BRNO UNIVERSITY OF TECHNOLOGY
VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ

FACULTY OF INFORMATION TECHNOLOGY
FAKULTA INFORMAČNÍCH TECHNOLOGIÍ

DEPARTMENT OF INFORMATION SYSTEMS
ÚSTAV INFORMAČNÍCH SYSTÉMŮ

AGGREGATION AND ANALYSIS OF SOCIAL NETWORK CONTENTS

AGREGACE A ANALÝZA OBSAHU ZE SOCIÁLNÍCH SÍTÍ

BACHELOR'S THESIS
BAKALÁŘSKÁ PRÁCE

AUTHOR **MATĚJ HORÁK**
AUTOR PRÁCE

SUPERVISOR **Ing. RADEK BURGET, Ph.D.**
VEDOUCÍ PRÁCE

BRNO 2019

Brno University of Technology
Faculty of Information Technology

Department of Information Systems (DIFS) Academic year 2018/2019

Bachelor's Thesis Specification

21491

Student: **Horák Matěj**
Programme: Information Technology
Title: **Aggregation and Analysis of Social Network Contents**
Category: Web

Assignment:

1. Study the principles of the most popular social networks and their application interfaces.
2. Study current methods of the content classification based on topic or other criteria.
3. Design the architecture of a system for the aggregation of the contents of selected sources on social networks and its filtering by relevance to a given topic.
4. Upon agreement with the tutor, choose the appropriate technology and implement the designed system.
5. Perform the testing of the implemented system on real-world data.
6. Summarize the results.

Recommended literature:

- Fiala, M.: Propojení sociální sítě Twitter s televizním vysíláním. Brno, 2018. Diplomová práce. Vysoké učení technické v Brně, Fakulta informačních technologií.
- Russell, M. A.: Mining the Social Web, 2nd Edition, O'Reilly Media, Inc., 2013

Requirements for the first semester:

- Items 1 to 3

Detailed formal requirements can be found at http://www.fit.vutbr.cz/info/szz/

Supervisor: **Burget Radek, Ing., Ph.D.**
Head of Department: Kolář Dušan, doc. Dr. Ing.
Beginning of work: November 1, 2018
Submission deadline: May 15, 2019
Approval date: October 30, 2018

Abstract

This thesis is focused on getting selected parts of social media content and their analysis. The thesis is aiming for a platform which is connecting individual social networks, aggregating their content by defined topics and which is opened for next improvements and extensions. A solution is a multi-container application that uses multi-label classification and support vector machines. The implemented system solves not shown content, filtering, and small statistics. Key parts are covered by tests and the system is opened for other analysis and advanced statistics.

Abstrakt

Tato práce se zabývá ziskem zvolené části obsahu sociálních sítí a jeho následnou analýzou. Cílem práce je platforma propojující jednotlivé sociální sítě, která dokáže agregovat obsah těchto sítí podle definovaných témat a zároveň je otevřená dalším rozšířením. Tento cíl byl vyřešen pomocí kontejnerové aplikace, štítkové klasifikace a metody podpůrných vektorů. Implementovaný systém řeší algoritmem nezobrazovaný obsah, filtrování a menší statistiky. Klíčové části systému jsou pokryté testy a systém je otevřený dalším analýzám a pokročilým statistikám.

Keywords

Social Networks, Text Topic Analysis, Support Vector Machines, REST, Docker

Klíčová slova

Sociální síťe, analýza tématu textu, metoda podpůrných vektorů, REST, Docker

Reference

HORÁK, Matěj. *Aggregation and Analysis of Social Network Contents.* Brno, 2019. Bachelor's thesis. Brno University of Technology, Faculty of Information Technology. Supervisor Ing. Radek Burget, Ph.D.

Rozšířený abstrakt

Sociální sítě jsou populární webové stránky, které používají lidé, firmy a další uskupení na celém světě. Mezi nejznámnější sociální sítě patří Facebook, Twitter atd. Použití sociálních sítí jako zdroje informací ale nemusí být optimální, protože v mnoha případech sociální sítě vydělávájí na personalizovaných reklamách a uživateli nebývá zobrazen veškerý obsah, nehledě na nutnost sledovat každou síť odděleně.

Tato práce se zabývá ziskem zvolené části obsahu sociálních sítí a následnou jeho analýzou. Konkrétně si dává za cíl vytvořit platformu, která dokáže propojit jednotlivé sítě, agregovat obsah podle definovaných témat a poskytnout statistiky obsahu. Tato platforma by měla být otevřená dalším rozšířením.

Sociální síť je tvořena uživateli, kteří si navzájem mezi sebou sdílejí příspěvky. Obvykle také poskytují REST aplikační rozhraní pro aplikace třetích stran, které ale může mít nějaké omezení v přístupu k příspěvkům, informacím o uživateli atd. Většinou je nutné zaregistrovat aplikaci, aby bylo možné toto aplikační rozhraní používat. Jednotlivé způsoby použití aplikačního rozhraní se liší podle platformy, nicméně systém popsaný výše většinou využíval tyto dva způsoby:

- Získání příspěvků od určitého účtu nebo z určité skupiny
- Získání příspěvků obsahující určité slovo nebo hashtag

Pro analýzu tématu příspěvku byla zvolena klasifikační metoda podpůrných vektorů. Tato metoda byla zvolena na základě porovnání s pravědopobnostním Bayesovským klasifikátorem. Vzhledem k tomu, že jeden příspěvěk může mít více témat, pro každé téma příspěvku je vytvořen vlastní klasifikátor. Při určování tématu textu je pak text analyzován všemi klasifikátory. Metoda podpůrných vektorů je založena na hledání rozdělující hranice ve vektorovém prostoru. Aby bylo možné tuto hranici nalézt, je nutné převést texty do obdobné reprezentace. Tato reprezentace je dosažena pomocí tokenizace, odstranění stop slov, lemmatizace a výpočtu relevance pomocí TF-IDF.

Jedním z hlavních požadavků pro navržený a implementovaný systém byla jeho rozšiřitelnost, např. aby bylo možné jednoduchým způsobem zaintegrovat službu pro analýzu obrázků. Z tohoto důvodu byl systém implementován jako kontejnerová aplikace, kde každá část systému je jeden Docker kentejner. Pro příspěvky byla zvolena NoSQL databáze (Mongo), aby byl možný rychlý přístup k datům v rámci strojového učení. Naopak u uživatelských dat (témata, zdroje atd.) byla zvolena SQL databáze (PostgreSQL). Velký důraz byl také kladen na aplikační rozhraní systému, konkrétně byl užit API-First Design. Samotné aplikační rozhraní je implementováno jako Spring aplikace pomocí programovacího jazyka Kotlin. Toto rozhraní pak volá interní služby pro zisk dat ze sociálních sítí a analýzu tématu. Tyto služby jsou implementovány v programovacím jazyce Python a využívají řadu knihoven. Grafickým uživatelským rozhraním systému je jednostránková aplikace, která je vykreslována na straně klienta. Tato aplikace byla vytvořena pomocí technologie React a sady komponent z kolekce Material UI. Systém ještě obsahuje kontejner pro pravidelné úlohy (zisk dat, návrh témat) a pro delegování požadavků na server pomocí technologie NGINX.

Implementace modelu pro analýzu tématu příspěvku má dostatečné výsledky. Systém byl také implementován s myšlenkou otevření zdrojů. Součástí vývoje bylo tedy i testování, aplikační rozhraní je z výrazné části pokryto testy. Funkce pro transformaci dat u grafického rozhraní a interních služeb jsou také pokryté jednotkovými testy. V rámci repozitáře

byla nakonfigurována i platforma pro nepřetržité dodávky. po každé změně ve zdáleném repozitáři dojde ke spuštění testů a pokud je zdrojový kód validní, dojde k nasazení systému.

Aggregation and Analysis of Social Network Contents

Declaration

I declare that I have prepared this Bachelor's thesis independently, under the supervision of Ing. Radek Burget, Ph.D. I listed all of the literary sources and publications that I have used.

.....
Matěj Horák
May 14, 2019

Acknowledgements

I would like express my sincere thanks to my supervisor, Ing. Radek Burget, Ph.D., for responsible guidance and providing valuable feedback.

Contents

Chapter 1

Introduction

Social networks are popular websites that are used by people, companies and other entities for content sharing all over the world. However, a social network may not be designed for using the social network as an information channel because in most cases, a social network profits from personalized advertisements and a user does not see all content from subscribed and followed sources, regardless of the need to use each social network separately.

The idea of creating an application for social networks content aggregation and analysis was based on the existence of public posts in the city context. News media share articles about the city and citizens share posts with information, question or with an attachment like a photo. Some aggregation and analysis tools might be useful for people who are interested in the city context.

The main goal of this bachelor's thesis is to design and implement a web application for content aggregation and analysis from several popular social networks. The application should be able to get content automatically from social media and networks, provide post topic analysis and display basic statistics. Assuming implementation will be open sourced, the application should be ready for easy configuration and extension integration.

First, the thesis focuses on principles of selected social networks and their application interface. Then, the thesis describes two classification algorithms for text topic analysis and text pre-processing for getting better results. The next chapters discuss application architecture and functional requirements, describe implementation and summarize results from implementation testing.

Chapter 2

Social Media and Networks

Social media and networks are internet websites used for communication and for creating and maintaining relationships between people, companies and other subjects. In January 2018, 69% of U.S. adults use at least one social media site, 68% use Facebook, and 53% of respondents use Facebook daily. Other platforms are Twitter, Instagram, Reddit, LinkedIn, etc. [7]

2.1 Principles

Users follow other users and subjects, share posts to other people and send messages in a chat.

A post can be a text, an image, a video, a link to an external source or some combination. Users can comment, share the post or give a reaction, e.g. “like”. A post can include a special tag called hashtag e.g. ”#universitylife”, which defines a topic associated with the post. Users can display posts containing a specified hashtag.

Social media and networks display content as a list called “feed”. A content rarely contains all posts from followed sources, and post order is not always by time. Feed generating depends on a user’s behavior, trends, paid advertisement, etc.

The next subsections describe platform specifics.

2.1.1 Twitter

Twitter does not have differences between business and user accounts. Twitter posts can have a maximal length of 280 characters, and by default each post is public. It is possible to enable the protected mode and if a user enables this mode, only those that user approve will see a user’s posts. Post can contain:

- Text
- Images
- Poll
- Location

User’s feed is personalized. It is a great platform for real-time posts. For example, users are posting tweets with assigned hashtags from some live events like concerts, elections, public disasters, etc.

2.1.2 Facebook

This platform has two account types: user account and public page. User accounts can send and accept friend requests from each other. Only user accounts can follow public pages but public pages (respectively their administrators and editors) are able to comment, share or give a reaction to a public post. Users can also enable the possibility of following without accepted friendship.

Each post from page is public. User has three options: share post with their friends, publicly or as private post which is visible only for that user. In addition, it is possible to create private and public groups where posts are shared only with group members.

Facebook supports hashtags but they are not so much used in comparison with Twitter. User's feed is personalized and Facebook post can be composed from many types of content:

- Text
- Images
- Feels or Activity
- Event
- Survey
- Location
- Offer
- Live Stream

2.1.3 Reddit

Reddit is a platform that has similar functions as a forum. Users can share posts on their profile or in dedicated forums for a particular topic called "subreddit". User feed is composed from posts from "subreddits" where the user is a member. By default, post order in a feed is personalized but user can sort posts by time, score, etc.

Post can be a text, image, video or some URL. Users can add "upvote" or "downvote" to some post and each post has a score. This score is used across "subreddits" and Reddit has a special feed for viral posts. Users can also comment on posts and comments are structured as threads under the post or comment.

2.1.4 News Media

News media publish and display articles on their website. Article has a title, summary and text body that can have subtitles. Some news media have article tagging and some news media have sections for articles. News media can focus on a particular topic or can cover all social topics.

