is trained, created, and saved in the file “`model.pickle`”. The `model` utilizes several essential functions to achieve the creation of the classification model, such as `tf_idf()`, `run_pca()`, and `save_model()` with support functions from module `support_functions`. Finally, the `evaluate_job_ad` module loads the trained model, preprocesses a new job ad’s text, and evaluates the different categories’ probability.

---

6[https://docs.python.org/3/library/pickle.html](https://docs.python.org/3/library/pickle.html)
Figure 4.1 Overview of Modules in Implementation
The objective of this thesis was to find a suitable solution for the categorization of job ads. To fulfill this objective, research and examination of feature engineering techniques and supervised machine learning classification algorithms were pursued using analyzed and preprocessed datasets to experiment with.

The results showed that SVM with RBF kernel for TF-IDF features performed best as measured by F1 score. The classification model combines the decision for the job content and title to get the final decision with weights of 0.46 for the title and 0.54 for the content. Furthermore, to improve the classification of the underperforming category Management and Finance, an offset of 0.2 was added to give a higher weight to content decisions. The final classification model has F1-score of 0.909. Although the algorithm had the lowest percentage of correctly categorized postings of 60 % for the Management and Finance category, consolidating Full-Stack Programming, Front-End Programming, Back-End Programming and DevOps and Sysadmin categories into a single Programming category improved the percentage to 98 % for those categories, a significant increase from lowest 19.7 % for Back-End Programming. All objectives of this thesis were achieved, and as an outcome, the classification model written in Python was given to up2staff.com.

In conclusion, the automatic job ads classification model developed in this thesis provides a systematic, effective and unbiased approach to categorization. This model will not only enhance the efficiency of job posting categorization on up2staff.com but also help job seekers in finding relevant opportunities and aid employers in finding suitable candidates. Future research can focus on refining the classification model for the Management and Finance category, creating new categories or removing existing ones, and exploring other feature engineering and classification techniques to improve the model’s performance further.
Appendix A

Requirements for Implementation

**Input**: Data from the database. Ideally in existing SQL file form. The script can expect data in the data.sql file e.g. in the same directory.

**Output**: Learned best model trained on this data. The model should be stored in a file, for example using the pickle package.

- The necessary packages should be described in the requirements.txt file and installed using the pip command.
- At least a brief tutorial should be available. The script should be commented.
- It is not necessary to do hyperparameter tuning, but maybe some basic tuning is possible. However, the best model resulting from the work should be used.
- The list of categories should be taken from the database only. i.e. it should handle the fact that a new category appears in the database.
- It would be good for the script to output some test error, e.g. the following can be done:
  1. Split the data into training and testing (e.g. 10%).
  2. Train the model on the training data and output the results on the test data (e.g. F1 score and confusion matrix).
  3. Then train the model again on all data, i.e. add the test data.

(a) Script to train and create a model

**Input**: Data from the database. Ideally in existing SQL file form. The script can expect data in the data.sql file e.g. in the same directory.

**Output**: Learned best model trained on this data. The model should be stored in a file, for example using the pickle package.

- The necessary packages should be described in the requirements.txt file and installed using the pip command.
- At least a brief tutorial should be available. The script should be commented.
- It is not necessary to do hyperparameter tuning, but maybe some basic tuning is possible. However, the best model resulting from the work should be used.
- The list of categories should be taken from the database only. i.e. it should handle the fact that a new category appears in the database.
- It would be good for the script to output some test error, e.g. the following can be done:
  1. Split the data into training and testing (e.g. 10%).
  2. Train the model on the training data and output the results on the test data (e.g. F1 score and confusion matrix).
  3. Then train the model again on all data, i.e. add the test data.

(b) Script for category prediction

- Figure A.1 Issues for Implementation of Final Categorization Script
Requirements for Implementation


Contents of Enclosed Archive

up2staff-job-categorization..............................directory with practical part
  _implementation........directory containing implementation submitted to up2staff.com
  _experiments.....directory containing code and Jupyter Notebooks used in experiments
  _data........................directory with data in the form of CSV files
  _tests........................pytest for the analytical part
  README............................short overview of the folder
  text..............................text of thesis
  BP_Petrilakova_Patricie_2023.tex........directory of \LaTeX{} source codes of the thesis
  BP_Petrilakova_Patricie_2023.pdf........pdf file with text of thesis