BRNO UNIVERSITY OF TECHNOLOGY
VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ

FACULTY OF INFORMATION TECHNOLOGY
FAKULTA INFORMAČNÍCH TECHNOLOGIÍ

DEPARTMENT OF COMPUTER GRAPHICS AND MULTIMEDIA
ÚSTAV POČÍTAČOVÉ GRAFIKY A MULTIMÉDIÍ

DOMAIN SPECIFIC DATA CRAWLING FOR LANGUAGE MODEL ADAPTATION
ADAPTACE JAZYKOVÉHO MODELU NA CÍLOVOU DOMÉNU VYUŽÍVAJÍCÍ STAHOVÁNÍ VEŘEJNÝCH DAT

MASTER'S THESIS
DIPLOMOVÁ PRÁCE

AUTHOR **Bc. SABÍNA GREGUŠOVÁ**
AUTOR PRÁCE

SUPERVISOR **Ing. MARTIN KARAFIÁT, Ph.D.**
VEDOUCÍ PRÁCE

BRNO 2021

Vysoké učení technické v Brně
Fakulta informačních technologií

Ústav počítačové grafiky a multimédií (UPGM) Akademický rok 2021/2022

Zadání diplomové práce

24957

Studentka:	**Gregušová Sabína, Bc.**
Program:	Informační technologie
Obor:	Zpracování zvuku, řeči a přirozeného jazyka
Název:	**Adaptace jazykového modelu na cílovou doménu využívající stahování veřejných dat** **Domain Specific Data Crawling for Language Model Adaptation**
Kategorie:	Zpracování řeči a přirozeného jazyka

Zadání:

1. Seznamte se jazykovým modelováním v systémech pro automatický přepis řeči.
2. Seznamte se přístupy pro analýzu textu a automatické stahování.
3. Navrhněte a vytvořte systém, který analyzuje data z cílové domény a automaticky stáhne podobná data z veřejných zdrojů.
4. Data automaticky vyčistěte a adaptujte obecny jazykový model.
5. Rozšiřte systém pouze na specifikaci cílové domény.
6. Experimentujte s různými systémovými parametry a otestujte jazykovou nezávislost.

Literatura:

- Le Zhang, Damianos G. Karakos, William Hartmann, Roger Hsiao, Richard M. Schwartz, Stavros Tsakalidis: "*Enhancing low resource keyword spotting with automatically retrieved web documents*". INTERSPEECH 2015

Při obhajobě semestrální části projektu je požadováno:

- Splnění prvních 3 bodů zadání.

Podrobné závazné pokyny pro vypracování práce viz https://www.fit.vut.cz/study/theses/

Vedoucí práce:	**Karafiát Martin, Ing., Ph.D.**
Vedoucí ústavu:	Černocký Jan, doc. Dr. Ing.
Datum zadání:	1. listopadu 2021
Datum odevzdání:	18. května 2022
Datum schválení:	2. listopadu 2021

Abstract

The goal of this thesis is to implement a system for automatic language model adaptation for Phonexia ASR system. System expects input in the form of source that, which is analysed and appropriate terms for web search are chosen. Every web search results in a set of documents that undergo cleaning and filtering procedures. The resulting web corpora is mixed with Phonexia model and evaluated. In order to estimate the most optimal parameters, I conducted 3 sets of experiments for Hindi, Czech and Mandarin. The results of the experiments were very favourable and the implemented system managed to decrease perplexity and Word Error Rate in most cases.

Abstrakt

Cieľom práce je implementovať systém pre automatickú adaptáciu jazykového modelu pre Phonexia ASR systém. Systém prijíma vstupný súbor, ktorý analyzuje a vyberie vhodné výrazy pre webové vyhľadávanie. Každé webové vyhľadávanie prináša množinu dokumentov, ktoré podstupujú čistenie a filtrovanie. Výsledný webový korpus sa zmieša s Phonexia modelom a vykoná sa evaluácia. Pre odhad optimálnych parametrov boli vykonané viaceré experimenty pre hindštinu, češtinu a mandarínsku čínštinu. Výsledky experimentov boli pozitívne a implementovaný systém bol schopný znížiť perplexitu a Word Error Rate vo väčšine experimentov.

Keywords

speech-to-text, automatic speech recognition, language model, language model adaptation, automatic web search, automatic web document scraping, automatic assessment of web documents

Kľúčové slová

speech-to-text, automatické rozpoznávanie reči, jazykový model, adaptácia jazykového modelu, automatické prehľadávanie webu, automatické čistenie webových dokumentov, automatické vyhodnotenie webových dokumentov

Reference

GREGUŠOVÁ, Sabína. *Domain Specific Data Crawling for Language Model Adaptation.* Brno, 2021. Master's thesis. Brno University of Technology, Faculty of Information Technology. Supervisor Ing. Martin Karafiát, Ph.D.

Rozšírený abstrakt

Reč je jedna z najdôležitejších prostriedkov ľudskej komunikácie. S vývojom technológii sa ľudia začali zaujímať o možnosť spracovania reči pomocou digitálneho počítača. Táto oblasť sa veľmi rozšírila a očakáva sa, že v budúcnosti budú existovať počítače, ktoré sa budú ovládať hlasom.

Takéto systémy sa globálne nazývajú ako systémy pre automatické rozpoznanie reči (Automatic Speech Recognition systems). V dnešnej dobe sa vyvíjajú systémy najmä pre prepis hovorenej reči, tzv. `Speech-To-Text` (STT) systémy. Ich úlohou je transformovať vstupnú nahrávku na jej čo najpravdepodobnejší prepis. Takéto systémy okrem iného vyvíja aj brnenská firma `Phonexia`, s ktorou som spolupracovala pri písaní tejto práce.

Automatické systémy pre rozpoznanie reči sú jazykovo závislé a skladajú sa z 2 modelov: akustický model so slovníkom a jazykový model. Úlohou akustického modelu je detekovať jednotlivé fonémy v krátkych úsekoch (25 ms) nahrávky. Tieto fonémy sa podľa slovníka prepíšu na slová. Celé vety sú potom ohodnotené jazykovým modelom. Jazykový model v sebe združuje informácie o konkrétnom jazyku, aká je jeho syntax, sémantika a ktoré slová sa často vyskytujú spolu. Veta, ktorá je logický správne bude jazykovým modelom ohodnotená ako veľmi pravdepodobná, zatiaľ čo veta s náhodnými slovami bude ohodnotená ako málo pravdepodobná.

Firma Phonexia vyvíja presne takéto systémy a okrem nich ponúka aj služby pre adaptáciu oboch modelov na cieľovú doménu. Adaptácia je veľmi dôležitá v prakticky nasadených systémoch. Môže totiž nastať situácia, že daný STT systém má vysokú presnosť na evaluačných datasetoch, ale po nasadení u zákazníka môže byť presnosť oveľa horšia. Toto sa deje najmä v prípadoch, keď zákazník používa STT vo veľmi špecifickej doméne, ktorá nebola zahrnutá v dátach pre trénovanie. Zákazník teda subjektívne hodnotí STT systém ako nevyhovujúci pre svoje potreby.

Táto práca sa zaoberá automatickým adaptovaním jazykového modelu na cieľovú doménu pomocou sťahovania verejných dát. Štandardne adaptácia vyžaduje získanie nového datasetu, jeho transformáciu, čistenie a následná tvorba jazykového modelu. Tento proces je veľmi náročný na čas aj ľudské zdroje, preto by sme ho chceli zjednodušiť.

Pre tento účel je implementovaný systém zreťazeného spracovania, tzv. `pipeline`. Táto pipeline je na základe súboru z cieľovej domény schopná stiahnuť podobné dáta. Vstupný súbor z cieľovej domény prejde analýzou a vyberú sa najvhodnejšie `n-tice` pre webové vyhľadávanie. Výber najvhodnejších `n-tíc` je ovplyvnený ich četnosťou výskytu v súbore a celkovou dĺžkou. Najoptimálnejšie `n-tice` sú použité vo webovom vyhľadávaní a pre každú `n-ticu` skript získa množinu vhodných `url linkov`.

Každý webový dokument prechádza niekoľkými štádiami kontroly a čistenia, pretože chceme zamedziť pridávaniu irelevantných alebo špinavých dát do jazykového modelu. Po získaní webového dokumentu je z neho vyextrahovaný text zo všetkých žiadaných webových tagov. Tento text je očistený od akéhokoľvek `HTML markupu` a následne prechádza čistením. Vrámci čistenia sa text normalizuje tak, aby výsledný korpus bol v porovnateľnej kvalite, akú majú čisté dáta vo Phonexii.

Čistý text najskôr podstúpi jazykovú identifikáciu. Ak prejde cez túto fázu, je posúdená jeho relevancia pomocou perplexity. Ak text uspeje, je možné ho ďalej pokročilo filtrovať na úrovni paragrafov alebo celého dokumentu. Výsledný text je zapísaný do webového korpusu.

Po vytvorení webového korpusu sa pripraví jeho jazykový model. Tento model sa zmixuje s pôvodným modelom z `Phonexie` a získame adaptovaný jazykový model. Pre rýchle porovnanie sa počas evaluácie vždy hodnotí perplexita pôvodného aj adaptovaného modelu.

Celá pipeline je ovládaná pomocou konfiguračného súboru s množstvom vstupných parametrov. Pre výber najoptimálnejších parametrov som vykonala 3 sady experimentov s rôznymi jazykmi. Chcela som overiť aj jazykovú nezávislosť, preto boli vybrané nasledujúce, vzájomne veľmi odlišné jazyky: hindština, čeština a mandarínska čínština.

Hindština nie je momentálne vyvíjaná vo Phonexii, preto bol s pipeline iba vytvorený webový korpus, ale mixovanie modelov bolo uskutočnené s Kaldi. Napriek tomu, že pôvodná perplexita bola relatívne nízka, nový model znížil perplexitu až o 7% a Word Error Rate o 1%.

Čeština má pochopiteľne vo Phonexii veľmi vysoký štandard kvality. Pre adaptáciu som zvolila technický dataset Phonexie z oblasti automobilového priemyslu. V týchto experimentoch bol skript schopný znížiť perplexitu až o 73.9% a Word Error Rate, teda mieru chybovosti slov, o 8.7% pre celý ASR systém.

Posledná sada experimentov bola vykonaná s mandarínskou čínštinou. Čínština je zo svojej podstaty veľmi neštandardný jazyk, preto som chcela otestovať, či bude skript dosahovať porovnateľné výsledky. Za dataset som zvolila jeden z Phonexia čínskych datasetov pre prirodzenú spontánnu reč. Pôvodná hodnota perplexity bola tak nízka, že sa jej zlepšenie ani nepredpokladalo. Experimenty potvrdili tento predpoklad, ale napriek tomu priniesli dôležitý náhľad na spracovanie neštandardných jazykov. Na základe dosiahnutých výsledkov verím, že pipeline má potenciál dosiahnuť dobré výsledky aj pre mandarínčinu ak by bol zvolený viac špecifický dataset.

Výsledky experimentov globálne hodnotím ako veľmi pozitívne. Práca potvrdila, že je možné využiť implementovanú pipeline pre automatickú adaptáciu jazykového modelu. Implementácia bude pridaná do repozitárov Phonexie a dúfam, že zlepší a zjednoduší tento proces pre mojich kolegov.

Domain Specific Data Crawling for Language Model Adaptation

Declaration

I declare that I wrote this master thesis on my own under the supervision of Ing. Martin Karafiát Ph.D. My colleagues from Phonexia have kindly provided me with other necessary information. I listed and cited all sources that were used in this thesis.

. . . .
Sabína Gregušová
May 17, 2022

Acknowledgements

I would like to thank my supervisor Ing. Martin Karafiát Ph.D. for his advice and support throughout this project. Next, I would like to thank my colleagues from Phonexia, who were always very kind and helpful. Lastly, I would like to thank my parents, Pavol and Anetta, and my boyfriend Filip for their maximal support and kindness during this difficult process.

Contents

Chapter 1

Introduction

Speech is one of the most natural and easiest mean of human interaction. Therefore, it is no surprise that for decades, humans have tried to analyze, capture and recognize spoken language with digital computers. It is expected that at some point in the future, there will be a full human-computer interface based solely on speech, making it a very promising field of research. *Automatic Speech Recognition* (ASR) system is implemented to transcribe spoken speech to its written form.

Typical ASR system consists of two components: *acoustic model* with *lexicon* and *language model*. Each component models different aspects of speech. In practice, both of these models can be *dynamically adapted* to various domains instead of re-training them from scratch. This is especially useful for real-life deployed systems, where re-training the model again and again would be unfeasible.

Real-life systems also encounter a different set of problems, namely customer's dissatisfaction with the system even though the accuracy on evaluation data was high. If a customer provides the target domain data, then the adaptation is fairly straightforward. However, there is often no customer data for the target domain adaptation, so a lot of extra labor to obtain and pre-process the target data is needed. This thesis aims to ease this issue by implementing a pipeline that automatically downloads and cleans web documents containing the target domain data for the *language model adaptation*. It explores *language modelling* and implements a fully automatic *language model adaptation pipeline* for a real ASR system of Brno's speech recognition company Phonexia.

Firstly, chapter 2 introduces all the theoretical prerequisites, namely: speech creation, speech propagation and perception and general architecture of any ASR system and its components.

Secondly, chapter 3 deals with language modelling, its methods, approaches, smoothing techniques, evaluations and difficulties in real-life applications.

Chapter 4 contains the design and implementation of the *language model adaptation pipeline*. The pipeline is implemented through multiple different components and each of them can be used separately. It includes description of every component of the pipeline, their limitations and some practical information for their usage.

Finally, chapter 5 presents a series of experiments conducted with the pipeline. The experiments were conducted on Hindi, Czech and Mandarin Chinese.

Lastly, chapter 6 concludes the thesis with a summary of this work. It briefly describes the experiments' results and pitches some ideas for further research.

Chapter 2

Automatic Speech Recognition

Automatic Speech Recognition (ASR), often also referred to as *Speech-To-Text* (STT) or *Speech recognition*, represents a field of computer science that deals with spoken language, its processing and recognition. The main goal of such system is to transcribe spoken speech in audio format to to its most probable written form, which can be processed even further (foreign language translation, sentiment analysis, dialogue interaction). This saves time and makes human actions more efficient, therefore, it is expected that there will be an increase in the use of speech technology in the future.

This chapter provides a brief overview of the most important theoretical prerequisites for the latter chapters. Each section contains references to multiple publications for further details. Section 2.1 provides fundamental knowledge about speech creation, propagation and perception. Section 2.2 deals with storing and processing the input speech signal inside digital computers. Section 2.3 explains the general architecture of automatic speech recognition system and its components. Lastly, section 2.5 introduces *acoustic modelling* with *lexicon* and some typical modelling approaches. Since *language modelling* is the central subject of this thesis, it is explored in detail in the separate chapter 3.

2.1 Speech

Speech is one of the most fundamental means for human interaction and communication. On an average day, humans can say up to 20000 words. Fundamentally, speech can be perceived as a series of sound signals that convey an idea of the speaker. The average person is equipped with organs that allow for natural speech production, with the main one being the *vocal tract*, *vocal folds* and *lungs*. Speech production starts with air in the lungs as the source of energy, which is then pushed up through the trachea and is modified by the *vocal folds* and the *vocal tract*. *Vocal folds* are two small muscles located in the throat and are part of the respiratory system. They can be closed and as a result vibrate when air passes through, which produces *voiced sounds*, or they can be opened, thus too far away to vibrate, which produces *unvoiced sounds*.

Voiced sounds have regular, almost periodic pattern in the time domain. All vowels and diphthongs (combination of two neighboring vowel sounds into one) are voiced. *Unvoiced sounds* have very irregular structure similar to noise. Consonants can be voiced or unvoiced. It is easy to distinguish between the two types by placing a hand on the throat when speaking. If it is possible to feel the vibrations of the vocal cords, then the consonant is voiced, otherwise it is unvoiced.

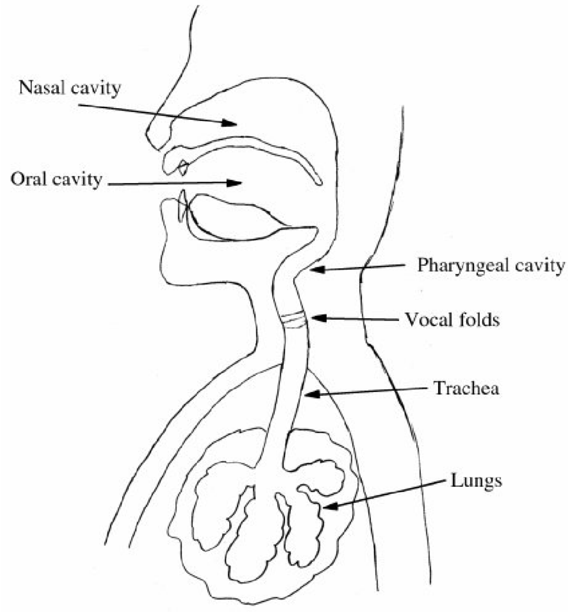


Figure 2.1: Human speech production system. Taken from [5].

The physical structure of the vocal folds directly influences *fundamental frequency* (F0). Fundamental frequency is subjectively perceived as a *pitch* of the voice. Male voices have $F0$ in the range of 80 – 150 Hz while for female voices, it is usually one octave higher at a range 160 – 250 Hz. Moreover, children's voices can have fundamental frequency of 300 Hz.

Finally, when the air reaches the vocal tract, it can pass through two pathways: *oral tract* or *nasal tract*. The resulting sounds are heavily influenced by various positioning of *tongue*, *teeth* and *lips* [15], [10].

The smallest unit of speech that distinguishes words from each other is called a *phoneme*. An acoustic realization of a *phoneme* is called a *phone*. All phonemes are grouped and standardized by the `International Phonetic Alphabet` (IPA), which represents all phonemes by a set of symbols. The speech production process is universal and does not depend on the language, but each language uses a characteristic subset of phonemes.

Speech propagation

Speech, as any other sound signal, travels through air in the form of *waves*. The speed of sound (distance travelled per unit of time) is influenced by various factors, but mainly temperature and the medium which the sound wave travels through. As the sound waves pass through space, they lose energy. In $20°C$, the speed of sound is believed to be approximately 343 m/s, but it is not always the same. The closer the molecules are to each other in the transport medium, the easier it is for the sound waves to pass through. That's why it is easier for a sound wave to pass through solids, rather than liquids or gases.

Additionally, the frequency of the sound wave also influences how it interacts with objects that appear in its way. High frequency sound waves are absorbed through concrete walls, whilst lower frequency sound waves pass through with minimal absorption. Therefore, structure of the sound wave at any point in space and time is greatly influenced by many external factors.

Essentially, sound wave causes changes in the air pressure. These changes can be picked by a *microphone* and converted to voltage and processed by a computer. Therefore, we want to minimize the influence of external factors during speech recordings. Ideally, speech should be recorded in a quiet environment with dedicated microphone and a single speaker speaking directly to the microphone. Achieving perfect recording conditions is difficult, especially when collecting large chunks of data from many different people. If we cannot guarantee ideal conditions, the general advice is to record speech in the same or similar conditions for all recordings [15], [10].

Hearing

Similarly to microphone, human ear can also distinguish changes in the air pressure and we subjectively perceive this passive process as *hearing*. Human auditory system consists of three parts: the *outer ear*, the *middle ear* and the *inner ear*. The acoustic wave firstly travels through the ear canal of the outer ear to the eardrum. The air pressure from the wave mechanically vibrates on the eardrum, whilst middle ear passes the information about vibrations to the inner ear. There is a small complex organ named *cochlea* that is located in the inner ear. Cochlea has a spiral shape resembling a snail shell and is filled with fluid that propagates these vibrations. *Organ of Corti* inside the cochlea is responsible for translating the mechanical vibrations to the electrical impulses and sending them to the brain.

The incoming impulses are analyzed by our brain, which results in the subjective perception of *hearing*. Humans can generally hear frequencies from 16 Hz to 20 kHZ, but as we age, the upper frequency limit can drop as low as 14 kHz. Hearing is the most sensitive to the frequencies in the range from 2000 Hz to 5000 Hz. On top of that, human hearing is not linear or flat, but logarithmic. The intensity of the energy from the sound waves is measured in decibels (dB). Humans are less sensitive to lower frequencies and such signals actually must have higher intensity in order to be audible for us [15], [10].