2.2 Application Interfaces

Social media and networks usually have an application interface that can be used for creating plugins, chatbot, subscription services, etc. The next subsections describe an application interface for each platform, its requirements and methods for obtaining data from selected sources because these methods are used in an implementation described in 5.

2.2.1 Twitter

This section draws from [10] and [3]. Twitter provides a free limited REST API. There are limitations like a limited number of requests[1] and a limited number of days for a search in a history. It is possible to get full access by switching to one of two paid plans.

API is running at this URL: https://api.twitter.com/1.1/

Requirements

It is necessary to enable a developer account and register an app for getting access to the API. This process requires providing information about the intents of API use. Using API also has to follow Twitter Developer Policy[2]. Each application has its own keys and tokens. These keys and tokens are used for authentication and authorization.

Tweet Object

Endpoints described below returns a response containing tweets with structure described in [3] and this section highlights some fields and its format. Twitter API provides all tweet information. It is unnecessary to request additional information like tweet author data because the tweet object contains structure "user" with fields "screen_name", "followers_count", etc.

The tweet object contains "id" and "id_str". Twitter recommends using "id_str" because some programming languages do not support numbers with a size bigger than 53 bits. Tweet time-stamp "created_at" is in format "EEE MMM d HH:mm:ss Z yyyy" and databases may not support this format for sorting and query operations in databases. The tweet object also contains the field for the tweet language "lang" and the field for tweet entities (hashtags, account mentions, urls, etc.) "entities". The tweet language is a machine detected.

Search Tweets Containing Particular Word or Hashtag

This endpoint searches tweets from the past seven days. The premium search endpoint described below is for a longer history.

- **Resource URL:** https://api.twitter.com/1.1/search/tweets.json
- **Auth:** OAuth 2.0 (1.0 also works but not for queries containing hashtag)
- **Parameters:**
  - **q:** word AND -filter:retweets / #hashtag AND -filter:retweets
  - **tweet_mode:** extended
  - **result_type:** recent

Get Tweets from Particular Account

This endpoint returns all tweets from a particular account.

- **Resource URL:** https://api.twitter.com/1.1/search/tweets.json

[1] https://developer.twitter.com/en/docs/basics/rate-limits
[2] https://developer.twitter.com/en/developer-terms/policy

- **Auth:** OAuth 1.0 / 2.0
- **Parameters:**
  - **screen_name:** username
  - **tweet_mode:** extended
  - **include_rts:** false

Premium Search

It is necessary to create a developer environment in the Twitter developer dashboard. The base URL is extended by environment type and name.

- **Resource Base URL:** https://api.twitter.com/1.1/tweets/search/
- **Resource Example URL:** https://api.twitter.com/1.1/tweets/search/30day/dev.json
- **Auth:** OAuth 2.0
- **Parameters:**
  - **query:** word #hashtag

2.2.2 Facebook

This section draws from [1] and from own exploration. Facebook provides Graph API but it has limitations in accessing the Facebook data. The Graph API is running on this URL: https://graph.facebook.com/v3.2/

Requirements

First, it is necessary to create an application. All requests require an access token but only reviewed applications have their own access token. There are not many resources with review experience and the chances for getting a public page access token that covers needed requests for this thesis. However, there is an option to generate user access token but user access token scope covers only that user data and data from pages and groups where the user has admin permissions.

Universal Endpoint for Group and Page Posts

This is not the only way to get posts from a page and group. The Graph API response object is defined by a special parameter "fields". There is a difference between accessible fields for a post from a page and from a group and it is possible to have two separate request types for pages and groups.

- **Resource URL:** https://graph.facebook.com/v3.2/page_or_group_id/feed
- **Auth:** None
- **Parameters:**
  - **access_token:** Your access token

- **fields:**
  created_time,shares,id,story,
  message,comments.limit(0).summary(true),
  reactions.limit(0).summary(total_count)

2.2.3 Reddit

This section draws from [2]. Reddit provides free limited API. There is a limitation 60 requests per minute per client[3]. In addition to JSON, it is also possible to get a response as XML. The response object for endpoints described below is a listing[4] containing 100 last posts and it is possible to send additional requests for getting next posts.

Requirements

For using Reddit API, it is necessary to register an application. Registering an application generates client ID and client secret key. These values are used for authentication. It is possible to use Reddit API without authentication, but there is a risk of stopping access to API by Reddit.

Authentication

Requests described below do not need permission from logged clients because response data are publicly visible. The Reddit API contains an endpoint with particular parameters for getting access token without client's permission - Application Only Authentication. It computes API rate limits for each device. This is a request for getting a Bearer token which is added as an authorization header:

- **Resource URL:** https://www.reddit.com/api/v1/access_token
- **Auth:** Basic Auth
- **Body:**
- – **grant_type:** client_credentials

The Basic Auth uses these application keys: User name is the client ID and password is the client secret key.

Reddit Post Object

Post time-stamp "created" is in unix time-stamp format. Post contains fields "id" and "subreddit_id" but it is better to use the field "permalink" for accessing the post. The field "text" is an empty string when the post is media or link only.

Get New Subreddit Posts

This endpoint is for getting new subreddit posts where the subreddit is specified inside resource URL.

[3] https://github.com/reddit-archive/reddit/wiki/API
[4] https://www.reddit.com/dev/api/oauth#listings

- **Authenticated Resource URL:**
  https://oauth.reddit.com/r/subreddit_id/new.json
- **Unauthenticated Resource URL:**
  https://www.reddit.com/r/subreddit_id/new.json
- **Auth:** Bearer Token or None
- **Parameters:**
- – **limit:** 100

Search Posts Containing Particular Word

This endpoint is for getting new posts containing a particular word where the word is specified as a parameter.

- **Authenticated Resource URL:**
  https://oauth.reddit.com/search.json
- **Unauthenticated Resource URL:**
  https://www.reddit.com/search.json
- **Auth:** Bearer Token or None
- **Parameters:**
- – **q:** word
  – **sort:** new
  – **limit:** 100

2.2.4 News Media

News media often provide RSS feed for displaying articles in third party RSS applications like Feedly. RSS (Rich Site Summary) is a an XML format for unifying contents (articles, podcasts, events, etc.) from websites. An RSS content is in chronological order. If the publisher does not provide RSS feed, it is possible to use technologies (for example python library Newspaper3k) for getting content from articles because articles have unified HTML structure (”h1” element for title, “p” element for paragraph, etc.).

There are examples of RSS feeds:

- The New York Times: http://rss.nytimes.com/services/xml/rss/nyt/HomePage.xml
- CNET News: https://www.cnet.com/rss/news/
- TED Radio Hour: https://www.npr.org/rss/podcast.php?id=510298

2.3 Privacy Policies and Data Protection Regulations

This section does not belong to the thesis topic but it is appropriate to mention that there are some restrictions and consequences in working with personal data.

Each platform has its own privacy policy (sometimes also developer policy) that needs to be followed. In addition, if some application (for example, an application from 4 and 5) is distributed to someone in some country, it is necessary to comply with the laws of that country or governmental entity like the European Union and its General Data Protection Regulation. Understanding and deriving overall requirements for the application may not be an easy task and legal advice can be useful.

There are examples of possible restrictions or consequences:

- Application cannot share processed personal data from a third party application.
- Application should have a method for deleting all data of some user. This data should be also deleted in third-party applications.
- User should have an option to export all its data which an application has.
- Application cannot display third-party content under monetization. For example, an article on a page with ads.

Chapter 3

Text Analysis and Aggregation

A text can be analyzed in several ways and according to several criteria. For example, it is possible to analyze and aggregate text by topic or other criteria and try to get the most similar texts or texts that relate to the selected text. The main goal is to represent texts in a vector space and build machine learning models for classifications or eventually for predictions. Afterwards, there is an option to perform statistical analysis.

The text in a vector space representation goes with text feature extraction. In 3.4, there are described methods of extracting features from text and representing text in a vector space.

There are two main categories of machine learning tasks: supervised and unsupervised learning. While the supervised algorithm uses an input dataset with prepared examples of output, the unsupervised algorithm uses data as it is and tries to identify similar patterns in the data. Supervised algorithms are used for classification and regression. Unsupervised algorithms are used for clustering, dimension reduction and anomaly detection. [18] In 3.2 and 3.1, there are described two supervised algorithms for classification. 3.3 describes how to use a classifier for more than two classes. An example of an unsupervised algorithm in terms of text topic analysis is Latent Dirichlet Allocation but this algorithm is not described in this thesis because the algorithm did not comply and was not part of experiments.

3.1 Multinomial Naive Bayes

Multinomial Naive Bayes is a probabilistic learning method. The method uses Bayes rule to compute the posterior (conditional) probability of the class for a text. The posterior probability is based on the distribution of the words in a text. There is an assumption that each word in a text is independent and the word position is not relevant to the text class. [4]

3.1.1 Method Definition

Multinomial Naive Bayes classification is defined as 3.1 but it is better to use 3.2 because of a possible floating point underflow. A transformation between 3.1 and 3.2 is using equation $\log(xy) = \log(x) + \log(y)$ and can be done because the most probable class also has the highest log probability. [15]

$$c_{map} = \arg\max_{c \in C} \hat{P}(c) \prod_{1 \leq k \leq n_d} \hat{P}(t_k|c) \tag{3.1}$$

$$c_{map} = \arg\max_{c \in C} [\log \hat{P}(c) + \sum_{1 \leq k \leq n_d} \log \hat{P}(t_k|c)] \tag{3.2}$$

Classifier is finding a class c from a set of classes C with best probability. $\hat{P}(c)$ is a probability that text is a class c and $\hat{P}(t_k|c)$ is a posterior probability that text with class c contains a word t_k. Accordingly, probability of a class is calculated as a product of probability of a class $\hat{P}(c)$ and probabilities of words in a class $\hat{P}(t_k|c)$ for each word in a text. [15] $\hat{P}(t_k|c)$ is a defined as 3.3 but it is better to apply add-one smoothing ([15]) and use 3.4 because probability from 3.3 for non-dictionary word will be zero and then a probability of a class will be zero. [9]

$$\hat{P}(t_k|c) = \frac{T_{ct}}{\sum_{t' \in V} T_{ct'}} \tag{3.3}$$

$$\hat{P}(t_k|c) = \frac{T_{ct} + 1}{\sum_{t' \in V} T_{ct'} + |V|} \tag{3.4}$$

T_{ct} is a count of occurrences of a given word t in texts with class c used for training. $\sum_{t' \in V} T_{ct'}$ is a sum of all lengths of texts with class c and $|V|$ is a vocabulary length. [9]

3.1.2 Example

In this example, there are two classes (a and b) and a set of texts. For simplicity, each text can contain only two words: "Happy" and "Sad". The goal is to classify this text: "Sad Sad Sad Happy Happy"

The first dataset has three sentences and each sentence contains 5 words.

