



Norwegian University of  
Science and Technology

# Improving recommender systems with machine learning and social media

**Petter Stenslie Ekern**

Master of Science in Informatics

Submission date: May 2017

Supervisor: Herindrasana Ramampiaro, IDI

Norwegian University of Science and Technology  
Department of Computer Science



---

# Abstract

This thesis studies the opportunity to utilize posts from social media in recommender systems. Recommender systems are used to help users find content they are interested in when there are many alternatives, and is very common in streaming services. One of the problems of recommender systems is called the cold-start problem, where the system does not have enough information about an item or a user to make accurate recommendations.

We want to mitigate this problem by gathering information from Twitter. Twitter is an extensive source of information about real-time events, but because of considerable noise it can be difficult to find the information relevant for a recommender system. In this thesis, we present a system capable of gathering and organizing information from Twitter in a way that makes it usable for a movie recommender system. This is done by filtering tweets about new movies through a classifier based on an artificial neural network that sorts movie-related tweets from non-movie-related tweets. After finding relevant tweets, our system can tell movie titles apart by clustering the data in such a way that clusters are assigned a movie title and a sentiment.

The solution gathers data from an unstructured source and organizes the data in such a way that makes it usable by a recommender system. The solution is evaluated against previous research concerning both clustering, sentiment analysis and classification. The results indicate that social media and Twitter in particular is a useful, extensive source than can be used to mitigate the cold-start problem of recommender systems.

---

---

# Sammendrag

I denne oppgaven studerer vi muligheten for å utnytte innlegg fra sosiale medier i anbefalingssystemer. Anbefalingssystemer brukes for å hjelpe brukere finne ting de er interessert i når utvalget blir stort, og er veldig vanlig i blant annet strømmingstjenester. Et av problemene til anbefalingssystemer kalles kaldstart problemet, der systemet ikke har nok informasjon om en gjenstand eller bruker til å kunne gjøre nøyaktige anbefalinger.

Dette problemet vil vi forminske ved å samle informasjon fra Twitter. Twitter er en stor kilde til informasjon om samtidshendelser, men på grunn av mye støy kan det være vanskelig å finne den informasjonen som er relevant for et anbefalingssystem. I denne oppgaven presenterer vi et system som er i stand til å hente informasjon fra Twitter og deretter organisere denne informasjonen på en måte som gjør den nyttig for et anbefalingssystem for film. Dette gjøres ved å samle tweets om nye filmer gjennom en klassifiserer basert på et kunstig nevralt nett som skiller ut tweets som diskuterer filmer fra tweets som ikke diskuterer filmer. Etter å ha funnet relevante tweets klarer vårt system å skille mellom forskjellige filmtitler ved å klynge dataene, der klyngene blir tilegnet en filmtittel og en sentimentverdi.

Løsningen samler data fra en ustrukturert kilde og organiserer dataen på en måte som gjør den anvendelig for anbefalingssystemer. Løsningen blir vurdert opp mot tidligere forskning innen både klynging, sentimentanalyse og klassifisering. Resultatene indikerer at sosiale medier og Twitter spesielt er en nyttig og omfattende kilde som kan brukes for å redusere kaldstartproblemet til anbefalingssystemer.

---

---

# Preface

This thesis is submitted to the Norwegian University of Science and Technology in Trondheim, as the final delivery of the course *IT3902 Informatics Postgraduate Thesis: Database Management and Search* and is the final submission during the Master of Science: Informatics program. The work presented in this thesis was conducted from August of 2016 to May of 2017. The supervisor for this thesis is Heri Ramampiaro.

---



---

# Acknowledgements

I would like to thank my supervisor, Associate Professor Heri Ramampiaro at the Department of Computer and Information Science for his advice and feedback during the course of this project. I would also like to thank my friend Torstein for providing feedback and support whenever I needed it, and my parents for giving me motivation and encouragement throughout my education.

---

# Contents

<b>Abstract</b>	<b>i</b>
<b>Sammendrag</b>	<b>iii</b>
<b>Preface</b>	<b>v</b>
<b>Acknowledgements</b>	<b>vii</b>
<b>List of Figures</b>	<b>xiii</b>
<b>List of Tables</b>	<b>xv</b>
<b>1 Introduction</b>	<b>1</b>
1.1 Background . . . . .	1
1.1.1 Recommender systems and their growing importance . . . . .	1
1.1.2 Online social media . . . . .	2
1.2 Todays general problem . . . . .	2
1.3 Motivation . . . . .	2
1.4 Suggested solution . . . . .	3
1.5 Goals and research questions . . . . .	3
1.6 Thesis outline . . . . .	4
<b>2 Theoretical background</b>	<b>5</b>
2.1 Twitter . . . . .	5
2.1.1 Noise . . . . .	5
2.1.2 Sparsity . . . . .	6
2.1.3 Retweets . . . . .	6
2.2 Recommender systems . . . . .	6
2.2.1 Content-based filtering . . . . .	7
2.2.2 Collaborative filtering . . . . .	8
2.2.3 Hybrid systems . . . . .	9
2.2.4 The cold-start problem . . . . .	10
2.2.5 Evaluating recommender systems . . . . .	10

# CONTENTS

---

2.3	Natural language processing . . . . .	11
2.3.1	Sentiment analysis . . . . .	11
2.3.2	Information retrieval . . . . .	12
2.3.3	Text analysis . . . . .	12
2.4	Machine learning . . . . .	14
2.4.1	Supervised learning . . . . .	14
2.4.2	Unsupervised learning . . . . .	16
2.5	Summary of background . . . . .	16
2.5.1	Twitter . . . . .	16
2.5.2	Recommender systems . . . . .	17
2.5.3	Natural language processing . . . . .	17
2.5.4	Machine learning . . . . .	17
<b>3</b>	<b>Related work</b>	<b>19</b>
3.1	Solving the cold-start problem using social media . . . . .	19
3.2	Twitter topic classification and text representation . . . . .	20
3.2.1	Text representation . . . . .	21
3.3	Clustering of tweets and movies . . . . .	21
3.4	Movies and social media . . . . .	22
3.5	Sentiment analysis . . . . .	22
3.6	Conclusion of related work . . . . .	23
<b>4</b>	<b>Proposed approach</b>	<b>25</b>
4.1	Overview . . . . .	25
4.2	Finding movie tweets . . . . .	28
4.2.1	Gathering tweets . . . . .	28
4.2.2	Representing tweets . . . . .	29
4.2.3	Pre-processing . . . . .	30
4.2.4	Classification . . . . .	31
4.2.5	Training the classifier . . . . .	32
4.2.6	Methods of evaluation . . . . .	33
4.3	Clustering movies . . . . .	34
4.3.1	Representing data . . . . .	34
4.3.2	Clustering method . . . . .	34
4.3.3	Methods of evaluation . . . . .	36
4.3.4	Summarizing clusters and finding movies . . . . .	39
4.4	Sentiment analysis . . . . .	39
4.4.1	Sentiment analysis on individual tweets . . . . .	40
4.4.2	Sentiment analysis on clusters . . . . .	41
4.4.3	Methods of evaluation . . . . .	42
4.5	Challenges . . . . .	42
4.5.1	The problem . . . . .	43

---

4.5.2	The solution . . . . .	43
4.5.3	Clustering after two step classification . . . . .	43
4.5.4	Assigning sentiment scores after two-step classification . . . . .	44
4.6	Final system . . . . .	44
4.6.1	Gathering and classifying movie tweets . . . . .	44
4.6.2	Two-step classification . . . . .	46
4.6.3	Clustering . . . . .	46
4.6.4	Sentiment analysis . . . . .	48
4.6.5	Final note . . . . .	48
<b>5</b>	<b>Experiments and evaluations</b>	<b>49</b>
5.1	Movie Classification . . . . .	49
5.1.1	The datasets . . . . .	49
5.1.2	Preparation . . . . .	51
5.1.3	Testing . . . . .	52
5.1.4	Evaluation . . . . .	53
5.1.5	Conclusion . . . . .	54
5.2	Clustering . . . . .	55
5.2.1	The dataset . . . . .	55
5.2.2	The Algorithm and parameters . . . . .	56
5.2.3	Testing . . . . .	56
5.2.4	Evaluation . . . . .	59
5.2.5	Conclusion . . . . .	63
5.3	Sentiment analysis . . . . .	64
5.3.1	The dataset . . . . .	64
5.3.2	Set-up . . . . .	64
5.3.3	Results . . . . .	65
5.3.4	Evaluation . . . . .	65
5.3.5	Conclusion . . . . .	66
5.4	Two-step classification . . . . .	67
5.4.1	The dataset . . . . .	67
5.4.2	Set-up . . . . .	67
5.4.3	Testing . . . . .	68
5.4.4	Evaluation . . . . .	70
5.4.5	Conclusion . . . . .	71
5.5	Evaluating the overall solution . . . . .	71
5.5.1	Final configuration . . . . .	71
5.5.2	Significance of results . . . . .	73
5.5.3	Comparing to previous studies . . . . .	73
5.5.4	Known weak points of the tests . . . . .	75

---

<b>6 Conclusion and Future Work</b>	<b>77</b>
6.1 Conclusion . . . . .	77
6.1.1 Contributions of the Thesis . . . . .	78
6.1.2 Answering research questions . . . . .	78
6.2 Future work . . . . .	79
<b>Appendices</b>	<b>81</b>
<b>A One-step classification clustering tests</b>	<b>83</b>
A.1 Cluster quality . . . . .	83
A.1.1 Threshold 0.2 . . . . .	83
A.1.2 Threshold 0.25 . . . . .	83
A.1.3 Threshold 0.3 . . . . .	84
A.2 Average F1 scores . . . . .	84
A.2.1 Average F1 of top five clusters 0.2 . . . . .	84
A.2.2 Average F1 of top five clusters 0.25 . . . . .	84
A.2.3 Average F1 of top five clusters 0.3 . . . . .	84
<b>B Two-step classification clustering tests</b>	<b>85</b>
B.1 Cluster quality . . . . .	85
B.1.1 Threshold 0.2 . . . . .	85
B.1.2 Threshold 0.25 . . . . .	85
B.1.3 Threshold 0.3 . . . . .	86
B.2 Average F1 scores . . . . .	86
B.2.1 Threshold 0.2 . . . . .	86
B.2.2 Threshold 0.25 . . . . .	87
B.2.3 Threshold 0.3 . . . . .	87
B.3 Movie title detection . . . . .	87
<b>Bibliography</b>	<b>89</b>

# List of Figures

4.1	Overview of the three main components of the approach . . .	27
4.2	Example of a tweet about a movie without mentioning "movie" words . . . . .	29
4.3	Illustration of the clustering algorithm adjusting a cluster centroid	35
4.4	Overview of system with one-step classification . . . . .	45
4.5	Overview of system with two-step classification . . . . .	47
5.1	Number of clusters created at different thresholds . . . . .	57
5.2	Number of outliers at different thresholds . . . . .	57
5.3	Titles and confidence scores given to clusters made with a threshold of 0.3 . . . . .	60
5.4	Clusters created at threshold 0.3 with sentiment assigned by vivekn API in the database . . . . .	67
5.5	Final result after clustering and assigning movie titles and sentiment to all clusters . . . . .	72
B.1	Titles and confidence scores given to clusters made with a threshold of 0.3, positive sentiment . . . . .	88
B.2	Titles and confidence scores given to clusters made with a threshold of 0.3, negative sentiment . . . . .	88

## LIST OF FIGURES

---



# List of Tables

2.1	Example of a User X Item matrix . . . . .	8
4.1	Pre-processing steps . . . . .	31
4.2	Calculation of sentiment score . . . . .	41
5.1	Datasets used for training and testing the movie classifier . . .	51
5.2	Performance scores for classification models tested on tweets with stemming . . . . .	53
5.3	Performance scores for classification models tested on tweets without stemming . . . . .	53
5.4	Accuracy measure for the chosen classifier model . . . . .	55
5.5	Dataset for clustering tests . . . . .	55
5.6	Top five clusters with threshold 0.3 . . . . .	59
5.7	Difference in clustering algorithms . . . . .	59
5.8	Sentiment tool testing . . . . .	65
5.9	Amount of clusters at different thresholds with two-step classi- fication . . . . .	68
5.10	Amount of tweets assigned to clusters . . . . .	68
5.11	Movies split into sentiment . . . . .	69
5.12	Top five clusters with threshold 0.25 with two-step classification, positive sentiment . . . . .	69
5.13	Top five clusters with threshold 0.25 with two-step classification, negative sentiment . . . . .	69
5.14	F1 score across one-step and two-step classification and different thresholds quality . . . . .	71
A.1	Top five clusters with threshold 0.2 . . . . .	83
A.2	Top five clusters with threshold 0.25 . . . . .	83
A.3	Top five clusters with threshold 0.3 . . . . .	84
B.1	Top five clusters with threshold 0.2 with two-step classification, positive sentiment . . . . .	85

## LIST OF TABLES

---

B.2	Top five clusters with threshold 0.2 with two-step classification, negative sentiment . . . . .	85
B.3	Top five clusters with threshold 0.25 with two-step classification, positive sentiment . . . . .	85
B.4	Top five clusters with threshold 0.25 with two-step classification, negative sentiment . . . . .	86
B.5	Top five clusters with threshold 0.3 with two-step classification, positive sentiment . . . . .	86
B.6	Top five clusters with threshold 0.3 with two-step classification, negative sentiment . . . . .	86

---

# Chapter 1

## Introduction

This chapter introduces the topic of this thesis. Section 1.1 presents some basic background information. Section 1.2 presents the problem we are facing today, while section 1.3 presents why it is worth solving. Finally, the suggested solution and research questions will be presented in section 1.4 and section 1.5.

### 1.1 Background

#### 1.1.1 Recommender systems and their growing importance

A recommender system is a system designed to improve the user experience. They operate by matching users of a service with items in a library to find content each user is interested in. One example of a platform where recommender systems are increasingly important, is in streaming services like Netflix <sup>1</sup> and HBO <sup>2</sup>. Because most streaming services run on a monthly subscription model, keeping the customers returning by keeping them satisfied with the product is a major concern. One way to do this is by keeping an expansive library of movies and series, to assure that every user can find something they will enjoy. A large library causes the problem of users finding it difficult to navigate all the items and struggle to actually *find* the movies and series they would enjoy. This is where recommender systems are useful. By analysing patterns, it can find out what users are interested in what content, increasing the probability of users finding content they enjoy.

---

<sup>1</sup><https://www.netflix.com>

<sup>2</sup><https://no.hbonordic.com/>

### 1.1.2 Online social media

Online social media allows people to create a network of friends and other contacts, where they talk to each other, like each other's posts or share the content of other people. Social media has seen increased usage in later years, and services like Twitter<sup>3</sup> are being used more than ever [33]. Twitter provides a service where users are able to post short messages, called *tweets*, to all their friends. Because social media is becoming easier and easier to access each year with increased usage of smartphones and convenient internet access all over the world, there is no reason to think this growth will stop any time soon.

Twitter is used to talk about many different topics, from politics to sports to entertainment. All kinds of people use Twitter, from many different layers of society, across religions, ages and genders. Twitter is for everyone and therefore gives us a broad perspective on what people are talking about.

Social media is the focus of many different fields of study, because it is a relatively new platform with growing potential and a large amount of available user data in the form of texts.

## 1.2 Today's general problem

This thesis will focus on one of the oldest problems of recommender systems and how we can utilize the huge amount of data that social media provides to solve that problem. The problem in question is known as the *cold-start problem*. While we will come back to this in section 2.2.4, the essence of the problem is this: A recommender system uses the knowledge it has about users and items and the interactions between them, such as reviews, to fuel its recommendations. In a dynamic recommender system, meaning a system that will receive new items or new users during its lifetime, the system cannot make accurate recommendations about new users and items because it knows nothing about them. This thesis aims to mitigate this problem using social media. Although the cold-start problem affects most recommender systems, this thesis will focus on how we can mitigate this problem for systems that recommend movies.

## 1.3 Motivation

This thesis focuses on the use of external sources to mitigate the cold-start problem in recommender systems. So-called *microblogging word of mouth*, or the spreading of information through microblogs such as tweets, have a great effect on how the audience perceives a movie [13], so it is natural to

---

<sup>3</sup><https://twitter.com/>

try to include this information in a movie recommendation system. Because recommender systems will only be more important in the future, and there is a large source of mostly unused but highly relevant data available in online social media, it will be the focus of this thesis to try to mitigate the cold-start problem using this untapped source of data. This will also allow us to study how effectively in general external sources can be utilized to improve recommender systems.

## 1.4 Suggested solution

We want to use Twitter in order to help recommender systems mitigate the cold-start problem. The reason Twitter was determined to be the most suitable media for this task is that the cold-start problem relates to completely new users and items. Since this thesis will focus on movie recommendations, new items in this case means new movies, and movies that have been recently released are often discussed on Twitter. We want to start by gathering tweets from the Twitter API. Then, we will classify the tweets into two categories: tweets about movies and tweets not about movies. After we have categorized the tweets, we need to find out what movies they discuss. We will do this by clustering the tweets, under the assumption that tweets that discuss the same movie would be clustered together. The system will then assign a movie title to each cluster, as this will further improve a recommender system by allowing it to gather additional information from external sources, as well as make it easier to aggregate data about the same movie. Finally, we will do sentiment analysis on each cluster. This will give us a good idea of whether or not a movie is being well received by the public or not. This information is then usable by a recommender system.