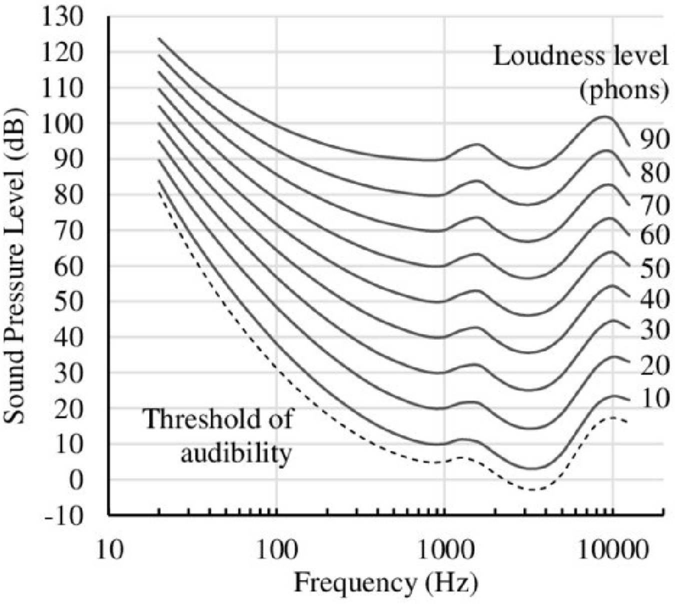


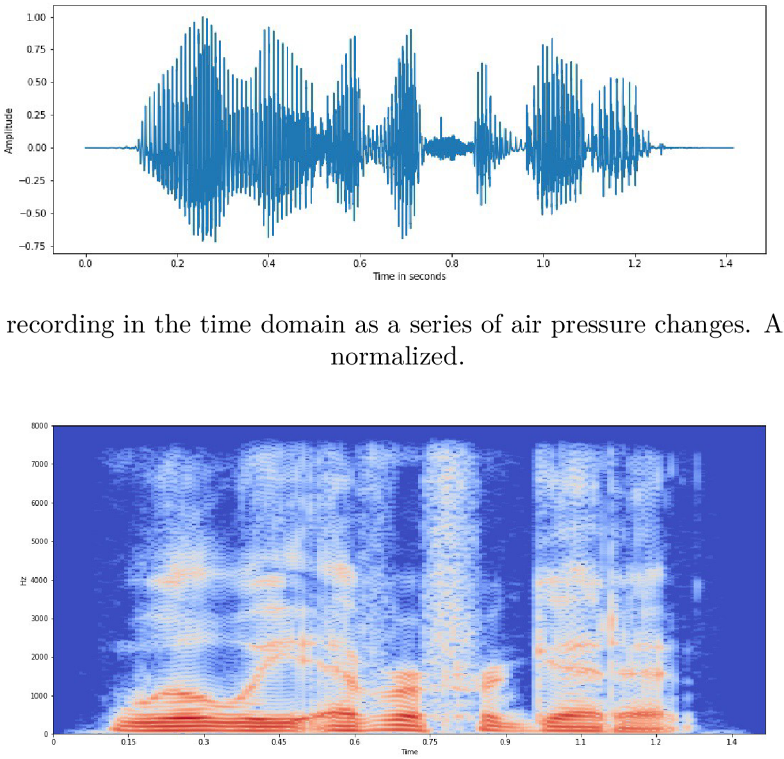
Figure 2.2: Graph of equal loudness for frequencies inside the human hearing range. Taken from [12].

2.2 Digital signal processing

When processing and storing speech digitally, the analog input speech signal must be firstly converted to digital signal by the `analog-to-digital` converter. The converter carries out the following:

1. **Sampling:** a process of converting analog signal to digital by recording values for n samples per seconds, where n is the `sampling frequency` (fs). *Nyquist-Shannon* sampling theorem establishes that a signal can be discretized and fully reconstructed if the sampling frequency is at least twice as high as the maximum frequency contained in the input signal. Sampling is often preceded by filtering with *anti-aliasing filter*. Anti-aliasing filter removes all the frequencies higher than half of the sampling frequency, so no *aliasing* occurs.
2. **Quantization:** a process of mapping any floating values of amplitude to a finite set of quantization levels. This finite set of quantization levels is given by the *bit resolution*, i. e. how many bits will be used to store each value of amplitude. Uniform quantization assumes uniform distribution of quantization levels. Input signal is assumed to be normalized in the range $(-1, 1)$ by default. For each sample, the value of amplitude is rounded to the closest value from the finite set of quantization levels.
3. **Encoding:** a process of assigning a clear binary value to each quantization level. The quantization levels are spread either uniformly or non-uniformly. Non-uniform encodings take advantage of the fact that humans are less sensitive to higher frequencies. Therefore, higher frequencies can be captured with fewer quantization levels than lower ones, simply because we subjectively do not hear the difference.

After this process, the signal can be stored and manipulated digitally. In speech processing, digital signal is usually represented either in *a time domain* or *a frequency domain*. Signal in the *time domain* simply plots the level of air pressure of samples over time. However, representing the signal in the time domain is generally ambiguous, thus using *frequency domain* is more prevalent for further processing and analysis. Signal in the *frequency domain* is usually visualized by a *spectrogram*. Spectrogram plots time on the x-axis, frequency (usually half of the `sampling frequency`) on the y-axis, and the intensity in dB for each frequency at every point in time as a coloured tile.



(a) Sound recording in the time domain as a series of air pressure changes. Amplitude is normalized.

(b) Sound recording in the frequency domain as a spectrogram. The redder the colour, the higher the intensity.

Figure 2.3: Speech recording visualized in both the *time domain* and *frequency domain*. Utterance of “Will we ever forget it”. Taken from[1].

The bridge between domains

As explained above, both of these domains come with their advantages and disadvantages. The most notable advantage of using frequency domain is simplification of mathematical operations and equations. Fouriers’s work [8] from the 19th century laid a base for the famous **Fourier transform** used to this day. Fourier claimed that any function can be expressed as a sum of sines, regardless of whether it is discrete or continuous.

Building upon that, it was discovered that we can use an integral to exploit the properties of the complex exponential. Complex exponential $e^{2\pi ft}$ provides a nice encapsulation for both the real and imaginary parts of the signal. By Euler’s rule, the complex exponential can be decomposed as:

$$e^{j2\pi ft} = jsin(2\pi ft) + cos(2\pi ft) \tag{2.1}$$

Essentially, the signal in the time domain can be represented as a sum of sines and cosines (i.e. the complex exponential) at different frequencies. These waves are then projected onto a frequency spectrum and we get the signal’s representation in the frequency domain,

[1] https://towardsdatascience.com/beginners-guide-to-speech-analysis-4690ca7a7c05

i. e. how much of each frequency is present in the signal. Collection of these intricate transformations is referred to as a **Fourier Transform**.

There are multiple versions of the Fourier transform formula that distinguish whether the input signal is discrete or continuous and periodic or non-periodic. However, the core idea behind all of them remains the same. There are also inverse transformations available, so we can transform the input signal back and forth between the two domains.

The output of the Fourier Transform is a series series of 2 values, one for `magnitude` and the other for `phase` respectively. In digital signal processing, most commonly used transform is the **Fast Fourier Transform** (FFT) based on the **Discrete Fourier Transform** (DFT). The Discrete Fourier Transform and its inverse is given as follows:

$$X[k] = \sum_{n=0}^{N-1} x[n]e^{-\frac{j2\pi kn}{N}} \tag{2.2}$$

$$x[n] = \sum_{k=0}^{N-1} X[k]e^{\frac{j2\pi kn}{N}} \tag{2.3}$$

where $x[n]$ is the signal in time domain, $X[k]$ is the signal in frequency domain and finally, there is a complex exponential. If we look closely at both of these formulas, they seem very similar. Essentially, both of them represent a base change of the input sequence and that can be easily computed with digital computers.

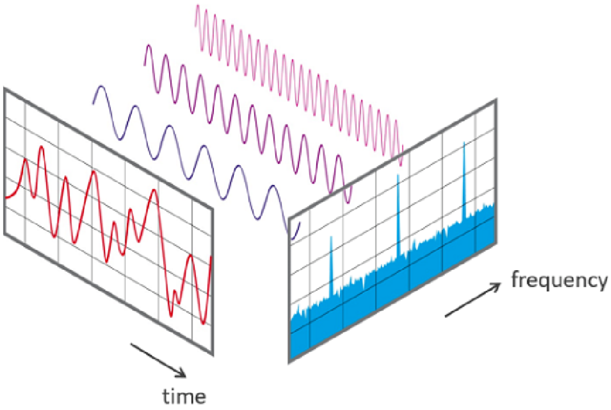


Figure 2.4: Visualization of what happens behind the scenes of Fast Fourier Transform. Taken from[2].

The DFT transforms a finite length input signal to a finite length output signal. The standard DFT calculations consist of N multiplications and N additions, resulting in a quadratic complexity $O(N^2)$. Quadratic complexity gets very high for a big enough N, so there was a need for optimization of the time complexity.

The FFT reduces the number of operations to $O(N \log_2 N)$ because of a smart algebraic manipulation of the original equation. It was proven that the DFT equation can be separated into odd and even indexed sub-sequences and those can be computed concurrently. This reduces the time complexity by the factor of 2. We can keep splitting by the factor of

[2] https://www.nti-audio.com/en/support/know-how/fast-fourier-transform-fft

2, which is possible only if N is a power of 2, and obtain the optimized time complexity of $O(N \log_2 N)$ [16].

The convolution theorem takes advantage of this reduced time complexity. The theorem states that convolution in the time domain is a simple point-wise multiplication in the frequency domain. Similarly, multiplication in the time domain results in a convolution in the frequency domain.

$$x[n] * y[n] = \sum_{k=-\infty}^{\infty} x[k]y[n-k] \xleftrightarrow{\text{DTFT}} X(e^{jw})Y(e^{jw}) \tag{2.4}$$

Not only is the reduced time complexity big advantage when processing in the frequency domain, but on the top of that, the frequency domain often contains clear patterns of frequencies in particular sounds, which are ambiguous in the time domain. This also makes the frequency domain more suitable for pattern recognition or machine learning [15], [10].

2.3 Architecture of Automatic Speech Recognition system

The input for ASR is a human voice, either extracted in real-time or on pre-recorded audio file. The target word sequence $\hat{\boldsymbol{W}} = \hat{w}_1 \ldots \hat{w}_n$ can be mathematically perceived as a sequence of words $\boldsymbol{W} = w_1 \ldots w_n$ originating from a human speech generator (vocal tract) tainted with noise from the communication channel. The system aims to process and decode the observed acoustic input signal $\boldsymbol{O}$ and produce the best statistical estimation of the original word sequence. This is done by extracting feature vectors from the input speech and generating the word sequence with the maximum posterior probability, given the input feature vectors.

Depending on the type of sequences that are being decoded, automatic speech recognition systems can be separated into the following classes [20], [10]:

- **Isolated Words:** decodes a single word or a single utterance surrounded with silence.
- **Connected Words:** similar to isolated words, but utterances require only minimal pause between them.
- **Continuous Speech:** users are allowed to speak naturally while the computer processes the content. There is a difficulty in determining the boundaries between the utterances.
- **Spontaneous Speech:** users are allowed to speak naturally and freely, so the speech contains slight imperfections, such as stutter or filler words. This system should be able to deal with a big variety of speech and speaker features.

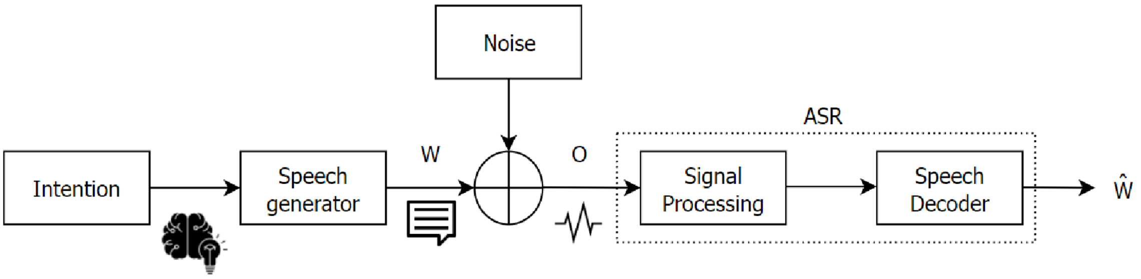


Figure 2.5: Basic model of ASR system.

This fundamental idea of ASR for finding the target word sequence $\hat{\boldsymbol{W}}$ is expressed by the following equation:

$$\widehat{\mathbf{W}} = \operatorname*{argmax}_{\mathbf{w}} P(\mathbf{W}|\mathbf{O}) \tag{2.5}$$

where $\mathbf{O}$ is the acoustic observation of feature vector sequence and $\mathbf{W}$ is the target word sequence. The variable $\mathbf{W}$ represents the actual word sequence as intended by the speaker and $\widehat{\mathbf{W}}$ denotes a word sequence that has been decoded by the ASR. The better the ASR system, the smaller the difference between $\mathbf{W}$ and $\widehat{\mathbf{W}}$. This equation can be modified using the Bayes rule:

$$\widehat{\mathbf{W}} = \operatorname*{argmax}_{\mathbf{w}} \frac{P(\mathbf{O}|\mathbf{W})P(\mathbf{W})}{P(\mathbf{O})} \tag{2.6}$$

Because the maximization is carried out with a fixed observation $\mathbf{O}$, the expression can be further simplified as:

$$\widehat{\mathbf{W}} = \operatorname*{argmax}_{\mathbf{w}} P(\mathbf{O}|\mathbf{W})P(\mathbf{W}) \tag{2.7}$$

where $P(\mathbf{O}|\mathbf{W})$ and $P(\mathbf{W})$ account for **acoustic modelling** and **language modelling** respectively. Creating accurate acoustic and language models constitute the biggest challenge in the recognition of any spoken language. Because spoken speech is essentially a series of continuous signals with possibly infinite number of phoneme combinations, this task goes beyond just recognizing simple fixed patterns. The role of both models is to account for language variability and statistical properties of a given spoken language [6]. The complexity of ASR system greatly varies with vocabulary size, speaker dependency, speech type, use of grammar or even training method [1].

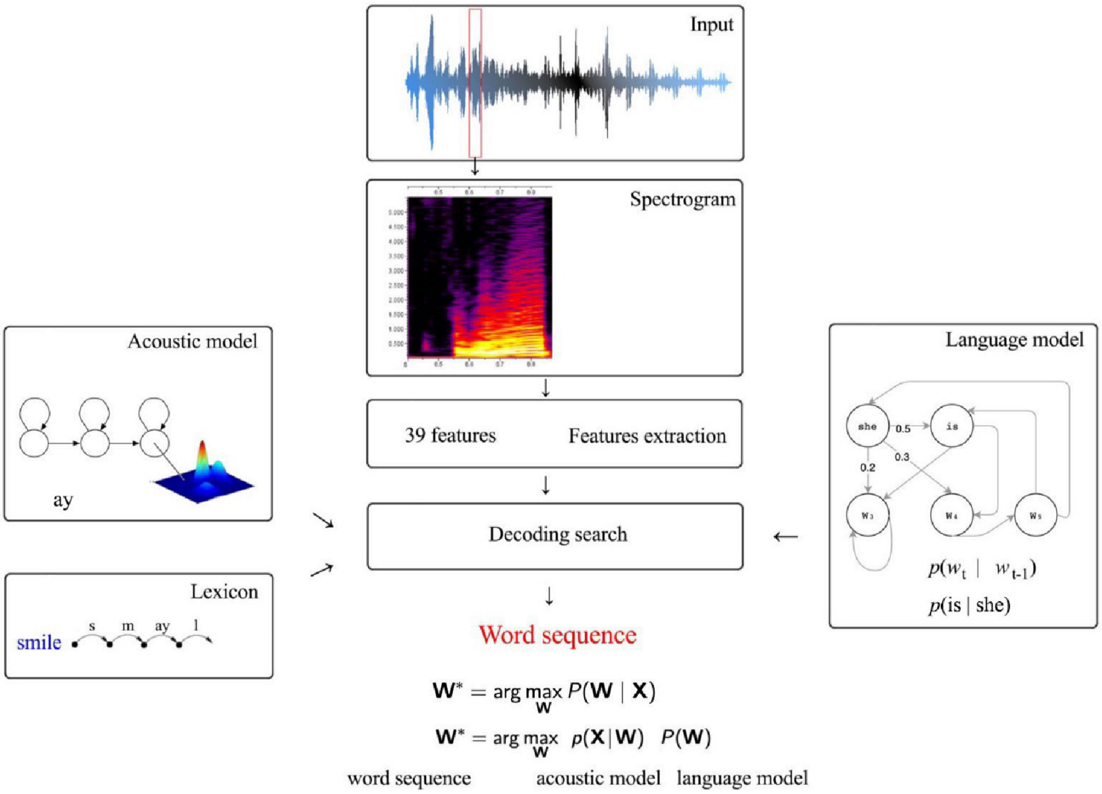


Figure 2.6: General architecture of any ASR system. Taken from [11].

Acoustic model aims to encapsulate the knowledge about acoustics, phonetics, dialect and gender differences or microphone and environment variability – this is reflected by $P(\mathbf{O}|\mathbf{W})$. Acoustic model is accompanied with *lexicon* for the target language. On the other hand, *language model* focuses on capturing the structure of a target language: what words are used, their frequency and which words are more likely to occur together, which is reflected by $P(\mathbf{W})$. Both of these models can be dynamically adapted to achieve a better overall accuracy of the ASR system [6].

Evaluation metrics

According to [17], a good evaluation metric should satisfy the following four conditions:

- it should be a *direct* measure of the desired ASR component
- it should be an *objective* measure that can be fully automated
- it should be an easily and clearly *interpretable* measure
- it should be *modular* in order to allow application-dependent analysis

One of the most commonly used metrics is the **Word Error Rate (WER)** derived from the *edit distance.* It is the minimum number of operations required to transform the reference word sequence to the word sequence estimated by ASR. These operations include insertions, deletions and substitutions. In order to compare different systems, we normalize the number of operations by the length of the reference word sequence. Then, WER is defined as follows:

$$\mathrm{WER} = \frac{S + D + I}{N_r} \tag{2.8}$$

where S is the number of substitutions, D is the number of deletions and I is the number of insertions respectively. N_r is the length of the reference sequence [17].

2.4 Feature extraction

Feature extraction is an essential step in the speech processing pipeline. The chosen features should be speaker invariant and the extraction process should be clear, deterministic and reproducible. The goal of feature extraction is to transform input speech signal into such space of observations, where the same class will be grouped together and different classes will be further apart.

There are various methods for extracting features, such as *Linear Discriminant Analysis* (LDA) or *Principle Component Analysis* (PCA), but *Mel-frequency Cepstral Coefficients (MFCC)* are the most prevalently used in ASR. MFCC uses the *Mel scale*, which models the non-linear sensitivity of human hearing at different frequencies (described in 2.1).

After the input signal passes through analog-to-digital converter, the features for MFCC are generally extracted as follows:

DC-offset removal

Direct current offset is removed from the signal to avoid signal distortion further on. DC-offset is calculated as a mean amplitude of the signal.

Windowing

The original input signal is split into smaller segments called *frames* by applying a windowing function. The interval of the segment can be arbitrary, but it is advised to use small ranges in the order of milliseconds. It is desired to choose small enough segments so that it contains enough relatively stationary information. The windows can be overlapping or not. Typical length of the window is 25 milliseconds with 10 milliseconds shift.

Fast Fourier Transformation

Each frame is transformed from time domain to frequency domain with the FFT. Only the power of absolute magnitude is used further on.

Pre-emphasis

The energy of high frequencies is boosted in order to improve the recognition accuracy of the acoustic model.

Mel Filter Bank

Filter banks model human hearing along the frequency axis with triangular filters that are spaced evenly below 1000 Hz and logarithmically above 1000 Hz.

Logarithm

Logarithm is applied because it has similar properties to human hearing. Small input values will be higher, but values that were already high will be lower.

Discrete Cosine Transformation

Decorrelates the data and outputs real-numbered values. Particularly important if the target model expects the data to be uncorrelated.

The output from this feature extraction pipeline is a set of MFCC coefficients. These coefficients constitute an *acoustic feature vector* for each frame. 13th-order MFCC are extracted and those can be supplemented with *1st order delta* MFCC and *2nd order delta* MFCC. These delta coefficients capture temporal changes over time and provide complementary information to the chosen acoustic model [18], [20], [10].

2.5 Acoustic modelling

The task of acoustic modelling is to find a relationship between extracted features and phonemes. It is not too difficult to create an acoustic model for a particular speaker in a particular language and speaking style. However, the underlying challenge lies in creating a truly robust model that can deal with variability of different speakers and speaking styles.

Acoustic model is trained with a large training corpus of audio files. It is necessaary to convert all audio files to the same format – the same sampling rate and the same number of bits per sample. The ASR works the best, if the training data encoding matches the real application encoding [10].

Hidden Markov Models

In the past, *Hidden Markov Models (HMM)* were very popular for acoustic modelling. Nowadays, acoustic modelling can also be done with *Neural Networks (NN)* or a hybrid model *NN-HMM* consisting of the aforementioned models or even a combination of *HMM* and *GMM (GMM-HMM)*. The basic Hidden Markov Model is based on the *Markov chain* model and is formally defined as a 5-tuple:

$$
\begin{aligned}
\mathrm{HMM} &= (Q, A, O, B, \pi) \qquad (2.9)\\
Q &= \{q_1, q_2, \dots, q_n\}\\
A &= \begin{bmatrix} a_{11} & \cdots & a_{1n} \\ \vdots & a_{ij} & \vdots \\ a_{n1} & \cdots & a_{nn} \end{bmatrix} \quad \text{where} \quad \sum_{j=1}^{N} a_{ij} = 1 \text{ for } \forall i\\
O &= o_1 o_2 \dots o_t\\
B &= b_i(o_t)\\
\pi &= \pi_1, \pi_2, \dots, \pi_n \quad \text{where} \quad \sum_{i=1}^{N} \pi_i = 1
\end{aligned}
$$

where Q is a discrete set of N hidden states, A is a transition probability matrix $N \times N$ for moving from state q_i to state q_j. O is a sequence of T observations and each o_t belongs to a predefined vocabulary V. B is a sequence of emission probabilities, where each emission probability expresses how likely it is that o_t has been generated from a state q_i. Finally, π is an initial probability distribution for N states that expresses how probable it is for q_i to be the initial state. Usually, not every state q_i can be initial state, therefore, its respective initial probability distribution π_i is a zero value.