- a: Happy Happy Happy Happy Sad
- b: Happy Sad Sad Sad Sad
- a: Happy Happy Happy Sad Sad

First, it is necessary to calculate the probability for each class and then the posterior probability for each word and class:

$$P(a) = \frac{2}{3} \quad P(b) = \frac{1}{3}$$

$$P(Happy|a) = \frac{7+1}{10+2} = \frac{2}{3} \quad P(Sad|a) = \frac{3+1}{10+2} = \frac{1}{3}$$

$$P(Happy|b) = \frac{1+1}{5+2} = \frac{2}{7} \quad P(Sad|b) = \frac{4+1}{5+2} = \frac{5}{7}$$

Then, a classifier calculates probabilities of classes:

$$P(a|t) = \frac{2}{3} * (\frac{1}{3})^3 * (\frac{2}{3})^2 = 0.01097$$

$$P(b|t) = \frac{1}{3} * (\frac{5}{7})^3 * (\frac{2}{7})^2 = 0.00992$$

The classifier would pick class a because it has a bigger probability. However, the tested text is more "Happy" than "Sad" and the result is inaccurate. This is an example that the short text classification depends more on class distribution than on used words. Now, the classifier will use a data set with an added word in each sentence:

- a: Happy Happy Happy Happy Happy Sad
- b: Happy Sad Sad Sad Sad Sad
- a: Happy Happy Happy Sad Sad Happy

The probability for each class will be the same. It is necessary calculate only posterior probability for each word and class and then probability of class for the tested document:

$$P(Happy|a) = \frac{9+1}{12+2} = \frac{5}{7} \quad P(Sad|a) = \frac{3+1}{12+2} = \frac{2}{7}$$

$$P(Happy|b) = \frac{1+1}{6+2} = \frac{1}{4} \quad P(Sad|b) = \frac{5+1}{6+2} = \frac{3}{4}$$

$$P(a|t) = \frac{2}{3} * (\frac{2}{7})^3 * (\frac{5}{7})^2 = 0.0039666$$

$$P(b|t) = \frac{1}{3} * (\frac{3}{4})^3 * (\frac{1}{4})^2 = 0.008789$$

The classifier would pick class b for the tested document and the second dataset.

3.2 Support Vector Machines

Support Vector Machine is a machine learning method for classifying objects into two classes. The method is non-probabilistic therefore a result of a classified object is a boolean value (not a probability of a class). The classification is based on finding a decision boundary in a vector space (line in two-dimensional, surface in three-dimensional) with the biggest margin. In Figure 3.1, there are two decision boundaries and the "boundary A" has a bigger margin than the "boundary B". Finding the best decision boundary is an optimization problem. [15]

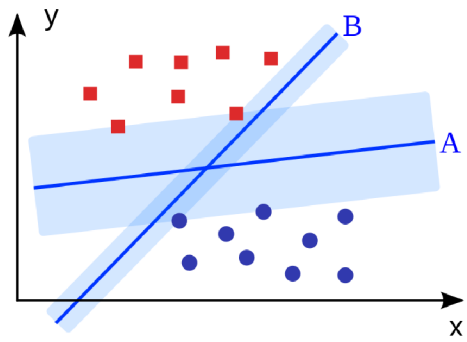

Figure 3.1: **Two Decision Boundaries Margin Difference** The "boundary A" has a bigger margin than the "boundary B". Taken from [20].

3.2.1 Method Definition

This subsection draws from [15] If data are separable into two classes, there is at least one decision boundary. The decision boundary called hyperplane is a line which is defined as 3.5. For example, in a two-dimensional space hyperplane is defined as 3.6

$$\vec{w}^T \vec{x_i} + b = 0 \tag{3.5}$$

$$w_1 x + w_2 y + b = 0 \tag{3.6}$$

w_1 and w_2 are features of a line normal (w) and b is a bias - line offset from center on y-axis. Linear classifier is defined as 3.7.

$$f(\vec{x}) = sign(\vec{w}^T \vec{x_i} + b) \tag{3.7}$$

Assuming that the Support Vector Machine requires a margin size greater or equal to one and the best decision boundary has at least one point belonging to a margin border, it is possible to derive a relation for overall margin width:

$$width = \frac{2}{|w|} \tag{3.8}$$

The optimization task is finding w and b that:

- $|w|$ is minimal.
- For all $(\vec{x_i}, y_i)$ in a training set, $y_i(\vec{w}^T \vec{x_i} + b) \geq 1$ holds.

Lagrange's method can solve this quadratic optimization problem. In the next section, there is an example of an algorithm for optimization.

3.2.2 Optimization Algorithm Example

This is an optimization algorithm example and this section draws from [12]. The algorithm starts with large vector w (almost zero margin) and trying to find a vector transformation and b which suits with provided dataset, then for best w trying to decrease vector size (increase margin) and repeat. The algorithm has room for improvements.

1. **max, min** - Find the maximal and minimal feature value
2. Define algorithm variables
   - **transformations** - Vector transformations - For example [[1,1],[-1,1],[-1,-1],[1,-1]] (Room for improvement)
   - **w_step_sizes** - Step sizes for w - [**max** * 0.1, **max** * 0.01, **max** * 0.001]
   - **b_step** - Step size for b
   - **b_range** - B multiplier
   - **latest_optimum** - Latest Optimum - At start as [**max** * 10, **max** * 10]
   - **latest_optimum_b** - Latest Optimum for b
3. For each **w_step_size** from **w_step_sizes**:
   (a) Define:

Table 3.1: **Data for Transformations Comparsion**

Class	x	y
-1	1	7
-1	2	8
-1	3	8
1	5	10
1	6	-1
1	7	3

- **optimum_dict** - Optimum dictionary
- **w** - Assign **latest_optimum**
- **optimized** - Assign false

(b) While **optimized** is not true:

i. For **b** in range between - **max** * **b_range** and **max** * **b_range**

A. Transform **w** to all **transformations**

B. If some **w** with **b** suits with dataset, add this **w** and **b** to **optimum_dict** with key nominal length of **w**

ii. If **w**[0] is below zero, set **optimized** to true, else decrease **w_step_size** from **w** (Room for improvement)

(c) Take the **element** from **optimum_dict** with lowest key

(d) Assign to **latest_optimum** value [**element.w** + **w_step_size** * 2, **element.w** + **w_step_size** * 2] (Room for improvement)

4. The decision boundary is defined by **element.w** and **element.b**

In [12] and in the algorithm example, the transformation array is [[1,1],[-1,1],[-1,-1],[1,-1]]. This means that the algorithm transforms the latest optimum vector only into four diagonal directions and also the decision boundary will be a diagonal at the end. It is better to generate points in all directions. For example, the algorithm using that four transformations did not found a decision boundary for the data from table 3.1 but algorithm using transformations obtained from a function below did. However, loop ending condition becomes imperfect after this change. This problem was not solved because the algorithm is only a demonstration.

```
def get_transformations():
  circle_points = []
  for x in range (-10, 10):
    for y in range(-10, 10):
      circle_points.append([x / 10, y / 10])
  return circle_points
```

The accuracy also depends on the iterating in the “b range” (range size and step size) although there is the same situation as with transformations. It is a wise choice between accuracy and computing time.

3.2.3 Non-Linear Classification

Sometimes data are not linearly separable as it is shown in the second graph in Figure 3.2. It is possible to make “the kernel trick” - map data to some higher-dimensional vector

space. [15] For example, in the third graph in Figure 3.2 there is an example of mapping data one-dimensional data to two-dimensional vector space using the quadratic function.

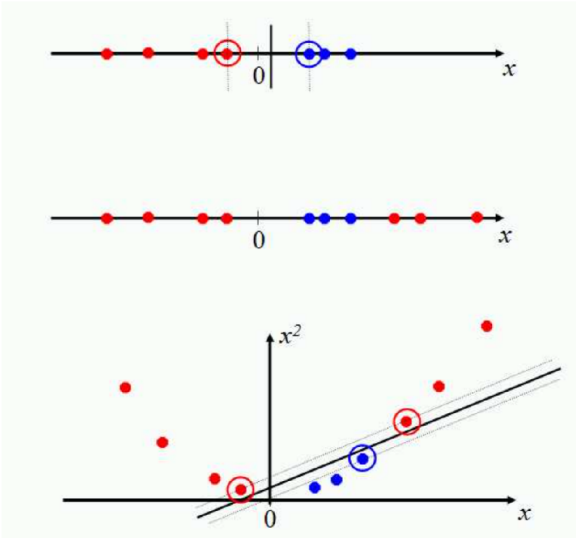

Figure 3.2: **Non-Linear Classification Problem and "Kernel Trick Solution"** Taken from [15].

3.2.4 Implementations

Implementing own models may not be an easy task. Support Vector Machine has many completed implementations and most times using one of this implementation may be sufficient. There are three examples of Support Vector Machine implementations by a programming language:

- **C/C++** - SVM^{light} - `http://svmlight.joachims.org/`
- **Python** - Scikit Learn - `http://scikit-learn.org/`
- **Java** - LIBSVM - `https://www.csie.ntu.edu.tw/~cjlin/libsvm/`

3.3 Classification With More Than Two Classess

This section draws from [15]. There are two methods for classification with more than two classes: any-of (also known as multilabel) and one-of (also known as multiclass). While the any-of classifier can classify an object to several classes or a single class or to none of the classes, the one-of classifier classify to a single class only.

This is the any-of classification algorithm:

1. Build a classifier for each class (dataset is transformed into datasets for each class - positive and negative values).
2. Classify an object by each classifier separately and assign all positive classes.

This is the one-of classification algorithm:

1. Build a classifier for each class (dataset is transformed into datasets for each class - positive and negative values).
2. Classify an object by each classifier separately.
3. Assign a class with maximal probability or a score or a confidence value.

3.4 Text Representation

The support vector machines algorithm described in 3.2 works with numeric data such as vectors, lines and points in a vector space. It is necessary to convert text data into similar representation. One way is convert text to tokens, lemmas, remove stop words and calculate term frequency-inverse document frequency. Applying the term frequency-inverse document frequency also for the multinomial Naive Bayes algorithm described in 3.1 increases a classifier accuracy. [16]

3.4.1 Tokenizing

"Tokenizing is the process of breaking a large set of texts into smaller meaningful chunks such as sentences, words, and phrases." [18]

Process output structure can be a tree or a list of tokens where a token represents a particular text part. Process implementation may vary with intentions. The following sections describe procedures which can be used inside the process.

Lower Case

It is recommended to convert all words to a small form to unify the letters and then represent the same words (with a different letter sizes) to the same token type. However, occasionally loss of meaning may occur. For example, "#Washington" and "#washington" will be same token and the meaning will be still same but in [15] there are listed examples with company names ("General Motors "vs. "general motors"), names ("Bush" vs "bush", "Black" vs "black") or acronyms ("computer-aided translation" as "CAT" vs "cat") where the conversion has meaning loss.

Special Terms

Texts often contain special terms like links to websites, phone numbers, etc. These terms can be converted to tokens representing term context but with some level of information loss. For example, the sentence "Visit https://example.com/posts/hello" can be after tokenizing:

- "visit LINK"
- "visit LINKTO example.com"
- "visit LINKTO example.com/posts/hello"

Text Noise

White characters, punctuation, and other special characters in text are usually not so significant in comparison with words. In that case, tokenizing should also remove these characters. For example, the sentence "Hello, welcome in Seattle. How was your flight?" can be after tokenizing "hello welcome in seattle how was your flight".

Table 3.2: **Lemmatization and Stemming Example**

Word	Stem	Lemma
was	wa	be
eating	eat	eat
she	she	she
worse	wors	bad

3.4.2 Stop Words

Text can contain words that are not so significant for the field of text and analysis type. These words are called stop words. Removing stop words can increase the accuracy of analysis methods. This procedure also speeds up analysis processing time because the dataset size should be lower. [17]

The field of text and analysis type is important for stop words selection. Most times, the selection is the taking the most frequent words across texts in the dataset with hand-filtering by field of text and analysis type. [15]

In [17] is listed an example with term "very high frequency radio". This term by the International Telecommunication Union signifies a frequency between 30 and 300 MHz. The word "very" can be evaluated as stop word but term "high frequency radio" signifies a frequency between 3 and 30 MHz.

3.4.3 Lemmatization and Stemming

"The goal of both stemming and lemmatization is to reduce inflectional forms and sometimes derivationally related forms of a word to a common base form." [15]. While stemming only crops words to the root of the word, lemmatization replaces words with their base form by a particular dictionary. "stemmer" applies specific transformation rules and "lemmatizer" works as a thesaurus. Decision between lemmatization and stemming can depend on text language and also on performance because "stemmers" based on algorithmic transformation are less performance-intensive than "lemmatizers". In Table 3.2 there is an example of differences between stemming and lemmatization.