## 1.5 Goals and research questions

In this thesis, we study how we can use online social media to mitigate the cold-start problem. We can measure this in several ways, and divide the task into smaller subtasks. In order to use data from social media, the solution needs to sort out the relevant data from the irrelevant data. The solution must then be able to cluster the data based on titles and determine the title of a movie to allow for easier aggregation of information. After relevant information has been gathered and sorted, the solution needs to extract sentiment about the different items.

We have split one main research question into two smaller subtasks to depict the challenges of this thesis.

- **RQ1:** *How can social media be used to mitigate the cold-start problem?*  
This is the underlying question this thesis will attempt to answer. This task has been divided into two smaller subtasks presented in research questions below. The main goal of this thesis is to study how effectively an external source such as Twitter can be used as a source to base recommendations on.
- **RQ1.1:** *How effectively can posts that relate to a recommender systems domain be filtered out on social media?*  
This question focuses on how the system can gather information that relates to the topic of the recommender system. To exemplify this, the movie domain will be used for this thesis, but the results should be applicable to other similar domains such as tv-shows, books or music.
- **RQ1.2:** *How can sentiment about specific items be extracted through these posts?*  
This question covers two distinct issues that need a solution before the information found in RQ1.1 can be used. For a recommender system to be able to use the information gathered, the system has to be able to determine the name(s) of the item(s) being discussed in the social media posts, as well as being able to extract sentiment about each item from the posts.

## 1.6 Thesis outline

We began in chapter 1 by introducing the topic. In chapter 2 an introduction to relevant background theories is presented. Chapter 3 will give an overview of relevant previous studies. Chapter 4 will present the details of the proposed solution for the problem. Chapter 5 presents, conducts and discusses all the tests and experiments for the different parts of the proposed solution. Chapter 6 concludes the work, summarizes the contributions of the thesis, and answers research questions.

---

# Chapter 2

## Theoretical background

This chapter will cover the background theories that relates to the problem and solution of this thesis. Section 2.1 presents Twitter and the problems associated with tweets. Theory on recommender systems is presented in section 2.2. In section 2.3, information regarding natural language processing is presented. In section 2.4, we present the concept of machine learning and how it relates to this thesis.

### 2.1 Twitter

Twitter is an online social media where users can post *microblogs* called *tweets*. They are called microblogs because they are limited in length. A tweet can typically be 140 characters long, excluding links to multimedia like pictures and videos. According to Twitter itself [35], the biggest usage and the main purpose of Twitter is to let people easily connect with and follow each other, allowing for quick and easy sharing of topical events, opinions and discussions. It is often used to share topics and events that are happening in real-time. For this reason, Twitter has great potential to be used for tackling the cold-start problem of recommender systems, because any newly introduced items have a high chance of being discussed there.

#### 2.1.1 Noise

Noise [36] is information stored in documents that is not necessarily related to the text, or otherwise creates irregularities in patterns in the text. The language in texts on Twitter is very informal, without any rules as to how users communicate with each other and no set syntax for tweets, which often leads to automated systems having more problems understanding the text and the meaning behind it. For example, misspellings like "movieeeee" is not unusual to see on Twitter. Informal spelling and a more "casual" atmosphere

is a characteristic of most social media. But words like these make it harder for information retrieval systems to accurately find relevant information and understand the text [34]. Some other sources of noise can be "emojis" or "smileys", which can be a series of symbols meant to express some kind of emotion. These can also be hard to interpret for information retrieval systems, especially if they are made using special characters.

### 2.1.2 Sparsity

A tweet is limited to be 140 characters long, which is a very short text to extract information from. From these 140 characters, a recommender system would need information about several different attributes, for example the name of the item being discussed and the sentiment portrayed about the item. This gathering of information is made harder by the lack of structure in tweets, meaning that not all tweets will have this information in them.

### 2.1.3 Retweets

One of the features of Twitter is *retweeting*. This means that a user copies a tweet from someone else, and spreads it out to all of his or her followers, increasing the number of people that tweet reaches dramatically. This results in the same tweet being sent several times from several different people, all pointing to the original tweet, only with the added "RT" tag which marks it as a retweet. Sometimes people retweet tweets they agree with, other times they retweet tweets they think their own followers should see.

## 2.2 Recommender systems

A recommender system [29] is a tool to help people find the content they want when they are overwhelmed with choices. They come in many different forms and recommend many different things, for example movies or music. When discussing recommender systems, it is useful to define a few key terms:

- *Item* - An object the recommender system knows about and wants to recommend to users.
- *User* - A user of the system the recommender system can recommend items to.
- *Domain* - What sort of setting the recommender system operates in. For example, is it a system for recommending movies or cars?



This article [20] presents a list of reasons why companies in many different businesses and areas choose to implement a recommender system in their business model. Some of the reasons given are:

- Increase sales by selling more items. This is achieved by showing the user more items the user is more likely to buy, and therefore more likely to spend money on. This could encourage more producers to sell their items on your platform.
- Sell a larger variety of items. Services like Netflix, HBO and other streaming services have nothing to gain from having a series or movie on their service that nobody watches, but they do not want to recommend these items to users with no interest in them. Therefore, the service needs an accurate recommender system that can find the right users to promote these items to. If they can sell more obscure and lesser-known items, it could promote a healthy growth to the library of items.
- Increase user satisfaction. Services like Netflix and HBO operate with a monthly subscription platform, meaning that users usually pay for one month at a time. Because of this, it is extremely important for services like these that they keep their consumers and user base happy and satisfied with the product they are receiving. This can be done by giving them a steady supply of items they enjoy.

Recommender systems are usually divided into two groups depending on how they generate their recommendations. They are called content-based recommender system and collaborative filtering recommender systems. There is also a third option, a hybrid approach, which utilizes techniques from both content-based filtering and collaborative filtering.

### 2.2.1 Content-based filtering

In content-based filtering systems, the attributes of the items in the system is used to make recommendations [29]. The word "content" in content-based filtering refers to the attributes the items can have. For example, when recommending movies, one simple approach can be to recommend movies in the same genre as movies the user has previously rated highly. This can be done for other attributes, like similar actors, the same director and so on. The benefits of content-based filtering is that they are independent of reviews from other users. As long as the user we are recommending to have rated some items and the system has information about the other items in its library, the system can look for items similar to the highly rated item. Because of this, users with very "simple taste" can often be given very accurate recommendations because the system can look for similar items to the ones it knows

they like. There are, of course, several disadvantages to this approach as well. For one, this method is not very good at detecting patterns that are more advanced. For example, if the user has rated only action movies with certain actors involved, there is no reason for the system to believe the user would like anything else. This is of course not necessarily true, as many people can enjoy movies from many different genres with many different actors. Systems that utilize content-based filtering are especially susceptible to cold items, or items they know nothing about, because most of their recommendations are based on general sentiment of items or the item attributes.

### 2.2.2 Collaborative filtering

Another type of recommender system uses a method called collaborative filtering. This method often performs better than content-based filtering methods if its given enough information [24]. Collaborative filtering systems are usually divided into user-based systems and item-based systems [1]. Collaborative filtering user-based systems utilize user ratings on items provided by multiple users, in order to predict the taste of the user it recommends to. The problem with this approach is the data sparsity it creates. Because most users will not have rated most items, most of the entries will be missing. The point made in [29] is that these missing values are not necessarily that important, and that the ratings the system knows about can be all it needs. The idea of collaborative filtering is that two users with similar ratings will have similar taste in items. So in the movie domain, the example in table 2.1 can be used: Consider two users, Alice and Bob. The system wants to recommend a movie to Bob. Alice has seen and rated "Die Hard", "Inception" and "Rush". Bob has only seen "Die Hard" and "Rush". Both Alice and Bob gave these two movies 4 out of 5 stars. The system will then assume that Alice and Bob have similar taste in movies, and will look for movies that Alice has rated highly that Bob has not rated, the assumption here being that Bob would like movies that Alice likes. It will discover that Alice rated "Inception" 5 out of 5 stars and assumes Bob would do the same, and recommends the movie to Bob.

User/Movie	Die Hard	Rush	Inception
Alice	4	4	5
Bob	4	4	?

Table 2.1: Example of a User X Item matrix

When Bob is done watching the movie he may rate it, and depending on his rating the system can adjust recommendations in the future if it was wrong. Because of this, the system can work well for users with special taste, because the system can find other users who are similar to the special user. It

can also recommend a big library of movies, because as long as someone with ratings similar to yours have seen a movie, it can determine whether or not to recommend that movie to you. Item-based collaborative filtering works in a similar way, but instead of finding similar users it searches for similar items. This is not the same way that content-based filtering systems search for similar items though. While content-based filtering bases similarity between items on the attributes or content of the items, collaborative filtering uses the ratings the items have received to look for similar items. If enough data is available, item-based collaborative filtering is one of the most powerful techniques for recommender systems [24] [31].

Besides the sparsity problem discussed earlier, these systems also suffer from having no way to evaluate movies with no recommendations, meaning that lesser known movies has a smaller chance of ever being recommended to anyone. In this particular area, content-based filtering works better. Collaborative filtering systems work best when there is sufficient ratings to work with, so unknown movies or brand new movies will be hard to recommend. This relates to the cold-start problem and the topic of this thesis.

### 2.2.3 Hybrid systems

It is clear that both content-based filtering and collaborative filtering systems have several different advantages and disadvantages. Hybrid recommender systems are systems that aim to take advantages from both systems in order to cover as many weaknesses as possible. A hybrid system is able to take advantage from both the representation of the content from content-based filtering and the similarities among users from collaborative filtering.

Hybrid systems can compute the correlation between users and item attribute weights. They evaluate the user's score on different item attributes instead of the user's score on the items. This means that all users will have more reviews and numbers to work with. For example, if a user rates the movie "Inception" positively, in addition to update the movie review for that movie for that user, the system can update the user's attribute weights. These can be genre attributes, actor attributes or director attributes. We could also have other tags created by the system, but the main point is that every review updates a series of weights, not just the item-user relation. Because of this, the problem where users have not rated enough movies that we know from collaborative filtering methods becomes mitigated, while we still achieve the deeper and more thoughtful recommendation method from collaborative filtering rather than straight attribute based recommendations from content-based filtering. The cold-start problem still applies to hybrid systems, even though they might mitigate it somewhat.

### 2.2.4 The cold-start problem

The cold-start problem [29] is one of the oldest problems in recommender systems. It describes the problem a recommender system encounters whenever a new user or a new item is introduced to the system. It affects recommender systems in all sorts of domains and both collaborative filtering systems, content-based filtering systems and hybrid approaches. The essence of the problem is that whenever a new item or a new user is introduced to the system, the system knows nothing about it. It is called cold-start because the word "cold" is used in recommender systems about anything that is new or unknown. When an item or user is in its cold-start phase it is difficult for the system to make recommendation regarding that item or user. There are three types [24] of cold-start situations:

- *Cold user* - Whenever a new user is registered in the system.
- *Cold item* - Whenever a new item is added to the system.
- *Cold system* - The system is brand new and knows nothing.

For example, consider a movie recommender system. When someone creates a new user, the recommender system cannot know what that user might like. If a system utilizes content-based filtering methods, it needs to know what types of movies the user has previously liked, which it cannot know. If we use a collaborative filtering method, the recommender system needs to find users with similar taste, which is impossible since it does not know what the user likes. This is an example of a cold-start user problem. As for a cold-start item problem, say that a new movie is released and the recommender system wants to add it to its library. The system have no way of knowing who to recommend this item to, because no one has rated it. In this case however, content-based filtering methods work better than collaborative filtering methods, because content-based filtering methods does not need anyone to have rated the movie, it just need to know the movie's content attributes. As long as the recommender system knows some attributes of the movie, like genre, director and actor and so on, a content-based filtering recommender system will be able to do recommendations with it. If it also knows the general sentiment about the movie, it can further adjust movies with similar attributes. As such, collaborative filtering methods are more vulnerable to cold-start problems, but out-performs content-based filtering methods if there is sufficient data available [24].

### 2.2.5 Evaluating recommender systems

There are several different measures used to evaluate recommender systems. For example, one way to evaluate performance is how well the recommender

system can recommend brand new items to the user. This could be the case when users are only looking for brand new items. Alternatively, the system could be measured on how it affected the revenue of the business where it was deployed.

## 2.3 Natural language processing

Natural language processing covers a large spectrum of topics. Subjects like word-counting and advanced systems able to respond to somewhat complex questions are all part of natural language processing [16]. A common factor in all these topics is that they deal with the interaction between computers and human, or natural, languages. This has proven to be quite a challenge, because there are so many rules and so many exceptions that make sense for humans but not necessarily for machines. For this thesis, natural language processing is relevant for understanding tweets, giving us the added challenge of understanding text that is often misspelled or with a very informal language. The natural language processing topics covered in this thesis are: Sentiment analysis, information retrieval and text analysis.

### 2.3.1 Sentiment analysis

Sentiment analysis deals with the problem of trying to extract sentiment from written text [19]. It is a highly active field of study, with new studies and methods constantly being produced. It is a field where there are still big problems to overcome, but it has seen improvements in later years. One of the reasons for the surge of research on sentiment analysis is that only after the internet and social media became a widespread phenomenon have there become a large amount of data to easily perform sentiment analysis on. One of the biggest sources of data to perform sentiment analysis on, are social media like Facebook and Twitter.

These sources are interesting because they represent what is called *word of mouth* [7], which is the passing of information through communication. It relates to how people often learn of products and their quality through talking to other people who have experience with these products. With the increased usage of social media in recent years, this kind of communication has become much more widespread, as anyone can have their opinions heard or read by everyone. This can happen through official reviews of products, or simply stating an opinion about a product in an unofficial tweet.

### 2.3.2 Information retrieval

Information retrieval is a field in computer science focused on providing people easy access to information they need [28]. It includes topics like storage and representation of information, and ways to easily search and access information stored in for example blogs, microblogs or other web pages. Information retrieval systems can be used to search documents for specific keywords, or to rank documents by order of how relevant they are likely to be for the user.

#### Precision, recall and F1 score

Because information retrieval is about *retrieval* of information, there needs to be a way to measure how well a system retrieves relevant information. To measure this, two measures, named *precision* and *recall* [28] are often used. These measures are defined as follows:

- **Precision** : The fraction of the retrieved documents that are relevant.
- **Recall** : The fraction of relevant documents which has been retrieved.

Precision and recall assumes that all documents in the collection have been examined.

A common way to combine these two measures is called the *F1* measure. This score is a combination of both recall and precision, and all three formulas are decried below.

$$precision = \frac{|relevantDocuments| \cap |retrievedDocuments|}{|retrievedDocuments|} \quad (2.1)$$

$$recall = \frac{|relevantDocuments| \cap |retrievedDocuments|}{|relevantDocuments|} \quad (2.2)$$

$$F1score = 2 \cdot \frac{precision \cdot recall}{precision + recall} \quad (2.3)$$

### 2.3.3 Text analysis

Text analysis relates to how a text is treated within a document and how a computer can try to make sense of it. In this section, we will discuss several techniques relating to text analysis and text manipulation and how they all contribute to making text easier to understand and how they make it easier to get to the relevant information.

### Text representation

When a computer needs to process and compare text, it is often useful to find another way to represent the data rather than using pure string text. Neural networks, which will be relevant for this thesis, need to get a representation of the text using vectors, which is an array of numeric values based on some sort of weight assigning method. One of these methods is called word embedding.

Word embedding [17][6] is used to capture information about words, by assigning real values to each words, giving each word a vector. It is a form of machine learning which utilizes unsupervised learning. By assigning a vector to each word, comparing words becomes easier, by looking at the distance between the two vectors. The main idea behind word embedding was presented by John Firth in 1957 [10]:

”You shall know a word by the company it keeps.”

He argues that words with similar meanings would be used in similar context. When used together with word embedding, this means that similar words can be found by looking at other words with the best matching vector, and therefore the shortest distance. The main use for this when it comes to topic classification is that a classifier can be trained on a relevant dataset like tweets, and find what words are most similar to the topic we are interested in within that medium. Text representation by vectors also become relevant when clustering data points, where there is a need to compare the similarity between two vectors.

### Tokenization

Tokenization refers to the act of replacing different elements in the text with appropriate tokens, or replacement strings. This technique is useful for several reasons, for example decreasing the noise in the text or removing irrelevant information. One example of noise that could cause trouble when attempted to convert into a word-embedding model are links and usernames. These cannot be appropriately translated into vectors because they do not appear at a high enough frequency for the model to be able to create accurate vector representations of them. These are then replaced by appropriate tokens, which the word embedder can understand. These tokens can be text in the form of replacement words, but can also be empty strings.