HMM is well-suited for acoustic modelling, because it models the *hidden states* Q that emit *observations* O according to its emission probability distribution. As the name suggests, hidden states are unobserved and we try to estimate their probability distribution from our observations O. In the case of speech recognition, the observations O are the features extracted from the spectrogram of the input signal sequence [15]. Refer to [10] or [15] for an exhaustive and precise explanation of the combined models, as well as the full training procedure for the standard HMM.

In general, modelling whole words for a large vocabulary spontaneous speech system is unfeasible. It is preferred to choose smaller modelling units that are: *accurate*, *trainable* and *generalizable*. One candidate choice is using *phones* as a basic unit. *Phones* are more trainable and generalizable than the whole words models, but they assume that each *phone* is identical in any context or position, which is not completely accurate. Furthermore, humans are not capable of strictly producing one phone right after the other, because we cannot adjust our articulatory tract instantaneously. Another option is using slightly larger *syllables* as a basic unit. A correct choice of the basic acoustic unit strongly depends on that particular target language and its properties. The acoustic units are then modelled with our chosen acoustic model. In HMM, each acoustic unit is usually modelled with at least 3 discrete states – beginning, middle and end of the acoustic unit [7], [10].

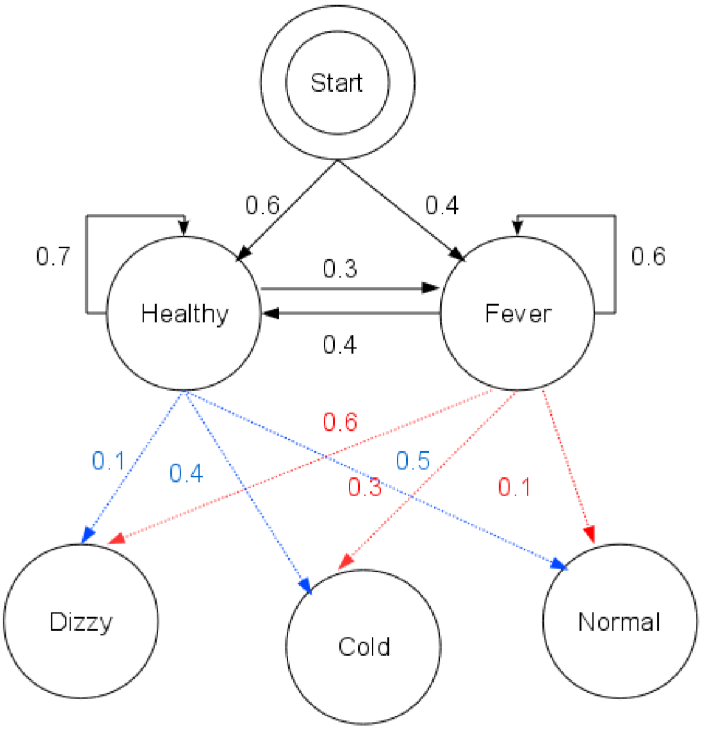


Figure 2.7: Hidden Markov Model sample diagram. Taken from [3].

Lexicon

Lexicon, alternatively also called a *pronunciation model* or a *dictionary*, closely collaborates with the acoustic model. Depending on the application and the target language, the format and complexity of the lexicon may vary.

Lexicon is created by human experts, usually professional *linguists*. The lexicon contains all the words from a pre-defined vocabulary V with their respective pronunciations. The pronunciation is described with the basic acoustic units of choice. For most modern languages, using *phonemes* as the acoustic unit of choice yields very good results.

Linguists have created the `International Phonetic Alphabet` (IPA) with a standardized notation of any speech sound in the written form. IPA is so extensive that it can accurately capture the smallest details of each sound, such as pre-voicing, devoicing, aspiration, and many others. This has led to the addition of various IPA-exclusive symbols to the Unicode character encoding. Unicode uses either 8-bits (1 byte) or 16-bits (2 bytes) in order to encode a single character. The first standardized computer encoding named `ASCII` has been used as a foundation for the Unicode encoding. Therefore, Unicode has a backward compatibility to `ASCII` and `ASCII` can be considered as a subset of Unicode. `ASCII` was developed in the 1960s and it utilizes 7-bits to encode 128 unique characters frequently used in American English. Those characters consisted of all lowercase and uppercase symbols of Latin alphabet, 10 Arabic digits and the rest was used for mathematical symbols and common punctuation, including spacing and non-printable characters.

[3] https://upload.wikimedia.org/wikipedia/commons/0/0c/An_example_of_HMM.png

Since the original goal of Unicode was to create unified and consistent encoding for characters in the World Wide Web, the 8-bit encoding has been quickly assigned to the most commonly used characters. As a consequence, most IPA exclusive symbols use the 16-bit encoding. Nowadays, it is no longer necessary to be extremely keen on the memory requirements for storing our data. However, training any ASR system is extremely data intensive, so every byte counts.

This issue has been partially solved by the `Speech Assessment Methods Phonetic Alphabet` (SAMPA). `SAMPA` is a computer-readable phonetic alphabet that is based on IPA, so that it maps the special IPA symbols to the most common characters that fit inside the `ASCII` encoding. Currently, concrete `SAMPA` has been officially created for around 30 worldwide languages and more are under the development. Additionally, there is also a language-independent version named `X-SAMPA`, making it one of the most universal and efficient approaches for storing pronunciation.

Currently, *SAMPA* and *X-SAMPA* are widely used in *lexicons* to ease the burden of memory requirements for pronunciation storage. In the simplest form, *lexicon* contains a list of words, with each word on a new line. Pronunciation in `SAMPA` or `X-SAMPA` is on the same line, usually separated from the word with a delimiter of choice. Lexicons can hold multiple pronunciations of the same word, but it is advised that all these pronunciations follow standard pronunciation rules of that particular language. Words that are still deemed as foreign should be kept in a separate lexicon. This requirement is amplified especially if that particular foreign word follows different pronunciation rules compared to the target language. Alternatively, many foreign words contain characters that are not used in other languages, so those should be kept in a separate lexicon as well.

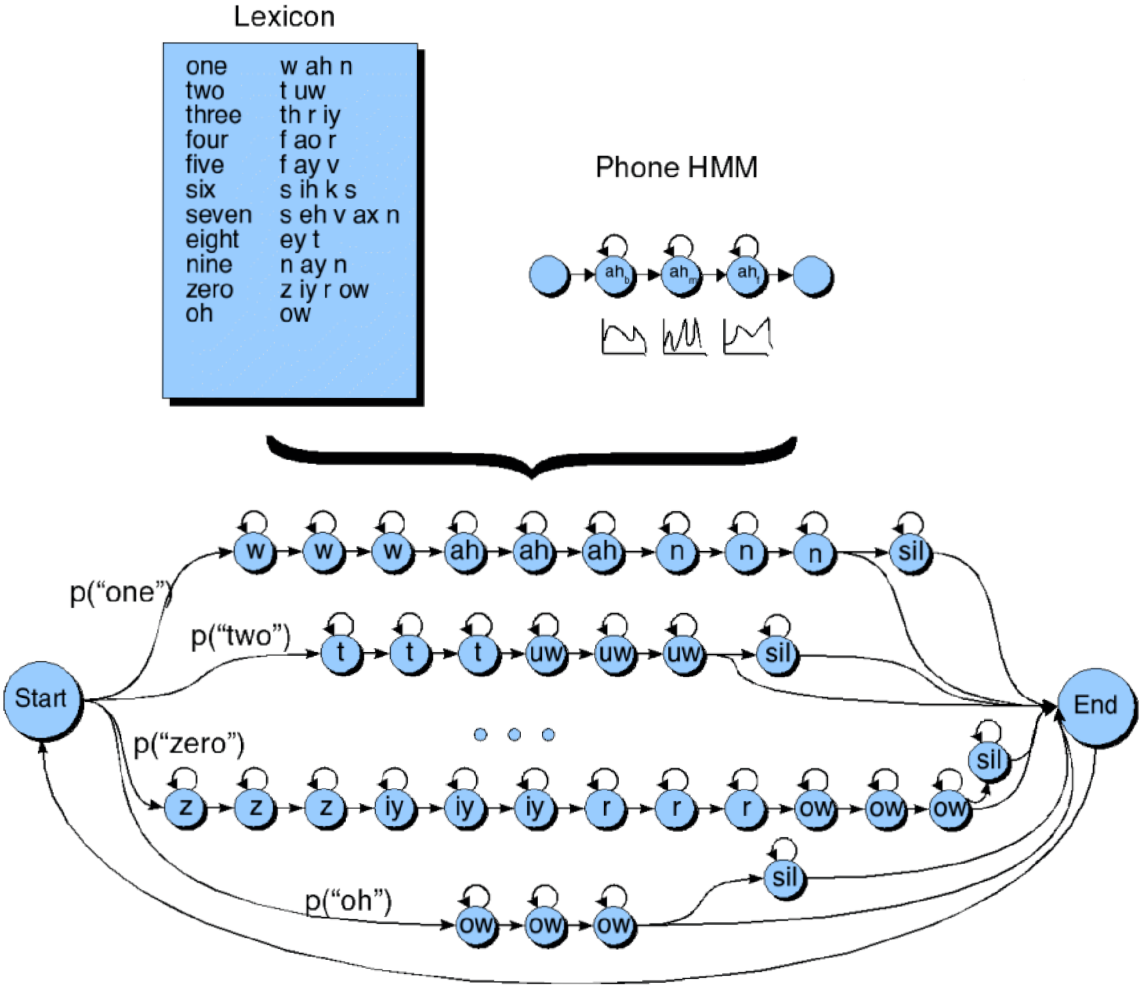


Figure 2.8: Sample of an acoustic model with lexicon for the task of isolated digit recognition. Taken from [14].

Chapter 3

Language modelling

Language model (LM) is a fundamental component of any ASR system. Language model encapsulates the knowledge about vocabulary, syntax and semantics of a particular language. Therefore, a well-trained language model predicts what words sequences are likely to occur. A correct, well-formed sentence would be accessed as very likely to occur by the LM, whilst a sentence consisting of random, unrelated words would be assigned a very low probability of occurring.

This chapter explains the most common approaches to languages modelling: formal language theory (Section 3.1) and the probabilistic language model (Section 3.2) with smoothing. Section 3.3 deals with evaluation of language models and finally, section 3.4 introduces the biggest challenges for language modelling in real-life applications.

3.1 Formal languages

Formal languages neatly preserve the information about language, what the language consists of and the rules for given language. The formal language theory is very extensive and is explained in detail in [9], but this paper covers only basics of the formal language theory.

Formally, properties of any language can be recorded by a *grammar*. Grammar is defined as a tuple $G = (N, T, R, S)$, where N is a set of *non-terminal symbols*, T is a set of *terminal symbols* and S is a starting *non-terminal* symbol. It is essential that $N \cap T = \emptyset$ so that no symbol is simultaneously *terminal* and *non-terminal*. Terminal symbols are words that belong to a language, for example `[i, love, tabby, cat]` is a possible list of terminal words. Non-terminal symbols are rewritten according to the rules from R.

Our goal is to either successfully rewrite all non-terminal symbols to terminal symbols (*top-bottom approach*); or rewrite terminal symbols by applying the rules backward in order to obtain the starting non-terminal S (*bottom-up approach*). The set of rules R consists of rules in the form of $\alpha \rightarrow \beta$, where α and β are arbitrary strings of symbols and α is not empty. Constraints posed on the rules result in a hierarchical order (*Chomsky's hierarchy*) of the grammars based on the format of the rule. The bigger the constraint on the rules, the weaker the grammar. However, the weaker the grammar is, it becomes less ambiguous, which is beneficial for automatic processing. The hierarchy is as follows (from strongest to weakest):

- **Phase structure grammar** : $\alpha \rightarrow \beta$
  - most general grammar
  - corresponding machine is *Turing machine*

- **Context-sensitive grammar** : $\alpha \rightarrow \beta, |\alpha| \leq |\beta|$
  - subset of phase structure grammar
  - $|.|$ signifies the length of the string
  - corresponding machine is *Linear bounded automata*
- **Context-free grammar** : $A \rightarrow \beta$
  - subset of context-sensitive grammar
  - only single non-terminal symbol on the left side of the rules; $A \in N$
  - corresponding machine is *Push down automata*
- **Regular grammar** : $A \rightarrow w$ and $A \rightarrow wB$
  - subset of context-free grammar
  - only single non-terminal symbol on the left side of the rules; $A \in N$
  - right hand side of the rule contains either only terminal symbol w, $w \in T$ or terminal symbol w and non-terminal symbol B, $B \in N$
  - corresponding machine is *Finite-state automata*

It is generally assumed that natural spoken language is at least context-sensitive, but the requirement for this level of grammar is quite scarce. Instead, the weaker context-free grammar is widely preferred in machine learning and natural language processing. It is capable of describing the basic structure of any natural language and powerful enough to efficiently parse various sentences. However, it is considered very unlikely that any grammar would have a complete coverage of the language, not to mention the fact conversational speech often does not adhere to the official standardized rules of the language [10].

3.2 Stochastic language models

Stochastic language models approach language modelling as a need to accurately calculate the probability $P(\mathbf{W})$ for a given word sequence $\mathbf{W} = w_1 w_2 \dots w_n$ given a *corpus*. This viewpoint tries to ensure that word sequences likely to occur have a higher probability. This probability dramatically reduces search space and makes the speech recognition more accurate. Currently, the most popular stochastic language model is a so-called `n-gram` model [10].

The joint probability distribution $P(\mathbf{W})$ can be decomposed as follows:

$$P(\mathbf{W}) = P(w_1, w_2, \dots, w_n) \tag{3.1}$$

$$P(\mathbf{W}) = P(w_1)P(w_2)\dots P(w_n) \tag{3.2}$$

This approach is rather naive, because it assumes that the words are completely independent of each other. However, it is more logical to assume that in spoken language, each following word w_i depends on all the previously said words, therefore we can adjust the decomposition as follows:

$$P(\mathbf{W}) = P(w_1, w_2, \dots, w_n) \tag{3.3}$$

$$P(\mathbf{W}) = P(w_1)P(w_2|w_1)P(w_3|w_1, w_2)\dots P(w_n|w_1, w_2 \dots, w_{n-1}) \tag{3.4}$$

$$P(\mathbf{W}) = \prod_{i=1}^{n} P(w_i|w_1, w_2 \dots, w_{i-1}) \tag{3.5}$$

Such decomposition means that we have to store history for every single word in the vocabulary. This is not feasible, and on top of that, most histories would be unique or occurred only a few times in the whole corpus. Consequently, we assume that $P(w_i|w_1, w_2 \dots, w_{i-1})$

depends only on a few equivalence classes. This equivalence class is based on n previous words, thus the name *n-gram*. *Unigram* $P(w_i)$ assumes naive independence of words, whereas *bigram* $P(w_i|w_{i-1})$ assumes dependence on one previous word. All the higher order n-grams follow the same logic, so *trigram* $P(w_i|w_{i-1}, w_{i-2})$ assumes dependence on the two previous words. Each sentence is supplemented with arbitrary start and end token to clearly mark the sentence boundaries.

The *trigram* model is especially powerful, because it assumes that each word has a strong dependence on the two previous words, and at the same time, it is still computationally feasible. It is possible to use a higher order *n-gram* if there are computational resources and large training corpuses available, but this paper primarily focuses on the *trigram* and lower *n-grams*.

The core of the *n-gram* model is based on counting the co-occurrence of words. For a vocabulary of the size V, we need a $V \times V$ matrix to cover every possible co-occurrence of the words in vocabulary. Naturally, such matrix will be very scarce, because most words generally co-occur only with a small subset of the vocabulary. A simple *trigram* model calculates the probability as a *relative frequency* ratio between the counts of our desired trigram and the bigram count of the two previous words:

$$P(w_n|w_{n-1}, w_{n-2}) = \frac{C(w_n, w_{n-1}, w_{n-2})}{C(w_{i-1}, w_{i-2})} \tag{3.6}$$

However, in a large training corpus, the resulting probability for each *trigram* tends to be a very small number. On top of that, multiplying probabilities between numerous trigrams would lead to a very small number with many decimal places, often resulting in an underflow.

Therefore, it is preferred to use *log probabilities* so that the results can be stored without underflow. Additionally, multiplication in linear space results in an addition in logarithmic space, so the operation is also simplified from multiplication to addition. Finally, if we require result to be a standard probability in the range $(0, 1)$, we can take the *exponent* of the final log probability as follows [10], [15] :

$$p_1 \times p_2 \times p_3 = \exp(\log p_1 + \log p_2 + \log p_3) \tag{3.7}$$

N-gram smoothing

The vanilla *n-gram* model has one big disadvantage – it automatically assigns the probability of zero to any n-gram not seen during the training. If the training dataset is very small, most n-grams will be assigned zero probability during the evaluation phase. However, this problem is still present even for a very large training corpus. Statistically, if we use several million-word collections, many n-grams would occur only once and the majority of them would occur less than 5 times, which is problematic. If we were to assign zero probability for any unseen n-grams, $P(\mathbf{W})$ could end up as a zero value. If we substitute 0 for $P(\mathbf{W})$ in the central equation of ASR (2.7), the whole expression would be annihilated to 0 and impossible to maximize.

Therefore, in order to ensure that $P(\mathbf{W})$ will always be non-zero value, *smoothing* (alternatively called *discounting*) techniques are used. The core idea behind these methods is to shave off a bit of probability mass from the often-occurring n-grams and assigning them to those n-grams that have never been seen before. This makes the whole model more robust to unseen data for the cost of hurting the training data slightly. There is a variety of smoothing techniques available to counter this problem [10], [15].

Add-one / add-k smoothing

`Add-one smoothing` and `add-k smoothing` are very similar in their nature. `Add-one smoothing` (alternatively `Laplace smoothing`) does exactly what the name suggests - arbitrarily adds 1 to every n-gram. Therefore, the previously seen n-grams would just increment their count by one and those never-before-seen n-grams would appear to have been seen at least once. For trigram with the vocabulary of size V, this would lead to a small adjustment:

$$P(w_i|w_{i-1}, w_{i-2}) = \frac{1 + C(w_i, w_{i-1}, w_{i-2})}{V + C(w_{i-1}, w_{i-2})} \tag{3.8}$$

Alternative to the `Laplace smoothing`, `add-k smoothing` generalizes the value of 1 to k, which results in the following smoothing for trigram:

$$P(w_i|w_{i-1}, w_{i-2}) = \frac{k + C(w_i, w_{i-1}, w_{i-2})}{kV + C(w_{i-1}, w_{i-2})} \tag{3.9}$$

Kneser-Ney smoothing

The aforementioned smoothing techniques became the basis for the `Kneser-Ney smoothing`, which is currently widely used in practice. It is considered to be the most effective state-of-the-art approach for smoothing. It is based on a smart discounting – subtracting a small amount δ from each count in order to save some probability mass for smaller counts. Furthermore, it includes the lower n-gram models in the calculation through $P_{CONT}(w_i)$ and its normalizing constant $\lambda(w_{i-1})$:

$$P_{KN}(w_i|w_{i-1}) = \frac{\max(C(w_i, w_{i-1}) - \delta, 0)}{C(w_{i-1})} + \lambda(w_{i-1})P_{CONT}(w_i) \tag{3.10}$$

The key difference between using P_{CONT} or lower order n-gram model is that P_{CONT} actually calculates the probability of w_i being a novel continuation. In order to calculate the probability of a word w_i as a novel continuation, we need to count how many n-grams the word w_i completes. The resulting novel continuation probability is normalized by dividing it with the total number of all n-grams:

$$P_{CONT}(w_i) \propto |\{w_{i-1} : C(w_i, w_{i-1}) > 0\}| \tag{3.11}$$

$$P_{CONT}(w_i) = \frac{|\{w_{i-1} : C(w_i, w_{i-1}) > 0\}|}{\sum_{w'} |\{w'_{i-1} : C(w'_{i-1}, w') > 0\}|} \tag{3.12}$$

Backoff and interpolation

Alternative approaches to smoothing techniques are `backoff` and `interpolation`. In case there is no history for particular n-gram, it can be useful to gradually `backoff` and search through $(n-1)$-grams until enough evidence is found.

`Interpolation` actually mixes the high order n-gram with lower order n-grams. This approach works generally better than simple backoff. The models are mixed according to their mixing weights λ. The mixing weights λ are estimated from a held-out dataset. It is necessary for the mixing weights λ to add up to one.

Linear interpolation for trigram model with lower n-gram models is defined as follows:

$$\begin{aligned} P(w_i|w_{i-1}, w_{i-2}) =& \lambda_1 P(w_i) + \\ & \lambda_2 P(w_i|w_{i-1}) + \\ & \lambda_3 P(w_i|w_{i-1}, w_{i-2}) \end{aligned} \tag{3.13}$$

$$\lambda_1 + \lambda_2 + \lambda_3 = 1 \tag{3.14}$$

3.3 Evaluation

The performance of any language model can be evaluated with an *extrinsic* or an *intrinsic* evaluation. *Extrinsic evaluation* requires the language model to be a part of a bigger system or application, which is evaluated as a whole. This approach accesses how do that particular component improves or setbacks the entire system. Repeated evaluation of the entire system can get very expensive and is almost impossible if the other components are not fully implemented at that time. Simpler approach is the *intrinsic* evaluation, because it accesses only the performance of our language model independent of any other components.

Fundamentally, *intrinsic evaluation* requires the chosen dataset to be split into disjunctive subsets, usually a *train set* and a *test set*, but it can be complemented with a *dev set* as well. It is crucial that no data from the test set are used in training, because it would lead to skewed results. Naturally, we want to use as much data as possible in training, so that the resulting trained model can generalize. The chosen dataset is split in such a way, that the smaller subsets (*test set, dev set*) are able to provide us with a statistically significant difference, so that it is possible to objectively compare various models. Common practice is splitting the dataset as 80% *train set*, 10% *test set* and 10% *dev set*, but the ratio can be adjusted at one's discretion. It is also important to include the arbitrary start and end tokens in our vocabulary.