3.4.4 Text to Vector Conversion

Furthermore, the text unit (in the thesis case, for example "tweet" text) is referred to as a document. Each document has properties like a topic, sentiment, a key message and so on. It is necessary to represent documents in a vector space for any analysis. Then, it is possible to compare distances between documents and their properties and do analysis and aggregation.

Assuming that document properties are defined by a particular word occurrence, it is possible to use word occurrence statistics for the document to vector conversion. For example, the simplest method is based on counting word occurrence in all documents. The second option is the term frequency-inverse document frequency referred to as TF-IDF. However, these methods (called "Bag of Words") neglect the order of words and document grammar. [18]

Each word in a document has a particular relevance to the whole document, sentence and words around the word. "In the area of information retrieval, TF-IDF is a good statistical

Table 3.3: **TF-IDF Values Example**

	0	1	2
am	0.000000	0.000000	0.795961
computer	0.680919	0.00000	0.000000
human	0.000000	0.57735	0.605349
is	0.517856	0.57735	0.000000
this	0.517856	0.57735	0.000000

measure to reflect the relevance of the term to the document in a collection of documents or corpus." [18].

TF-IDF is defined as 3.9 where tf is defined as 3.10 and idf is defined as 3.11:

$$tfidf = tf * idf \tag{3.9}$$

$$tf = \frac{\text{Number of times term appears in a document}}{\text{Total number of terms in the document}} \tag{3.10}$$

$$idf = \log\left(\frac{\text{Total number of documents}}{\text{Number of documents with a given term in it}}\right) \tag{3.11}$$

For example, Table 3.3 shows TF-IDF results for these three documents:

- **0:** This is a computer.
- **1:** This is a human.
- **2:** I am a human.

Differences between these documents are in a simple machine-readable format because documents are represented as vectors. For example, the first document has this representation: [0.000000, 0.680919, 0.000000, 0.517856, 0.517856].

An alternative to statistical methods is doc2vec[1]. It is possible to train own doc2vec model on a particular dataset and use this model for the document to vector conversion. In [8] is a good comparison between doc2vec and TF-IDF.

[1]Doc2vec is based on word2vec but it adds a paragraph vector. [13]

Chapter 4

System for the contents aggregation of selected sources

This chapter is about the design of the system for getting content from selected sources and for content aggregation and statistics. Section 4.1 describes basic design principles for the system and Section 4.2 describes system architecture itself.

4.1 Application Design

It is possible to split system requirements for an application design into two sections by two points of view: From the user's perspective and its functional requirements, and from the developer's point of view. In other words: define requirements for developer's convenient development.

4.1.1 Functional Requirements

An ideal application use scenario has three main parts. A user will subscribe to some sources (posts from a particular account, posts containing a word or a hashtag) and the application automatically gets contents from subscribed sources. The application will provide analysis and some statistics, and the user will display selected contents, analysis results and statistics. The sections below describe the main functional requirements.

Contents from Selected Sources in One Place

A user wants to subscribe to selected posts from multiple platforms and have that posts in one place. For example, the user can select several sources depending on a platform like posts from a particular account, publisher or posts containing a particular word or hashtag. Posts have to be synchronized minimally each hour and the application should contain a way to get data on demand. The user also wants to sort posts.

Contents Analysis

A user wants to define topics that can be assigned to a post. A topic has to represent a post classification to some criteria. For example, the user can define topics for post topic, sentiment, etc. Application should be able to suggest topics for each post and user should be able to confirm particular suggestion. For example, the application suggests two topics:

“Technology” and ”Negative”. The user selects only “Technology” and the second topic will be discarded. The user also wants to filter contents by topics.

Contents Statistics

The application should provide at least basic content statistics like number of posts in time, by topic and platform. Also, other statistics may be useful to the user. For example, posts with the most reactions, the most posting authors, a count of posts without topic, overall topic distribution or the most frequent words.

4.1.2 Architecture Requirements

This section describes three main architecture requirements. The thesis author set these requirements considering that the implementation will be open sourced. It is not a single approach. For example, it is possible to design the application as a closed system running on one configuration only, with a fixed graphical interface, without the possibility of adding new extensions and third-party usability.

API-First Architecture

API-First Architecture (or API-First Design) is an application development approach that prefers designing the application interface based on the domain analysis as opposed to the user-interface driven design, which is quite commonly used. [19] The next reason for API-First Architecture is a parallel backend a frontend development possibility and simple openness to third-parties. For example, an application has a graphical user interface only as a website and someone will create a mobile application.

Application development starts with an Application Programming Interface (in most times a REST API) design. The mindset “Your API is the first user interface of your application” can be helpful for the design process. API definition should cover all product functionality before the backend and frontend implementation. API should not change frequently. Adding new features to a well-designed API should not affect the overall structure. On the other hand, it is possible to change, refactor or optimize an implementation without affecting the second side. For example, refactoring backend to a new version with same API does not affect frontend side, mobile applications or third-party API consumers. [19]

A big emphasis was also placed on making the interface RESTful, for example keep collection pattern which is decrebed in [5].

Operation System Independency

The open sourced implementation should be easy to configure and set up. A Software Developer who wants to try the implementation should be able to run this system on a local machine regardless of operation system (Linux, OSX, Windows). The repository should contain a README file that describes system configuration and setup. An ideal configuration is as a one file (in JSON format) for inserting API keys and other third-party values and the setup is as one line command in the terminal.

Extensibility

The system should be ready for new features, improvements and extensions. The key consequence in development is having a set of tests (Chapter 6 describes testing) that

covers the API. The set of tests ensures safe adding new features without affecting existing features. For example, the system should be well designed and implemented for easy and safe addition or integration of:

- New social media platform
- Another database
- Further analysis and statistics
- User groups and an authorization layer
- New automation
- Another internal or third-party API

4.2 System Architecture

The basic idea of the system architecture is one public REST API that has some functionality and also works as a proxy for internal APIs. This is because of the possibility of using different technologies for different operations. For example, Python has many libraries for machine learning but it does not have good support for functional constructions like collection map, filter, etc. in comparison with Kotlin which was chosen for fundamental operations like getting and editing the data, eventually for calculations for statistics. HTTP Request forwarding can be defined in a request proxy configuration or it is a possible create forwarding endpoints in the main backend application.

The graphical user interface for the application is a single page application. System API serves on a root endpoint "/" a set of HTML and JS files. Serving these files handles the internal Node.js server. and in a client's web browser executes JS files for starting the single page application which handles sending requests to an API (with a prefix "/api/"), dynamic HTML changes and URL routing.

The system contains two databases: SQL and NoSQL. The reason for using two databases is suitability for particular cases. Content from social media and networks are big data and fast access is necessary. The thesis author chose a NoSQL database for this use case after consultation with the supervisor. However, NoSQL database does not guarantee data consistency. Taking into account that the application in future could contain more sophisticated data entities and its relations like user groups with permissions, SQL database were chosen for other data such as sources and topics.

In Figure 4.1, there is a system scheme for clarity. The sections below describe each part of the system in more detail.

4.2.1 Database Model

The SQL Database contains four tables. Table for topics, sources, users and analysis model trainings. The NoSQL contains a set of collections of posts. Each collection contains posts for one platform.

SQL Database

All data types are from PostgreSQL because this database type was used for the implementation. Table 4.1 shows columns for topics and Table 4.2 shows columns for sources.

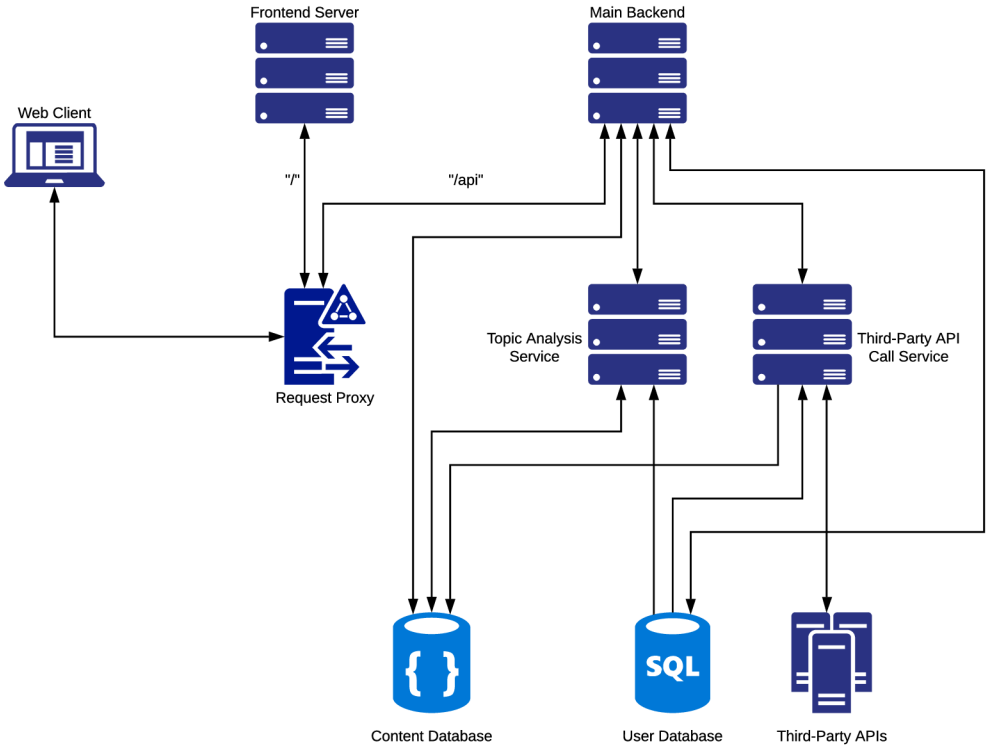

Figure 4.1: **System Architecture Scheme**

Table 4.1: **Topics Table Columns**

Name	**Type**
id	SERIAL NOT NULL PRIMARY KEY
text_id	TEXT NOT NULL
name	TEXT NOT NULL UNIQUE

There are columns *platform* and *value_type*. Each platform can contain different source types. For example, Twitter has sources for a hashtag, an account or a word but Facebook has only pages and groups. These source types are important not only for social platform API calls but also for validations in a graphical user interface. Source-type definitions are hard-coded because the application uses only 4 platforms. However, the API prevents from wrong combination inserts and in the future, there is a possibility to create another table containing source-type definitions for more dynamic configuration.

All analysis model trainings are in one table, the table has these six columns and these columns are shown in Table 4.3.

The last table for users contains five columns and its names and types are shown in Table 4.4.

NoSQL Database

There was a decision between one NoSQL collection for all posts from all platforms and several collections for each platform. The first case would support queries for last 20 posts

Table 4.2: **Sources Table Columns**

Name	Type
id	SERIAL NOT NULL PRIMARY KEY
platform	TEXT NOT NULL
value_type	TEXT NOT NULL
value	TEXT NOT NULL UNIQUE

Table 4.3: **Trainings Table Columns**

Name	Type
id	SERIAL NOT NULL PRIMARY KEY
model_id	TEXT NOT NULL
is_done	BOOLEAN NOT NULL
start	TIMESTAMP NOT NULL
end	TIMESTAMP
accuracy	DECIMAL

by time from all platforms, but then each post would have to contain a field for a platform. The second case avoids "platform switch" in a code. The post structure can differ for each platform but fields in Table 4.5 are uniform across platforms.