### Stopwords

Words that occur too frequently are close to useless in terms of deciding the topic of the text. Words that occur frequently across different kinds of texts and topics are called *stopwords* [28] and it is common practise to ignore these

words when doing a word-count or creating a representative vector for the text. It also allows us to increase the speed at which a collection of documents can be processed, because irrelevant parts are removed or ignored. Some examples of common stopwords are: "The", "is", "a", "are".

### **Stemming**

Stemming comes from the word *stem*, which is the portion of the word that is left after the affixes are removed. Stemming is a technique used to make sure that all words are written the same way and with the same affix, by reducing all words to its stem. For example, "watched", "watches" and "watching" all becomes "watch" when stemmed. This is important, as it decreases the number of variables in the word embedding, which in turn gives less noise and a bigger chance of accurate text representation.

## **2.4 Machine learning**

Machine learning concerns a field of computer science that focuses on finding ways for computers to learn how to achieve something without being specifically programmed to do so. In this thesis, machine learning is used for different tasks like classification and clustering. In this section, we cover the different challenges and applications of machine learning that apply to our proposed solution.

### **2.4.1 Supervised learning**

Supervised learning refers to a group of machine learning approaches that requires more guidance than other approaches. Supervised learning requires the user to teach the system what to do. For example, in this thesis, machine learning will be used for classification of tweets into movie tweets and non-movie tweets. This in turn means that the system needs to be taught how to recognize these tweets. This is done through *training* the system into recognizing relevant tweets.

### **Classification**

Classification is a field where deploying machine learning methods often yield better results with better accuracy than other static methods. The classification problem can be defined as trying to assign a class to an object. For this thesis, we will be focusing on text classification.

Text classifications has to do with how the topic of a text can be determined. It is often referred to as topic classification. Document classes can



be used to describe the topic of the document, the quality of the content, the genre of the text and much more. For this thesis, we focus on binary classification. A binary classification problem is a problem where there are only two possible classes, and is as a result usually easier to classify than classification problems with multiple classes.

Classification can be done through both supervised and unsupervised learning methods. For this thesis, supervised learning is used to train the classifier, because we know how many classes there should be in our dataset.

### **Artificial neural networks**

An artificial neural networks [38], abbreviated to ANN, is in essence a system comprised of any number of simple, connected processing elements called neurons. These neurons are placed in layers, and the neurons job is to process input and provide an output. The underlying workings and mathematics of an ANN is quite complex, but the overarching idea and principals that power an ANN are simple. ANNs have three types of layers, the input layer, the hidden layers and the output layer, and each layer consists of several neurons that can be adjusted. When the system has sufficient time and data to adjust each neurons weighting and sufficient layers and neurons in each layer, it can achieve high accuracy in classification. ANNs can be utilized on data that may have very obscure patterns in them, or where processing the data would take too long or give too inaccurate results with other techniques.

### **Training and testing**

Training and testing are vital parts of the machine learning process. Training [32] is the process where a selected batch of input data is processed by the system in order to teach it how to recognize the data the system needs to identify. During training, the system adjusts its weights in order to improve upon its output. When training a system, the training data must consist of an example of an object to classify and the the correct classification of the object. For example, to train a classifier to determine if a text is representing positive or negative sentiment, a training set with predetermined sentiment must be given to the system so that it can learn from it.

When evaluating a machine learning system, testing data is implemented. Testing data is similar to training data, and is meant to be the test to determine how accurate the system is. It is important that the testing data and training data is different, to avoid testing on the same data that adjusted the weights, which could give skewed results.

## 2.4.2 Unsupervised learning

In contrary to supervised learning, unsupervised learning does not need to be taught to recognize any sort of data. This can give less accurate results, but does not require the user to go through the time-consuming process of teaching the system what to look for. Because it is never taught anything about the data, unsupervised learning works well when we are unsure how many classes there are in the data. Unsupervised learning is used for a number of different approaches, one of which is very relevant for this thesis, namely clustering. Clustering works well with unsupervised learning, because it can then determine for itself how many classes, or clusters, there are in the dataset.

### Clustering

When we talk about clustering, we mean the act of placing data points that are the most similar to each other in the same group and separating them from other groups with data points that are less similar. Clusters can be used to easily separate the data into different topics and can make it easier to spot patterns and classes. There are several different methods and algorithms that can be used to cluster data, and they are suitable for different kinds of data. We can choose what attributes we want to cluster the data around. For example, when clustering tweets, an algorithm can cluster based on both the text in the tweet, the hashtag or the user who tweeted it. Clustering is not only used for text, but can also be used to cluster data points in a recommender system matrix. A recommender system can for example cluster together all movies with a high score based on their genre.

## 2.5 Summary of background

This chapter has introduced related background about Twitter, recommender systems, natural language processing and machine learning.

### 2.5.1 Twitter

Data from Twitter in the form of tweets has several inherent problems, the most prominent problems for this thesis is the noise and the sparsity in tweets. The sparsity is a problem because the system needs to determine several variables from a relatively short text, such as topic, name of the item and sentiment about the item. Noise contributes to this problem, as several of the 140 characters in a tweet could be used up with emoticons, usernames or other data irrelevant to our solution.

## 2.5.2 Recommender systems

Among the three types of recommender system, content-based filtering systems is the method that could most easily make use of the research in this thesis. While the other systems also benefit from knowing more about new items in their library, content-based filtering systems makes most of its recommendations based on the similarity between items and their attributes. If this approach could determine the exact title of a new item from Twitter, gathering its attributes could be done through external tools with information about new releases, such as open movie APIs. Because the solution in this thesis would also provide the recommender system with the sentiment about each item, it could further rank items with similar attributes differently based on what item has a higher sentiment score.

## 2.5.3 Natural language processing

A major part of this thesis was the analysis and processing of text in the form on tweets. Because of this, several techniques from natural language processing becomes relevant. In order to evaluate how well the approach presented in this thesis could extract information about a specific topic or item, the quality measures precision, recall and F1 from information retrieval was used. Sentiment analysis would be necessary to make information about the items found on social media useful for recommender systems. From text analysis, both stopword removal and stemming would be applied to mitigate the noise in the data and to make detecting patterns easier for the classifier.

## 2.5.4 Machine learning

Both supervised and unsupervised machine learning would be relevant for this thesis. The classification method responsible for finding movie related tweets would be a supervised learning based artificial neural network, while the clustering algorithm would be responsible for determining how many clusters would be suitable for each dataset, meaning it would be unsupervised.



---

# Chapter 3

## Related work

In this chapter, we cover previous studies relevant for this thesis. Section 3.1 presents social media and how it has previously been used to mitigate the cold-start problem. Section 3.2 presents previous attempts at topic classification and methods for text representation, while section 3.3 presents previous attempts at clustering methods. Section 3.4 presents studies related to predicting movie success using social media, and section 3.5 discuss papers relating to sentiment analysis on tweets. Section 3.6 concludes the research of previous papers.

### 3.1 Solving the cold-start problem using social media

Because this thesis focuses on improving recommender systems by mitigating the cold-start problem using social media, this section looks at previous attempts to combine the two and what can be learned from them.

In [3], they presented a solution for improving recommendations for brand new users. They assumed that new users in their recommender system could sign up with a Twitter account. They then used this account to gather information upon which they could base their recommendations. This was done by looking at the users Twitter followers and what they wrote on Twitter. They determined the sentiment each follower had about specific items, and then weighted the reviews differently based on the person who reviewed it and that persons relationship with the user of the recommender system. This was done through what they called implicit social trust. Implicit social trust, meaning trust that can be inferred without directly asking about it, was modelled based on whether or not the two users were following each other and whether or not they re-tweeted each other. This allowed the system to get information about brand new users through their friends. They also described the algorithm they

utilized in detail but for this thesis, the focus was on how social media was used. By using these methods, they reported an increased accuracy compared to the baseline with no such cold-start mitigating system. The downsides to this system is that it requires the user of the recommender system to have a Twitter account. Another drawback to this system is that it takes no steps to improve the problem of new items being introduced to the system, only new users.

In [18] they took a slightly different approach using Twitter, in which they look at Twitter accounts of the *items* they want the system to know more about. They studied how to improve recommendations for mobile applications, by mitigating the cold-start problem of trying to recommend brand new applications. In this case, the applications are the items of the recommender system. Whenever a new item was added, they suggested that if that item had a Twitter account, which mobile applications often have to keep in touch with their customers, they could look at the followers of the application's Twitter account and use this as an estimation of the application's popularity and sentiment. They did this by calculating the probability of Twitter user A liking application C because Twitter user A followed another user, Twitter user B, which in turn followed application C. This method was also expanded to include all the reviews of a user. For example, if Twitter user B had liked application C and disliked application D, and Twitter user A followed user B, then the system could recommend application C to user A and avoid recommending application D. Using these two methods, they reported an increase in accuracy of their recommendations. While this method did improve the cold item problem in recommender systems, it requires the new items to have a Twitter account and a social media presence, which not all items in all domains will have.

## 3.2 Twitter topic classification and text representation

One of the tasks in this thesis was to classify tweets whose contents were related to a specific topic. There were many different ways to do this, with very different approaches and accuracy. In [8] they took a 3 step approach to movie classification, where they tried to find movie titles based on matching words with the title of the movie, first by completely identical words and then by almost identical, such as hashtags. After this they did a third step where they utilized a support vector machine to catch any remaining tweets not yet classified as movie tweets. A problem with this approach is that their first two steps in classifying text is to look for exact matches. This requires the system to know the name of the items instead of discovering them dynamically.

In a master thesis, Øystein Repp [27] found artificial neural networks to be an effective method for topic classification on tweets with decent results. Here, the data being classified was the same as the data used in this thesis, tweets. In addition, classification was used for a similar task, topic classification, making it comparable to our thesis.

### 3.2.1 Text representation

A common way to represent text that can be as short as tweets are with numeric vectors and word-embedding. Word-embedding assigns values and vectors to words, in order to make them easier to compare. Two popular models are the *word2vec* [21] model and the *Glove* model [25]. Out of these two models, more support and available pre-trained models was found for the word2vec model. Word2vec is also widely used and accurate [37].

## 3.3 Clustering of tweets and movies

Clustering refers to the task of placing data points in a vector space and dividing them into easily recognizable groups based on what points are closest to each other. This is done in such a way that points belonging to the same group, or cluster, are more similar to other points in the same group than points located in other groups. Clustering was a key part of the proposed solution in this thesis.

In [30] they attempted to cluster tweets into topics with both unsupervised and supervised methods. They report that supervised methods outperformed unsupervised methods, but state that "unsupervised methods tend to classify tweets based on language similarity rather than topical coherence". They also found it easier to cluster tweets in broad genres rather than specific topics. They achieved F1 scores for their clusters between 60% and 70%. The problem in comparing our thesis to [30] is that their paper tried to classify very broad topics. For our case, the topics will be narrower, as every tweet we cluster would be related to movies.

Article [26] deals with the problem of first story detection, and attempts to solve this problem through clustering. Their clustering algorithm is based on locality-sensitive hashing, and look for the closest neighbours using the cosine between two document vectors.

In [4] they report that an effective measure of similarity between documents is the cosine similarity, which calculates how similar two document vectors are. This, in addition to the previous studies regarding clustering of tweets, indicate that a clustering method based around finding the closest tweets in a vector space using cosine distance could be a viable solution.

## 3.4 Movies and social media

Because this thesis focuses on the movie domain in recommender systems, it was relevant to study previous papers concerning the relation between movies and social media. In two papers in particular, the relation between Twitter and movies discussed on Twitter was the focus. These papers also studied how to gather movie related tweets and how to predict movie success using Twitter.

In [15], they used the Python library Tweepy <sup>1</sup> and added search terms to the tweets they got from the Twitter API. This is probably the simplest way of getting somewhat relevant tweets, although getting some noise in a dataset made of tweets is close to impossible. The downside to this method is that you are limited to the movies you list in your search query to Twitter, and you are dependent on getting an exact match to your query.

In [2] an attempt was made to combine data from Twitter with data from other sources in order to predict whether a movie would be popular or not. From sites like IMDB, they took the names of actors, and then went to those actors' Twitter accounts. They then summarized the number of followers those actors had, along with other social media statistics and a sentiment score from tweets for the movie, and they used this in the popularity estimation. By doing this, they achieved a good accuracy score of up to 95%, however this was an estimation of movie *popularity*, such as ratings and gross income. The problem with this approach is that it requires all or most of the actors of the movie to utilize social media. It also discovers the popularity of a movie, which is not necessarily relevant for a recommender system that might care more about pure sentiment values. The dataset used in [2], a dataset of movie tweets, was not published.

## 3.5 Sentiment analysis

Sentiment analysis has been important for several of the papers mentioned in this chapter, and this section looks closer at this aspect.

In [14] they used classification methods to determine the sentiment of tweets about movies. They gathered tweets about six movies from different places in the world and classified them into either positive sentiment, negative sentiment or cognitive sentiment. The cognitive sentiment category was used to classify tweets that did not strictly express positivity or negativity about a movie. The classification methods used were Naïve Bayes and Max Entropy. They found that Max Entropy gave the best results with an accuracy of 84%. Accuracy was measured with the F1-score.

---

<sup>1</sup><http://www.tweepy.org/>



In [3], they did not state how they found tweets relevant to products. Once they found relevant tweets, they used a set of techniques to determine the sentiment, such as stopword removal, removing usernames, negating whenever "not" was found and using an online lexicon to identify sentiment behind emoticons and abbreviations such as "LOL" or " :) ". To assess sentiment, they used SentiWordNet [5], a lexical resource for opinion mining.

In [2], sentiment analysis was performed on movie-related tweets to determine the overall sentiment of movies. They found that the most important factor in trying to classify rating and movie income was the sentiment score. This was done by using a decision tree called C4.5 with a method called J48, a Java library, and applying the sentiment score to this method gave an accuracy of 77% at best.

Sentiment analysis of movie reviews was the focus of [22], which used a training set consisting of movie reviews from IMDB. To determine whether a word or a sentence was positive or not, they used the Naïve Bayes classifier enhanced with several techniques to increase the accuracy, such as handling negations and feature selection. By doing this an accuracy of close to 90% was achieved. One of the authors of the article, Vivek Narayanan, created an API and a website <sup>2</sup> where users can send in texts and the system will return a classification of the sentiment along with a confidence score. Because the system was trained on IMDB movie reviews, it works well when trying to classify the sentiment of movies, but works best with longer texts, 2-3 sentences or longer.

## 3.6 Conclusion of related work

While there have been several previous successful attempts at estimating the popularity of movies using Twitter data, to the best of our knowledge no study was found that focused on gathering unfiltered tweets and clustering them to determine the exact items. Because features like user following and trust levels between users have already been explored, it is more interesting to attempt a different approach, with a focus on unfiltered tweets and the sentiment about the movies they discuss and how effectively we could determine what movies they are discussing. The approach suggested in this thesis has the advantage of being more general and is able to be applied to a list of collected tweets without the need for specific query words.

For classification, we have seen several different accuracy scores in the literature. The highest accuracy was reported in [14] where they achieved an F1 score of 84%. This was done with a classifier trained on tweets about only six different movies. This indicates that the classifier might not require a large

---

<sup>2</sup><http://sentiment.vivekn.com/>

variety of movies in the training set in order to be properly trained. This report gives an idea of what accuracy our classifier and sentiment analyser should aim for. However, this is not completely comparable with the work in this thesis, because there are no pre-existing datasets that can be used to train the classifier, making it more challenging to achieve a high accuracy. In addition, the classifier in [14] was trained as a sentiment analysis tool, and not a topic classifier.

In [30], we get an estimate for cluster quality of between 60% and 70% F1. We also saw that using a clustering algorithm based around the cosine distance between tweet vectors could be a viable solution.

Because of the high accuracy of the sentiment analysis tool achieved in [22], and because they created an open API usable by anyone, this could be used in this thesis as a baseline to compare our own solution with. Although the work in this thesis included creating a sentiment analysis tool, the vivekn API would serve as a control to compare our solution against. If necessary, this API can be used in the final solution if the sentiment tool made for this thesis proves to perform worse, as sentiment analysis is not the focus of this thesis. It would still need to be tested, as the data it would be used on would be very different.

---

# Chapter 4

## Proposed approach

This chapter presents the proposed approach to improve recommender systems by finding a way to mitigate the cold-start problem. The chapter is divided between different parts of the solution and the overarching structure and the challenges. First, section 4.1 presents the overall architecture for our solution. Then, the approach is broken down and explained in three separate sections, from section 4.2 to section 4.4. In section 4.5 we discuss some problems with this approach and how to overcome them. Finally, section 4.6 presents the final system, how it is connected and how the output data can be used to improve recommender systems.

### 4.1 Overview

This thesis suggests an approach to collect, organize and aggregate information about new items discussed on social media in a way that makes the data usable by recommender systems. The first step was to collect movie-tweets from Twitter. The system then clustered all related tweets together to discover groups of items and then aggregated the sentiment about each item. Because the discussions on Twitter are usually about real-time events, most of the movies discussed there will be new movies that have recently been released. This makes Twitter an ideal source of data for a system that needs information and sentiment about new items.