Language models are not evaluated with simple probability, but instead with a metric called *perplexity* (often shortened to *PP* or *PL*). *Perplexity* is calculated as the inverse probability when using the test set, normalized by the number of words in the test set. For a test set consisting of $W = w_1 w_2 \dots w_n$, where n is the number of words, perplexity for a trigram model is given and simplified as follows:

$$PP(W) = P(w_1 w_2 \dots w_3)^{-\frac{1}{N}} \tag{3.15}$$

$$PP(W) = \sqrt[N]{\frac{1}{P(w_1 w_2 \dots w_3)}} \tag{3.16}$$

$$PP(W) = \sqrt[N]{\prod_{i=0}^{N} \frac{1}{P(w_i|w_1 \dots w_{i-1})}} \tag{3.17}$$

$$PP(W) = \sqrt[N]{\prod_{i=0}^{N} \frac{1}{P(w_i|w_{i-2}, w_{i-1})}} \tag{3.18}$$

The inverse probability causes the resulting perplexity to be lower when the conditional probability of the words sequences is higher. Another view on perplexity suggests that it represents a *weighted average branching factor* of a language. This basically translates to how many possible words usually follow any word [10], [15].

3.4 Practical challenges

When sampling sentences from a pretrained language model, the obtained sentences will vary depending on the original training dataset. If we use a very limited corpus that only deals with a small range of topics, then the language model is useless for any different topics than those it was trained on. In a general ASR system, we aim to collect as much data as possible, so that the language model can deal with a variety of word combinations [15].

One huge challenge in practical applications is **insufficient data** for training of the language model. Especially if a given language is generally considered to be a **low-resource**, it tends to suffer due to the scarcity of training data. In such cause, we can try to ease this burden by **expanding the existing language model** naturally with similar data from the internet.

On the other hand, there is another underlying challenge usually hidden to the naked eye. Our well-trained ASR system has a good performance on the evaluation dataset, but once its sold to a customer, it may appear to not perform so well on the customer's data as originally anticipated. The reason behind this is often the *niche specialisation* that the ASR system was not trained for.

Oftentimes, the ASR system performs well on general spontaneous speech, but it lacks the knowledge of words and words sequences occurring in the customer's specialisation. Consequently, the ASR system outwardly appears to have poorer performance than originally advertised, which may lead to confusion, disappointment and distrust of our customers or end users. Fortunately, it is possible to tweak isolated components, so the language model can be expanded with words sequences from customer's target domain. This process is called a **dynamic language model adaptation to target domain** and is one of the central source problems for this thesis.

Both of these aforementioned challenges can be partially overcome with downloading data and web documents from the internet. This whole process should be fully automated without any need for interventions and it should be efficient and not too time-consuming. Furthermore, this process should bring at least reasonable gain to make it justifiable [23]. Zhang et al.'s work [23] experimented with this approach to improve a *keyword spotting system* and they achieved substantial improvements.

Chapter 4

Language model expansion pipeline

Automatic language model expansion or adaptation has proven to be very useful when dealing with practical challenges in language modelling described in the section 3.4. Furthermore, it removes the need for re-training the whole language model from scratch. This challenge of dynamically adapting or expanding the language model is the central problem explored in this thesis. Naturally, these problems can be fully explored only on a real-life Speech-To-Text system.

Thankfully, the Brno based company *Phonexia* has offered to collaborate with me on this thesis and kindly provided me with much needed information, tools and datasets. Therefore, the language model expansion pipeline developed in this paper is specifically meant for Phonexia system, but similar approach for different system shall yield comparable results.

Firstly, section 4.1 shortly introduces Phonexia and their system. Only the bare minimum of structural information is provided so as not to betray any confidentiality. Secondly, section 4.2 explains the design of the pipeline. Finally, section 4.3 describes the implementation of the pipeline and its components. It also includes some practical advice for correct usage of the pipeline, particularly on how to choose some of the input parameters.

4.1 Phonexia

Phonexia is a Brno based company founded by a few researchers from the Brno University of Technology speech group `BUT speech@fit`. Created in 2006, it has quickly become one of the key players in speech and voice recognition technologies. Currently, Phonexia is working on many interesting projects in this field, ranging from commercial to government solutions. The technologies researched and developed in Phonexia can be divided into 3 basic categories:

- Automatic speech recognition technology: Speech-To-Text, keyword-spotting system
- Voice biometry technology: speaker identification, language identification
- Supporting technologies: voice activity detection, speech quality estimation

Further on, this thesis focuses only on the automatic speech recognition technology called `Speech-To-Text (STT)` in Phonexia, because it deals with the biggest variability of the input data. The most commonly used types of data are audio files with matching textual data. The textual input data is often referred to as *annotations* or *transcriptions* interchangeably, but these two terms are not complete synonyms. *Annotation* is a time-stamped orthographic transcription in the standardized written form of the language, whilst the term *transcription* does not necessarily imply there are any time-stamps present. Furthermore, the term *transcription* can also refer to a purely phonetic transcription, which has no use for the STT training. Therefore, it is always better to clarify this with the data providers and customers beforehand, because for STT training, we always need the time-stamped *annotations*.

The quality and amount of the available data often determines the overall accuracy of a STT system. Phonexia usually obtains data by purchasing it from providers or directly from customers. In some cases, public sources can be used free of charge. It is also possible to carry out *annotation projects* for data without annotations, but such are usually more time-consuming and expensive than the other methods. When purchasing data for development, the price hugely depends on the target language, amount of data and its quality. Telephony data collected in call-centers or help-desks are less expensive to obtain, while read sentences recorded with professional or semi-professional gear in a quiet environment tend to have the highest cost.

When Phonexia obtains a new dataset, it must be firstly transformed to a predetermined structure and format. The unprocessed data is kept in separate folders away from the processed data. Already processed datasets are stored in the Phonexia data server. The processed dataset structure looks as follows:

```
dataset_name + version
├── data
├── source_data
├── transcriptions
├── dataset_name + version.txt
├── LICENSE.md
├── metadata.csv
└── README.md
```

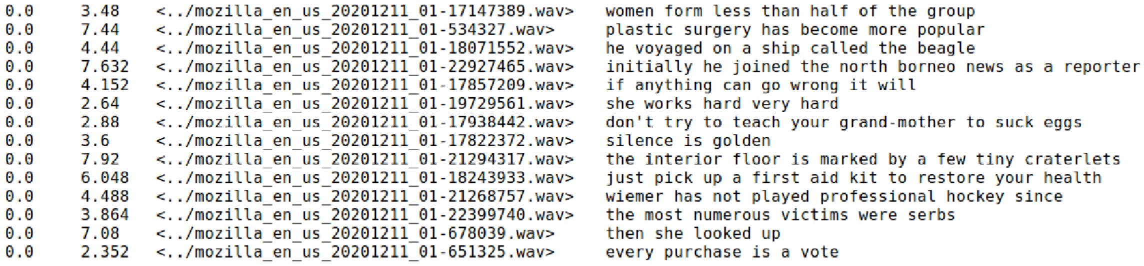
Figure 4.1: Tree structure of a processed dataset.

The dataset name is joined with a two-digit version code by an underscore. This becomes a new internal name of the dataset, so that if there is ever an update to the data, it can be clearly distinguished by the version code. Completely processed directory for any dataset consists of 3 subfolders and 4 files:

- `data` : folder with all audio files that have been converted to a `wav` mono-channel recording with 16-bit encoding (one codec for all files) and a frame rate of 8 kHz (less commonly 16 kHz)
- `source_data` : folder with the original data in the original form
- `transcriptions` : folder with transcriptions or annotations in the original form. The form of transcriptions greatly varies, some datasets have separate annotation file for each audio file, others gather the transcriptions in one single file
- `dataset_name + version.txt` : text file with all relative paths (each on a new line) to the audio files in the `data` folder
- `LICENSE.md` : detailed information about license and usage
- `metadata.csv` : detailed information about every audio file, including information (if available) about the speaker, for example: gender, age, accent, native language etc.
- `README.md` : general information about the dataset, such as providers, content of the dataset, amount of audio files, quality of the audio file etc.

The next step after transforming the dataset to its desired form is to create files that will be used in the ASR training: `stt.phxstm`, `test.list` and `dataset.info`. These files are then stored in a `Gitlab` repository and from that point on, they can be used for training. `test.list` contains the paths to the audio files used for evaluation (usually 1 hour of annotated audio) and `dataset.info` has some basic information about the dataset, duration of the audio files and its license. The longest file named `stt.phxstm` contains information about all the segments with their respective annotations, timestamps and absolute paths. It consists of 7 columns separated with a tabulator.

The exact structure of the `stt.phxstm` will not be displayed, but only a cropped version of the file is shown. Unsurprisingly, the `stt.phxstm` contains start and end time of each segment, its transcription and the absolute path to the audio file containing that particular segment. The transcriptions are located in the last column of the `stt.phxstm` file.



```
0.0     3.48    <../mozilla_en_us_20201211_01-17147389.wav>   women form less than half of the group
0.0     7.44    <../mozilla_en_us_20201211_01-534327.wav>     plastic surgery has become more popular
0.0     4.44    <../mozilla_en_us_20201211_01-18071552.wav>   he voyaged on a ship called the beagle
0.0     7.632   <../mozilla_en_us_20201211_01-22927465.wav>   initially he joined the north borneo news as a reporter
0.0     4.152   <../mozilla_en_us_20201211_01-17857209.wav>   if anything can go wrong it will
0.0     2.64    <../mozilla_en_us_20201211_01-19729561.wav>   she works hard very hard
0.0     2.88    <../mozilla_en_us_20201211_01-17938442.wav>   don't try to teach your grand-mother to suck eggs
0.0     3.6     <../mozilla_en_us_20201211_01-17822372.wav>   silence is golden
0.0     7.92    <../mozilla_en_us_20201211_01-21294317.wav>   the interior floor is marked by a few tiny craterlets
0.0     6.048   <../mozilla_en_us_20201211_01-18243933.wav>   just pick up a first aid kit to restore your health
0.0     4.488   <../mozilla_en_us_20201211_01-21268757.wav>   wiemer has not played professional hockey since
0.0     3.864   <../mozilla_en_us_20201211_01-22399740.wav>   the most numerous victims were serbs
0.0     7.08    <../mozilla_en_us_20201211_01-678039.wav>     then she looked up
0.0     2.352   <../mozilla_en_us_20201211_01-651325.wav>     every purchase is a vote
```

Figure 4.2: Sample of cropped `stt.phxstm` from the Mozilla common voice dataset for American English. Full audiopaths have been shortened for privacy reasons.

In Phonexia, there is a huge focus on cleaning the transcriptions and annotations correctly. If not done properly, it can hinder the training of subsequent components and degrade the overall performance of the STT. As can be seen in the figure 4.2, correctly cleaned transcription is stripped of all the punctuation marks (excluding apostrophe and tightly-knit hyphens), then redundant spacing is removed and finally, all the letters are converted to lowercase. Although this is the general process for cleaning any transcriptions, depending on the dataset, there are often some additional steps required to fully normalize them. Datasets from providers usually have their own annotation tags, so those must be converted to one of the internal Phonexia tags:

- `<hes>` : represents hesitation, affirmative/negative sound or a filler word that has a sound, but no meaning
- `<sil>` : represents silence, non-verbal expressions (coughing, laughing, sneezing) or a short background noise (dropped object, car honking, door slamming)
- `<unk>` : represents unknown, unfinished, unintelligible or mispronounced words

Additionally, most modern languages have adapted a fair share of foreign words. Most borrowed words are gradually adapted to their official written form in the target language. However, some contain special characters not present in the target language and those should be marked somehow. Usually, tightly-knit square brackets (e. g. `[möbelix]`) are used to denote foreign words or words with non-standard pronunciation for that particular language.

Once the dataset is fully processed, the resulting `stt.phxstm` file is used for training the STT. During training, *acoustic model* (2.5) and *language model* (3) are trained and fine-tuned. In order to generalize for unseen words in the *lexicon* (2.5), a `G2P` (grapheme to phoneme) is trained. Well-trained `G2P` can map *graphemes* to their typical phonetic realization of *phonemes*, with respect to their context or position in the word. This allows us to generate pronunciation in terms of phonemes for virtually any word in the target language if it follows the standard pronunciation rules. This entire process of training each component is very computationally intensive and requires a lot of memory (RAM) and parallel GPUs to be truly efficient.

Once all 3 models: `AM`, `LM` and `G2P` are trained for the target language, they can be packaged to a single binary file containing the STT system. This binary file can be used for decoding with `BSAPI` (Brno Speech Application Programming Interface). `BSAPI` provides a command line interface to various Phonexia systems and is frequently used by the customers of Phonexia.

Phonexia constantly tries to improve and optimize their standardized training procedures, but they also offer a range of services regarding the adaptation of already existing *acoustic* and *language* models. *Acoustic models* can be adapted with standard data from the provider or directly on the customer's data, which naturally yields the best results. As for *language models*, the underlying challenges are a little bit tricky and are fully described in the chapter 3.4. If a customer can provide specific data for the language model adaptation, then the process is very straightforward. However, this situation is mostly rare and a customer often just subjectively accesses the overall performance of the STT as unsatisfactory for their needs.

Therefore, the adaptation usually requires additional dataset searching, pre-processing and cleaning. It might end up being a very time-consuming task for the company. This thesis aims to explore, implement and experiment with a pipeline that partially or fully alleviates the need for human labor in the language model adaptation or expansion.

4.2 Design

The design for the language model adaptation pipeline is general and its components can be fully or partially reused for different purposes. Ideally, the pipeline accepts input in the form of configuration file and outputs web corpora, web model, mixed model, logger file and statistics.

Key input parameter is the **input source file**, because it directly determines what terms are used during the web search. The default input source file is either `stt.phxstm` or `kws.phsxtm`, which are standard formats used in Phonexia. Ideally, the pipeline shall search the web for arbitrary terms, download the most appropriate and relevant web documents for each of them, clean the documents, extract textual data and use the extracted data for automatic language model adaptation or expansion.

As explained in the section 4.1, the `stt.phxstm` has 7 columns, with the transcriptions being in the last column. Therefore, this file can be easily used by Phonexia as an input for this pipeline, since the `stt.phxstm` file is kept in storage long-term. Using `stt.phxstm` as the input source file results in a natural language model expansion, because similar terms are used for web search and consequent corpora creation.

If we need to adapt the language model to a particular domain, the source file for the pipeline is a list of keywords inside a `kws.phxstm` file. Each keyword (or alternatively a single sentence) is on a new line. The list of keywords can be provided by our customer or it can be created internally. Fortunately, the list of keywords for particular domain may contain only the most basic keywords, because the related, more complex vocabulary is believed to be found and extracted during the web search.

Furthermore, even a different type of file can be easily converted to a `kws.phxstm` format. Fundamentally, the only notable difference between the two input types is the number of columns. `Stt.phxstm` has 7 columns, with the transcriptions being in the last one. `Kws.phxstm` has only one column and it contains keywords or transcriptions, each on a newline. The script looks specifically for these two filenames. If the name differs, the file is assumed to have the same format as `kws.phxstm` with only one column.

Both of these input source files contain only **fully normalized transcriptions**. In this case, a transcription is considered to be fully normalized if it meets the following requirements:

- contains only lowercase characters characteristic for that target language
- foreign words are inside the square brackets
- numbers are expanded to their full written form
- all punctuation marks have been removed
- redundant spacing has been removed
- spelling is denoted as a standalone character, immediately followed by an underscore

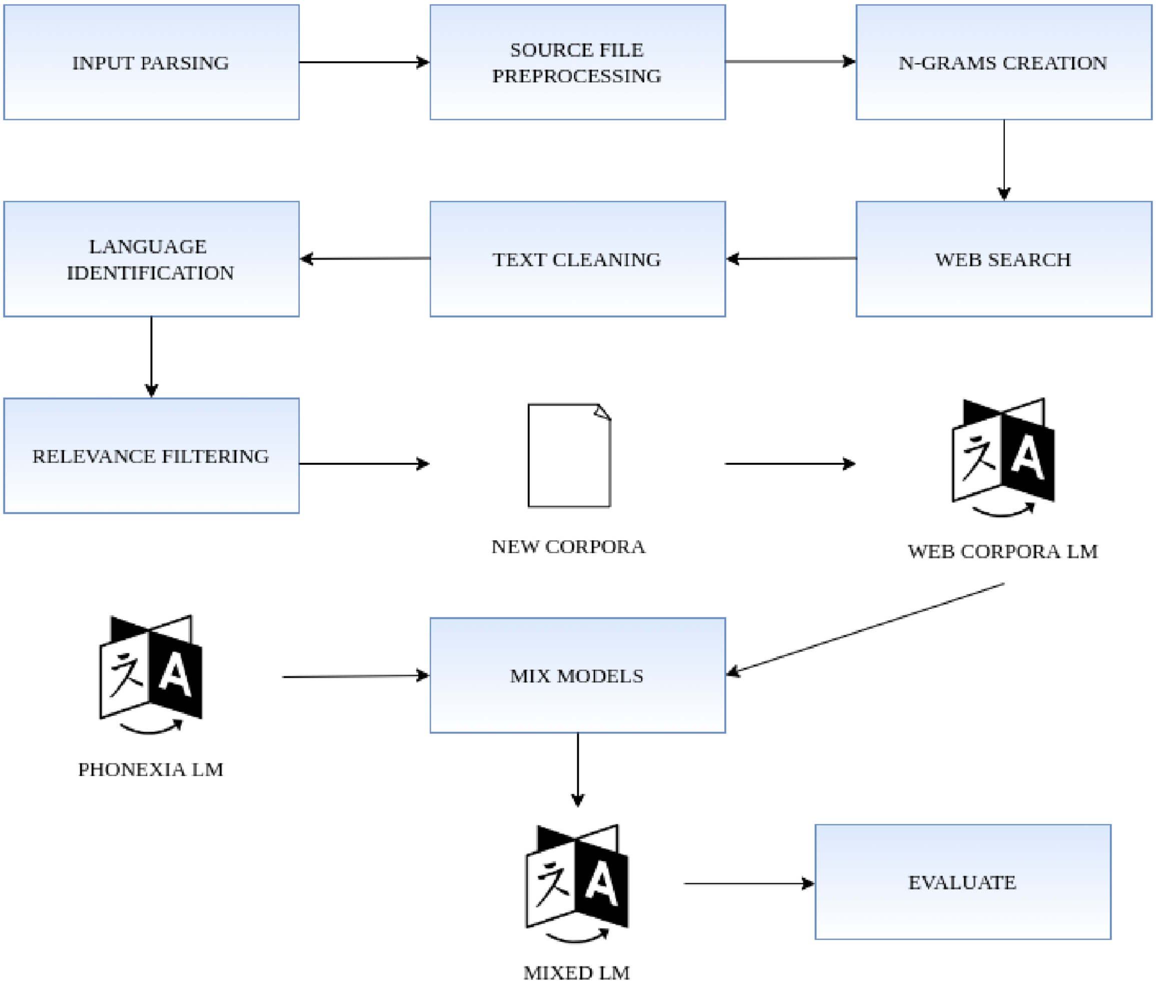


Figure 4.3: The design for the language model adaptation pipeline.

The designed pipeline consists of the following sequential stages:

INPUT PARSING

Input parameters are accepted in the form of a `JSON configuration` file. It holds all the hyperparameters for the pipeline, as well as paths to the input source file and results directory. The configuration file is parsed and checked for any invalid input parameters. Each hyperparameter has a constraint type assigned to it. If there is an invalid parameter or the parameter violates the imposed constraint, a respective error message is displayed. The next stage is initiated only if the configuration file passes all checks.

SOURCE FILE PREPROCESSING

The input source file, either `stt.phxstm`, `kws.phxstm` or a different file treated as a `kws.phxstm`, is loaded and pre-processed if necessary. Generally, these files contain transcriptions, which are already fully normalized. However, if the source file happens to be different or unprocessed, a cleaning module is provided for a quick pre-processing. This step is fully optional and the decision to use it is up to the programmer.

N-GRAMS CREATION

The input source file is analysed and occurrences of all the n-grams are estimated. By default, `trigrams` are estimated and used throughout this pipeline. The trigram model provides a nice balance between sufficient context and inexpensive memory requirements, making it a very good candidate for a quick web search. Trigrams also provide enough information, whilst they do not create too many word combinations, so their re-occurrence is more likely. Each n-gram is accessed and ranked relative to the others. Only the best scoring n-grams are to be used for the web search.

WEB SEARCH

Each chosen n-gram is converted to a query term and used for the web search. The web search is carried out with an arbitrary search engine or API. It is assumed that the choice and quality of the web search approach may directly influence the final results. Not only that, but paid, professional API has a potential to greatly decrease the time required for the web search and therefore, for the entire pipeline as well. The result of the web search phase is a set of links for documents that can be easily obtained with a `HTTP GET` request. The downloaded web documents get stored on the local disk and are re-used later on by default. Another reason for storing them is purely pragmatic – in case we need to look through a particular document in the future, it will be readily available.

TEXT CLEANING

After a web document is downloaded, it must be thoroughly cleaned. Firstly, only the paragraphs enclosed by the desired web tags are extracted from the web document. The extracted paragraphs must be then stripped of all the `HTML` tags and `markup`. Since web documents use vast number of different characters, it is very essential to remove all the characters not used in a standard written form of the target language. Furthermore, we want to clean the text so that it can be considered fully normalized. Higher quality of the cleaning process reduces the amount of invalid data that might get introduced to the language model.