4.2.2 Representational State Transfer Application Interface

As described in 4.1.2, a big emphasis was placed on the API structure. The thesis author chose a Representational State Transfer (further referred to REST) style for the API. REST is a software architectural style with a set of constraints like Client–server architecture, Statelessness, Uniform interface, etc.[5]

The API design is based on the database model. Because the database contains several collections for each platform, each platform has its own endpoint. The same goes for an analysis. It is very likely that the analysis model will not be the only one in the future. Therefore, API design contains endpoint "analysis" and nested endpoint "topic". The API design also contains endpoint "auth/signup". This endpoint is available only for the first use and it is for creating first user after first system deploy. For statistics, this endpoint returns a small set of statistics and in that time it did not make sense to use a different structure. The API has the following structure:

- **/auth**
  - **/accessToken** - POST
  - **/firstUser** - POST (Available only for the first user)

Table 4.4: **Users Table Columns**

Name	Type
id	SERIAL NOT NULL PRIMARY KEY
username	TEXT NOT NULL UNIQUE
name	TEXT NOT NULL
email	TEXT NOT NULL UNIQUE
password	TEXT NOT NULL UNIQUE

Table 4.5: **Post Document Required Fields**

Name	**Type**
_id	String
timestamp	String (ISO format)
text	String
topics	List of strings
suggestedTopics	List of strings

- **/users** - GET, POST
  - **/me** - GET
  - **/:id** - GET, DELETE
- **/topics** - GET, POST
  - **/:id** - GET, PUT, DELETE
- **/sources** - GET, POST
  - **/:id** - GET, PUT, DELETE
- **/contents**
  - **/twitter** - GET
    - **/:id** - GET, DELETE
      - **/topics** - PUT, DELETE
  - **/news** - GET
    - **/:id** - GET, DELETE
      - **/topics** - PUT, DELETE
  - **/reddit** - GET
    - **/:id** - GET, DELETE
      - **/topics** - PUT, DELETE
  - **/facebook** - GET
    - **/:id** - GET, DELETE
      - **/topics** - PUT, DELETE
- **/analysis**
  - **/topic**
    - **/trainings** - POST
      - **/last** - GET
      - **/running** - GET
      - **/:id** - GET
    - **/suggestions** - POST
    - **/accuracy** - GET
- **/statistics** - GET
- **/jobs**
  - **/contentDownloads** - POST

4.2.3 Contents Analysis Service

The main requirement for analysis service is an asynchronity. The application should be able to retrain analysis models in order to get better results. For example, a user will confirm suggested topics for several posts and these posts can have a big effect on new topics. However, time of model training can take a few seconds, hours or even days and it is necessary to send a response to request immediately after processing the request.

When a request for a training is sent, the content analysis service creates a row in the table for trainings with status *is_done* equal to *false* and then the service starts the learning itself. Once the learning is completed, the service changes status *is_done* to *true*. The trained model is a saved as file in case of the service restart.

The communication interface between the main backend and analysis service is a REST API after consultation with the supervisor and it should be uniform across all analysis services.

Chapter 5

Implementation

A source code repository contains *README.md*, *.gitignore*, *.editorconfig* and directories for the system parts. The following sections describe the multi-container application approach, continuous integration configuration and implementations of particular parts of the system.

5.1 Multi-Container Application

Container is a lightweight execution environment which can be used for an application isolation and which is sharing the operating system kernel. In comparison with virtual machines, containers are not completely isolated and container can share a computing power and a memory with another container. Thanks to this, operations with containers such as start-up and shutdown are very fast. [21] Docker[1] is one of container platforms for a container management and this technology was used in this implementation because Docker has a majority in the industry and there was a presumption of good documentation. [6] Docker container is created from a Docker image. It is possible to get predefined images such as Node.js, OpenJDK or Fedora.

At the beginning of the implementation, containers were used for databases but then also for a deployment. Docker Compose was chosen as a container management tool and in the directory called *docker* there are two files: *docker-compose.yml* and *docker-compose.prod.yml*. The first file contains a definition of containers for databases. Containers for backend, request proxy and other services are defined in the second file. This division is because of the possibility to set up only database containers for the development. Database containers must have a definition of a volume for the database data because if a new database container is created with an existing volume, the container will use that volume and data from the volume persists.

It is possible to connect to the container on a localhost on a particular port (for example, Mongo on 27017) but if an application inside a container wants to access another container, it is necessary to change the host value from localhost to container name. For example, Mongo container is defined as *content_database* and the backend application has to connect to *content_database:27017*.

Some project directories contain a file called *Dockerfile*. This file is used for building a new Docker image and a deployment process builds Docker image and creates a container from this image for each part of the implemented system.

[1] https://www.docker.com/

5.2 Topic Analysis Service

Topic Analysis service is implemented in Python. It has its REST API which can be called by main backend or by NGINX proxy. API server is implemented using a Flask[2] framework which is not so huge library for REST API implementation. Unit tests can be executed by PyUnit. [14] was used as an inspiration for the classification algorithm. All dependencies are listed in a file called *requirements.txt*

5.2.1 Data Pre-Processing

All helper functions for text pre-processing have a unit test coverage. Once the dataset is retrieved from a database, it is converted into a Pandas data frame[3] which is better than a simple list of posts in case of data exploration and model training. The data frame has columns for topics and for post text and id. Each row in a data frame represents a post.

Then, it is necessary to clean up a text in each post. In the thesis appendix, there is a list of characters which is used for not necessary character removal. Before the removal, post text is converted to tokens using NLTK TweetTokenizer[4] with a combination of regular expression for replacing date tokens to "DATE". URL tokens are left as they are. This tokenizer is better for a social media content in comparison with WordTokenizer because word tokenizer converts "#hashtag" to two tokens - "#" and "hashtag". Text tokens are replaced by their lemmas if it is known. As a lemmatization library was chosen Majka[5] because this library besides main languages like English, German, Spanish also supports Czech language. Text language is detected by langid library[6]. All words from a text are converted to lowercase form and then stop words are filtered out from the text. The implementation uses a stop words from a Python library called stop-words[7]. TfIdfVectorizer[8] converts cleaned text to the TF-IDF vector.

It was necessary to consider whether add text properties like platform, author or publisher directly to the text or create new dimensions in a vector for these properties. Considering that an author in not trained data can be unknown but a platform is always defined, the author and the platform are attached to a text before the vectorization. However, the platform index could be added as a new feature to the vector.

5.2.2 Classification Model

Topic classification is using the Support Vector Machines method described in 3.2 but it is possible to use Multinomial Naive Bayes method by changing one parameter in a pipeline shown below. The reason for using this method was better overall accuracy in comparison with Multinomial Naive Bayes model.

On attached CD is enclosed a reference dataset for comparison. The data are from the context of the city of Brno. It is a combination of tweets and news articles. Any data that might be sensitive were replaced by the appropriate substitute. These topics were chosen for analysis: traffic, work, sport, culture and politics. Posts were naively labeled by

[2] http://flask.pocoo.org/
[3] https://pandas.pydata.org/pandas-docs/version/0.23.4/generated/pandas.DataFrame.html
[4] https://www.nltk.org/api/nltk.tokenize.html#module-nltk.tokenize.casual
[5] https://nlp.fi.muni.cz/czech-morphology-analyser/
[6] https://github.com/saffsd/langid.py
[7] https://github.com/saffsd/langid.py
[8] https://scikit-learn.org/stable/modules/generated/sklearn.feature_extraction.text

Table 5.1: **Accuracy Difference for labeled 168 posts**

Topic	Support Vector Machines	Multinomial Naive Bayes
traffic	0.98	0.93
sport	0.89	0.79
work	0.89	0.86
culture	0.88	0.88
events	0.82	0.80
politics	0.89	0.89

thesis author. Accuracy difference between these models is shown in Table 5.1 and topic distribution for the dataset is shown in Table 5.2. In Figure 5.1, there is a visualization of the most common words. The dataset was divided into two sets (for training and testing) using *train_test_split* function[9] with these parameters:

- **random_state**: 42
- **test_size**: 0.33
- **shuggle**: True

Figure 5.1: **Most Frequent Words**

[9] http://scikit-learn.org/stable/modules/generated/sklearn.model_selection.train_test_split

Table 5.2: **Topic Distribution for 168 posts**

Topic	Count
traffic	38
sport	59
work	18
culture	19
events	33
politics	23

The classification algorithm uses pre-implemented Support Vector Machines model. The algorithm uses scikit-learn SVM[10].

```
Pipeline([
  ('tfidf', TfidfVectorizer()),
  # Not used - ('clf', OneVsRestClassifier(MultinomialNB(...), ...)),
  ('clf', OneVsRestClassifier(LinearSVC(), n_jobs=1))
])
```

The pipeline reference can be used for a fit function call but it is necessary to call this function for each topic because the classification is a multi-label classification problem. Once the model learning is finished, models are saved as files with name *topic_id_model.pkl* using Python object serialization[11]. These files are loaded on a REST API server start.

On the suggestion endpoint, the server checks that models are loaded and then call the predict function for each topic. In the case of missing models, the server return error code.

5.3 Third-Party API Call Service

The second Python REST API server is for third-party API calls. The API server has endpoints for getting the social media a networks data. These endpoints are called by job scheduler or by user request forwarded from a backend. The server implementation is in similar technologies as a topic analysis service and does not have a bigger complexity.

5.4 Backend

Backend is developed in Kotlin using Spring framework[12]. Tests are written in KotlinTest framework[13] which supports descriptive test names and adds a set of advanced matchers. JUnit platform can be used for the test execution.

5.4.1 Configuration

All dependencies are defined in a file called *build.gradle* and this file also contains a configuration for a JOOQ generator. JOOQ[14] is a Java library for object-relational mapping. In addition to simplifying query writing, it is possible to generate classes for the database

[10] https://scikit-learn.org/stable/modules/generated/sklearn.svm.LinearSVC
[11] https://docs.python.org/2/library/pickle.html
[12] https://spring.io/
[13] https://github.com/kotlintest/kotlintest
[14] https://www.jooq.org/

structure. JOOQ generator is configured only for the user database. JOOQ does not support NoSQL databases. The code is generated into the package *db.generated.user_database*. This package contains classes for data access objects and plain old Java objects for each table in the database.

The file *build.gradle* also configures a Flyway library[15]. This library is a solution for database migrations. Besides a database connection definition, *build.gradle* configures that the Kotlin code compilation depends on JOOQ code generation task and this task depends on database migrations. In other words, database structure is in accordance with the code and each compilation checks the structure and generated code and eventually migrations and code generation is executed. The database structure definition is in a package *db.migration* in a directory called *resources*. In this directory can be SQL files with name *Vx__name.sql* where the x is a number which defines a SQL script order.

In the directory resources, there is also a file application.yaml. This file defines database connections (PostgreSQL and MongoDB) for the Spring Application. Databases need to be configured also in the Spring Application configuration. In the package *configuration* are classes with Spring annotation *@Configuration* for the Spring Application configuration. Specifically, there are classes for both databases, authentication and CORS filter settings.

As mentioned in 5.1, it is necessary to configure hosts by its container name. However, JUnit tests are not executed inside a container and hosts need to be configured as *localhost*. This problem is solved with a second configuration. This configuration is in a directory with tests. There is also a directory called *resources* and its content is the same except host values. Regarding configuration classes, there is a difference and these classes must have an annotation *@TestConfiguration*.

5.4.2 Data Classes

MongoDB database for social media contents does not have code generation support and it was necessary to define appropriate classes for plain old java objects. Package *pojos* is determined for these classes. The same situation is for objects returned as an API response. Kotlin provides a data class concept for this use case. By default each class should be a data class but classes for platform post (for example *Tweet*, *RedditPost*, etc.) are basic classes extending from the abstract class called *Post* because of common fields _id, timestamp, text, etc. It is important to add annotation *@Document(collection = "collection_name")* to a class representing a platform post. This annotation defines a MongoDB collection containing objects defined by that class.

Although there are data access object classes in the generated code, some queries are more complex and it is advisable to wrap them into a special class. This type of class is called a repository. These repositories are in a package *domain.impl*. For example, there is a *SourceRepository* extending a *SourceDao* and overriding insert methods with an implementation containing a source type validation. It is possible to define an interface extending a generic interface called *MongoRepository*. The generic type is a class with *@Document* annotation for the collection document. The *MongoRepository* interface is from a MongoDB library for the Spring framework and automatically implements basic queries like "find all documents", "find by id", etc. All methods can have a parameter called *pageable* of class *Pageable*. This parameter transforms returned result into a page and it is possible to load the content as a page by page in order to decrease response size and API consumer performance demands. A class extending the *MongoRepository* interface

[15] https://flywaydb.org/

can define its own methods that have an annotation *@Query(value = "")* for the MongoDB database.