The approach was divided into three main parts. First, the system would collect tweets from Twitter. It would then classify each tweet depending on whether or not the tweet discussed movies or not. Afterwards, it would save all movie related tweets for further use, and ignore the rest. This process is discussed in section 4.2. After all tweets relating to movies had been stored, the next step was to sort them. This was done by clustering them, under the assumption that tweets discussing the same movie would be clustered together. Afterwards, the titles of the movies had to be determined. Clustering and title

determination is described in section 4.3. Finally, the system had to determine the sentiment of the tweets, so that it could be used in a recommender system. This process is explained in section 4.4. The challenges related to the approach and how they were overcome is described in section 4.5. An overview of the architecture of the system can be found in figure 4.1.

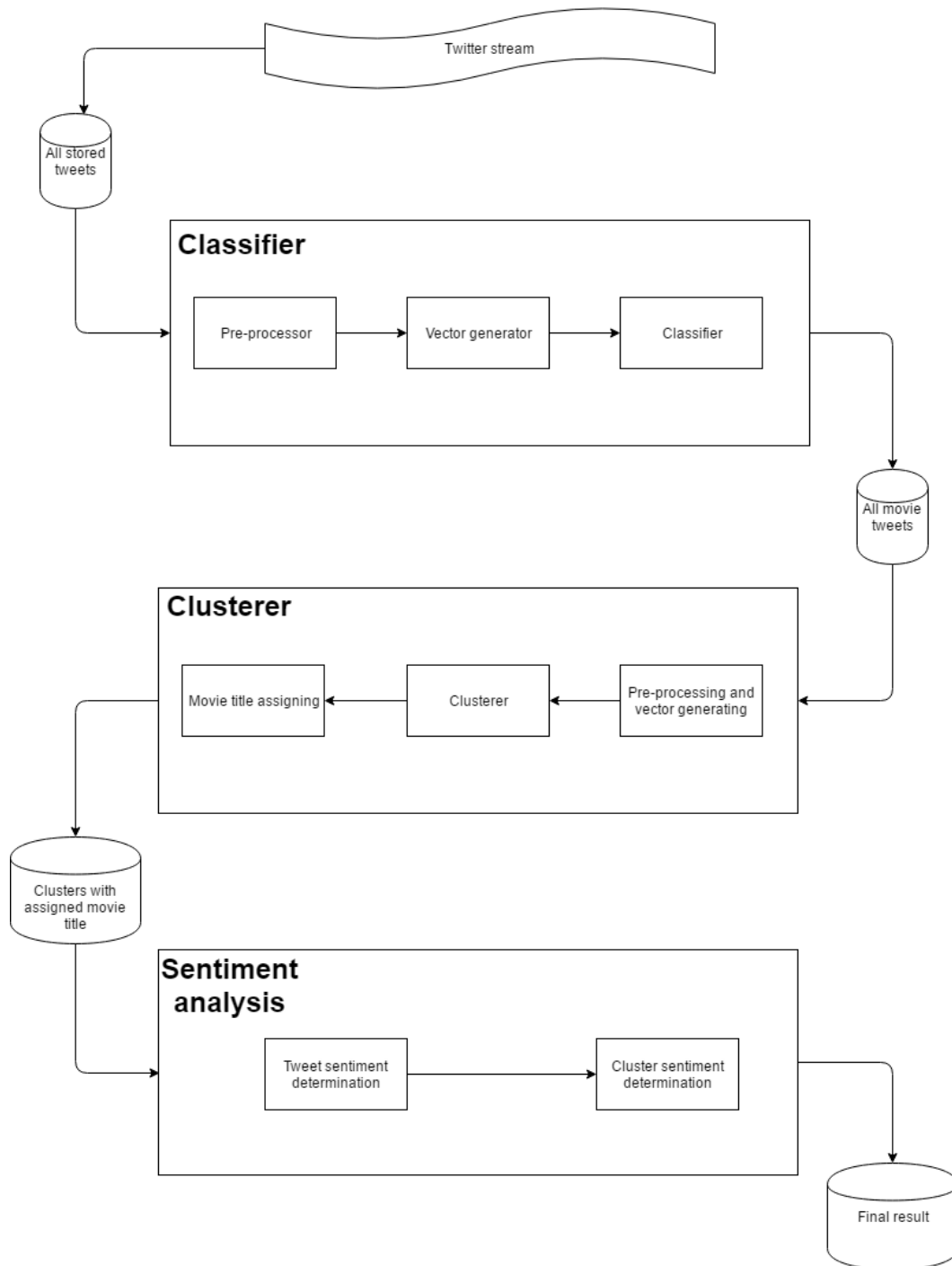


Figure 4.1: Overview of the three main components of the approach

## 4.2 Finding movie tweets

The first task in the proposed solution was to gather tweets and classify them. This section explains this process further, and explores some alternate ways of doing it.

### 4.2.1 Gathering tweets

Twitter has an API <sup>1</sup> which allows users to receive a stream of tweets and metadata. However, using the API directly gives unnecessary room for mistakes. That is why wrappers made by independent developers is often used, instead of using the Twitter API directly. These wrappers come in many different versions, and because this approach would be based on Python the two most relevant were Tweepy <sup>2</sup> and Twython <sup>3</sup>. Both of these offer a way to easily access the Twitter API. We chose Twython, as Tweepy had several bugs causing it to crash when receiving special characters from Twitter at the time of testing.

To access the Twitter API, developers have to register an application with Twitter, which is necessary to get the keys and tokens needed to authenticate the application and receive Twitter data. When the system was ready to gather tweets using Twython, there were two main options of doing so, the `filter` function and the `sample` function. The difference was that the `filter` function allows for query terms, meaning only tweets containing any of the words in the query will be returned. The `sample` method allows users to define a language, like English, and then returns all available tweets for that language, without needing to define filter words. For this thesis, we would only be working with English tweets. On the surface, the `filter` method might seem better, because it would seemingly make the classification task easier. However, the problem with this method was how the querying words and phrases were defined. One method was to use phrases that often occurred in discussions about movies. Sites like the Cambridge dictionary <sup>4</sup> provides a dictionary of words that are often associated with movies. The problem with this approach was that most people do not usually specify that they have seen a movie, and sometimes just say that they like the movie by specifying the name of the movie. An example of this can be seen in figure 4.2.

Another possible solution was to use movie titles as filters. The problem with this approach was that the system would need to keep a continuously updated list of all movies being released, and if it missed any titles, that

---

<sup>1</sup><https://dev.twitter.com/rest/public>

<sup>2</sup><https://github.com/tweepy/tweepy>

<sup>3</sup><https://github.com/ryanmcgrath/twython>

<sup>4</sup><http://dictionary.cambridge.org/topics/entertainment/cinema-general-words/>



Figure 4.2: Example of a tweet about a movie without mentioning "movie" words

would mean missing most tweets relating to that movie. The system would also fail to recognize any movies with a name that was abbreviated, like saying "LotR" instead of "Lord of the Rings". Misspelled titles would also be ignored. Because of these points, it was decided to use the `sample` function and simply classify them afterwards.

## 4.2.2 Representing tweets

To classify the tweets effectively, the system had to represent them numerically in a way that made it easier for a classifier to use. The representation model we ended up using was the *word2vec* model, because it had more support available at the time of conducting this study. This model had several pre-trained models available for free use, which would save the time it would take to train the model ourselves, which would require an enormous dataset of relevant data. There were several models available, but the two most noteworthy were Google's model <sup>5</sup> and a Twitter trained model made available on this website <sup>6</sup> by independent researchers in association with this paper [12]. Both of these models were trained on millions of words on different domains. The Google model had been trained on Google news event documents, while the Twitter model was trained on 400 million English tweets.

These models place similar words closer together in a vector space, thereby displaying what words are the most likely to be used in the same setting as other words. For example, we compared the nine most similar words to the word "movie" in both models. In the Google model, these similar words were, from most similar to least similar: film, movies, moive, Movie, horror flick, sequel, Guy Ritchie Revolver, romantic comedy, flick. While a lot of these made sense because of natural association with synonyms like "movies" and "film", it also included very specific and narrow phrases near the end of the list. Only one of these nine most similar words accounted for misspellings of common words, which is often found on Twitter. The same input into the

<sup>5</sup><https://code.google.com/archive/p/word2vec/>

<sup>6</sup><http://www.fredericgodin.com/software/>

Twitter model gave the following output, from most to least similar word: film, Movie, movieee, moviee, movies, movieeeee, mopvieeee, moive, mivie. In this list, there is more of what one would expect on Twitter, with many different ways to spell a single word, as well as less obscure matches near the bottom of the list.

The Twitter model was chosen for this thesis. Because it was bigger and trained on data very similar to what was relevant for this thesis, it was deemed to be the most suitable dataset. The Twitter model represented each word with a 400 dimensional vector, and had a vocabulary of a little over 3 million words.

Now that a method to create vectors for individual words had been found, the next step was to make a vector representation for the entire tweet. This was done by adding up all the individual term vectors and dividing by the number of terms, giving us the average vector. The formula is given in formula 4.1, where  $t$  is the vector of a single term, or word,  $T$  is the collection of vectors for all terms,  $n$  is the number of terms and  $t_n$  is the vector for term  $n$ .

$$AverageVector = \frac{\sum_{t \in T} t_1 + t_2 + \dots + t_n}{n} \quad (4.1)$$

### 4.2.3 Pre-processing

Because the system would utilize a pre-trained model, the pre-processing steps needed to recreate the pre-processing steps used when creating the model. Therefore, some pre-processing steps were necessary before sending each tweet into the word2vec embedder. This would improve our accuracy, in addition to decrease the amount of data to analyse and thereby give less room for errors. The Godin paper [12], which was the paper behind the word2vec Twitter model, explained the steps they took when preparing their dataset. Because the model was trained only on English tweets, the first step was to filter out any non-English tweets, which was taken care of assuming the Twitter API wrapper worked as intended. The system also had to replace any mentions of users and URLs with replacement tokens. Other steps included removing punctuations, question marks, exclamation marks, and numbers. A complete list of pre-processing steps can be found in table 4.1.

Converting all letters to lower-case was done for consistency, and should not affect the accuracy of the model. Removing URLs removes unnecessary noise. Because the approach would not consider the content of the URLs, they had no effect on how tweets were classified. The same could be said for usernames. Because there was no need for them in any way in our solution, and because each tweet vector was only based on its own text and not on who sent it or whom it was directed at, usernames were pointless to consider and only added noise. Special characters were removed, including punctuations. This was done



Table 4.1: Pre-processing steps

Filtering step	Description
lower_case	Make all letters lowercase
url_removal	Replace all URLs with an empty string
username_removal	Replace all usernames with an empty string
special_removal	Remove any special characters
hashtag_removal	Remove any words with hashtags
numbers_removal	Remove all numerical characters
stopword_removal	Remove all stopwords
stemming	Reduce all words to their stem

to avoid false mismatches. For example, if the system differentiated between "movie" and "movie.", it would affect our accuracy. The informality of tweets was another factor. Many tweets use many special symbols to underline a point or to express emotion. For example, writing a sentence and ending it with several punctuation marks or several exclamation marks. Trying to determine the sentiments behind these symbols or otherwise make sense of them in a vector space model would have been more work outside of the main purpose of this thesis, so they were removed for consistencies sake. They are usually not important for movie titles either, and the same can be said for numbers. Numbers could make an appearance in movie titles, like sequels, but in general, they are more often used to abbreviate sentences to make them fit tweets' 140-character limit, making them harder to classify for our classifier. An example of this would be replacing the words "to" or "too" with the number 2, or the word "for" with the number 4.

#### 4.2.4 Classification

There were two main ways to do classification, through either an unsupervised method or a supervised method. Because our data would be very diverse coming into the classifier, an unsupervised method was not a good choice. An unsupervised method is left to itself to determine the most important features of the text and the number of classes, and because Twitter is full of topics with more precedence than movies, it would be too easy for an unsupervised classifier to simply ignore all movie related texts and terms. Therefore, a supervised learning method would be used. By feeding the algorithm a training set that had been pre-classified, the classifier was trained to recognize what text belonged in what class. For this thesis, the classification task was treated as a binary classification problem. This meant only two classes were considered. The two classes were: Tweets that were about movies, and tweets that were not about movies. There were many classification methods to choose from,

but after consideration and looking at previous work like [27], a decision was made to implement an artificial neural network (ANN). To keep the code strictly to Python, Keras <sup>7</sup> would be used. Keras was chosen for its simplicity as a frontend, making it easier and quicker to get results. The underlying mechanics had to come from either TensorFlow <sup>8</sup> or Theano <sup>9</sup>. For this thesis, TensorFlow was chosen, as it outperformed Theano in training speed on the machine that would be used to run the program.

### 4.2.5 Training the classifier

The classifier had to be trained to recognize the correct tweets. Finding a suitable training set was therefore the first step. The choice of training data was very important, as the wrong kind of data might cause the classifier to fail to identify movie tweets. For the purpose of this thesis, it would be best to get training data as similar to the actual data as possible. That meant that tweets about movies would be used to train the classifier. The problem with this approach was that no such dataset was found available anywhere, however some similar alternatives were considered.

#### The IMDB review dataset

Some studies, like [9] had made a system that collected tweets sent from the IMDB website. IMDB allows users to tweet out what movies they rated and how they rated it, for example: "I rated The Matrix 9/10 <http://www.imdb.com/title/tt0133093/> # IMDb". The problem with this approach was that if our classifier was trained on this data, where all tweets would have the same structure and wording, the classifier would not identify other tweets to be relevant. For example, tweets without a number score or tweets without a link might be ignored. Even the word "rated" might cause trouble, since most users of Twitter do not explicitly state in their tweets that they are rating something.

#### The SAR14 dataset

Another dataset that could be more relevant for our case despite not consisting of tweets was the SAR14 dataset, put together in association with this study from 2014 [23]. This dataset consists of 233 600 movie reviews from IMDB, which meant the dataset had both good size and good quality in terms of topical relevance. This dataset was still not optimal, because the syntax and language of an IMDB review is inherently more "official" than that of a movie

---

<sup>7</sup><https://keras.io/>

<sup>8</sup><https://www.tensorflow.org/>

<sup>9</sup><http://deeplearning.net/software/theano>

tweet. It was still worth taking this set into consideration, and conduct tests to see how it performed when used to identify text from Twitter.

### **Creating a custom dataset**

One possibility to get the most relevant dataset possible is to spend some time during this thesis to create a dataset to train our classifier. If we chose to do this, there were two main methods of doing it.

*Collecting relevant tweets manually* - By searching Twitter manually it would be possible to gather only hand-picked tweets, ensuring that all tweets used in the dataset would be relevant and of a high quality. The problem with this approach was that while the relevance and quality would be high, the quantity would be very low. Even by finding one tweet in one minute, it would take 162 days of non-stop searching to get to the same amount as the SAR14 dataset. Realistically, even finding 1000 relevant tweets would take a long time to do manually.

*Collecting relevant tweets automatically* - Instead of manually looking for tweets, another solution was to use the Twitter API wrapper to get all tweets with specific keywords in them. By using movie titles as these keywords, the wrapper returned all tweets referencing this movie by its title, making it highly relevant data. To get a large dataset, it needed to be used on movies that were discussed a lot on Twitter, which usually means the newest movies, or movies that are the topic of discussion due to trailers or other reasons. By doing this we would get close to the relevance of the manual method, while also get the size of the SAR14 dataset.

### **Summary of training the classifier**

This leaved us with four methods, the automatic gathering of tweets, the manual gathering of tweets, the SAR14 dataset and the IMDB review tweet set. We chose to use the automatic gathering of tweets and the SAR14 dataset. These two were chosen in order to see the difference in accuracy when using training set from Twitter and when using a general movie review set, and to get a training set with sufficient size. These datasets would be larger than the manually collected tweets and they would be more relevant than the IMDB scoring tweets.

### **4.2.6 Methods of evaluation**

As mentioned in section 2.3.2, the precision, recall and F1 score are good ways of evaluating the rate of retrieval of relevant information. These three metrics was used to evaluate the performance of the classifier.

## 4.3 Clustering movies

In order for movie-tweets to be usable by a recommender system, the approach needed a way to sort tweets based on what items, or movies, they discussed. This sorting was done through clustering, and there were several ways to cluster these tweets as discussed in section 3.3. One possibility was to group the tweets together based on genre or sentiment instead of movie title. However, because the information should be usable by a recommender system, the most logical choice was to cluster movies based on title. Determining the titles of the movies would let us gather any additional information about the movie from external sources, and with the tweets giving their sentiment that should be all the information a content-based filtering system needs to make recommendations. Because tweets with more words in common would have a more similar vector representation, a clustering algorithm based around cosine distance between tweet vectors would give decent results.