LANGUAGE IDENTIFICATION

The cleaned text undergoes language identification to make sure the entire text is in the target language and not only a few sentences. Already pre-trained model is used to do the language identification. The decision threshold for language identification is a hyperparameter inside the configuration file and it can be freely adjusted. Ideally, the decision threshold should be strict and no lower than 0.75. It is undesirable to mistakenly add data in an incorrect language to our web corpora, and the threshold should reflect that.

RELEVANCE FILTERING

Finally, web documents are judged according to their relevance. We aim to keep those web documents that contain similar data as in the input source file. This stage can be eased or even skipped with a good search engine that naturally *ranks* the most relevant documents higher. A file marked as relevant can undergo further advanced filtering for its paragraphs or the document as a whole.

CREATION OF WEB CORPORA AND WEB LM

The data that successfully passed through all the previous stages is used to create the *web corpora*. The *web corpora* is recorded in a text file and contains all the extracted sentences, each one on a new line. This file is used to create the n-gram *web corpora* language model. We do not want to add this data to the *Phonexia language model corpora* directly, but rather create a separate *web corpora* and *web corpora language model* that gets mixed with the *Phonexia language model* afterwards.

MIX MODELS

Phonexia language model is mixed with the *web corpora language model* according to the mixing weights. Mixing weights are estimated with a characteristic text file. The Phonexia model is always used as the primary model during mixing of models. The resulting mixed model is an optimal combination of both language models given the characteristic text file. Therefore, the data used in the characteristic text file should be as close to the real target domain data as possible.

EVALUATION

After the mixed model is successfully created, it can be evaluated on the evaluation dataset. Ideally, *Phonexia language model* is firstly evaluated on the same evaluation dataset. After the mixed model is created, it can be evaluated with the same dataset and the results objectively compared.

Additionally, the *results* and *statistics* about performance of the pipeline are to be logged. *Experiments* module is to be implemented in order to easily run any pipeline experiments. The optimal hyperparameters of the pipeline will be established through experiments, but they can be adjusted at one's discretion.

4.3 Implementation

The language expansion pipeline is implemented in `Python` was tested on unix-based systems with the current latest version of `Python 3.6`. Object-based approach is used in order to nicely encapsulate all the individual components as objects.

The pipeline is controlled by an input configuration file in the `JSON` format. Each run of the pipeline is highly customizable, which is reflected by the amount of adjustable hyperparameters in the configuration file.

Input hyperparameters

Hyperparameters are split into 2 categories: **mandatory** and **optional**. Mandatory hyperparameters must be provided in the configuration file, otherwise the pipeline cannot be initialized. Optional hyperparameters do not have to be provided, although it is advised to specify as many hyperparameters (especially those language-dependant) as possible in the configuration file. If no value is provided for the optional hyperparameter, the default value is used.

This section provides an alphabetical list of all the hyperparameters, their constraint types, basic explanation and their default value. Mandatory hyperparameters are clearly denoted as such. Interesting hyperparameters which are used and modified throughout the experiments are marked with an asterisk *.

The implemented constraint types are:

- `bool_val` : equivalent to a standard `Bool`
- `filter` : filter type for advanced filtering of the text
- `lang` : 2-character string
- `negative_float` : negative float number range
- `path` : absolute path; if missing and not in `REQUIRED_VALUES`, the path is be created
- `positive_integer` : positive whole number range
- `positive_float` : positive float number range
- `search_pref` : preference for the web search, completely arbitrary and can be expanded if a new web search possibility arises
- `web_tags` : list of strings of web tags we wish to extract from web documents; the strings do not contain square brackets, i. e. `['p']` is valid, whilst `['<p>']` is not
- `zero_one_range` : only float values in the range of 0 to 1, used for probabilities

Alphabetical list of all the hyperparameters with a short description:

`create_ngrams` : `bool`, default is `True`
Determines whether n-grams will be created from the input source file. If `False`, the entire sentences from the input source file are used for web search instead of n-grams. The type of input source file does not matter, since this parameter only controls the n-grams creation. Throughout the experiments, this parameter is always set to `True`.

`dictionary` : **mandatory**, `path`
Absolute path to the dictionary file in the target language. The dictionary file is a `lexicon` in the format as described in the section 2.5.

`doc_default` : `positive_integer`, default is 25
For every searched term, its number of available web documents is estimated. If such information is unavailable or cannot be estimated, a default value of `doc_default` is used.

`doc_limit` : `positive_integer`, default is 50
Maximum number of documents that are searched per one term. If more than `doc_limit` documents are chosen to be searched, that value is trimmed to the `doc_limit` value.

`download_path` : `path`, default is `current working directory + /download`
Absolute path to the folder, where web documents are downloaded to. If the path does not exist, it is created in the current working directory. This approach allows us to reuse the already downloaded web documents without any redundant downloads.

`evaluation` : `bool`, default is `False`
Determines whether the final mixed model and the `source_model` will be evaluated on the `evaluation_datasets`, all located in the `evaluation_path`.

`evaluation_datasets` : no constraint, list of `sub paths`, default is `empty list`
List of sub paths for files used for the evaluation. All of these sub paths share the same prefix of `evaluation_path`.

`evaluation_path` : **mandatory** if `evaluation` is `True`
Absolute path to the directory, which is a shared common prefix for all sub paths in the `evaluation_datasets`.

`filter_threshold*` : `negative_float`, default is -4.5
Threshold for advanced filtering in the `log` domain. All segments or documents with the total score lower than the `filter_threshold` are accepted. Otherwise, they are filtered out and are not included in the final web corpora.

`filter_type` : `filter`, default is `median`
The type of filter used for advanced filtering. Possible values are `median`, `avg` (average) and `None`.

`is_standard_lang*` : `bool`, default is `True`
Determines whether the language is considered to be standard. This option allows for an analysis of characters of the input source file, which creates a clear constraint of graphemes for the web documents. Obtained paragraphs from the web documents are judged based on this grapheme constraint.

`k_ngrams*` : `positive_integer`, default is 500
Given an input source file, estimate the most common n-grams and use the top *k* best scored n-grams, where `k = k_ngrams`. If there are fewer n-grams than the value of `k_ngrams`, all found n-grams are used.

len_penalty* : positive_integer, default is 15
The optimal length of the entire n-gram in characters. All n-grams with length smaller than the len_penalty are penalized for bringing too little information. This hyperparameter is very language dependant and should be considered carefully based on the nature of the target language.

lid_threshold : zero_one_range, default is 0.9
Only documents with a confidence for the target language higher than the lid_threshold pass the language identification. The default value is so high because we want to avoid mistakenly adding data in an invalid language to our web corpora. It is advised not to use a threshold lower than 0.75.

ngrams_percentage : zero_one_range, default is None
Alternative to k_ngrams, it is possible to use a percentage range from 0 to 1. If the percentage value is valid and not None, the top ngrams_percentage of n-grams is used for the web search. If the value ngrams_percentage is available, it has higher priority than k_ngrams. If the value is None, k_ngrams is used by default.

order_ngram* : positive_integer, default is 3
Order of the n-gram model used throughout the pipeline. It is advised to primarily use n-grams for $n \in \langle 2, 6 \rangle$.

output_path : path, default is current working directory + /output
Absolute path to the folder where web corpora, web corpora language model and mixed language model will be stored. If the path does not exist, the folder is automatically created in the current working directory.

ppl_path : path, default is current working directory + /ppl
Absolute path to the folder where relevance filtering results are stored. If the path does not exist, the folder is automatically created in the current working directory.

ppl_threshold* : positive_integer, default is 1200
The threshold value of perplexity for relevance filtering. All documents with higher perplexity value than this threshold are filtered out and not considered for the final web corpora.

search_preference : string with possible values of google and bing, default is google
Search preference for the web search. This list can be updated or adjusted with the introduction of another web search engine or API. Currently, google from the python google module is the best choice that is also free of charge. Bing uses standard Bing API, but the free version is limited to only around 1500 transactions per month.

source_model : **mandatory**, path
Absolute path to the Phonexia language model in the target language.

source_path : **mandatory**, path
Absolute path to the input source file. This file is used as a sole source of the input text data for conducting the web search.

`statistics_path` : `path`, default is `current working directory + /statistics`
Absolute path to the folder, where the statistics files are stored. Each statistics file is named after that particular pipeline's characteristic `fingerprint`. If no directory is given, it is automatically created in the current working directory.

`target_language` : **mandatory**, `lang`
Case-insensitive ISO 639-1 2-character target language code.

`timeout` : `positive_integer`, default is 90
Time limit for processing a single link. It the link is not fully processed within the set `timeout` limit, its processing is terminated and the next link is processed.

`trim_input` : `bool`, default is `True`
Clean the input source file if not yet fully normalized. Useful for quick use of the pipeline without pre-processing the source input file. For the best possible results, it is advised to pre-process the source file manually and check it for any discrepancies beforehand.

`use_doc_filter*` : `bool`, default is `False`
When set to `True`, the whole web documents undergo an advanced filtering with filter of a given `filter_type` and only documents with the score lower than `filter_threshold` are accepted.

`use_window_par_filter*` : `bool`, default is `False`
When set to `True`, all paragraphs of the downloaded web document undergo an advanced filtering. If `window_len` is set to non-zero value, each paragraph is considered to be a single segment. If `window_len` is set to a given integer value, every paragraph is split into segments with a given `window_len` length. All segments are scored and filtered with filter of a given `filter_type` and only segments with score lower than `filter_threshold` are accepted.

`web_tags` : `web_tags`, default is `['p','span']`
List of web tags that are extracted from each web document.

`window_len` : `positive_integer` or `None`, default is `None`
Parameter used for advanced filtering when `use_window_par_filter` is set to `True`. It determines, whether each paragraph is split into smaller segments for filtering or not.

Classes description

This pipeline is implemented through smaller components in the form of objects. Each class provides at least one public function and numerous private functions. Public functions of each class are introduced in their description below. If a class uses one of the input hyperparameters, it is mentioned in the description together with some practical advice on how to correctly choose the hyperparameter.

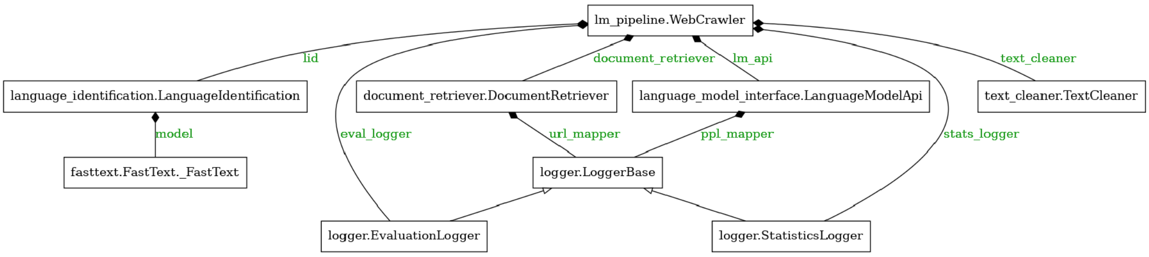


Figure 4.4: Implementation of the pipeline.

The `WebCrawler` class is the main class that initializes all the smaller components. It is initialized with an instance of the `ConfigParser` class. This is a list of all the classes implemented in the pipeline:

ConfigParser

Class for parsing the input configuration `JSON` file and storing values of all the input hyperparameters. This class does not have any public methods, but it loads all the hyperparameters into its public attributes. An instance of this class is initialized with absolute or relative path to the configuration file. During initialization, all the input hyperparameters are checked for validity and if there is any incorrect parameter, a proper error message is displayed.

The `ConfigParser` is implemented so that only mandatory input parameters are required to initialize the pipeline. All non-mandatory parameters which are missing from the configuration file are supplemented with their respective default values. Adding a new parameter or adjusting the existing ones is very straightforward as well.

List of all the parameters and their constraints is kept in a separate header file named `config_parser_values.py`. It contains a list of `REQUIRED_VALUES`, a dictionary for default values `DEFAULT_VALUES` and a dictionary of constraints `CONSTRAINT_TYPES`.

In order to successfully add a new input parameter to the pipeline, we only need to add 3 lines of code. Firstly, add the new parameter name either inside `REQUIRED_VALUES` list or `DEFAULT_VALUES` dictionary together with its default value. Secondly, add the new parameter name and its constraint type to the `CONSTRAINT_TYPES` dictionary. Lastly, add a line of code inside of the `ConfigParser` class initialization and load the newly added parameter into its public attribute with preferably the same name as the hyperparameter.

WebCrawler

The main class that instantiates all the other classes and uses their public methods. Each instance of the `WebCrawler` must have its characteristic arbitrary fingerprint. It is either given during the initialization or it gets generated if `None` is provided. In order to keep the naming style concise, the automatically generated fingerprint is actually a hash of all the input parameters rolled out into a string. The default hash function is `Message Digest (MD5)`, which is known to be suitable for creation of a unique identifier. It outputs a fixed length 128-bit vector, which is a great option for the fingerprint. This approach allows for easy recollection and identification of experiments and their hyperparameters. Furthermore, if there is a model with the identical fingerprint available, it means that such experiment with the same input parameters was already carried out, so it can be skipped.

If `is_standard_lang` input parameter is set to `True`, it automatically estimates the allowed characters from the input source file. All Arabic numerals are automatically included in the allowed character's set, as well as all lowercase and uppercase versions of all the found graphemes. This character's set is used during document cleaning in one of the `TextCleaner` methods.

Additionally, each link exploration (downloading, cleaning, language identification, relevance filtering, logging…) is limited with a time limit. Many websites require additional authorization before download, which may halt the simple `HTTP` communication inside the pipeline. Furthermore, the pipeline relies on different freeware components that may malfunction at any moment, which led to the introduction of timeout.

Each web link exploration is run as a standalone process guarded by a timeout. Usually, very short documents take a couple of milliseconds, whilst middle and large size documents need a few seconds to get fully processed. Of course, there are some outlier cases which require more time and the time limit should reflect that. Therefore, the default value of the time limit is set to be 90 seconds.

Initially, there was an effort to give all the timeout links a second chance. However, after some observations of the experiments, it was concluded that over 90% of all the timeouted links that are given a second chance result in timeout for the second time as well. Therefore, this part of the design was scrapped.

The pipeline can be used by calling the `lm_pipeline` function. This function firstly parses the parameters with `ConfigParser` and creates an instance of the `WebCrawler` class named as `pipeline`. Then, class method `already_done` is called to verify, whether a model with the same fingerprint already exists. If it does, the experiment is skipped. If not, a function `search_web` is called to create the web corpora. This is the most time-consuming function of the pipeline. Finally, `mix_models` and `evaluate` functions are called. These two functions can be omitted if we wish to only download the web corpora and not mix or evaluate the models.

NgramsProvider

Class for analysing the input source file, creating n-grams and choosing the terms for web search. If `create_ngrams` is set to `True`, n-grams will be created and ranked based on the analysis of the input source file. If set to `False`, the input source file is read line after line and no n-grams are created, but the whole sentences are used instead. Such file may contain keywords, but also full sentences if we want to be very specific.

The order of ngram specified in the `order_ngram` is used throughout this class. The default value is 3, as trigram models offer a good trade-off between accuracy and memory requirements. Once all n-grams are created during initialization, they can be obtained by calling the public method `get_top_ngrams(self, k, percentage)`. The value of `k` corresponds to `k_ngrams` and `percentage` to `ngrams_percentage`. If `percentage` is not `None`, then the top `percentage` of n-grams will be chosen. If `percentage` is `None`, then the top `k` n-grams will be chosen. The ranking of all n-grams is taken from Zhang et al. [23] as their initial document count estimate:

$$DC(\text{ngram}) = DF(\text{ngram}) \times P(\text{ngram}) \tag{4.1}$$

The document count metric $DC(\text{ngram})$ represents the number of all the documents we expect to obtain if we do a web search for that particular ngram. It consist of two subcomponents: *document frequency* $DF(\text{ngram})$ and *precision score* $P(\text{ngram})$. The document frequency is just a simple count of all occurrences of that particular ngram in the input source file, which can be done in one command with `collections.Counter`. The precision score ($0 < P(\text{ngram}) < 1$) aims to penalize n-grams that are too short, because we hypothesize that longer n-grams are more likely to return a document in the target language. Our adjusted precision ad hoc formula is then calculated similarly to [23] as:

$$P(ngram) = \min\left(1.0, \left(\frac{\text{length}(ngram)}{\text{len\_penalty}}\right)^2\right) \tag{4.2}$$

where the `penalty_len` is a language dependant input parameter. The ngram is joined to a single string with spaces and the number of characters in the resulting string is the n-gram's length. In the original paper [23], the `len_penalty` is set to 20 by default. However, we decided to allow the user to dynamically set the value for `len_penalty` inside the input configuration file.

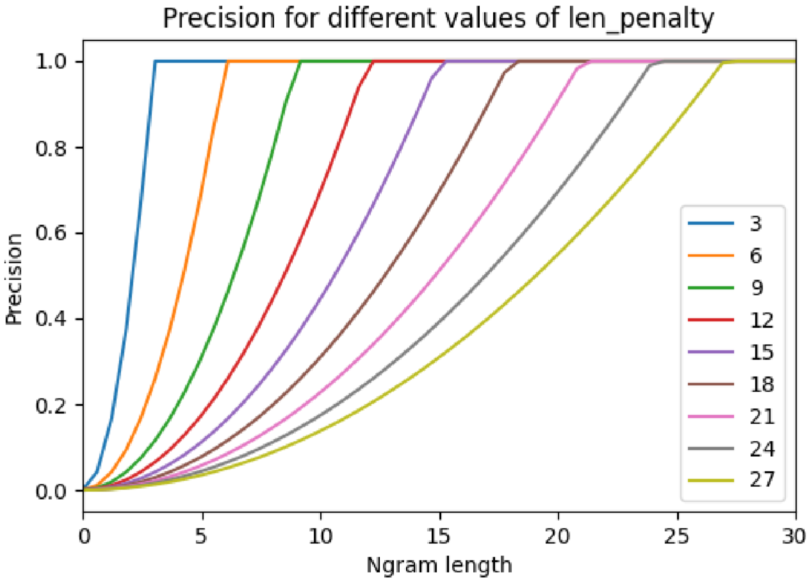


Figure 4.5: Graph of precision for different initial values of `len_penalty`.

Although it may seem insignificant, `len_penalty` directly influences ranking of all n-grams and should be considered carefully. Ideally, `len_penalty` should be estimated by a skilled linguist. If such option is unfeasible, we can do a naive estimation based on a simple language analysis. Firstly, we need to find an average length of a single word in the target language. It can be estimated either from the input source file or from a large corpus of text data in the target language. Then, we can naively estimate the optimal value for `len_penalty` as follows:

$$\text{roundup}(\text{order\_ngrams} \times \text{avg\_word\_len} + (\text{order\_ngrams} - 1)) \tag{4.3}$$

The average word length is multiplied by the order of n-grams and $(\text{order\_ngrams} - 1)$ represents spaces used for joining the n-gram tuple into a single string. The entire expression is then rounded up to a whole number.

The last public function provided by this class is `get_expected_count(self, term)`. It outputs the expected number of documents (DC) for given term. This value is then used by the `DocumentRetriever` class.

DocumentRetriever

The DocumentRetriever class nicely encapsulates all the web communication and document retrieval. It is initialized with search_preference, download_path, doc_limit, doc_default, web_tags and target_language. The class implements functions for web searching that are built on either BingAPI or Googlesearch python module. What type of web search is used depends on the search_preference value. Functions for different API or web search can be easily implemented in the future if necessary. Included BingAPI is set up with a free of charge subscription and therefore allows only for around 1500 transactions per month.

On the other hand, the Googlesearch python module is built upon native Python modules, so it can be used with no charges. The only downside of this module is the occasional request limit exceeding. In that situation, the server stops responding and the pipeline automatically terminates. We need to wait for some time before running the pipeline again, otherwise the server will keep responding with the same limit excess error code.

I aim to keep the web requests at minimum, so the pipeline stores and recycles the web documents by default. This means every document is stored in the download_path folder under a unique name. Since web links are usually quite long, we cannot store the documents under their web links directly. In order to compress the web link and create a unique identifier, hash function called Message Digest (MD5) is used. Storing the web documents on the disk reduces the number of consequent web requests, which makes using the Googlesearch python module more feasible. Additionally, it speeds up the experiments, since many documents can be re-used from storage instead of being redundantly downloaded all over again. The mapping of links to their identifiers is stored in the url_mapper file inside the download_path directory.

The class implements 2 public functions. Firstly, search_term(self, term, cnt) uses the standard search_preference to search cnt number of documents for the given term. The value of cnt is estimated in the NgramsProvider class. If the value of cnt cannot be estimated for any reason, then the doc_default value is used. If cnt exceeds maximum number of documents given in doc_limit, then doc_limit value is used as cnt. If target_language is passed as an argument during class initialization, all web searches are given the language code, which increases the hit rate of documents in the target language. This function returns a list of web links.

There was an effort to incorporate .pdf and .doc files processing in this class. I experimented with various Python modules and then APIs, but none of them provided satisfactory results. If the downloaded documents were only in English, those tools would suffice, but they were not reliable enough for other non-standard languages. Thus, the idea was scraped and all web links with the filename extension .pdf or .doc are filtered out.