In the package *domain*, there is an interface called *GenericPostRepository*. The *GenericPostRepository* interface is an abstraction for platform post repositories and also needs to be used with a generic type - in this case a class for a platform post. Extended interface *GenericPostRepository* defines four methods:

- Find posts without topics: "{$or: [{'topics': {$exists : false}}, {'topics': {$eq: [] }}]}"
- Find posts with suggested topics: "{$and: [{'suggestedTopics': {$exists : true}}, {'suggestedTopics': {$ne: [] }}]}"
- Find posts containing topics: "{ 'topics' : {$in : ?0 }}"
- Find posts in timestamp range: "{'timestamp': {'$gte': ?0, '$lte': ?1}}"

An example below shows the simplicity of repository definition - in this case for Twitter posts:

```
@Repository
interface TweetRepository : GenericRepository<Tweet>
```

5.4.3 Services

Calculations for statistics did not have a simple implementation on a few lines and all more complex code for calculations and data transformations should be wrapped into services. In package *services*, there is a class called *StatsService* implementing necessary calculations for statistics.

The implementation manifested a disadvantage of splitting post to several collections for each platform because it is necessary to query the data several times and map them to a list of post objects. Methods for calculations use Kotlin collection functions like the map, filter, etc.

One of the more difficult things in this implementation was a right choice for the timestamp representation in Kotlin respectively in a Java language. The Java language, unlike for example JavaScript, has many classes for date and time data and the choice is confusing sometimes. Finally, the *OffsetDateTime* class was chosen.

5.4.4 Controllers

All API endpoints are handled by controller classes. These classes have annotation *@RestController* and are located in package called *api*. It is necessary to implement public method with one of these annotations: *@RequestMapping*, *@GetMapping*, *@PostMapping*, *@DeleteMapping*, etc. Method parameters with annotation *@RequestParam* define request parameters. Method return type defines a request response type. Plain old Java objects are returned as JSON. Sending a HTTP status code is invoked by throwing a *ResponseStatusException* with a constructor parameter of class *HttpStatus*.

As with repositories, the controller has the appropriate abstraction. In the package *api.content*, there is an abstract class called *GenericPostController* with a generic type representing a platform post. For the platform controller definition, it is necessary to extend this class with a *@RequestMapping* annotation for an endpoint prefix and with a particular type and repository as is in the example below - also in this case for Twitter:

```
@RestController
@RequestMapping(''/contents/twitter'')
class TwitterController(
  tweetRepository: TweetRepository
) : GenericController<Tweet>(tweetRepository)
```

5.4.5 Authentication

Requests starting with prefix "/auth" are without required authentication. All other requests require authentication with a bearer token. This authentication is configured as a request filter in a class *AuthConfiguration* in a package with Spring configuration. The bearer token has a JSON Web token standard that uses a signature algorithm called HS512. A signing key and expiration time is configured in a file *application.yaml* in a directory with Spring resources.

5.5 Frontend

The frontend part is a client side rendered single page application. The application is implemented in JavaScript using a React[16] library and Material UI components[17]. Jest was chosen as a unit testing framework. The only difference between development and production configuration is an environment variable called *REACT_APP_API_URL*. If this variable is set, the frontend application will use it for API calls. One statement in *Dockerfile* for frontend is the environment variable definition. Almost all implemented components are stateless. Non-stateless implemented components are listed in this section. Application screenshots are in thesis appendices.

The application has implemented routing in order to keep application browsing history. In addition, it is possible to directly display particular view by entering an URL thanks to the routing implementation. React Router[18] library was used for the implementation.

The main component *Application* in a file *Application.js* contains a definition of platforms and loads topics, sources from a backend. The *Application* component displays a dashboard by default. It is possible to change the dashboard view to another from a side menu. For example, the side menu has items for dashboard, posts, topics, sources. All these views are defined in separate components. The *Application* component is not stateless.

The dashboard is composed from material cards showing statistics. The dashboard components loads statistics and model information from the backend and is not stateless. React Google Charts[19] library is used for displaying graphs. A user also can retrain topic classifier directly from the dashboard. The posts view is for viewing posts and it is possible to enable "interactive mode" for browsing posts one by one. Topics and Sources views are for editing only.

Posts view displays posts in tables. Each table represents one platform and posts are paged. Each table is configurable in terms of page size, post sorting by some property and filtering. A component *PlatformPostTable* was implemented for these tables. This component is not stateless. In listing below, there is an example of platform definition in the *Application* component and *PlatformPostTable* component usage.

[16] https://reactjs.org/
[17] https://material-ui.com/
[18] https://reacttraining.com/react-router/
[19] https://github.com/RakanNimer/react-google-charts

```
const platforms = [
  {
    id: ‘‘twitter’’,
    name: ‘‘Twitter’’,
    columns: [
      {
        id: ’timestamp’,
        valuePath: [’timestamp’],
        label: ’Timestamp’,
        valueFormatter: getDate
      },
      ...
    ],
    getPostsAsPage: getTwitterPostsAsPage,
    deletePost: deleteTwitterPost,
    savePostTopics: saveTwitterPostTopics,
  },
]
...
<div>
  {
    platforms.map((platform, index) =>
      <PlatformPostTable
        key={index}
        platformName={platform.name}
        platform={platform.id}
        columns={platform.columns}
        topics={topics}
        deletePost={platform.deletePost}
        getPostsAsPage={platform.getPostsAsPage}
      />
    )
  }
</div>


  }
/>
```

In the platform definition in the listing, there is a property *columns* which defines table structure and value formatting. Compared to this approach, a component for post detail has defined sub-components for each platform. Sub-component selection is made by a *Switch* component from the React Router library.

If a user signs in to the application, the frontend application saves the user's access token to a local web browser storage. The user can log out from the frontend application and remove the token. If the access token is not stored in the local web browser storage, a login view is shown. This approach was achieved through a component called *PrivateRoute*. [11] The frontend does not contain a sign up view. It is necessary to use API endpoints for adding new users and their management.

All data requests are in folder data. Requests are divided into files by a data subject. For example *Topic.js*, *Posts.js*, etc. Request are sent by Axios[20] library. Data from API sometimes need a transformation for components like graphs. These transformations are in a directory *lib* and all functions from this directory have a particular unit test coverage.

5.6 Request Proxy

Another important part of the implemented system is a request proxy. A user can send two request types: API requests and frontend application requests. Container with configured NGINX[21] was used as a request proxy. Appropriate *Dockerfile* and NGINX configuration file called *nginx.conf* are in a directory *nginx*. The request proxy applies these two rules:

- "/api" request is passed for http://backend:8080/
- "/" request is passed for http://frontend:3000/

5.7 Job Scheduler

Internal API servers have endpoints for jobs for getting data, suggesting topics for all posts, etc. These endpoints can be called by user or automatically. A Docker image called Ofelia[22] was used as a job scheduler. This image is based on CRON but this image simplifies job definitions. A configuration contains these three jobs:

- Get social network data each 10 minutes
- Suggest topics each 30 minutes

5.8 Continuous Integration and Delivery

Each last commit in a master branch in a repository should have a source code that has passing tests. It is advisable to automatically run all tests on each commit push to a repository. The repository is hosted on Github[23] and in the repository, there is a hidden directory called *.circleci*. This directory configures a CircleCI platform[24] for automatic test runs and automatic deployment. In Figure 5.2, there is a scheme of continuous integration of the implemented system. If some parts fail, followed parts will not be executed and continuous integration platform sends a mail notification.

[20] https://github.com/axios/axios
[21] https://docs.nginx.com/nginx/admin-guide/web-server/reverse-proxy/
[22] https://github.com/mcuadros/ofelia
[23] https://github.com/Horm/socialclusters
[24] https://circleci.com/

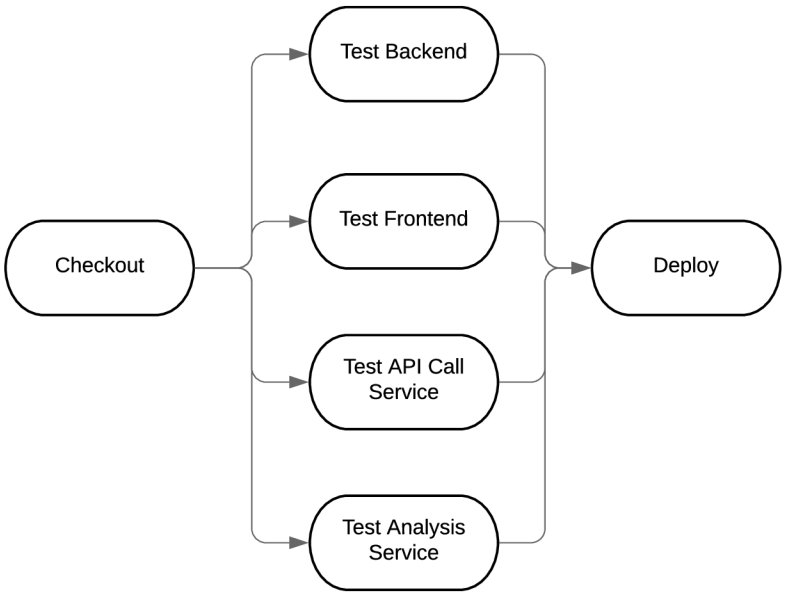

Figure 5.2: **Continuous Integration and Delivery Scheme**.

Chapter 6

Testing

Testing was partially mentioned in chapter 5 because tests were written during the development process. However, this chapter presents results from the testing which can be useful for future extensions. System monitoring was not implemented.

6.1 Analysis Model Validation

The analysis model is the main part of the implemented application. Key aspect of this model is its overall accuracy. However, it is not possible to measure the accuracy because users can retrain the model with new data. In 5.2.2, there is a analysis method comparison on a reference dataset. These comparison results can be also used for the model validation. Support Vector Machines method has better overall accuracy for this use case. An implementation of the analysis model based on support vector machines has sufficient results for the system implementation and the analysis service can be used as an advisory service.

6.2 Unit and E2E Testing

It can be said that all API endpoints of main backend are covered by E2E tests and all data transformation functions are covered by unit tests. The main backend classes has 86% test coverage and a test directory contains a package with 82 test cases divided into 16 classes.

Classes testing requests for internal APIs use an implemented mock that partially simulates the internal server operations with the database. Internal API for analysis is tested only for the training entry creation in the database and there is room for improvement to implement asynchronous tests.

A class *AuthControllerTest* contains static methods for getting an authentication request mocks. This class also tests the functionality of these static methods. These methods are used in all other test classes that require authentication because controller tests should use requests with the same headers and parameters as in normal use.

6.3 Manual Testing

Manual testing approach was used for internal APIs and for the frontend. Individual frontend components were tested in a web browser during development to test their basic

functionality. Internal APIs and its functions were tested using a Postman[1] application and a local Python console.

The main result from manual testing is a finding that many frontend views can be more responsive. Further, it could be worked on better latency and state maintenance of the graphical user interface after the webpage reload.

[1] https://www.getpostman.com/

Chapter 7

Conclusion

The main goal of this bachelor's thesis was to design and implement a web application for content aggregation and analysis from several popular social networks. This goal has been achieved.

The thesis describes the principles and application interface of Facebook, Twitter, Reddit and News Media. The thesis also contains a description of the text to a vector space conversion, Support Vector Machines method and Multinomial Naive Bayes. An extensible system was designed and it was implemented using Support Vector Machines method, Docker, Kotlin, Python and JavaScript.