### 4.3.1 Representing data

The clustering algorithm would use the same representation of tweets as the classifier did, a word2vec model trained on tweets. The same pre-processing steps would be repeated, where irrelevant information like usernames and URLs were replaced with empty strings and all letters converted to lowercase. The exception was that for clustering, stopwords would not be removed, as they can relate to the title of a movie.

### 4.3.2 Clustering method

There were several methods to sort tweets based on titles. The fundamental assumption was that tweets that discuss the same movie would be grouped together. The reason for this was that they must contain some information related to the title in the text, and should therefore be more similar to other tweets discussing the same movie than tweets discussing other movies. There were several methods to cluster movie titles, some of which were more suitable than others. Before the proposed method is presented, we discuss some reasons why other more traditional methods were not chosen.

Many of the traditional clustering methods, like the k-means clustering algorithm, requires input in the form of the number of clusters the algorithm is expected to create. This is true for most supervised learning methods. For the purpose of this thesis, the number of clusters was unknown. The system should be able to determine the number of clusters without any required input. This would make the system more dynamic as well as make it easier to utilize the system in other domains.

### The clustering algorithm

The algorithm would try to mitigate the outliers caused by the noise on Twitter and use cosine distance as a measure for similarity. Because we are using Python to construct our solution, the similarities between vectors would be measured with cosine distance, through the SciPy <sup>10</sup> library. Pseudocode for the clustering algorithm can be found in algorithm 1. The essence of this algorithm was that the distance from one *data point*, or tweet, to a cluster would be measured by getting the cosine distance from the point vector to the centroid vector for the cluster. The centroid vector is the average vector for all points in a cluster. Whenever a point is added to a cluster, the centroid is adjusted according to the new point, moving it around.

This is illustrated in figure 4.3, where we start in (1) with a cluster with four assigned points and a centroid. In (2), a new point is discovered and assigned to the cluster, which results in the centroid being recalculated in (3). Another point is added in (4), which further adjusts the centroid in (5).

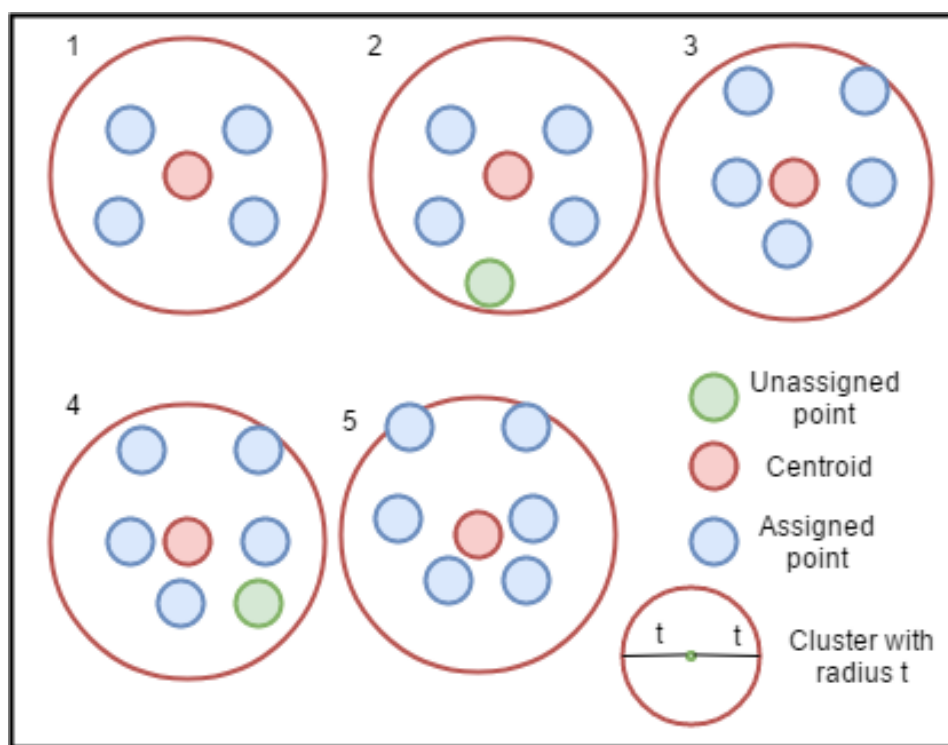


Figure 4.3: Illustration of the clustering algorithm adjusting a cluster centroid

This incentivizes each cluster to only contain tweets about one single movie, however there may be more clusters for each movie. An important aspect of

<sup>10</sup><https://docs.scipy.org/doc/scipy/reference/index.html>

tweets as data and this algorithm was that the data would contain many outliers, meaning clusters with a single point. To somewhat mitigate this, one method applied was to check all discovered clusters after the algorithm is done clustering. We then repeat the first step in the algorithm for any outlier points, assuming here that some of the points previous classified as outliers were now within the minimum distance threshold of a cluster, both because more clusters had been added and because the cluster centroids had been moving around.

Another way to mitigate outliers was to implement a method we named the "refresh clusters" algorithm. This algorithm is described in algorithm 2. "Refresh clusters" was triggered whenever a new cluster was created. Upon activation, "refresh clusters" would make a list of all the closest clusters to the newly created cluster, and then go through all the points of these closest clusters, to see if any of the points were closer to the newly created cluster than the one they were assigned. This would get very costly as more data was added, and might not be worth the cost of the extra data management.

An important factor of this algorithm is the way it handles duplicates. Whenever a point is added to a cluster, the algorithm checks all previous points in the cluster to see if any of them are identical to the point we are trying to add. This is done by checking the cosine distance between all points in the cluster and the point we want to add, and check if any has a cosine distance of approximately 0. If any duplicate is found, the point is not added to the cluster, and the algorithm continues with the next point.

### 4.3.3 Methods of evaluation

The clusters was evaluated using the same metric as the classifier, with precision, recall and the harmonic mean F1 score. For evaluating clusters, we can define these scores as the following

- **Precision** - Measures how well the algorithm performed in regards to only assigning the correct data points to the cluster. If the clusters were full of data points that did not belong there, it resulted in a lower precision score.
- **Recall** - Measures in what degree the data points belonging to a certain cluster had been assigned to that cluster. In order to score a perfect score on recall, all data points belonging to a cluster must be assigned to that cluster.
- **F1 score** - the harmonic mean of precision and recall.

---

**Data:** Threshold *threshold*, list of tweets *tweetList*  
*knownClusters*  $\leftarrow \{\}$  ;  
 assignToCluster() starts here;  
**for** *tweet* in *tweetList* **do**  
   *tweet.vector*  $\leftarrow$  vector generated for *tweet*;  
   **if**  $|knownClusters| == 0$  **then**  
     *newCluster.centroid*  $\leftarrow$  *tweet.vector*;  
     add *newCluster* to *knownClusters*;  
   **else**  
     *nearestCluster*  $\leftarrow$  nearest cluster from *tweet*;  
     *shortestDistance*  $\leftarrow$  distance from *tweet* to  
       *nearestCluster.centroid* ;  
     **if** *shortestDistance* < *threshold* **then**  
       **if** *tweet* not in *nearestCluster.clusterPoints* **then**  
         add *tweet* to *nearestCluster.clusterPoints*;  
         recalculate *nearestCluster.centroid*;  
       **end**  
     **else**  
       *newCluster.centroid*  $\leftarrow$  *tweet.vector*;  
       add *tweet* to *newCluster.clusterPoints*;  
       add *newCluster* to *knownClusters*;  
       refreshClusters() See next algorithm for this  
     **end**  
   **end**  
**end**  
*outliers*  $\leftarrow \{\}$  ;  
**for** *cluster* in *knownClusters* **do**  
   **if**  $|cluster.clusterPoints| == 1$  **then**  
     add *cluster.clusterPoints*[0] to *outliers*;  
     remove *cluster* from *knownClusters*  
   **end**  
**end**  
 Repeat the first for loop for the outliers list;  
**Algorithm 1:** Main clustering algorithm

**Data:** List of known clusters *knownClusters*, newly added cluster *newCluster*, Threshold *threshold*  
*newCluster*  $\leftarrow$  Newly created cluster ;  
*clustersToRefresh*  $\leftarrow$  {} ;  
**for** *cluster* in *knownCluster* **do**  
    *distance*  $\leftarrow$  *dist(cluster, newCluster)*;  
    **if** *distance* < *threshold* \* 2 **then**  
        add *cluster* to *clustersToRefresh*;  
    **end**  
**end**  
**for** *cluster* in *clustersToRefresh* **do**  
    **for** *point* in *cluster* **do**  
        *oldDistance*  $\leftarrow$  *dist(point, cluster)*;  
        *newDistance*  $\leftarrow$  *dist(point, newCluster)*;  
        **if** *newDistance* < *oldDistance* **then**  
            remove *point* from *cluster.clusterPoints*;  
            add *point* to *newCluster.clusterPoints*;  
            recalculate *cluster.centroid*;  
            recalculate *newCluster.centroid*;  
        **end**  
    **end**  
**end**

**Algorithm 2:** Refresh clusters algorithm



### 4.3.4 Summarizing clusters and finding movies

After the tweets had been clustered, one cluster was going to consist of several different data points, or tweets. In order to use this information in a recommender system, there had to be a way to extract what movie these tweets were discussing. To do this, we would create a word-counting dictionary for each cluster. By doing this, we count the frequency of every word in each cluster, which could then be used to create a textual representation for each cluster. If, for example, we return the top 10 words for each cluster, the idea was that even though some of those words would be irrelevant words, some of them should spell the title of a movie. Stopwords can not be removed in this process, as movie titles often contain stopwords like "the". We can then use external sources like Wikipedia or open movie APIs to check if there is a movie released around the time of running containing any of these words. For example, after clustering a set of tweets, this was one of the clusters text representation: "it about see the and you just is Animals Nocturnal". By comparing these words to a list of movie titles recently released, chances were high we would find a match, for example the movie "Nocturnal Animals". After deciding on a movie title, we would extract a sample of tweets from the cluster and see how many of them contains the title we have assigned. By doing this on a large enough amount of tweets we could get an accurate confidence score, which would represent how certain we are that this is the correct title for this cluster. For this thesis, we chose to use "The Movie DB"<sup>11</sup> API. An example call to the API written in Python can be found below.

---

```
import tmdbsimple as tmdb
tmdb.API_KEY = "key"
dis = tmdb.Discover()
res = dis.movie(sort_by='popularity.desc')
for s in dis.results:
    print(s['title'])
```

---

## 4.4 Sentiment analysis

Once the relevant tweets were classified and clustered into movie titles, the next step was to determine the sentiment of each movie. Because each cluster had one movie title associated with it, this could be done by assigning sentiment values to clusters. This was done by performing sentiment analysis on each individual tweet in a cluster, then aggregating all sentiment values to determine the overall sentiment of the cluster.

---

<sup>11</sup><https://www.themoviedb.org/>

### 4.4.1 Sentiment analysis on individual tweets

The first step of the sentiment analysis was to determine the sentiment of each individual tweet. As discussed in sections 2.3.1 and 3.5 there were several ways to perform sentiment analysis. One way to do this was to use another artificial neural network classifier trained for sentiment classification instead of movie-tweet classification. Another method was to assign sentiment to each word, and calculate a final sentiment based on the score of each of the individual words. This could have been done by comparing each word in the tweet to two dictionaries, one containing words associated with positive sentiment, and one containing words associated with negative sentiment. A third option was to utilize already existing solution which had proved to work well, as discussed in section 3.5 with the `vivekn` sentiment API.

In order to compare two different approaches and to determine if we could create a better analysis tool ourselves than the dedicated tools we found during research, two methods of sentiment analysis was used. Firstly, a sentiment classifier based on an artificial neural network. This ANN would be trained on the SAR14 dataset, which consisted of movie reviews with scores. The sentiment classification would be treated as a binary problem to give the best chance of a higher accuracy. These two sentiments would be positive and negative. The SAR14 dataset ranks from 1, being negative, to 10, being positive. To create an even split, this meant that reviews with scores of 5 or lower were treated as negative and reviews with scores of 6 or higher would be treated as positive. Because this was treated as a binary classification task, the ANN output would be 0 for negative text and 1 for positive text.

In order to be able to compare two different methods of sentiment analysis, this thesis would also use the `vivekn` API to determine sentiment of the tweets. The API can be found here <sup>12</sup>, and was presented in this paper [22]. This solution had the benefit of not having to be trained, and had already documented a good accuracy score for its domain, which was IMDB movie reviews. Because the `vivekn` sentiment analysis was trained on movie reviews, it was very suitable for the needs of this thesis. It would still need to be tested on tweets, as they are very different from well-structured IMDB reviews.

The API required a string on which to perform sentiment analysis, and returned a JSON object with the fields *sentiment* and *confidence*. The sentiment was returned as one of either positive, neutral and negative. The confidence was a score indicating how sure the system was that the sentiment assigned to the text was the correct sentiment. The confidence score was a number between 50 and 100, with a higher score indicating less doubt concerning the assigned sentiment. To assign tweets an accurate sentiment score, the scoring in table 4.2 was applied. Each tweet was assigned a final sentiment score based

---

<sup>12</sup><http://sentiment.vivekn.com/>

on the sentiment and confidence from the API. The reasoning for the scoring was based on the assumption that a negative sentiment text with 90% certainty was "more negative" than a negative sentiment text with a 60% certainty.

Sentiment	Confidence	Score
Positive	81 - 100	2
Positive	50 - 80	1
Neutral	50 - 100	0
Negative	50 - 80	-1
Negative	81 - 100	-2

Table 4.2: Calculation of sentiment score

In order to effectively compare the classifier and the vivekn API, *testing* would consider neutral and positive as the same sentiment, in order to compare the result to the binary output produced by the artificial neural network classifier. However, during usage in the final product, the scoring system describes in table 4.2 would be used.

Example call to the vivekn API written in Python:

---

```

from urllib.parse import urlencode
from urllib.request import urlopen, Request
import json as js
url = "http://sentiment.vivekn.com/api/text/"
post_fields = {'txt': 'Text to classify goes here'}
request = Request(url, urlencode(post_fields).encode())
json = js.loads(urlopen(request).read().decode())
sentiment = json['result']['sentiment']
confidence = round(float(json['result']['confidence']))

```

---

#### 4.4.2 Sentiment analysis on clusters

After finding and saving each tweet and its corresponding final sentiment score, the next step was to find the overall sentiment for each cluster of tweets, and thereby, each movie title.

If the vivekn API yielded the best accuracy, this could be done by simply adding up the final sentiment score of all the individual tweets within that cluster and assigning that score to the cluster. A score above 0 indicates overall positive sentiment, while a score below 0 indicates negative sentiment.

If the ANN classifier gave the best accuracy, we would divide the aggregated tweet score by the number of entries in the cluster, giving a final cluster sentiment score of between 0 and 1, with 0 meaning only negative tweets in the cluster and 1 meaning only positive tweets in the cluster.

### 4.4.3 Methods of evaluation

Evaluating sentiment is a challenging task. Because of the nature of sentiment, it is often unclear or uncertain what the meaning behind the words are and what sentiment they really portray. Factors like negators, irony and sarcasm can all affect the sentiment perceived in a sentence. One way to evaluate the accuracy of our sentiment analysis was to perform user tests. This would require gathering a large group of people and give them a set of tweets that had been given sentiment by our system. They would then evaluate how accurately our system had scored the tweets. This is possibly the most accurate method of evaluation, in the sense that humans would be more reliably able to identify sentiment in a sentence than a machine. However, in order to get reliable test results, a substantial amount of tweets would need to be tested. Each tweet would have to be evaluated by at least two people to counteract personal bias, and each person would require some time to determine the sentiment of each tweet. For example, if it took a single person 30 seconds to evaluate a single tweet, 100 people would be able to evaluate 12 000 tweets in an entire hour, assuming they never did anything other than evaluating tweets. In addition, because each tweet would need to be evaluated by two people, we can halve this amount down to 6000 tweets. This might sound like a decent amount, but if we consider the amount of entries in some of our datasets, like the SAR14 set that contains over 321 000 entries, the percentage tested becomes relatively small.

After further research into similar solutions determining sentiment of tweets, it was decided that an automated system of evaluation would be better, and assume the accuracy of human testers could be made up for by testing more entries. The most suitable dataset we found was the Stanford sentiment140 dataset <sup>13</sup>, used to train a Maximum Entropy sentiment classifier. This dataset contains 1.6 million tweets with an associated sentiment score, ranging from 0 (very negative) to 4 (very positive). The tweets in the dataset were gathered using a variety of queries. In this report [11] they describe their methods and accuracy scores, which averaged around 80%, which was good enough as a baseline for comparison for this thesis. When evaluating our system, we took tweets from the Stanford dataset with their assigned sentiment, and run the same tweets through our sentiment analysis tool and compared the results.

## 4.5 Challenges

In sections 4.2 to 4.4 we presented individual steps and challenges related to those steps, but in this section we discuss some of the overarching challenges.