Second public function get_document_by_url(self, url) is given an url link and returns extracted list of paragraphs from the web document. Firstly, the script checks whether that web document is not already downloaded. If it is found in storage, the file is read and returned. If not, the web document is obtained with a simple HTTP GET request through Python module requests. The obtained web document is parsed with BeautifulSoup and web tags specified in web_tags are extracted. You can refer to this page[1] for a full list of available HTML tags. The script can function for any type of web tag, but naturally, it is strongly advised to extract only web tags containing text. Once

[1] https://way2tutorial.com/html/tag/index.php

the text is extracted, it is stripped of any lingering HTML tags and stored locally. However, the downloaded documents contain text only from the tags defined in `web_tags`, so it is important to keep that in mind when re-using the downloaded data in-between experiments.

TextCleaner

Small class that implements methods for quick, easy and automatic text cleaning. It provides a method `clean_document(self, content)` for complex document cleaning, which can be constrained with a chosen characters constraint set. The characters constraint set is estimated in the `WebCrawler` class initialization from the input source file. When cleaning a document, each paragraph is cleaned separately and empty ones are removed right away. If the characters constraint set is not `None`, the paragraphs are checked for any constraint violations. All paragraphs are searched through and those that contain any characters outside of the allowed characters constraint set are filtered out.

Initially, there was an effort to fully clean the web documents automatically, but they are often contaminated with various unpredictable characters. All web documents undergo a basic cleaning procedure, but it is difficult to predict everything, so some special characters might remain. We want to avoid clustering our language model with unnecessary rubbish, so it is safer to filter out those affected paragraphs. That led to the introduction of the characters constraint set. The characters constraint set is estimated only if `is_standard_lang` is set to `True`.

Therefore, the level of data pre-processing for the input source file directly influences the quality of our resulting language model. Furthermore, `clean_document` method automatically removes paragraphs, where more than half of the total number of characters are numbers. If the language is standard, all numbers are expanded to their full written form with Python module `num2words`. All of these steps are to ensure the highest possible quality of the automatically created corpora file.

`TextCleaner` also provides a method for trimming the input source file if it had not been properly processed. Trimming of the input file includes adjusting spacing, removing general interpunction marks, removing underscores for spelling and converting all letters to lowercase. However, it is still advised to fully pre-process the input source file and not rely on the pipeline for that. If it is already fully pre-processed, `trim_input` can be set to `False` in order to save some computational resources. If set to `True`, public method `trim_input(self, content)` is called and the input source file is quickly pre-processed.

LanguageIdentification

Instance of this class is initialized with the `target_language` ISO 639-1 code and a threshold value `lid_threshold`. It utilizes `Python fastText module` and a pre-trained `fastText` language identification model. When this class is initialized for the first time, it attempts to download a fastText language identification model named `lid.176.bin` [13]. The `lid.176.bin` model is capable of recognizing 176 standard world languages. The target language ISO code is checked with the `Python module pycountry`. It holds database of all the ISO codes for languages, countries and currencies, so it can be used to check the inputted `target_language` code before passing it to the fastText model.

This class implements a single public function `is_target_lang(self, text)` that returns a Boolean value. If the inputted text is judged to be in the target language with confidence higher than the `lid_threshold`, `True` is returned. Otherwise, `False` is returned.

The fastText model outputs a list of languages and their confidence rates. Language with the highest confidence rate is considered to be the model's choice.

In order to only obtain the web documents in the target language, the lid_threshold has to be at least 0.75. This threshold needs to be even higher for a language with close sister languages. There were several experiments conducted with lid.176.bin before it was added to the pipeline.

Firstly, the same Wikipedia web document[2] was downloaded in the following language pairs: Czech and Slovak, Spanish and Portuguese, Russian and Ukrainian. These language pairs were chosen deliberately, because they are very similar in grammar, vocabulary, writing style and origin. Furthermore, each pair share a lot of lexical similarity, so its native speakers often understand each other even though they are not speaking the same language. The figure 4.6 shows Indo-European languages, which language family each one belongs to, and finally, their *lexical distance* to other languages. It is advised to always research what languages belong to the same language family so that an informed decision about the lid_threshold can be made.

Initially, each document was accessed by lid.176.bin to explore, how confident the model is for each document to be in its target language when it really is in the target language.

	Czech	**Slovak**	**Spanish**	**Portuguese**	**Russian**	**Ukrainian**
Confidence (%)	99.85	98.44	97.22	98.44	99.74	99.97

Table 4.1: Confidence of lid.176.bin for each document in its respective language.

The model has a very high, almost 100%, confidence that each document is really fully written in its target language. Subsequently, a new document was sampled from the two same documents in the language pair. In the new mixed document, each paragraph is randomly selected from one of the documents. Sampled document is then judged by lid.176.bin in multiple experiments. I carried out 1000 experiments for each language pair. The concise results in the Table 4.2 show obtained minimal, maximal and median values of the model's confidence.

[2] https://en.wikipedia.org/wiki/Cat

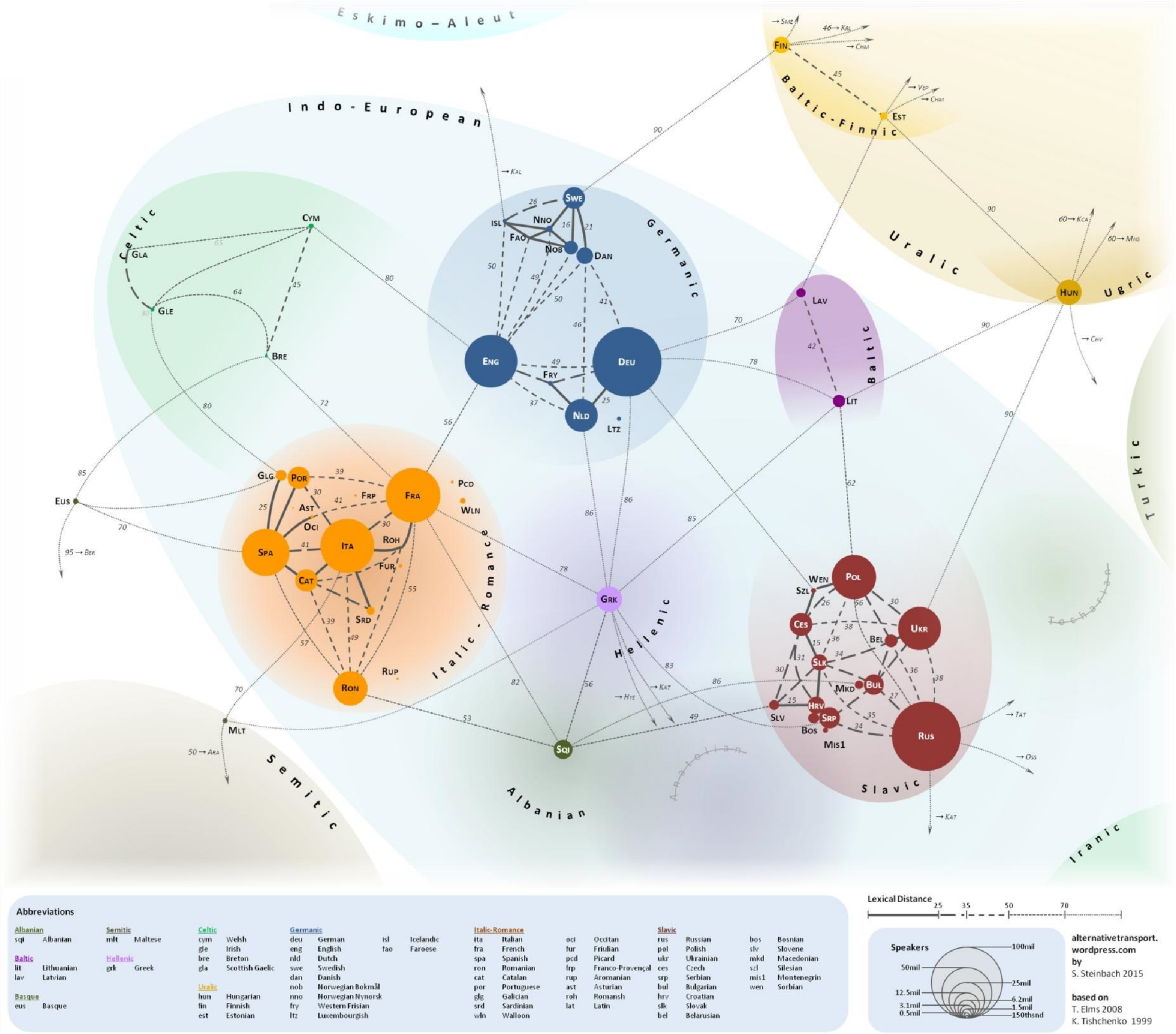


Figure 4.6: Lexical distance among European languages. Taken from [21] and adjusted.

.

Czech ratio (%)	**Slovak ratio (%)**	**Identified language**	**Confidence (%)**
40.78	59.22	Slovak	50.47
55.70	44.30	Czech	84.47
71.90	28.10	Czech	97.61

(a) Experiment results for Czech and Slovak language pair.

Russian ratio (%)	**Ukrainian ratio (%)**	**Identified language**	**Confidence (%)**
40.94	59.06	Ukrainian	49.97
43.22	56.78	Ukrainian	61.60
53.31	46.70	Ukrainian	91.64

(b) Experiment results for Russian and Ukrainian language pair.

Spanish ratio (%)	**Portuguese ratio (%)**	**Identified language**	**Confidence (%)**
52.16	47.84	Portuguese	46.13
42.48	57.52	Portuguese	68.25
27.62	72.38	Portuguese	88.07

(c) Experiment results for Spanish and Portuguese language pair.

Table 4.2: Experiments of language identification model `lid.176.bin` for randomly mixed documents in the language pair.

Although the documents were mixed randomly, the `lid.176.bin` model made very rational choices. Slovak and Czech share a lot of similarities, but Czech contains some special characters (*ř*,*ě*) that appear quite frequently in words. Therefore, it usually identifies the document as being fully written in Czech. Out of the 1000 experiments, the model identified 997 mixed documents as being written in Czech.

The same thing happened for Spanish and Portuguese. Moreover, these two languages look mutually intelligible in their written form, the only difference is that Portuguese has some additional special letters (ç, ã …), similarly to Czech. Therefore, the model almost always chose Portuguese - 987 out of 1000 experiments.

For Russian and Ukrainian, the results were more mixed. The model chose Ukrainian for around 72% of experiments and Russian in the remaining 28%.

Conclusively, the `lid.176.bin` model makes very reasonable estimates. When faced with a mixed document, it tends to make a decision based on the occurrence of graphemes.

LanguageModelApi

This pipeline utilizes the SRI Language Modelling Toolkit (`SRILM`) which has been under development since 1995 and is freely available for non-commercial purposes. `SRILM` is a collection of C++ libraries, executable programs and other miscellaneous scripts for statistical language modelling [22]. It is available for download on its official website[3]. Unfortunately, `SRILM` is available only in the source form, so it must be manually downloaded, compiled, installed and added to the environment variable `PATH`. If it is not available in `PATH`, the entire pipeline cannot be initialized. Successfully installed `SRILM` can then be directly used inside Unix terminal, including pipelining of multiple `SRILM` commands.

[3] http://www.speech.sri.com/projects/srilm/

Since the rest of the pipeline is fully implemented in Python, I decided to build a simple application interface layer in Python over the SRILM toolkit. The toolkit allows for language model creation, as well as evaluation. The implemented API provides functions for the following:

- creation of a new language model limited with a given dictionary
- perplexity evaluation of an existing language model with test data
- mixing of two existing language models

When the pipeline is initialized, a new language model is created from the input source file with `create_source_lm(self, content)`, where `content` is a list of sentences from the input source file. By default, the script uses the Kneser Ney smoothing (Section 3.2) technique for every language model creation. Language model is automatically reduced with dictionary defined in the input hyperparameter `dictionary`. The input source file language model is created during initialization, so it can be used to evaluate the downloaded web documents.

The decision threshold for accepting or rejecting a web document is given in the input parameter `ppl_threshold` and must be chosen experimentally. All documents above the aforementioned threshold are rejected, since we aim to choose documents with reasonably small perplexity.

Besides the standard filtering with `ppl_threshold`, more advanced means of filtering were implemented. By default, the advanced filtering is not used unless specified so in the configuration file. The advanced filtering is based on the Zhang et al. [23] relevance filtering and expanded a little further. The goal of the advanced filtering is to give the user more detailed control of what documents (or their parts) are accepted or rejected. Zhang et al. approached this problem by selecting a fixed size window of 9 and then split the documents into segments of that window's length. Each segment is then scored and judged against the given threshold. They use a statistical scoring model created from the training documents to assign document frequency to each word in a segment. The score of the entire segment is therefore just the average document frequency of all the words in a segment.

Instead of building a statistical model, I decided to utilize the `unigram` probabilities from the Phonexia `source_model`. Although probability is usually expressed in the range from 0 to 1, very small probability might result in a numerical `underflow` in the digital computers. Because of that, probabilities in the language model are generally expressed in the `log` domain. The higher the value in the log domain, the more probable it is and vice versa. An unknown word is assigned the value of `negative infinity`.

The advanced filtering is controlled with `use_doc_filter` for the whole document filtering and `use_window_par_filter` for paragraphs filtering. If any of them is set to `True`, the `unigram` log domain probabilities from the Phonexia LM are read and stored during the class initialization.

I demonstrate the advanced filtering possibilities and their results with the following concrete example. All of the used `unigram` log values are actual values from Phonexia's language model for English. Consider a document with the following 3 fully cleaned paragraphs as they are represented internally in the pipeline:

[„that's my spot said dr sheldon cooper“,
„leonard looked annoyed but moved eventually“,
„there's no point in arguing with sheldon anyway“]

Document-based filtering joins all the document's paragraphs into a single `string`. Each word in the string is assigned its unigram log domain probability of occurring. Finally, the scored segment vector is then passed through the `median` or `average` filter. The resulting score is compared with the `filter_threshold` and a decision is made.

For document-based filtering, it is strongly advised to only use the `median` filter. The `average` filter has a disadvantage in this case. Once any word in the document is unknown to the `source_model` and is assigned the `-inf` value, its score with `average` filter ends up being `-inf` as well. Therefore, `median` filter is better suited for document-based filtering.

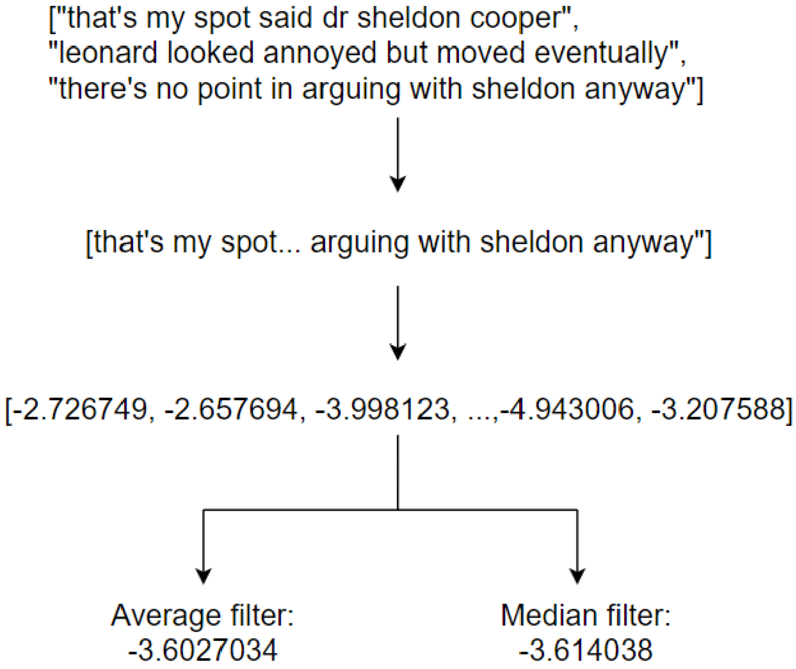

["that's my spot said dr sheldon cooper",
"leonard looked annoyed but moved eventually",
"there's no point in arguing with sheldon anyway"]
[that's my spot... arguing with sheldon anyway"]
[-2.726749, -2.657694, -3.998123, ...,-4.943006, -3.207588]
Average filter:
-3.6027034
Median filter:
-3.614038

Figure 4.7: Example of document-based advanced filtering.

Paragraph-based filtering judges each paragraph separately, giving the user closer control over what is included in the web corpora. If `window_len` is set to `None`, each paragraph is considered to be a separate segment. Otherwise, each paragraph is split into windows of a given `window_len` length. When all the segments are created, each of them is judged in the same way as shown in the figure 4.7.

Every web document that passes through the pipeline is immediately added to the final corpora file with `write_corpora(corpora, path, append)`, where `corpora` is a list of sentences, `path` is the absolute path to the corpora file and `append` is a self-explanatory Boolean value. The corpora file is named after the pipeline's `fingerprint` as `corpora_fingerprint`. Once the web searching is finished, `corpora_fingerprint` is used to create a web corpora language model named `model_fingerprint`.

The last step is mixing the Phonexia language model with `model_fingerprint` over a test file. The models are mixed with a public method `mix_models(self)`. The test file is needed in order to accurately estimate the mixing weights λ for both models. Ideally, the test file should contain representative data from the target domain. The resulting mixed language model is stored as `mixed_model_fingerprint`.

Next public function is `evaluate_ppl_doc(self, corpora, exp_fingerprint, link_fingerprint)`, where `corpora` is a list of sentences for evaluation, `exp_fingerprint` is the characteristic fingerprint for the pipeline and `link_fingerprint` is the characteristic fingerprint for the web link. The given corpora is automatically evaluated against the language model created from the input source file. The evaluation results are stored in the `ppl_path` directory inside a file named after `exp_fingerprint` joined with `link_fingerprint` by an underscore. Evaluation results are stored by default so that we can refer to them later if necessary. The mapping of the ppl files is stored in a file named `ppl_mapper` in the `ppl_path` directory. This function returns a measure of `perplexity`.

Last public method is `already_done(self)` and returns a Boolean value. It checks whether there is a model with the same `exp_fingerprint` available. If there is, it means experiment with the same input parameters was already carried out, so it can be skipped.

Logger

The class `LoggerBase` creates a foundation for its inherited classes: `ExperimentsLogger`, `EvaluationLogger` and `StatisticsLogger`. We do not only want to log standard information, but also information about the experiments, evaluation and overall statistics.

The parent class `LoggerBase` provides a basic function for writing any arbitrary content to a file. The inherited classes are equipped with additional functions for parsing and logging the results. Initially, only the `LoggerBase` class was used and all the results were formatted before being passed to the logger function. However, this made the other parts of pipeline very cluttered and cumbersome, so the inherited classes were implemented. Each of them accepts data, which is then formatted and organized within its public logging function.

Firstly, `ExperimentsLogger` is used to hold information about the experiments. Each experiment run with this pipeline is logged in the experiment file. Each line contains unrolled input configuration parameters for that experiment, with the `exp_fingerprint` in the last column. Since all corpora results and models are named after the `exp_fingerprint`, this experiments logger file holds the exact mapping of what input parameters correspond to that fingerprint.

Secondly, `EvaluationLogger` deals with the evaluation results. Each experiment has its own evaluation file named after its `exp_fingerprint`. Firstly, the original Phonexia language model is evaluated and logged. Then, the mixed model is evaluated and logged, so the results can be directly compared and judged. Each line contains the absolute path to the model, absolute path to the evaluation file and then the output from `SRILM`'s *ngram* perplexity evaluation. That includes number of words and sentences in the evaluation file, oov rate, logprobability and two values for perplexity, one normalized and the other not.

Lastly, `StatisticsLogger` is used to log results from every single link exploration. Each experiment has its own statistics file. Links that resulted in timeout are not logged. Each line contains the searched term, explored link, document length right after the download, whether the document passed language identification, document's perplexity evaluation, document length after it is fully cleaned and finally, percentage gain. The percentage gain represents, how much information was actually obtained after it passed through the pipeline. It is calculated from the cleaned document length and the raw document length right after download.

The statistics file is very important for a quick system recovery. Since it holds information about all the previously processed terms, it is used for swift continuation of the script if the script or any external component fails. When the pipeline is initialized with its `exp_fingerprint`, the script searches for statistics file with the same fingerprint. If the statistics file exists, but the mixed model with the same fingerprint does not, it means the script was interrupted during corpora preparation. In that case, statistics file is read and all searched terms are extracted. The last term from all the extracted terms is removed, since its search might not have been fully finished when it was interrupted. Once the NgramsProvider chooses the top n-grams, the script carries out set difference with the extracted terms, so only those yet unprocessed terms are searched. This approach reduces redundant web searches and allows for easy recovery at the approximate point of failure.

Statistics visualization

Small script named create_statistics.py that processes any arbitrary statistics file and outputs accumulated statistics and graphs. This script is not integrated in the pipeline and must be run manually. It is used to quickly visualize results from each experiment. The absolute path to the statistics file is set up manually at the beginning of the script.

First graph shows histogram of raw document length and clean document length side by side. Second graph plots histogram of how many links were used for each search term. Third graph plots a histogram of language identification results. Last graph plots a histogram of how many documents were actually extracted per each term. These results provide data about web documents and how they were assessed by the pipeline at each stage. However, these results alone do not truly reflect the overall contribution of the pipeline, they are only informative.

Experiments module

Experiments module is implemented to run multiple experiments with the same various configuration settings. Of course, the pipeline can be used as is, but if we want to run multiple experiments or create multiple web corpora LM models, this script can take care of it. It accepts input in the same type of configuration file with a little adjustment – each configuration parameter value must be enclosed in a Python list. These lists can contain multiple values for each parameter.