The system automatically obtains data from social networks defined by a user and the system displays content statistics. The user also can gradually train a topic analysis service and aggregate posts by topic. The system can be used in many contenxts and the system is ready for new extensions and features because all main system functions have test coverage.

The other success of this thesis is a possibility of an easy system setup regardless of the machine type. The author of the work gained experience in machine learning, multi-container applications and React framework. In the future, thesis author would like to improve the system with advanced analysis and statistics, user groups and clear up a legal aspect.

Bibliography

[1] Facebook Graph API Developer Documentation. https://developers.facebook.com/docs/graph-api. accessed: 2019-04-30.

[2] Reddit API Documentation. https://www.reddit.com/dev/api. accessed: 2019-04-30.

[3] Twitter Developer Documentation. https://developer.twitter.com/en/docs.html. accessed: 2019-04-30.

[4] Aggarwal, C., Charu C.; Zhai: *Mining Text Data*. vol. 9781461432234. New York: Springer. 2012. ISBN 9781461432227.

[5] Amundsen, M.; Ruby, S.; Richardson, L.: *RESTful Web APIs*. O'Reilly Media, Inc. first edition. 2013. ISBN 9781449358068.

[6] Carter, E.: 2018 Docker Usage Report. May 29 2018. Retrieved from: https://sysdig.com/blog/2018-docker-usage-report/

[7] Center, P. R.: Social Media Fact Sheet. 2018. Retrieved from: https://www.pewinternet.org/fact-sheet/social-media/

[8] Currie, D.: Predicting Similarity: TfidfVectorizer & Doc2Vec. 2017. Retrieved from: https://www.kaggle.com/currie32/predicting-similarity-tfidfvectorizer-doc2vec

[9] Dočekal, M.: Pokročilé metody strojového učení pro klasifikaci textu. 2016.

[10] Fiala, M.: Propojení sociální sítě Twitter s televizním vysíláním. 2018.

[11] Jason Watmore: React - Basic HTTP Authentication Tutorial & Example. 2018. accessed: 2019-04-30. Retrieved from: https://jasonwatmore.com/post/2018/09/11/react-basic-http-authentication-tutorial-example#private-route-jsx

[12] Kinsley, H.: Support Vector Machine Optimization in Python part 2. Retrieved from: https://pythonprogramming.net/svm-optimization-python-2-machine-learning-tutorial/?completed=/svm-optimization-python-machine-learning-tutorial/

[13] Le, Q. V.; Mikolov, T.: Distributed Representations of Sentences and Documents. 2014.

[14] Li, S.: Multi Label Text Classification with Scikit-Learn. https://towardsdatascience.com/multi-label-text-classification-with-scikit-learn-30714b7819c5. 2018. accessed: 2019-04-30.

[15] Manning, C. D.; Raghavan, P.; Schütze, H.: *Introduction to Information Retrieval*. New York, NY, USA: Cambridge University Press. 2008. ISBN 0521865719, 9780521865715.

[16] Rennie, J. D. M.; Shih, L.; Teevan, J.; et al.: Tackling the Poor Assumptions of Naive Bayes Text Classifiers. In *Proceedings of the Twentieth International Conference on International Conference on Machine Learning*. ICML'03. AAAI Press. 2003. ISBN 1-57735-189-4. pp. 616–623.
Retrieved from: http://dl.acm.org/citation.cfm?id=3041838.3041916

[17] Savoy, J.: Working with Text. Tools, Techniques and Approaches for Text Mining. Emme L. Tonkin & Gregory J.L. Tourte. Chandos Publisher, Cambridge (MA). 2016. 330pp. (ISBN 978-1-84334-749-1). *JASIST*. vol. 69. 2018.

[18] Swamynathan, M.: *Mastering Machine Learning with Python in Six Steps: A Practical Implementation Guide to Predictive Data Analytics Using Python*. Apress. first edition. 2017. ISBN 9781484228661.

[19] Trieloff, L.: Three Principles of API First Design. https://medium.com/adobetech/three-principles-of-api-first-design-fa6666d9f694. 2017. accessed: 2019-04-30.

[20] Wikipedia, the free encyclopedia: Support Vector Machines. 2010. accessed: 2019-04-30.
Retrieved from: https://de.wikipedia.org/wiki/Support_Vector_Machine

[21] Yegulalp, S.: What is Docker? Docker containers explained. *InfoWorld.com*. Sep 06 2018. copyright - Copyright Infoworld Media Group Sep 6, 2018; Last updated - 2018-09-07.
Retrieved from:
https://search.proquest.com/docview/2100097722?accountid=17115