---

<sup>13</sup><http://help.sentiment140.com/for-students/>

### 4.5.1 The problem

The biggest problem stems from the assumption made during the clustering phase. We assumed that tweets discussing the same movie would be clustered together due to the same movie titles occurring in the tweets. The problem with this assumption was that the clustering algorithm might cluster together tweets that shared the same sentiment instead of tweets that discussed the same movie. This could happen because two tweets that share the same sentiment might have more similar words than two tweets discussing the same movie. So the result might be two tweets in the same cluster discussing different movies, but because they shared the same sentiment, they would have been similar enough as to put them in the same cluster.

### 4.5.2 The solution

To test if this would cause problems, the different modules of the program would be run in two different orders. One version would classify the tweets into movie tweets and non-movie tweets, then perform clustering on the movie tweets and finally perform sentiment analysis on the finished clusters. This was named the *one-step classification* method. The other version would classify the tweets into movie and non-movie, then classify the movie tweets into two lists depending on their sentiment, and finally cluster both these lists. For this *two-step classification* method, only two sentiments would be used, negative and positive. The reason for not having a third list with neutral tweets was that this might cause one or more of the list to become very sparse with not enough entries to cluster in any meaningful way. In addition, pure neutral tweets could still be considered positive, as they generate discussion and interest around a movie. For these reasons, the results from the vivekn API would put neutral and positive entries into one list and negative tweets in the other, as this also lined up with the way the sentiment classifiers were tested.

With this method, the tweets being clustered together already shared the same sentiment, which might yield clusters with higher F1 scores.

### 4.5.3 Clustering after two step classification

After classifying the tweets into two sentiment-sorted lists, the system would cluster the tweets. The two-step classification should make it easier for the clustering algorithm to differentiate between different movies being discussed in the tweets, and cluster those tweets together with higher precision and recall. This meant that all movies that were present in both lists would get two clusters, one for each sentiment. The same clustering algorithm would be used for two-step classification and for one-step classification, but the best threshold for the distance between vectors may vary.

#### 4.5.4 Assigning sentiment scores after two-step classification

The clustering algorithm should create two clusters for each movie, assuming there are both negative and positive tweets about that movie. In order to determine the overall sentiment of the movie, the size of each of the two clusters had to be compared. If the cluster that contained positive tweets was larger than the cluster which contained negative tweets, then this would represent overall positive sentiment for the movie. If the cluster with negative tweets was larger, this would indicate overall negative sentiment.

### 4.6 Final system

Now that all parts of the system has been explained and the sentiment-clustering problem has been explored we can finally take a look at how the system would work as a whole. In figure 4.4, an overview of the system with one-step classification is illustrated.

#### 4.6.1 Gathering and classifying movie tweets

This is be the first step of the system, and covers everything from collecting tweets to identifying movie tweets.

##### Gathering tweets

Tweets would be gathered by using the Twython wrapper for the Twitter API. Here, we gathered tweets without any sort of filtering phrase, the only filter applied at this stage was that only English tweets will be received.

##### Representing tweets

The tweets would be represented by a 400 dimensional word2vec model, after some pre-processing steps to reduce the noise in each tweet. The pre-processing steps include stemming, stopword removal, tokenization and more. A complete list can be seen in table 4.1.

##### Classifying movie tweets

To classify what tweets were about movies, we would be applying an artificial neural network using a combination of Keras and Tensorflow. The classifier would be trained with two datasets, to determine which gave the best results. The SAR14 [23] dataset, which consisted of IMDB movie reviews, and a

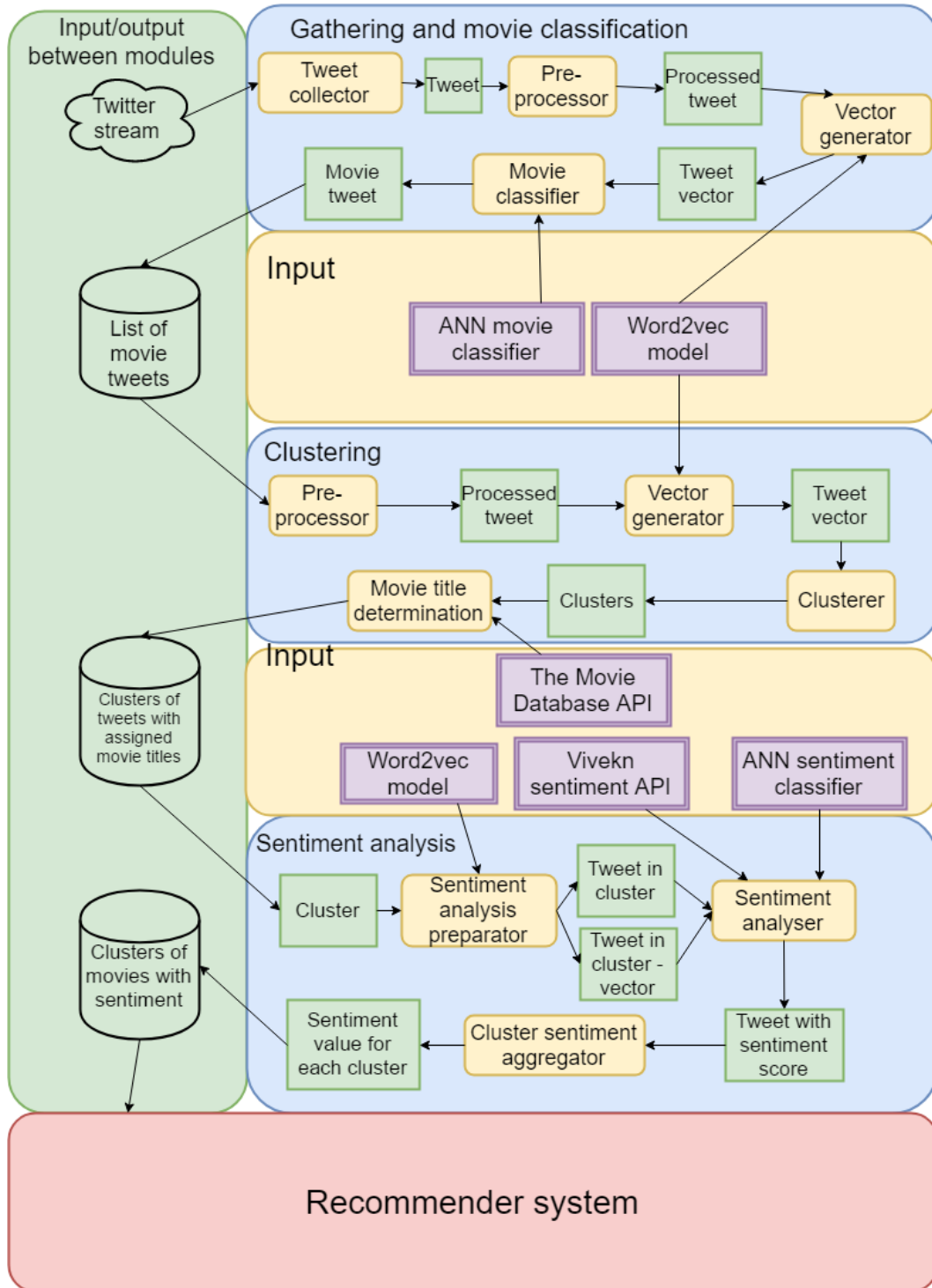


Figure 4.4: Overview of system with one-step classification

dataset created by automatically gathering tweets that discuss certain movie titles through the Twitter API.

### 4.6.2 Two-step classification

In order to achieve optimal clusters, one solution would be developed where the sentiment of the tweets was determined straight after determining what tweets were relevant. This would be done with the same method as the movie classifier, an artificial neural network with the same configuration. To train this classifier, the SAR14 dataset was used. The classifier would classify all movie tweets into two classes: Negative sentiment and everything else. We would also use the vivekn API where positive and neutral sentiment is grouped together to see what method gave the most accurate result. The two-step classification method is illustrated in figure 4.5.

### 4.6.3 Clustering

After the tweets had been classified, we would cluster them together. This would be done by using the algorithm explained in section 4.3. Using cosine distance, we clustered tweets together based on distance between vectors. The tweets would be represented the same way when clustered as when classified, with a word2vec model trained on twitter data and with the same pre-processing steps found in table 4.1.

#### **With one-step classification**

With one-step classification, the clustering would take place straight after the movie classifier decided what tweets were about movies. The assumption was that the clustering algorithm would create clusters in such a way that each movie had at least one cluster, which would contain all tweets about that movie.

#### **With two-step classification**

If we perform two-step classification, then the system would have already done the sentiment analysis and produced two lists at this point. One list of all negative movie tweets and one list of all positive or neutral movie tweets. The clustering algorithm would then be run one time for each list, which should create two cluster for each movie, one for each list.



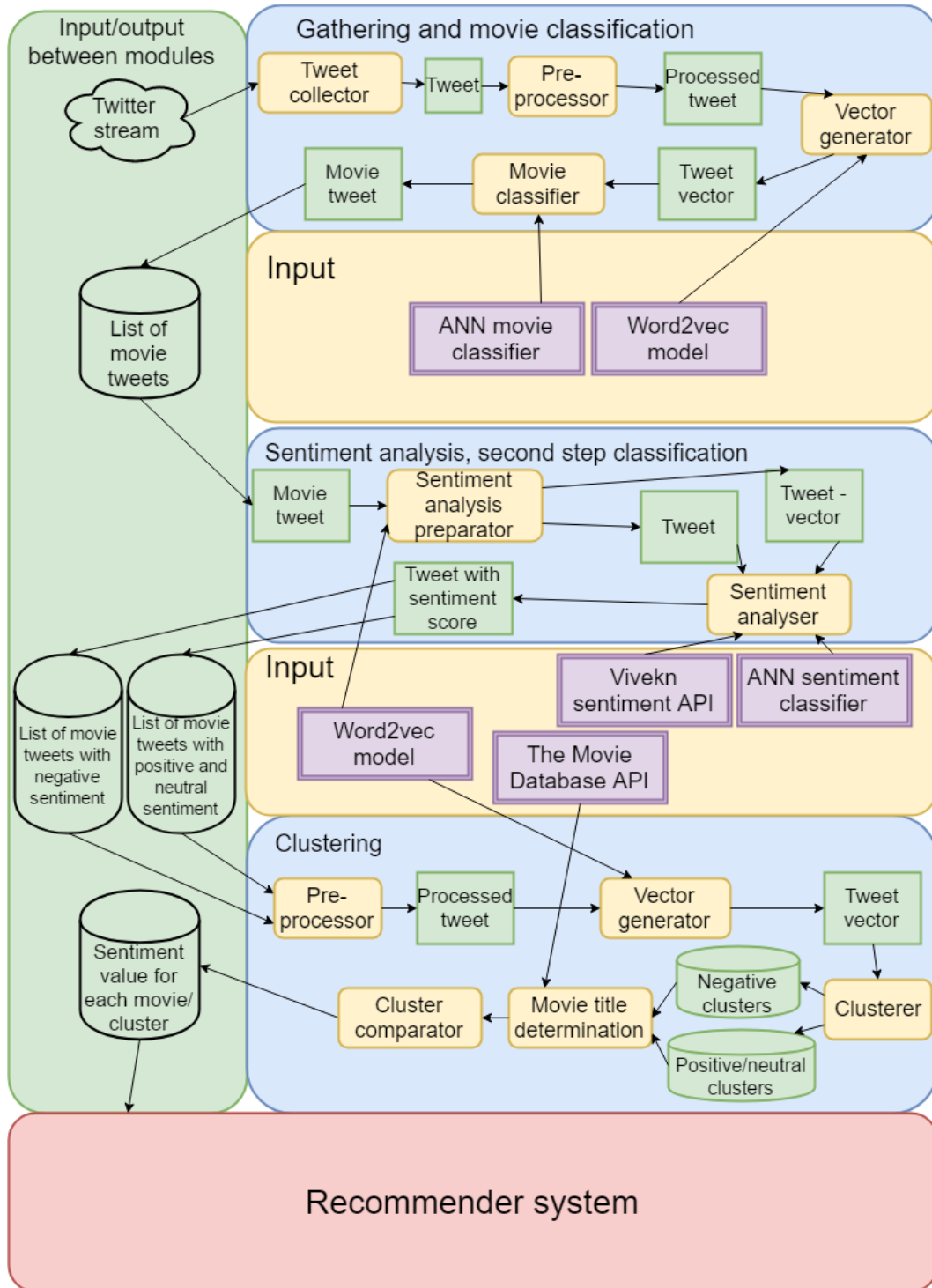


Figure 4.5: Overview of system with two-step classification

#### **4.6.4 Sentiment analysis**

##### **Sentiment analysis with one-step classification**

If we did not utilize two-step classification, this step is the last step. Here, all tweets in each cluster will be analysed and a given a sentiment score dependant on whether the ANN binary classifier or the vivekn API was used. Then, according to the method used, the sentiment scores are aggregated on a cluster basis and each cluster is given a total sentiment score.

##### **Sentiment analysis with two-step classification**

First, the tweets would be split into two lists with negative sentiment entries in one group and everything else in the other. If the cluster with positive and neutral sentiment was larger than the cluster with negative sentiment, then the overall sentiment was positive.

#### **4.6.5 Final note**

After all these steps have been completed, both the two-step classification method and the one-step classification method should have created several clusters of movies and determined what movies have a positive sentiment attached to them according to Twitter and its users. This can be used to improve recommender systems, and this is as far as this thesis aims to get.

---

# Chapter 5

## Experiments and evaluations

This chapter describes the methods used to test the solution presented in chapter 4. It also evaluates the results from these tests and makes conclusions based on these evaluations. Each component is given its own section of preparation, testing and discussion. Section 5.1 presents the movie classifier and how different training sets and other parameters gave different results. In section 5.2, the clustering algorithm and how it performed with different parameters are discussed. Section 5.3 presents the testing of both the neural network sentiment classifier and the vivekn API classifier. Section 5.4 discusses and compares results from one-step classification and two-step classification. Section 5.5 will cover the final result and whether or not it can be used to improve recommendations.

### 5.1 Movie Classification

In order to classify the tweets into movie related and non-movie related tweets using an artificial neural network, the ANN model needed to be trained. Because the training set determines how effectively the ANN can classify the data, the training set needed to have both quality in terms of relevance and quantity in terms of size.

#### 5.1.1 The datasets

##### Twitter

Because no dataset of movie tweets was found as of conducting these tests, a decision was made in section 4.2.5 to create a custom dataset. We found in section 3.5 that six movies can be sufficient to create a training set. The dataset was made by entering six different movie titles into the Twython Twitter API wrapper and gathering all available tweets discussing these movies. These

tweets were gathered in the period 14.12 2016 - 18.12 2016, and the six movies collected were:

- Rogue One: A Star Wars Story
- La La Land
- Doctor Strange
- Nocturnal Animals
- Assassin's Creed
- Collateral Beauty

These movies were chosen because they represent very different genres of movies, and should therefore cover a bigger group of users. For example, it is possible that people tweeting about fantasy movies never see movies that are romances or dramas. Therefore, we chose movies that cover multiple genres to get input from as many different users as possible. They also varied greatly in their critical acclaim. Rotten Tomatoes <sup>1</sup> is a site that aggregates movie reviews to create a summary score for each movie from reviewers. It rates "Assassin's Creed" at 17%, a low score, while "La La Land" is rated at 93%, a very high score. We can assume that this sentiment would be reflected in some of the tweets collected. By including both "bad" and "good" movies, the dataset represents both tweets that praises a movie and tweets that talk negatively about a movie.

This gave a great number of relevant tweets, although not all tweets displayed sentiment about the movie. Some tweets were marketing, others were tweets that linked to videos discussing the movies.

### **SAR14**

The SAR14 dataset was one of the movie related datasets that already existed. It consisted of 234 000 movie reviews from IMDB with matching scores. For the movie classifier, the score was removed, and only the review itself was used. This dataset was very different from the Twitter set as all entries were longer than tweets and consisted of more formal and structured sentences.

---

<sup>1</sup><https://www.rottentomatoes.com/>

### Gathering irrelevant tweets

Both of the two previously mentioned datasets consisted only of entries that belonged to one class. They were both examples of text that should be classified as a movie related text by the classifier. The training set had to consist of examples from both available classes in order to provide proper training for the ANN. To achieve this, the Twython wrapper was used to gather tweets without any sorting query, meaning all available tweets were collected. This would provide the ANN with non-movie related tweets. Because of the size of the set, it was impossible to read all the tweets gathered to confirm none of them were about movies. However, from general experience working with tweets, the percentage of discussions on Twitter that revolve around movies are very low. The collection of irrelevant tweets was done in one day, on 03.01 2017.

### The sets and their sizes

A total of 124 078 "irrelevant" tweets, meaning non-movie tweets, were collected, and 321 750 movie related tweets were collected. It was important that the tweets used to test the accuracy of the classifier was not part of the training set. The reason for this was that it would affect the result of the accuracy test by giving the classifier bias toward vectors it had been trained on. Therefore, 94 727 tweets were put aside from the movie tweet set along with 36 717 tweets from the non-movie tweets set. This total of 131 444 tweets would be used to test the classifier. This number was chosen because the common norm in machine learning with ANNs is to have a 70-30 split between the test set and training set, meaning that 30% of the total size of the set should be used for testing. Both the classifier trained on tweets and the classifier trained on the SAR14 set would be tested on tweets because this was the most relevant set. The size of all sets can be seen in table 5.1.