Once the configuration file is read, all possible parameter combinations are created. The script then initializes the language model expansion pipeline for each set of parameters and carries out experiments one by one. Since all web documents are stored locally by default, it is advised to carry out experiments with higher value of k_ngrams or ngrams_percentage first. Consequently, the initial experiment downloads a lot of web data and the subsequent experiments can re-use a big chunk of it, speeding up the entire process.

Chapter 5

Experiments

Experiments are conducted in order to evaluate the overall contribution of the implemented language model expansion pipeline. The experiments were carried out on the Phonexia's `SGE` server and on my personal computer.

This chapter presents experiments and the obtained results. The overall goal of experiments is to run the pipeline multiple times, compare the results and assess its contribution. All experiments are language dependant. After each set of experiments is finished, the best model is chosen, packaged and used for decoding.

Firstly, section 5.1 conducts experiments on Hindi, section 5.2 on Czech and section 5.3 on Mandarin Chinese. Each experiment section begins with language specific information, followed with some setup information. Lastly, obtained results are presented and model with the best evaluation score is chosen and packaged for decoding.

5.1 Experiment A - Hindi

Experiments for Hindi were conducted in the earlier stages of the pipeline development. At the time of writing this paper, Phonexia is yet to develop their own ASR system for Hindi. However, I wished to test the pipeline on generic system and task to see, whether it could be quickly adjusted and used for other non-Phonexia systems.

Therefore, the following experiments use data from the `HINDI ASR CHALLENGE 2022`[1]. The data was collected by a social technology enterprise Gram Vaani and includes spontaneous telephone speech recordings in regional variations of Hindi. This dataset is said to contain speech with natural background noise and timestamped transcriptions with various levels of accuracy. The accuracy of transcriptions varies, because this project was crowd sourced.

Hindi characteristics

Hindi is one of the official languages currently used in India. Although India has no national language, it is home to over 400 unique languages. In the 20th century, there was an effort to establish Hindi as a sole official language. However, this proposal was met with resistance and dissatisfaction in many regions of the country [4].

Currently, Hindi and English are established as the official languages. Additionally, each region in India has its own list of officially recognized languages. This makes most of

[1] `https://sites.google.com/view/gramvaaniasrchallenge/home`

the people in India at least bilingual (English, Hindi), with many of them being trilingual (English, Hindi, regional language) or even polylingual (4+ languages). However, Hindi remains the common language with over 500 million speakers.

Hindi belongs to the Indo-Aryan language group and its writing system is based on Devanagari. Devanagari is a syllable-based system, so each syllable consists of a consonant followed by an inherent vowel. Altogether, Hindi has 33 consonants and 11 vowels. The vowels can be nasalized or not. Written form of each vowel changes depending on whether the vowel follows a consonant or is standalone.

The Devanagari writing system is used for other regional languages and dialects used in India. In order to convey a rough idea of what the writing system visually looks like, here is a sample of all the Hindi consonants:

कखगघङचछजझञटठडढणतथदधनपफबभमयरलवशषसह

Grammatically, Hindi is a highly inflected language. The inflections are realized through prefixes and suffixes. The typical word order is **Subject** – **Object** – **Verb** (SOV) [2]. According to the FSI's ranking of language difficulty (by default for native English speakers), Hindi is classified as a category IV language. That makes Hindi one of the harder languages to learn with an estimated 1100 hours of study required to achieve proficiency in speaking and reading [3].

Mixing models in Kaldi

These experiments utilized the implemented pipeline for corpus creation, but mixing and evaluation was done with Kaldi. Kaldi is a toolkit for speech recognition written in C++ used by the ASR research community [19]. The projects inside Kaldi are organized in the form of recipes.

The web corpora and training corpora of Gram Vaani are pre-processed by removing some special tokens and then, G2P is used to generate pronunciation of the newly obtained vocabulary. This recipe's G2P uses Phonetisaurus[2] that needs to be manuallly installed beforehand. Finally, the two models can be mixed. Before mixing, it is advised to reduce the corpora by given dictionary. Since Phonexia did not have any dictionary for Hindi, it was kindly provided by my supervisor.

In experiments with Hindi, the source_model we mix with the web corpora is actually a language model created from the training data of Gram Vaani dataset. The web corpora language model is mixed with the source_model to create the mixed_model. The characteristic file for estimating the mixing weights is the development data file of Gram Vaani dataset. Lastly, both of the models are evaluated on the development set from Gram Vaani dataset to determine their *perplexity*.

Experiments

There were 15 experiments conducted for Hindi. Each experiment used different set of values for the following input hyperparameters: k_ngrams, ppl_threshold and len_penalty. Other hyperparameters had their default values. Experiment or language specific parameters were set accordingly and remained the same throughout the experiments. After all corpora files were downloaded, each of them was manually setup in a Kaldi recipe, mixed and then evaluated. The perplexity of source_model for the development set was quite low

[2] https://github.com/AdolfVonKleist/Phonetisaurus

at 251.96. Out of 15 experiments, only the 8 top results are shown in the following table. Results are sorted by the improvement rate.

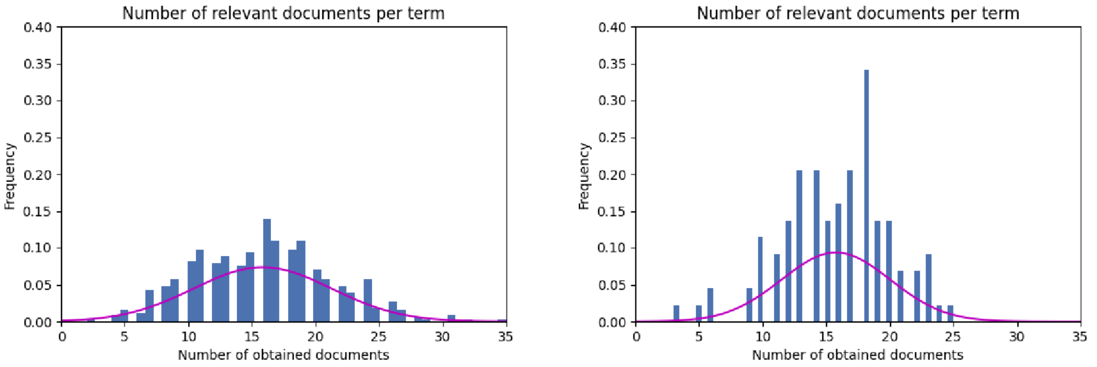
k_ngrams	ppl_threshold	len_penalty	perplexity	improvement (%)	# of lines
500	1200	6	234.18	7.06	229785
500	1300	6	234.63	6.88	255076
500	1200	9	235.42	6.56	232708
250	1000	9	237.77	5.63	141627
250	1000	3	237.95	5.56	121025
250	1000	6	238.73	5.25	116062
100	1100	9	243.66	3.29	60287
100	1200	3	243.69	3.28	65195

Table 5.1: Top 8 experiments with the highest improvement rate for `trigrams` in Hindi (original perplexity was 251.96).

Although the initial perplexity of 251.96 was already low, the experiments managed to decrease the perplexity even lower. Initially, I did 9 experiments for k_ngrams equal to 100. Then, I did 3 experiments each for `k_ngrams` equal to 250 and 500. The results of experiments suggest that `k_ngrams` strongly influences the number of downloaded lines and consequently, the improvement rate. Since `len_penalty` directly influences what n-grams are selected for the web search, experiments with the same `len_penalty` and `k_ngrams` values search for the exactly same n-grams. Throughout the experiments, the `ppl_threshold` with the value of 1200 had tendency to choose the most optimal documents. Furthermore, when the `ppl_threshold` was set to 1300, more paragraphs were extracted, but the improvement rate was lower, meaning the additionally accepted documents were lower quality. The most optimal `len_penalty` from $\{3, 6, 9\}$ for Hindi is 6.

Finally, the best language model with the perplexity of 234.18 was packaged in Kaldi and decoded. The best mixed model decreased the original `WER` from 30.3% to 29.3%.

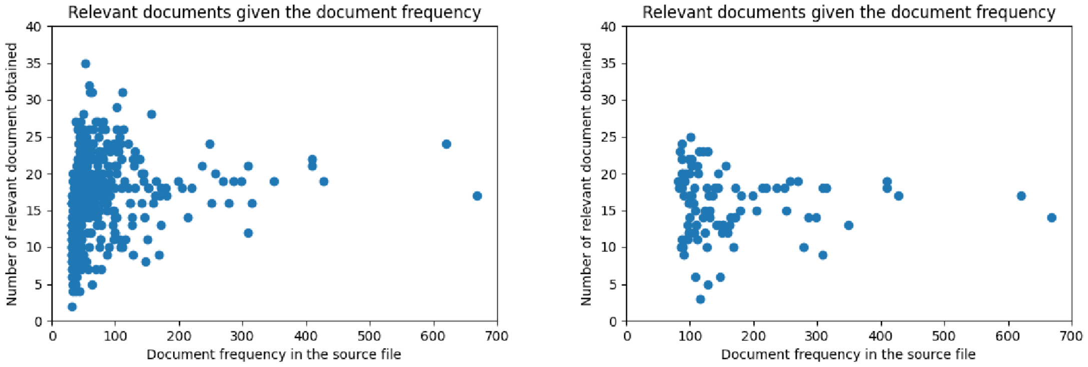
The number of obtained and used web documents from each experiment can also be compared. The following histograms created from the `statistics` files show, how many documents were useful per web search term in a given experiment. Only the best and the worst models are compared. The last set of graphs show the relationship between document frequency in the source file and number of relevant documents.



(a) Best model for Hindi.

(b) Worst model for Hindi.

Figure 5.1: Histograms of relevant downloaded documents for the best and the worst model for Hindi.



(a) Best model for Hindi.

(b) Worst model for Hindi.

Figure 5.2: Comparison of number of relevant documents given its document frequency for the best and the worst model for Hindi.

Taking into consideration that the development corpora was quite small with only around 1800 lines, the experiments were successful, since they managed to lower the perplexity by as much as 7.06% and WER by 1.0%. These experiments also confirmed the pipeline can be used for creation of corpora even for non-Phonexia system. The downloaded corpora must be mixed and processed to match the system-dependant format of the language modelling.

5.2 Experiment B - Czech

Since Phonexia is a Czech based company, there's a very high standard for the quality of their ASR system for Czech language. Furthermore, Phonexia has acquired various datasets for Czech throughout the years and therefore, it has a lot of potential to be used for experimenting with the pipeline.

Czech characteristics

Czech belongs to the Slavic language family and is spoken by over 10 million speakers. It is very closely related to Slovak, and to certain degree to Polish as well. Czech is very rich in morphology. Although Czech has a basic word order of **Subject** – **Verb** – **Object** (SVO), the rules are quite relaxed. Depending on what we need to emphasize, the word order can be flexibly adjusted.

According to the FSI's ranking, Czech belongs to the category `IV` language with around 1100 hours of study required to achieve proficiency [3]. It is ranked so high because of its variable structure and morphology.

The nouns, adjectives and verbs are inflected to modify their meanings. Omission of subject in a sentence is also very common, since it is expressed through conjugations of that verb. There are also complex rules about capitalization and the use of i/y in different words. For non-native speakers, these things can be quite complex, especially if there are no equivalent terms in their native language.

Experiments

The experiments for Czech were conducted with one of Phonexia's evaluation dataset `CS_CZ_SKODAv1`. This dataset is very specific, because it contains data used in the technical field of automobile industry. Visual inspection of the dataset concluded that all the annotations contain specific technical terminology and almost no general conversation. Therefore, this dataset is an excellent adept to test this pipeline, since it simulates the situation of obtaining data from our customer with the goal of adapting it to the target domain. Here is an example of what types of `trigrams` were selected for the web search based on the document count estimate:

Term (in Czech)	**Translation**
únik oleje z	oil leakage from
z hydraulického agregátu	from hydraulic aggregate
práce na víkend	work for weekend
01 ucpaný filtr	01 clogged filter
první dva tisíce	first two thousand
číslo stroje m	machine number m

All of these terms are quite specific, so their web search is expected to have a good yield of relevant documents. The distribution of the estimated document count is as follows:

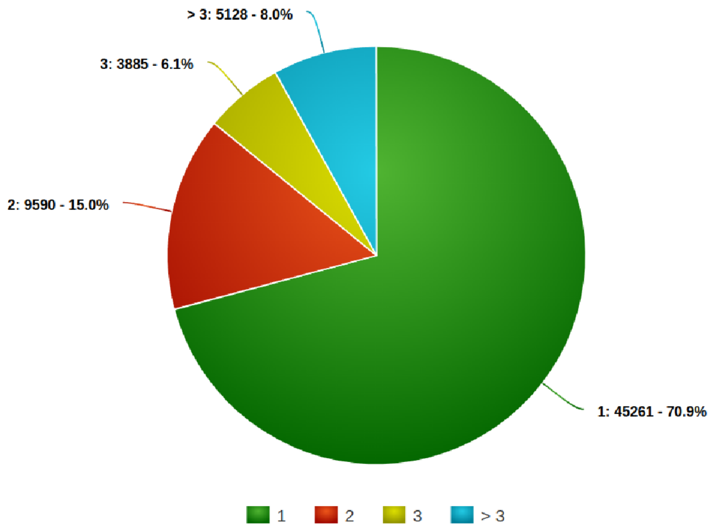


Figure 5.3: Estimated DC for the Czech data (`CS_CZ_SKODAv1`). All the estimated document counts have been rounded up to the closest integer.

The main hyperparameters that were tuned are: `ppl_threshold`, `len_penalty`, `k_ngrams` and `order_ngram`. Firstly, I carried out experiments for trigram web searches, as trigrams capture a lot of contexts (Section 3.2). Each resulting web corpora is mixed with the Phonexia Czech language model and the resulting mixed model is then evaluated with *perplexity*. In order to compare the results objectively, the Phonexia Czech language model is evaluated with `perplexity` over the same evaluation file.

Firstly, perplexity of `CS_CZ_SKODAv1` dataset was measured with the `source_model` from Phonexia. The initial value was quite high at 15032.27. Altogether, 21 experiments were carried out for `trigrams`. The following is a table of hyperparameters, obtained perplexity, improvement rate and number of lines extracted for each web corpora. The table is sorted by the improvement rate and only the top 10 experiments with their hyperparameters are shown.

k_ngrams	ppl_threshold	len_penalty	perplexity	improvement (%)	# of lines
500	1200	25	4952.77	67.05	178086
500	1200	20	4994.80	66.77	224220
500	1500	25	5011.72	66.66	127594
500	1100	20	5183.83	65.52	204799
500	1300	25	5235.75	65.17	106628
500	1000	20	5251.15	65.07	180342
500	1200	15	5252.07	65.06	281681
500	1500	20	5355.56	64.37	194706
500	1100	15	5458.11	63.69	269138
500	1300	20	5518.33	63.29	131659

Table 5.2: Top 10 experiments with the highest improvement rate for `trigrams` in Czech (original perplexity was 15032.27).

The experiments were carried out for all combinations of the following hyperparameters:

$$\begin{aligned} \text{k\_ngrams} &\in \{500, 250\} \\ \text{ppl\_threshold} &\in \{1000, 1100, 1200, 1300, 1500\} \\ \text{len\_penalty} &\in \{25, 20, 15\} \end{aligned}$$

These experiments once again confirmed the value of `k_ngrams` directly influences the number of lines extracted and transitively, also the improvement rate. The experiments for `k_ngrams` = 250 did not even make it to the top 10 results. The usual `ppl_threshold` with the highest improvement rate was 1200 as in the Hindi experiments. Experiments were more favourable for higher values of `len_penalty`, and its lowest possible value of 15 consistently ranked in the lowest range of improvement rates for 500 `k_ngrams`.

After the trigrams experiments, I wanted to test whether comparable results could be achieved with lower order n-grams. I carried out 3 experiments for `bigrams` and 1 experiment for `unigram`. For the following experiments, `k_ngrams` was always set to 500 for all of them.

ppl_threshold	len_penalty	perplexity	improvement (%)	# of lines
1400	15	3918.56	73.93	394328
1200	15	4144.76	72.43	340744
1200	10	4456.50	70.35	371654

Table 5.3: Experiments for `bigrams` in Czech (original perplexity was 15032.27).

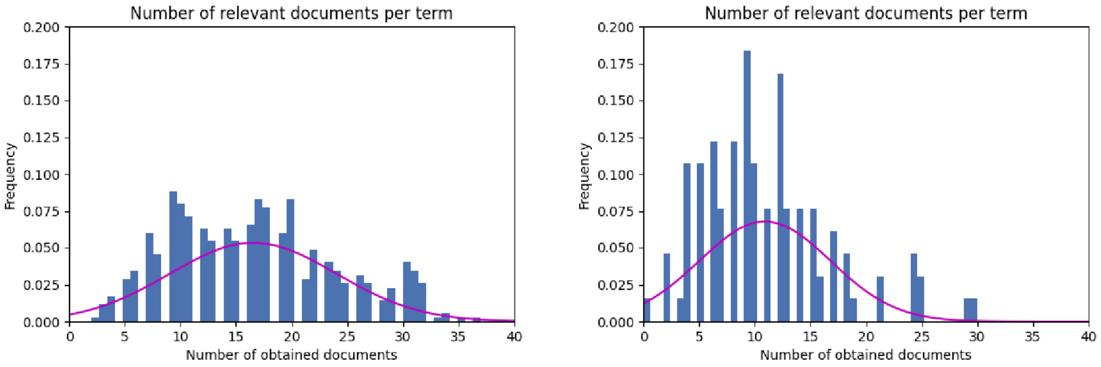
Although there's a high improvement rate, the number of extracted lines is also quite high. The best corpora for `bigrams` has more than twice the number of lines than the best corpora for `trigrams`. Since bigrams are less specific, it is reasonable that more data is obtained from these experiments. Furthermore, these experiments reached the highest improvement rate of almost 74%.

Lastly, I did one `unigram` experiment with the most optimal hyperparameters so far and expected the improvement rate to be lower but still comparable. Because `unigrams` do not capture any surrounding context, the resulting web corpora is then quite ambiguous as well.

ppl_threshold	len_penalty	perplexity	improvement (%)	# of lines
1200	8	4866.02	68.07	649237

Table 5.4: Experiment for `unigram` in Czech (original perplexity was 15032.27).

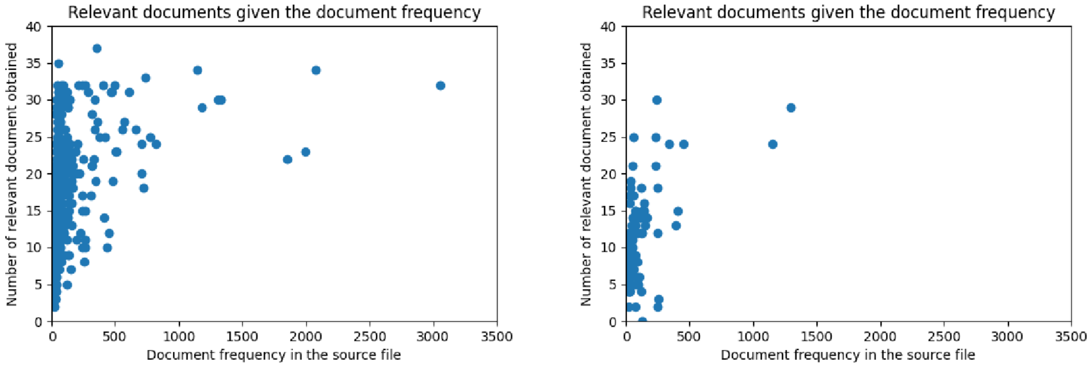
As expected, the improvement rate was lower compared to the other experiments. Still, it managed to obtain a nice improvement rate of around 68%, which is very good for a `unigram` model. Although both `bigram` and `unigram` experiments achieved favourable improvement rates, using `trigrams` as default is still highly advised since it captures the most context. The experiments showed `trigrams` and `bigrams` provide a nice balance between the improvement rate and the amount of relevant data downloaded.



(a) Best model for Czech (best bigram).

(b) Worst model for Czech (worst trigram).

Figure 5.4: Histograms of relevant downloaded documents for the best and the worst model for Czech.



(a) Best model for Czech (best bigram).

(b) Worst model for Czech (worst trigram).

Figure 5.5: Comparison of number of relevant downloaded documents given its document frequency for the best and the worst model for Czech.

After all the experiments were conducted, I chose the best model for `trigram`, `bigram` and `unigram` and packaged each of them into a single ASR system. The results decreased the original WER of 55.7% as follows:

category	**WER (%)**	**improvement (%)**
trigram	48.5	7.2
bigram	47.0	8.7
unigram	48.7	7.0

Table 5.5: Word Error Rate (WER) for the best models in each category. Original WER was 55.7%

These experiments proved to be very succesful. The perplexity was decreased in all of the experiments, so the pipeline positively contributed to the adaptation process.

5.3 Experiment C - Mandarin

Last set of experiments was conducted with Mandarin. Mandarin is considered to be a non-standard language, mainly due to its complicated writing system and speaking. I chose Mandarin to test, whether the pipeline can be used with non-standard languages as well.

Mandarin characteristics

Mandarin is based on the Beijing dialect and belongs to the Sinitic (Chinese) language group. Contrary to popular belief, Chinese and Mandarin are not completely equivalent terms. Chinese refers to a group of languages spoken by the ethnic Han people. Mandarin is just its Beijing dialect, which has been selected to be the official language of China in the 20th century. The reason for doing so was to establish a common mean of communication between speakers of different dialects of China, since the dialects are quite different from each other. Therefore, many people in China are bilingual (Mandarin, local dialect) or even trilingual (Mandarin, two local dialects).