Appendix A

Screenshots

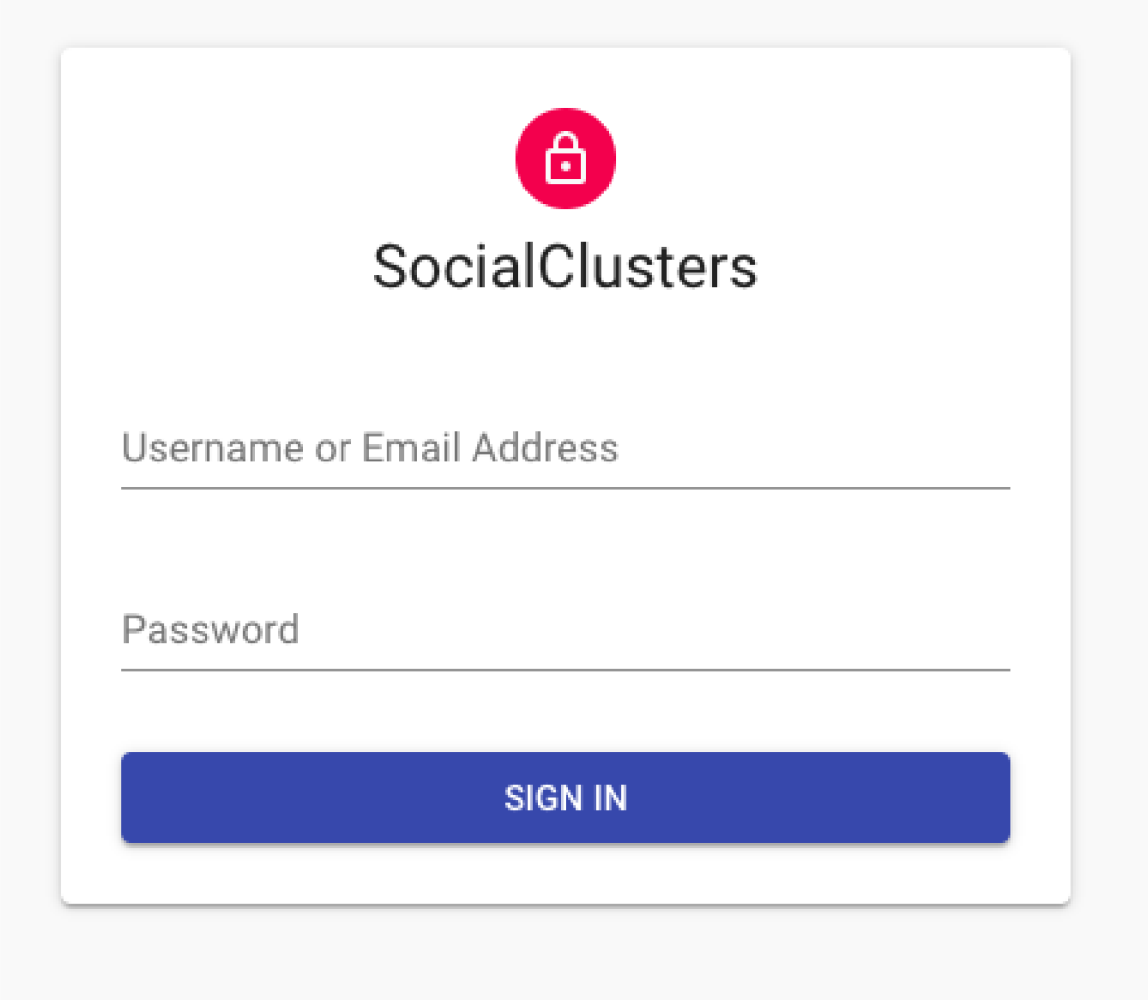

Figure A.1: **Login**

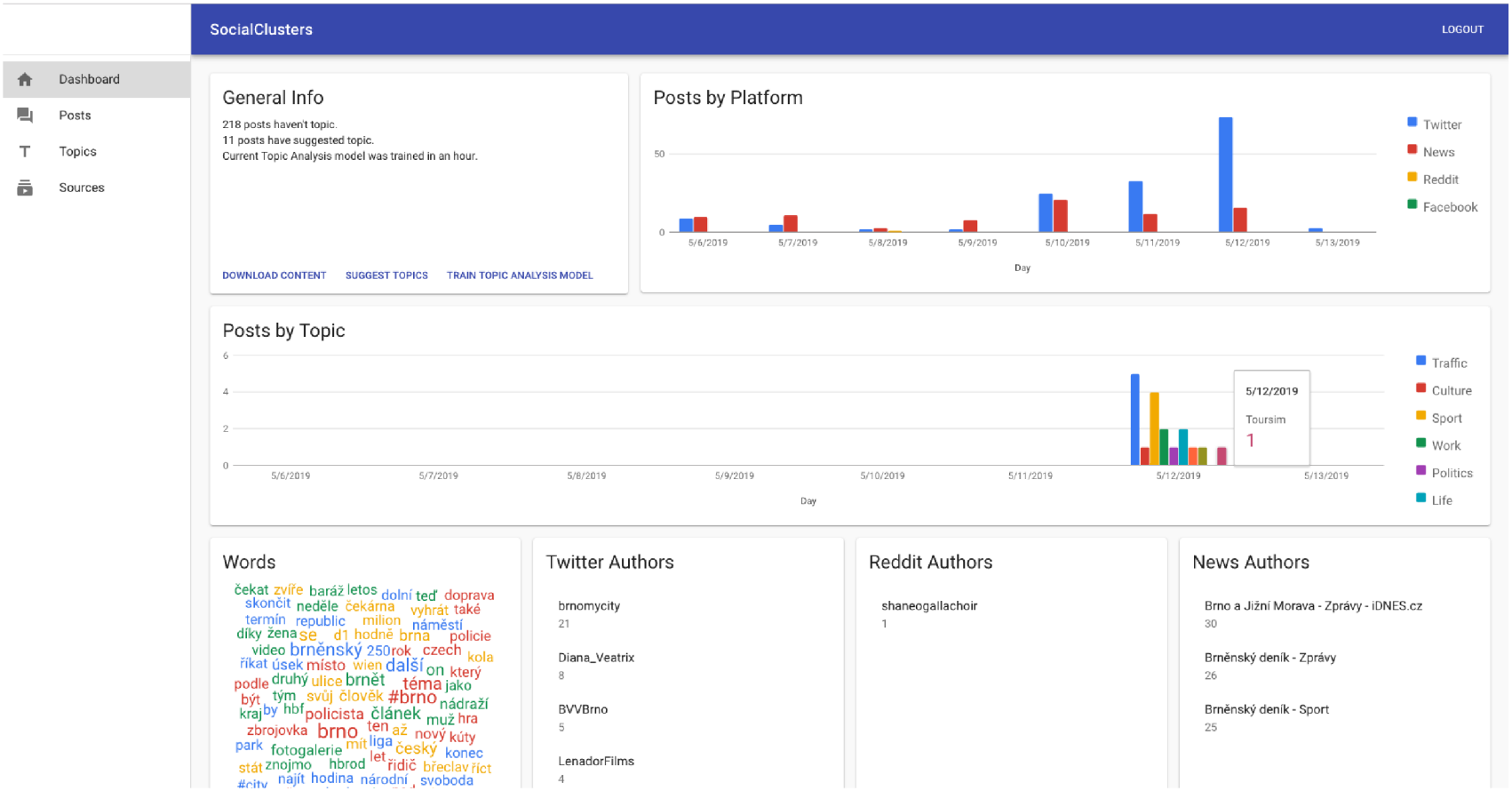

Figure A.2: Dashboard

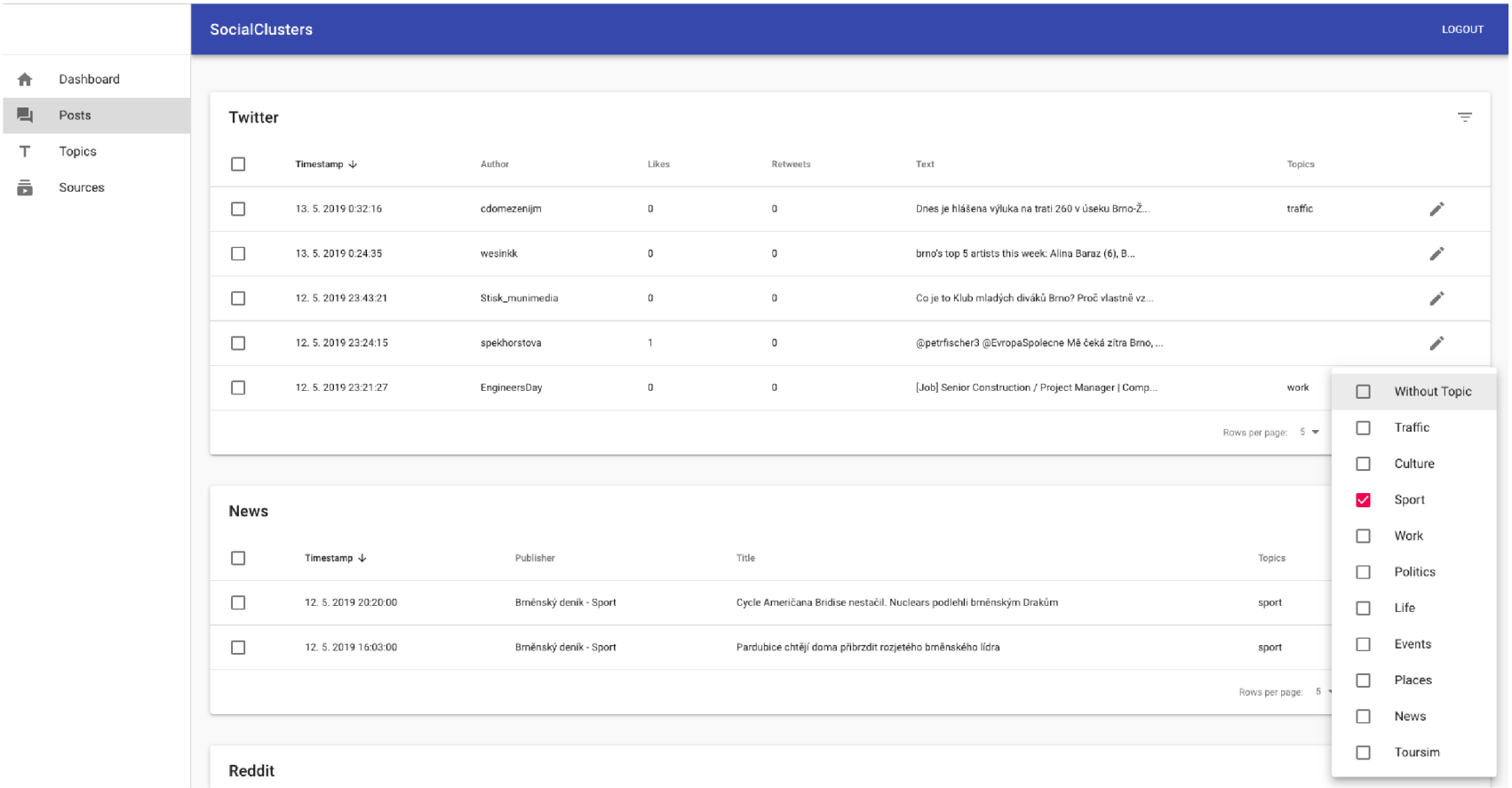

Figure A.3: **Posts**

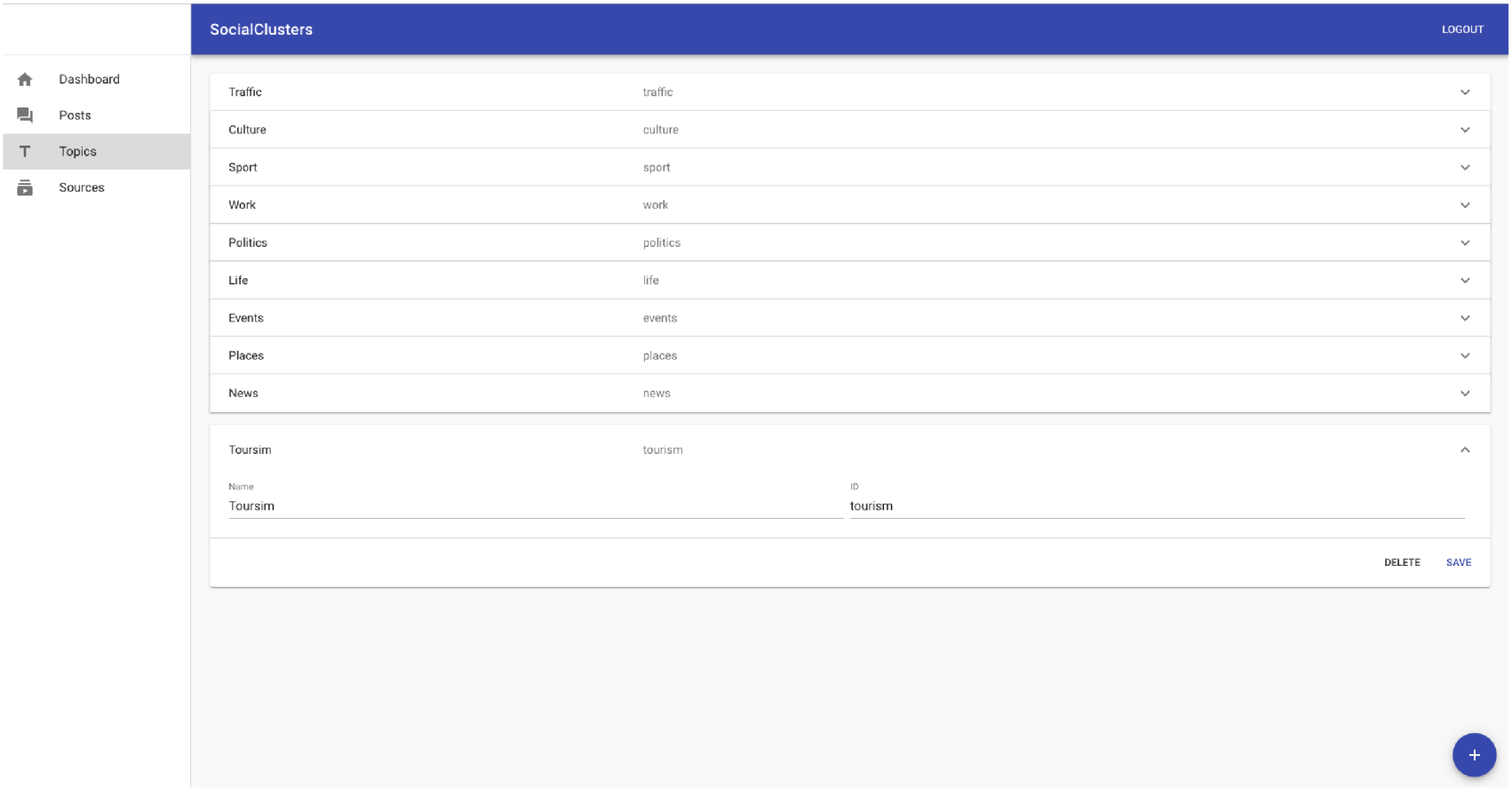

Figure A.4: **Topics**

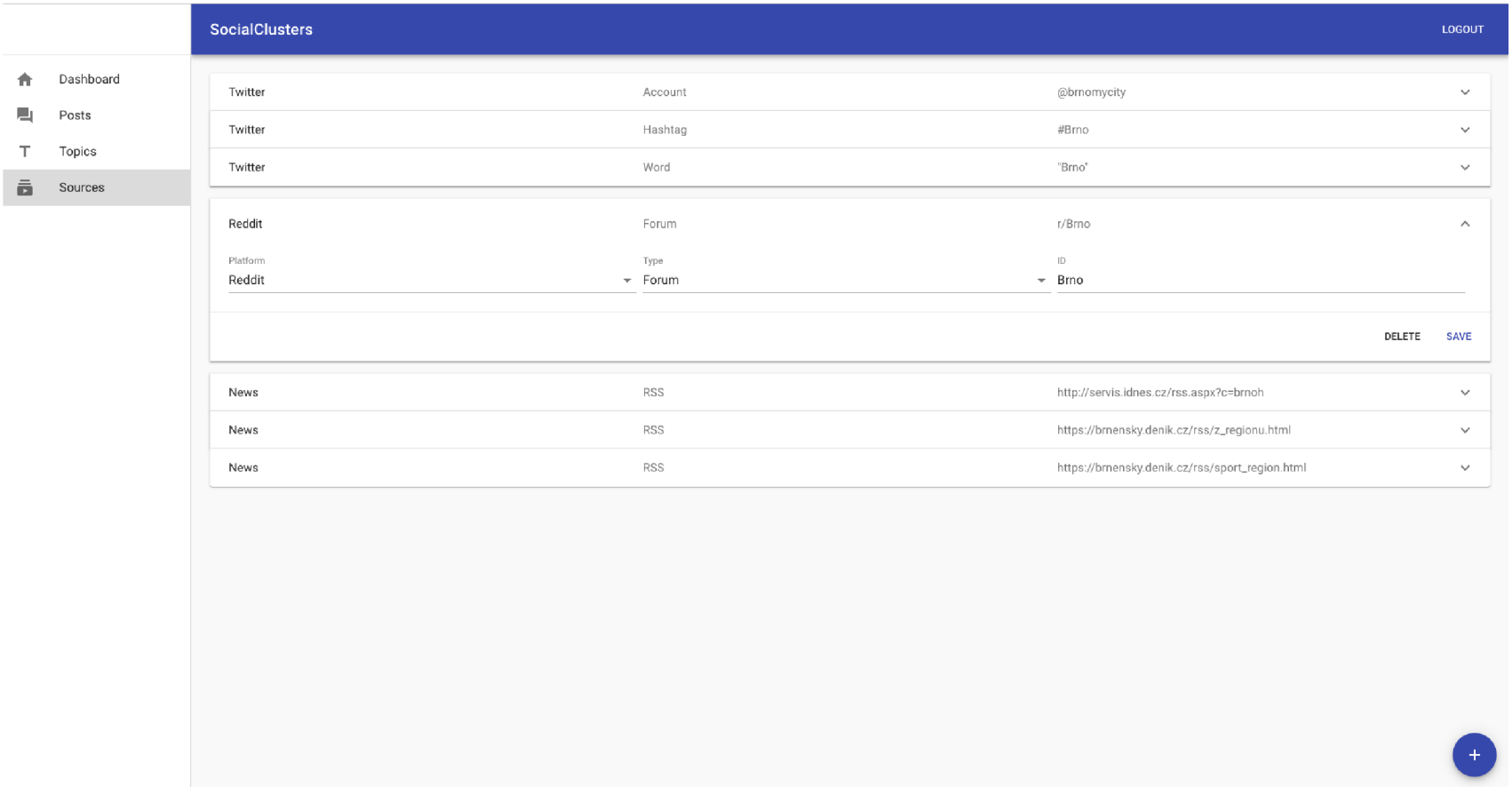

Figure A.5: **Sources**