Dataset	Movie related entries	Non-movie entries	Total entries
SAR14	234 000	87 361	321 361
Twitter Set	227 023	87 361	314 384
Testing Set	94 727	36 717	131 444

Table 5.1: Datasets used for training and testing the movie classifier

### 5.1.2 Preparation

The ANN was based on the Python libraries Keras and Tensorflow. Tensorflow provided the backend and Keras provided a simplified interface to access the different Tensorflow functionality. In preparations for these tests, smaller

subsets of the datasets were tested to see what configuration of the ANN gave the best results. The chosen parameters for the ANN was: A sequential model, consisting of three layers. The first layer was the input layer. It had 400 neurons, took in a vector with 400 attributes and used the "relu", or rectifier, activation function. The second layer had 200 neurons and the same activation function as the input layer. Finally, the output layer had 1 neuron with sigmoid activation function to ensure an output of between 0 and 1, making it easy to round off and get the binary classes. When compiling the model, the loss function, also known as the optimization score function, was specified as binary cross entropy. The optimizer chosen was "Adam" and the metrics "accuracy".

When fitting the models to the training set, the network worked with a batch size of 10 and with 150 complete runs, called epochs. Other amounts of epochs and batch sizes were tested during preparations for the main tests, but these configurations were found to give the best results, although the variance was relatively small when compared to the effect of adding or removing layers and neurons.

Keras comes with an automatic measure for accuracy by returning the loss value and metrics value for the model. This accuracy was the overall accuracy for how well the model could predict the results. This would be used in addition to the previously mentioned precision, recall and F1 scores. All scores were assigned with the testing set.

### 5.1.3 Testing

#### With and without stemming

Both the Twitter set and the SAR14 set was tested with and without stemming the training data in order to see how stemming affected the accuracy of the model. The tests would be performed both on stemmed and not-stemmed testing sets. Because all vectors in the test set were marked with the correct class, measuring the accuracy was done using the

`sklearn.metrics.precision_recall_fscore_support` method from the `scikit` <sup>2</sup> library. These first results are from both models trained with and without stemming, tested on tweets that were stemmed, and can be seen in table 5.2.

---

<sup>2</sup><http://scikit-learn.org/stable/>

Dataset	Keras accuracy	Precision	Recall	F1 score
SAR14 with stemming	42.81%	99.99%	20.97%	34.67%
SAR14 without stemming	29.93%	100.00%	3.18%	6.16%
Twitter with stemming	95.54%	97.48%	96.33%	96.90%
Twitter without stemming	69.38%	96.89%	59.60%	73.80%

Table 5.2: Performance scores for classification models tested on tweets with stemming

The same models were tested a second time on tweets that were not stemmed, and the results can be seen in table 5.3.

Dataset	Keras accuracy	Precision	Recall	F1 score
SAR14 with stemming	43.46%	97.91%	22.33%	36.36%
SAR14 without stemming	59.07%	86.99%	51.06%	64.35%
Twitter with stemming	93.07%	96.77%	93.55%	95.13%
Twitter without stemming	96.38%	98.08%	96.90%	97.48%

Table 5.3: Performance scores for classification models tested on tweets without stemming

### 5.1.4 Evaluation

#### The criteria

When evaluating the classifier, the most important parts are the precision, recall and F1 scores. The Keras accuracy is regarded as an additional pointer and not the primary index of performance, although in our case the scores were often correlated.

In the scenario for this thesis, where the classifier would be implemented on a Twitter feed where most of the tweets received were already irrelevant, it is very detrimental to turn away any possible movie tweets, because they are rare. Therefore, it was determined as valuable for the classifier to score decent on both precision and recall, and not necessarily valuable to have a perfect precision score if that affected the recall score.

#### Testing on tweets with stemming

It is clear from both sets that stemming affects the accuracy of the models. In the first test, where models were tested on stemmed tweets, it is clear that both the SAR14 and the Twitter model trained without stemming performed worse than their stemmed counterparts did. Although they performed well

in precision when tested on tweets with stemming, the recall scores are much lower, resulting in a lower F1 score. The best model of this set is the model trained on tweets with stemming, getting a good precision score and a good recall, giving an F1 score of almost 97%.

### **Testing on tweets without stemming**

In the tests conducted on tweets without stemming, both models trained without stemming greatly improves. Both achieved a Keras accuracy score of almost 30% more than when tested on stemmed tweets. As expected, both models trained on tweets with stemming performs worse in this test. The best F1 score in this test was 97.48%.

### **Final evaluation**

From both these tests, it is clear that the Twitter trained models performs much better than the SAR14 trained models on both accounts. The highest score achieved is 97.48%, which is an unnaturally high score. This is most likely due to the fact that while the dataset has been split to ensure that the models were trained and tested on different tweets, the tweets in the training set still discusses the same six movies as the tweets in the testing set. It is natural to assume the scores would drop if the system was tested on tweets of other movies, as there would be a greater variance in words in each tweet. This variance would be in the form of different movie titles and would change the vector representation of each tweet. After running the classifier on an unfiltered, new dataset from Twitter, the classifier appears to be able to filter out tweets about movies, along with some noise from other tweets. It is therefore concluded that the tests holds enough validity to continue with the other parts of the system.

The two best results are achieved with a Twitter model either with or without stemming, as they both outperform the SAR14 models. One problem with stemming that was discovered during testing is that some stemmed words are less likely to be registered in the word2vec model. Any stemmed word not in that model results in a 0 vector, which could potentially skew the results. It was therefore decided to continue the thesis using the model trained without stemming, and to avoid problem in the future in regards to creating vector representations of stemmed words, the stemming step was removed from the pre-processing steps list found in table 4.2.

### **5.1.5 Conclusion**

The classifier is of course a very important part of the overall solution, because it decides what tweets the rest of the system will be working with. Because



of this, it is important that the classifier does not gather more "garbage", that is, irrelevant tweets, than necessary. The Twitter trained model without stemming provides an overall good accuracy, with high precision and good recall. It is also good enough without having to further tune the ANN, as testing showed different epochs and batch sizes does not affect the result in any big way.

Because of these tests, the suggested approach will be using the model trained on tweets without stemming, and will no longer stem any tweets that need a vector representation. It is concluded that the final results shown in table 5.4 is good enough to be usable, and will therefore used as the basis for the rest of the approach.

Model	Keras accuracy	Precision	Recall	F1 score
Twitter without stemming	96.38%	98.08%	96.90%	97.48%

Table 5.4: Accuracy measure for the chosen classifier model

## 5.2 Clustering

The focus of the clustering tests was to study if it was possible to group together tweets discussing the same movie, and then determine the title of the movie. The assumption made was that tweets about the same movie would be grouped together. Because of this, we assume that any tweets being sent to the clustering algorithm have been determined to be about one or more movies.

### 5.2.1 The dataset

In order to test the clustering algorithm and different distance thresholds, a dataset consisting of tweets about six different movies were used. These tweets came from the big dataset used to train the classifier. The amount were distributed in the way seen in table 5.5.

Movie	Quantity
Assasin's creed	11
Rouge One: A Star Wars Story	106
La la Land	102
Collateral Beauty	100
Nocturnal Animals	102
Doctor Strange	100
<b>Total</b>	<b>521</b>

Table 5.5: Dataset for clustering tests

In order to see the effect of tweets about movies with smaller coverage, one movie was purposefully represented with few tweets in order to see if the algorithm would group these tweets together or if this movie would be lost among the other movies with more representation. In addition, there were slight variances in the quantity of tweets about different movies because some tweets discussed more than one movie. We include these tweets to see how the algorithm clusters tweets that discuss more than one movie.

### 5.2.2 The Algorithm and parameters

The algorithm used for clustering can be found in section 4.3. For testing purposes, the two main parameters to focus on was the threshold value and the "refresh clusters" method. The threshold value indicated how far apart two data points could be and still be considered in a group, and the "refresh clusters" method was used to further allow for outliers in the dataset to become part of a cluster. The "refresh clusters" method is explained in detail in section 4.3, and its essential purpose is to check all nearby clusters whenever a new cluster is created in order to see if any of the neighbour clusters points should be reassigned to the newly created cluster. Testing on the "refresh clusters" method would focus on difference in results with and without this functionality, because it could be a time consuming calculation to perform, for a potentially small decrease in the number of outliers.

### 5.2.3 Testing

When testing the clustering algorithm it was interesting to look at several aspects. As discussed in section 4.3, precision, recall and F1 score would be used to measure the contents of the clusters and how well they gathered relevant information. It was also interesting to see how many clusters and outliers the algorithm created with different thresholds, because it would be relevant to mitigating the noisiness of Twitter data.

#### Clusters and outliers

Since the algorithm assigns every point to a cluster no matter how many points that cluster has, an outlier in the data was defined as a cluster with only a single point in it. Figure 5.1 shows how many actual clusters, defined as a cluster with more than one point, the algorithm created at different thresholds. Figure 5.2 shows how many outliers, or clusters with only one point, the algorithm created at different thresholds.

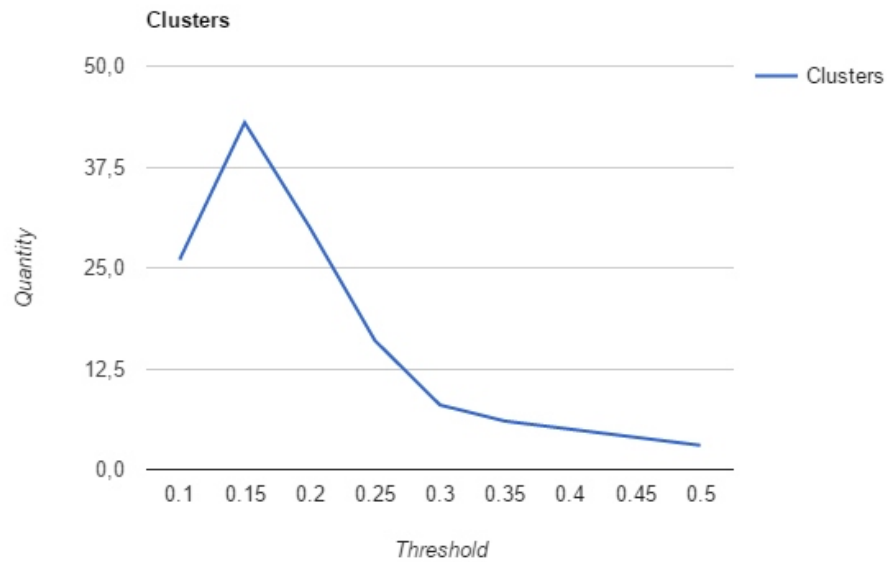


Figure 5.1: Number of clusters created at different thresholds

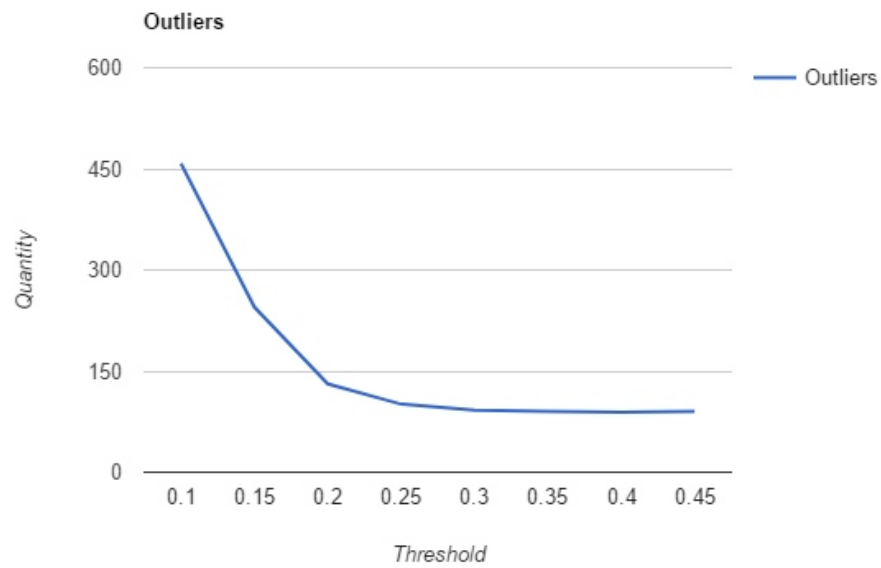


Figure 5.2: Number of outliers at different thresholds

### Quality of clusters

For these tests, the focus would be on thresholds 0.2, 0.25 and 0.3, as these thresholds created the most suitable amount of clusters for how many unique movies were being discussed in the dataset, which was six. This is explained further in section 5.2.4. To measure the quality of the clusters, the accuracy scores precision, recall and F1 would be used. The system created a text summary of each cluster which indicated what movie was in the majority in that cluster, and the accuracy scores of each cluster would be based on that movie being the target of the cluster. Only the result-table of the best threshold is presented in this section, the other tables can be found in their appropriate appendixes. To evaluate all thresholds equally, only the five biggest clusters at each threshold in terms of size, or amount of data points, were considered.

It is important to remember that for this thesis, a cluster had to have more than one data point in it, otherwise it was counted as an outlier. Because of this, only clusters containing more than one point were part of the results.

With a threshold of 0.2, seen in appendix A.1.1, the algorithm created a total of 30 clusters. Out of those 30 clusters, 20 had less than five members. The biggest cluster had no central movie. For example, it had 33 entries that mentioned "Collateral Beauty" and 46 entries which mentioned "Rogue One" and 36 entries about "Nocturnal Animals". For the sake of getting data comparable with other clusters, "Rogue One" was considered this clusters main movie, since it had the most entries. All other clusters pointed towards one specific movie through its textual summary. The average F1 score for this threshold was 0.43, and can be seen in appendix A.2.1.

For the results of threshold of 0.25, seen in appendix A.1.2, we experienced the same problem with the biggest cluster. It simply covered too much diverse data to have a "main" movie. The distribution of movies seemed to be close to the same as with a threshold of 0.2, in that "Rogue One" and "Nocturnal Animals" covered the biggest parts. Because of this, "Rogue One" was again considered the main movie. The total number of clusters with this threshold was 16, and 5 of these had 5 or fewer members. The average F1 score for this threshold was 0.45, and can be seen in appendix A.2.2.

In testing with a threshold of 0.3, seen in table A.3, the algorithm created a total of 8 clusters. Unlike the two previous thresholds, this threshold's summary of the biggest cluster indicated a majority of "Rogue One" tweets, meaning it would be used as the main movie for this cluster.

Cluster	Size	Movie	Precision	Recall	F1 score
1	238	Rogue One: A Star Wars Story	$71/238 = 0,30$	$71/106 = 0,67$	0,41
2	67	La La Land	$67/67 = 1$	$67/102 = 0,66$	0,80
3	42	Nocturnal Animals	$42/42 = 1$	$42/102 = 0,41$	0,58
4	37	Doctor Strange	$37/37 = 1$	$37/100 = 0,37$	0,54
5	26	Rogue One: A Star Wars Story	$26/26 = 1$	$26/106 = 0,25$	0,4

Table 5.6: Top five clusters with threshold 0.3

The average F1 score score for the top five clusters when threshold was 0.3:

$$Average = \frac{0.41 + 0.80 + 0.58 + 0.54 + 0.4}{5} = 0.55 \quad (5.1)$$

All three thresholds that were tested indicated that several of the tweets assigned to the largest cluster discussed more than one movie. These tweets that discussed more than one movie were rarely seen outside of the largest cluster.

### With and without refreshing clusters

This test was conducted to see the effect of the "refresh clusters" implementation in the clustering algorithm. It turned out that most of the results were very similar both with and without this functionality. As an example, table 5.7 presents the results of clustering the same dataset used in every other test on threshold 0.3, but executed both with and without refreshing the clusters whenever a new cluster was added.