All Chinese languages are tonal, meaning pitch of a voice determines meaning of a given linguistic unit. Mandarin uses syllables as a basic unit and each syllable has its set tonal variants. Each syllable consists of either consonant followed by a vowel, or a standalone vowel sound. There are 4 full tones and 1 neutral tone in Mandarin, traditionally numbered from 1 to 5. This fact needs to be clearly reflected in the *lexicon* file for Chinese ASR.

Mandarin uses simplified Chinese characters (Hanzi) in Mainland China and traditional characters in Taiwan. The simplified Chinese characters were created in order to boost the literacy rate. The literacy steadily improved after the introduction of Chinese characters with fewer strokes. Every Chinese character is logically composed of smaller units named radicals. Pronunciation of a composed character is generally based on one of its radical's pronunciations, giving the characters some further logic. Although there are truly many characters, only a handful of them is actually frequently used. Some reports suggest that there are as many as 50000 characters, but only 20000 of them are used. Average Chinese person usually actively knows around 3000 characters, but even knowing the basic 1000 covers a lot of everyday vocabulary.

Words with the most basic meaning, such as eat, go, car, cat etc. are usually 1-character words. Small percentage of words consists of 3 or 4 characters. Most of the other words are 2-character words. More complicated terms are built from the simpler words; multiple words can be combined to create a new term. When it comes to grammar, Mandarin is very simple – grammatical meaning is generally expressed through short (1 or 2 character) particles. The basic word order is same as in English: **Subject** – **Verb** – **Object** (SVO).

The FSI categorizes Mandarin as a class `V` language with 2200 hours of study required to achieve proficiency. The category `V` groups together the most difficult languages for native English speakers [3]. Besides Mandarin, this category also includes Cantonese, Japanese, Korean and Arabic, all of which are languages with vastly different writing styles and specific pronunciation.

Experiments

The experiments for Mandarin were quite different from Czech. I used a Phonexia dataset named `MOMvzh_01`. It contains around 8 hours of spontaneous telephone speech in Mandarin. This dataset was pre-processed quite recently. At that time, there was a discussion on how to pre-process Mandarin datasets, since Mandarin is actually written without any spacing. Therefore, no clear boundary between words can be established.

It was concluded that each Chinese character is to be considered as a separate unigram. Therefore, the Mandarin `dictionary` file is actually a list of Chinese characters with their pronunciation, each one on a new line. Current version of Phonexia's Mandarin dictionary contains just over 20000 characters, so occurrence of out-of-vocabulary character during corpora creation is not very likely. The `stt.phxstm` for `MOMvzh_01` is already pre-processed so that each Chinese character is separated from the others with space.

However, this poses a great challenge for the pipeline. There are 2 things to be considered before the experiments:

- If we use the standard `trigram` approach for `MOMvzh_01`, the selected trigrams are not actually 3 full words as in the other experiments, but average 1.5 – 2.5 words per trigram depending on the characters used. Only if all 3 characters are standalone words it would be equivalent to 3 words in a `trigram`.
- The document count estimate (Equation 4.1) in the class `NgramsProvider` uses *precision* to penalize `n-grams` shorter than `len_penalty`. However, Mandarin renders the *precision* metric useless, since all `n-grams` (containing only Chinese characters) have the same length. Therefore, `n-grams` for web search are ranked and chosen solely on their *document frequency*.

Additionally, I did 2 manual adjustments to the pipeline for Mandarin before any experiments were conducted. Firstly, I adjusted the `TextCleaner` function for cleaning web documents to follow the spacing rules for Mandarin as explained above. Secondly, I added filtering to `TextCleaner`, so only paragraphs with at least 70% of Chinese characters are considered. It automatically excludes spacing from its calculations, so only actual characters are counted and compared. The threshold is set to 70% and not higher, because we still want to keep paragraphs with a few words in Latin. Such words are usually names of companies, foreign names or simply terms that some Chinese tend to write in Latin.

The initial perplexity evaluation of the `MOMvzh_01` by the `source_model` from Phonexia resulted in a very low score of 73.77. Therefore, it was expected that addition of web data is more likely to increase the perplexity, rather than decrease it.

Firstly, I carried out 16 experiments for Mandarin. Since `len_penalty` was the same for all the experiments of a given order of ngram, I tried to experiment with `order_ngram`, `k_ngrams` and `ppl_threshold`. Due to the problems explained above, Mandarin could benefit from using higher order n-grams. I experimented with combinations of the following hyperparameters:

$$\begin{aligned} \text{order\_ngram} &\in \{3, 4, 5\} \\ \text{k\_ngrams} &\in \{500, 1000\} \\ \text{ppl\_threshold} &\in \{1100, 1200, 1300, 1400\} \end{aligned}$$

All the conducted experiments led to the increase of perplexity. Furthermore, throughout the experiments, many selected web links did not respond to the `HTTP GET` requests. Some servers responded with `RST packet`, which immediately closed the connection, and an exception of `connection reset by peer` occurred. Some servers responded only if the maximum number of retries was exceeded. Other servers did not respond at all, leaving the script hanging and waiting for the server's response. Because each link exploration is run as a standalone process guarded with a set `timeout`, the script is not left hanging forever and after the timer runs out, the process is terminated.

Although the same problems arose for the other language experiments too, it was still very rare. However, for Mandarin, these exceptions occurred definitely more frequently than in the other experiments. It is possible that some corporas have diminished potential because of these difficulties.

The best 3 achieved results are shown here:

order_ngram	k_ngrams	ppl_threshold	len_penalty	perplexity	# of lines
3	1000	1100	5	166.24	22517
3	1000	1300	5	167.58	27173
3	1000	1200	5	170.65	27554

Table 5.6: Experiments for various `n-grams` in Mandarin (original perplexity was 73.77).

As expected, the perplexity was not increased. Still, `ppl_threshold` at around 1200 consistently ranked higher, while higher values of 1400 ranked lower. The higher the value of `k_ngrams`, the higher the improvement rate, similar to the Czech experiments. Surprisingly, Mandarin did not benefit from using higher order `n-grams` as originally anticipated. Since none of these experiments managed to decrease the perplexity, I decided to investigate a bit further.

The first shortcoming is definitely the limited `document count` estimation. Out of the total distinct 230224 trigrams that were found, only around 3.7% had the document count estimation higher than 3. If we compare it with the Figure 5.3 for Czech document count estimate, Mandarin obviously suffers due to the limited `n-grams` document count estimate.

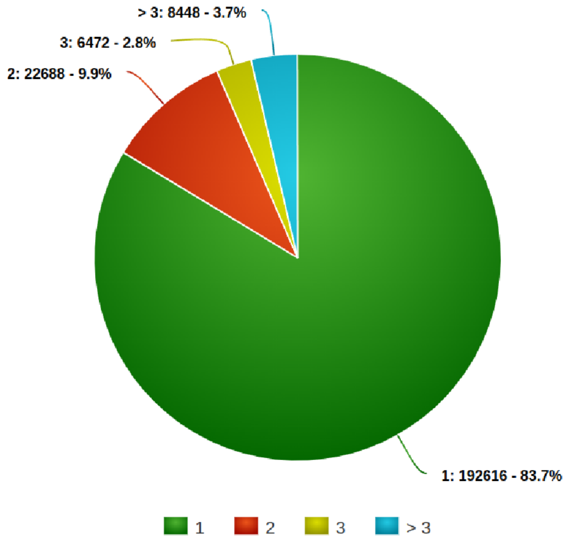


Figure 5.6: Estimated DC for the Mandarin dataset (`MOMvzh_01`).

Firstly, I tried to improve the results with advanced filtering. I used paragraph-based filtering without any `window_len` specified and carried out 3 experiments for different values of `filter_threshold`. Only the `median` filter type was used. The results improved a little bit:

filter_threshold	perplexity	# of lines
-4.0	157.20	9727
-4.5	157.93	8939
-5.0	158.45	9478

Table 5.7: Experiments for `trigrams` in Mandarin with advanced paragraph-based filtering (original perplexity was 73.77).

The advanced filtering managed to decrease the perplexity more than the previous experiments. Still, the perplexity was higher than the original one, so I decided to try using a more language-specific approach, so I manually went through the content of this dataset.

Since this dataset is not specialized, but a general conversational speech, I took a look at what `trigrams` are actually selected for the web search with the limited document count estimation. Here is a sample of terms that were selected:

´ **Term (in Mandarin)**	**Translation**
告诉我	tell me
就是那	eventually is that
跟你说	speak with you
是今天	is today
没有车	not have a car
完了这	finished that

As anticipated, the terms selected for the web search are quite ambiguous, so the downloaded corpora is transitively ambiguous as well. Since selecting the `trigrams` according to their estimated document count did not contribute much, I decided to try the inverse approach – select rarer n-grams for the web search. By selecting the rarer n-grams, we avoid searching for the ambiguous terms and repeated grammatical structures.

In order to select the rare `trigrams`, I manually adjusted the selection process inside the `NgramsProvider` class for the following experiments. Firstly, I filtered out trigrams that occurred more than once. Secondly, I filtered out trigrams that included multiple occurrences of the same Chinese character. Then, the first `k_ngrams` were picked:

Term (in Mandarin)	**Translation**
是最贵	is the most expensive
电子版	electronic version
忘了做	forgot to do that
发韩国	send to Korea
湿度高	high humidity
忙天的	in a busy day

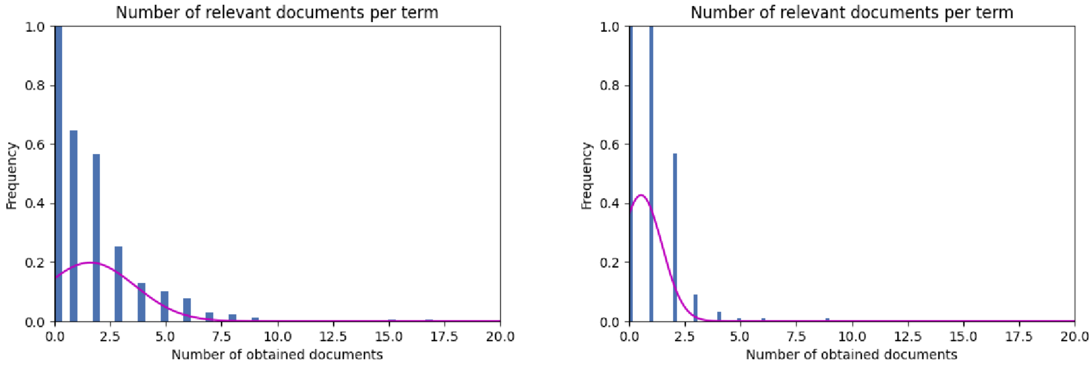
I manually went through the terms that are to be searched with this approach. Although there were still some general terms, this subset of `trigrams` is certainly more specific than the original one. Even the newly selected general terms were a bit more useful – the original set of `trigrams` included a lot of pure grammatical structures without any actual meaning. This subset's general terms were mostly numbers and numeral classifiers, which is actually more useful, since the basic grammatical structure is already captured by the Phonexia `source_model`.

I conducted 5 more experiments with the manually selected `trigrams`. All the experiments used the same `ppl_threshold` of 1200, `len_penalty` of 5 and `k_ngrams` equal to 500. The first experiment was basic, without any advanced filtering. The rest of the experiments utilized the advanced filtering.

I expected to obtain comparable results to the previous experiments. However, the opposite happened and *perplexity* increased substantially. Therefore, this approach was unsuccessful and it was concluded the original approach was more suitable despite its limitations.

The best model is therefore the one with perplexity of 157.20 and the worst is the one with perplexity of 230.57. If we compare the histograms of number of relevant documents for Mandarin between the best and the worst model, the difference is immense.

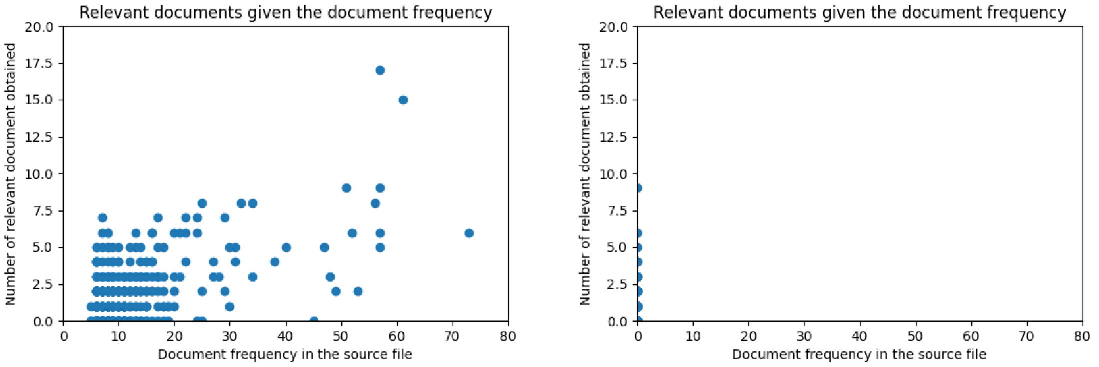
On the top of that, since Mandarin suffered from the web search difficulties, the histograms are very sparse in comparison to the other experiments. These results also strongly suggest there is some diminished potential from the web search for Mandarin.



(a) Best model for Mandarin (natural selection).

(b) Worst model for Mandarin (manual selection).

Figure 5.7: Histograms of relevant downloaded documents for the best and the worst model for Mandarin.



(a) Best model for Mandarin (trigram with advanced filtering).

(b) Worst model for Mandarin (trigram).

Figure 5.8: Comparison of number of relevant downloaded documents given its document frequency for the best and the worst model for Mandarin.

5.4 Results summary

Altogether, 3 sets of experiments were conducted with vastly different languages. Therefore, I believe the pipeline's possibilities were fully explored and the results were favourable.

Firstly, I experimented with corpora creation for Hindi in section 5.1. Although the initial perplexity was already low at around 252, the pipeline managed to decrease it by as much as 7%. Finally, when the best model was decoded, the `WER` decreased by 1%. This set of experiments also confirmed that the implemented pipeline can used for corpora creation even for `non-Phonexia` systems.

The second set of experiments was with Czech in the section 5.2. The chosen dataset was technical and the initial perplexity was very high at over 15000. In this case, the pipeline contributed very much and managed to decrease the perplexity by as much as 73.9%. This set of experiments can be considered to be the most successful. All of the experiments managed to decrease the perplexity. The value of WER also decrease by as much as 8.7%.

Last set of experiments was conducted with Mandarin in the section 5.3. The chosen dataset was from `Phonexia` and consisted of natural spontaneous speech. The initial perplexity was already very low at around 73, so a substantial decrease was not expected. Furthermore, these experiments faced some additional difficulties and limitations. Although no model achieved better perplexity, these results still provided an insight on how to handle a non-standard language like Mandarin.

In conclusion, the experiments confirmed the pipeline can be utilized for automatic language model adaptation. However, even this pipeline has its limitations. If the dataset is very general and the initial perplexity already low, there is not much to improve with the simple web searches. I still do believe that this pipeline has the potential to alleviate some number of resources for `Phonexia` during the adaptation process.

Chapter 6

Conclusion

The goal of this thesis was to explore and implement a pipeline for automatic language model adaptation for Phonexia ASR system. The pipeline is given a domain-specific input file, which is analysed and the optimal terms for web search are selected. During the web search, each document is thoroughly filtered and evaluated to determine, whether it should be included in the final web corpora or not. Because filtering of the data is so important, some advanced means of filtering were implemented as well.

The experiments were conducted on 3 vastly different languages - Hindi, Czech and Mandarin. Furthermore, the experiments for Hindi used only the corpora creation part of the pipeline, but the actual model mixing and evaluation was done with Kaldi. This proved the pipeline can be utilized for a web corpora creation even for a non-Phonexia ASR system.

The results were very promising. Hindi used general dataset with very low initial perplexity. The pipeline managed to decrease its perplexity by around 7% and WER by 1%. The second set of experiments for Czech was carried out on a Phonexia system. The chosen dataset was very specialized and technical, which perfectly simulated the situation of obtaining the data for adaptation to a specific domain. The pipeline managed to decrease the perplexity of the language model by as much as almost 74%. When it was packaged to the final ASR system and evaluated, WER decreased by 8.7%. The last set of experiments was conducted with the non-standard Mandarin Chinese. The dataset for Mandarin was very general and its initial perplexity was already very low compared to the other experiments. Therefore, this set of experiments did not achieve better perplexity, but it still provided an insight on how to handle the non-standard languages in the pipeline.

It can be concluded the implemented pipeline can be used as language-independent, since the best hyperparameters between sets of experiments were consistently similar. I believe the goals of the thesis were fulfilled and hopefully, the implemented pipeline will be used by my colleagues in Phonexia to ease the language model adaptation process.

The future research building on this thesis can focus on many different aspects. For example, the pipeline can be easily updated to work with subscription-based API or use a different language identification model. For the non-standard languages like Mandarin, a different, more specific dataset could be used for experiments. Another possible feature is the inclusion of movie subtitles in the target language. Since subtitles contain a lot of conversational data, they could prove to be useful for natural expansion of the language model for general spontaneous speech. Finally, I believe it would be very interesting to conduct the same experiments with Mandarin, but with VPN in the target language country.

Bibliography

[1] Available at: http://wwwhomes.uni-bielefeld.de/gibbon/Handbooks/gibbon_handbook_1997/node303.html.

[2] *Hindi language - structure, writing alphabet - mustgo.* Available at: https://www.mustgo.com/worldlanguages/hindi/.

[3] *Language difficulty ranking.* Dec 2019. Available at: https://effectivelanguagelearning.com/language-guide/language-difficulty/.

[4] *Languages with official status in India.* Wikimedia Foundation, May 2022. Available at: https://en.wikipedia.org/wiki/Languages_with_official_status_in_India#Eighth_Schedule_to_the_Constitution.

[5] Bäckström, T. Linear Predictive Modelling of Speech ‑Constraints and Line Spectrum Pair Decomposition. *951-22-6946-5.* january 2004.

[6] Damerau, F. J. and Indurkhya, N. *Handbook of natural language processing.* Taylor & Francis, 2010.

[7] Fabien, M. *Introduction to automatic speech recognition (ASR).* May 2020. Available at: https://maelfabien.github.io/machinelearning/speech_reco/#.

[8] Fourier, J.-B.-J. *The analytic theory of heat.* Stechert, 1888.

[9] Hopcroft, J. E., Motwani, R. and Ullman, J. D. *Introduction to automata theory, languages, and computation.* Pearson Education, 2020.

[10] Huang, X., Acero, A. and Hon, H.-W. *Spoken language processing: A guide to theory, algorithm, and system development.* Prentice Education Taiwan, 2005.

[11] Hui, J. *Speech Recognition - Phonetics.* Medium, Dec 2019. Available at: https://jonathan-hui.medium.com/speech-recognition-phonetics-d761ea1710c0.

[12] Jones, N. and Goehring, A. Aeroacoustic Façade Noise Validation: A Comparison of CFD and Wind Tunnel Tests. In:. September 2019. DOI: 10.26868/25222708.2019.210880.

[13] Joulin, A., Grave, E., Bojanowski, P. and Mikolov, T. Bag of Tricks for Efficient Text Classification. *ArXiv preprint arXiv:1607.01759.* 2016.

[14] Jurafsky, D. *Lecture 8: Asr: Noisy channel, hmms, evaluation.* Available at: https://web.stanford.edu/class/cs224s/lectures/224s.20.lec8.pdf.

[15] Jurafsky, D. and Martin, J. H. *Speech and language processing (3rd Ed. draft).* Available at: http://www.web.stanford.edu/~jurafsky/slp3/.

[16] Maklin, C. *Fast fourier transform.* Towards Data Science, Dec 2019. Available at: https://towardsdatascience.com/fast-fourier-transform-937926e591cb.

[17] Mccowan, I., Moore, D., Dines, J., Gatica Perez, D., Flynn, M. et al. On the Use of Information Retrieval Measures for Speech Recognition Evaluation. january 2004.

[18] Motlíček, P. *Feature extraction in speech coding and recognition.* Available at: https://www.fit.vutbr.cz/~motlicek/publi/2002/rp.pdf.

[19] Povey, D., Ghoshal, A., Boulianne, G., Burget, L., Glembek, O. et al. The Kaldi Speech Recognition Toolkit. In: *IEEE 2011 Workshop on Automatic Speech Recognition and Understanding.* IEEE Signal Processing Society, December 2011. IEEE Catalog No.: CFP11SRW-USB.

[20] S, K. and E, C. A review on Automatic Speech Recognition Architecture and approaches. *International Journal of Signal Processing, Image Processing and Pattern Recognition.* 2016, vol. 9, no. 4, p. 393-404. DOI: 10.14257/ijsip.2016.9.4.34.

[21] Steinbach, S. *Lexical distance among languages of Europe 2015.* Apr 2019. Available at: `https://alternativetransport.wordpress.com/2015/05/05/34/`.

[22] Stolcke, A. *SRILM – an extensible language modeling toolkit.* ISCA Archive. Available at: `https://www.isca-speech.org/archive/pdfs/icslp_2002/stolcke02_icslp.pdf`.

[23] Zhang, L., Karakos, D., Hartmann, W., Hsiao, R., Schwartz, R. et al. Enhancing low resource keyword spotting with automatically retrieved web documents. In:. September 2015, p. 839–843. DOI: 10.21437/Interspeech.2015-262.