Version	Number of clusters	Average cluster size	Size of biggest cluster
With refreshing clusters	8	53	238
Without refreshing clusters	8	53	250

Table 5.7: Difference in clustering algorithms

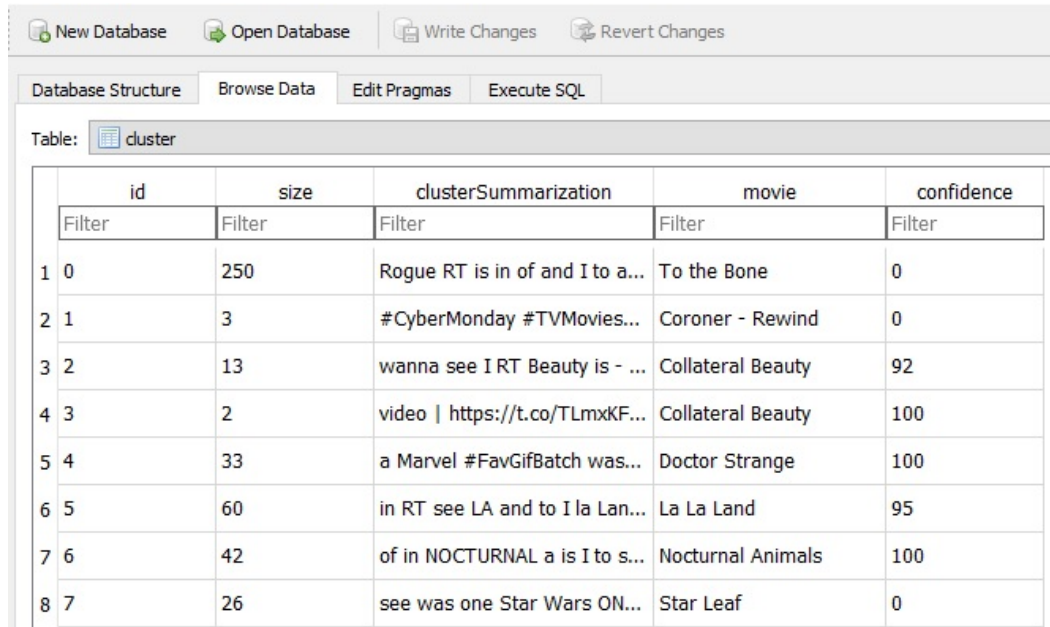
### Assigning movie titles to clusters

The result of assigning titles to the movies can be seen in figure 5.3. This was done on clusters created with a threshold of 0.3

## 5.2.4 Evaluation

### Clusters and outliers

The best-case scenario for the clustering method is to create exactly one cluster for each movie, and gather all relevant tweets in that cluster, thereby achieving a 100% precision and 100% recall. This is obviously very difficult, if not



	id	size	clusterSummarization	movie	confidence
	Filter	Filter	Filter	Filter	Filter
1	0	250	Rogue RT is in of and I to a...	To the Bone	0
2	1	3	#CyberMonday #TVMovies...	Coroner - Rewind	0
3	2	13	wanna see I RT Beauty is - ...	Collateral Beauty	92
4	3	2	video   <a href="https://t.co/TLmxKF...">https://t.co/TLmxKF...</a>	Collateral Beauty	100
5	4	33	a Marvel #FavGifBatch was...	Doctor Strange	100
6	5	60	in RT see LA and to I la Lan...	La La Land	95
7	6	42	of in NOCTURNAL a is I to s...	Nocturnal Animals	100
8	7	26	see was one Star Wars ON...	Star Leaf	0

Figure 5.3: Titles and confidence scores given to clusters made with a threshold of 0.3

impossible, to achieve with data as noisy as tweets. Because the ideal situation is to create one cluster for each movie, the recall scores given to each cluster is calculated independently from other clusters. At threshold 0.3, it can be argued that for the second "Rogue One" cluster in the table, the recall should be  $26/35$ , as that was the remaining amount of "Rogue One" tweets. Because we calculated the recall based on the total amount of "Rogue One" tweets and not the amount available to the cluster, the score was lower than if they were considered part of the same cluster.

As can be inferred from the clusters and outlier graphs, between thresholds of 0.2 and 0.3 the algorithm creates enough clusters to cover one or more for each movie, while also reaching a stable amount of outliers. For all thresholds past 0.25, the amount of outliers were constantly around 90. If the threshold is very low, like 0.1, very few clusters are created, as nearly all tweets are assigned as outliers. At threshold 0.15 the algorithm creates the most clusters, as more and more outliers are put together. However, having many small clusters is not ideal for this system. For example, if the system wants to see what people think about "La La Land", it has to analyse and assign sentiment scores for up to 40 different clusters, wasting a lot of time in database queries and requests to the sentiment classifier. It is therefore better to have fewer, more concentrated clusters, even if that means there is more noise in each cluster.

At the higher thresholds, once the algorithm applies a threshold value of 0.4 and above, there are fewer clusters created than there are unique movies in the

dataset, which is not ideal. The goal was to achieve at least one cluster for each major movie. Any fewer meant all other clusters would get a smaller precision score as other movie's tweets were assigned to that cluster. Once the threshold was set to 0.6, only one cluster was created. Therefore, a threshold between 0.2 and 0.3 was deemed the best choice. At 0.3, the algorithm created 8 clusters, and the summary created for each cluster revealed that most movies were represented by at least one cluster. As for outliers, the data shows a steady decrease throughout the different thresholds, and after 0.25 the amount of outliers reached a constant level. These are the reasons for only testing the quality of thresholds 0.2, 0.25 and 0.3.

### **Tweets about multiple movies and movies with few tweets discussing them**

The movie "Assassin's Creed" was intentionally underrepresented in the testing set to see how the algorithm would cluster movies with less attention than others. While most of the thresholds from 0.1 to 0.3 creates a cluster for this movie, they are usually small with only 2 or 3 members, giving a recall between 0.2 and 0.3. In fact, most of the tweets about this movie can be found in the biggest clusters. It is interesting to see that even at the low threshold of 0.2, none of the tweets about this movie was classified as outliers, meaning they are all assigned to a cluster with more than one member in it.

The tweets that discuss more than one movie are also usually clustered into the largest cluster, possibly because this cluster contained examples of most of the movies and therefore shows the most similarities with all tweets that discusses more than one movie.

### **Cluster quality**

We focused on thresholds between 0.2 and 0.3 for the quality analysis of the clusters. An important part of finding the correct threshold for the clustering algorithm, is finding what threshold has the best F1 score. To do this, the average F1 score of the top five clusters was calculated.

First, we consider the scores when the threshold was 0.2. The largest cluster was very unclean, with low precision and a decent recall, giving it an overall low F1 score. This presumably occurs because many of the tweets share many similar words and sentiment other than the title of the movie. This causes two tweets about different movies receiving a small cosine distance from one another, while two tweets about the same movie but portraying different sentiments can get a higher distance between them. This phenomenon of having one big "diverse" cluster was true for all thresholds, so it was decided not to put much weight on this when evaluating cluster quality at different thresholds. Threshold 0.2 was also the threshold with the highest number of

clusters, many of which were very small, with only 2-5 points in each. This gives a better precision for the small clusters, but gives a poor recall score, because all tweets about one movie is spread across multiple clusters.

When the threshold was 0.25, the overall quality of each cluster saw very small changes. The biggest difference in the two thresholds is the amount of clusters created, which in turn should mean more noise in other clusters. However, despite reducing the number of clusters from 31 to 16, the noise of the 5 biggest clusters did not increase at all. The exception was a slight decrease in precision in the largest cluster, though, the amount of relevant data points assigned to the largest cluster increased. The reason the precision declined despite the algorithm correctly assigning more correct data points, was that even more irrelevant points were assigned to the cluster, resulting in less precision but a higher recall. As we can see from the average F1 score for this threshold, the result is very similar to the previous threshold of 0.2.

Finally, the threshold was adjusted to a cosine distance of 0.3, which seemed to be the point where most outliers died off and we got close to one cluster for each movie. First off, the trend from earlier continued, the biggest cluster got even bigger. In fact, all clusters increased in size. This can result in increased recall but decreased precision. However, as can be seen in the results table, none of the clusters suffered decreased precision. With the exception of the largest cluster, they all stayed at a perfect 100%. Even the biggest cluster got increased precision, despite increasing drastically in size. If the size of the cluster is increased, but precision stays the same or is increased, it follows that the recall has to increase, because not only are we assigning more points to each cluster, we are assigning more *relevant* points. This is reflected in the F1 score scores, which are the highest of any of these three tests. Another positive factor for this threshold was that despite only creating eight clusters, all movies were in some degree represented by the clusters.

### **Refreshing clusters**

The refresh clusters method had a very small impact on our test set. Because this procedure was time and resource consuming, it would have to have a big effect in order to be worth it. It had the expected outcome, causing the biggest cluster to be smaller, since more points are redistributed after new clusters are created. However, as we can see, the result were very small, changing the size of the biggest cluster from 250 to 238.

### **Assigning titles to each cluster**

Overall, the method that assigned titles to clusters worked very well, only really limited by variables beyond the control of the system. For example, if the movie title was missing from the list we received from the API, there



was not much to do about it. One such limitation was that if the interval of months was too big, the system limits the number of pages retrieved from the API to 100, to not spend too much time receiving data. Even if the wrong titles was assigned to a cluster, it was clearly marked by the system because it also assigned it a confidence score of 0, clearly separating it from the correctly assigned titles. Because of the method the system uses to assign titles, comparing each word to a list of movie titles and finding the one that had the closest match, movies with long titles are often hard to correctly label. The reason for this is that long titles, like "Rogue One: A Star Wars Story" are often only discussed as "Rogue One". Since the full title of the movie contains six words and the system only found two of them in the text summary, the algorithm correctly calculated that it had a match of 33%. Meanwhile, another title like "One Night" scores a match of 50%, because the cluster summary contains half of the words in the title. Despite these problems, the system did assign the correct title in most cases.

### 5.2.5 Conclusion

The clustering section of the proposed solution was predicted to be the most difficult part of the project, since accurately separating different items in data as similar as tweets is challenging.

The result is satisfactory. We discovered that when the threshold was in the interval of 0.2 to 0.3, the algorithm creates a desirable amount of clusters and outliers.

We stated before that the ideal goal of the clustering algorithm is to create exactly one cluster for each movie, and ideally score perfect in precision and recall. In the end, with a threshold of 0.3, the algorithm created eight clusters, where all movies had at least one cluster to represent it. In addition, all clusters have equal or better precision scores than their counterparts from other thresholds, and nearly all of them receives drastically increased recall scores.

As for the "refresh clusters" method, it proved to have too small of an impact compared to the costs, and will not be implemented.

The title assigning works very well and can easily be implemented for future use in the system. Being able to determine the title of a movie makes the recommendation job and assigning a sentiment score easier.

It is important to remember that tweets are noisy and very unpredictable data. It might be that the results we have achieved from testing these six movies might not be representative for all other movies, as different movies tend to be talked about in very different ways. However, because it is impossible to test the clustering algorithm on all future movies, we need to draw a conclusion based on the data and results available to us.

The conclusion drawn from the clustering tests is this: the ideal threshold for these types of data in regard to ideal amount of clusters and outliers is between 0.2 and 0.3, and for these data the best F1 score was achieved with a threshold of 0.3. Refreshing clusters after creating a new cluster had relatively low impact on the result. Therefore, the basis for clustering should be with a threshold of 0.3 with no "refresh clusters" method.

## 5.3 Sentiment analysis

In this section, we present the tests and results from both the vivekn sentiment classifier API and the ANN sentiment classifier.

### 5.3.1 The dataset

The dataset we used for this testing was the Stanford sentiment140 dataset [11], described in detail in section 4.4. It contains 800,000 entries with negative sentiment and 800,000 entries with positive sentiment, giving a total dataset size of 1.6 million entries. The dataset consists of several values per entry: The sentiment score, the id of the tweet, the data of the tweet, the Twitter query used to get the tweet, the user who tweeted and the text of the tweet. Out of these attributes, only the sentiment score and the text was used.

### 5.3.2 Set-up

When testing the sentiment analysis part of the approach, we tested both the ANN classifier trained on the SAR14 dataset and the vivekn sentiment tool. The reason we wanted to test the vivekn API was to study how it performed with data it was not trained to classify, which was tweets, and to see if it gave good enough accuracy to be used instead of our own trained sentiment classifier.

#### Vivekn API

When using the vivekn API, the scores were set by our application to a value between -2 and 2, with -2 representing negative sentiment and 2 representing positive sentiment. For testing purposes, both scores of -2 and -1 are converted to 0, and scores of 1 and 2 are converted to 4, in order to match the format of the Stanford sentiment140 set. In addition, the vivekn API also returns "neutral", whenever the sentiment is not classified as either positive or negative. Whenever this occurred during testing, it was regarded as positive sentiment. This was done because we wanted the vivekn API method to produce the same

amount of classes as output as the artificial neural network classifier to be able to directly compare the two.

We then fed the vivekn API with entries from the Stanford sentiment140 dataset, and compared the result given by vivekn API and the sentiment determined by the Stanford classifier.

### ANN binary classifier

For comparing the ANN binary classifier to the Stanford sentiment140, a similar approach was used. The ANN classifier was trained on the SAR14 dataset and would output 0 for negative entries and 1 for positive entries. The sentiment140 used 0 for negative entries and 4 for positive. The positive entries were treated as 1 instead of 4 to allow for direct comparison with the output from the classifier. We then converted all of the dataset entries into vectors according to the pre-processing steps we found the most optimal when testing the movie classifier, meaning no stemming, and fed these to the ANN model. All other variables in the ANN, such as epochs, batch size and layers were all chosen based on what gave the best results when testing the movie classifier.

### 5.3.3 Results

The results of testing can be seen in table 5.8.

Analysis tool	Errors	Error percentage	Accuracy
Vivekn API	612784	38.3%	61.7%
ANN classifier	795120	49.7%	50.3%

Table 5.8: Sentiment tool testing

### 5.3.4 Evaluation

At first glance, the numbers and results in table 5.8 might seem unsatisfactory. However, the reality of sentiment analysis is that it can often be very difficult to determine, and determining the sentiment of tweets is no different.

Even after tuning the parameters, the ANN classifier performed very poorly. An accuracy of about 50%, when the outcome was one of two classes, meant the classifier was just guessing. There can be several reason for a performance this bad. The classifier could be trained on bad or unsuitable data, that being the SAR14 dataset. This might give the ANN incorrectly adjusted weights, because it was trained to classify sentiment in movie reviews and could therefore not perform very well when used on tweets. It could also be a problem with the size of the training set or that the data in the Stanford dataset contained too much noisy data.

The vivekn API performed significantly better than the ANN classifier, giving an accuracy of 61.7%. While this is not a great accuracy score, it validates the suspicion that using a pre-made tool for determining sentiment would be more useful for this thesis than creating one from scratch, since determining sentiment was not the focus of this thesis.

### **Taking the test set into consideration**

Some problems with the test data, the Stanford sentiment140 dataset, should be disclosed and discussed. The first problem is that the dataset contains several tweets that are so short they practically contain no real sentiment. One example of such a tweet, taken from April 6th 2009: "Up early". This tweet is classified as containing negative sentiment by the dataset, but to most humans it would probably be difficult to agree that this tweet portrays an exclusively negative sentiment. Another tweet in similar fashion comes from April 7th: "off to work". Again, the tweet is so short that determining an accurate sentiment should be impossible, but it is still classified as negative.

The reason for bringing these examples up was for us to consider that the test set was not by any means perfect. It was simply the best fitting dataset that could be used for the difficult task of evaluating the sentiment analysis section of our solution. However, the extremely sparse or misclassified tweets were far between in the dataset, and the results that the testing gave us should therefore be good enough to base a conclusion on.

### **5.3.5 Conclusion**

For the purpose of this thesis, the vivekn sentiment API gave good enough results to be able to use in further testing. Even though the accuracy was not that good compared to previous tests in both movie classification and clustering, sentiment analysis by its nature contains a bigger grey area with more unclear borders than topic classification or clustering. In addition, getting an accurate sentiment classification is still something worked on by researchers and there are currently no perfect solutions. It was therefore decided that instead of taking the time to create a better sentiment classifier ourselves, either by using an improved ANN or some other method, we would use the vivekn API as the sentiment analysis tool for this thesis.

In figure 5.4, we can see the result after the vivekn sentiment tool had analysed all points in the dataset used for testing the clustering algorithm.

Because the vivekn API was determined to be the best way to assign sentiment to our tweets, this was what we would be using when moving on to the final step of testing our solution, the two-step classification.

id	size	clusterSummarization	movie	confidence	sentiment
Filter	Filter	Filter	Filter	Filter	Filter
0	229	is Rogue RT of in and I to a the	Cezanne and I	0	51
5	68	RT in the see and to Land la I La	La La Land	95	19
2	13	the Baranski wanna Beauty RT Collateral - is coll...	Collateral Beauty	92	8
4	33	was @CumberbatchSpam Marvel to watching R...	Doctor Strange	100	5
3	2	(2016): TV Experience <a href="https://t.co/TLmxkF2a0k...">https://t.co/TLmxkF2a0k...</a>	Collateral Beauty	100	3
1	3	<a href="https://t.co/aEfxP2ZvTX">https://t.co/aEfxP2ZvTX</a> Series <a href="https://t.co/ht4F...">https://t.co/ht4F...</a>	Assassin's Creed	0	0
6	55	of animals and I a to see is Animals Nocturnal	Nocturnal Animals	100	-4
7	26	#RogueOne was Star one Wars ONE ROGUE On...	Rogue One: A Star Wars Story	0	-9

Figure 5.4: Clusters created at threshold 0.3 with sentiment assigned by vivekn API in the database

## 5.4 Two-step classification

During testing of two-step classification, the focus of the tests were on how splitting a list of tweets into two separate and sentiment sorted lists would affect the results of the clustering algorithm. This was based on a concern that the algorithm might cluster tweets based on sentiment instead of items. To find the optimal solution, we would compare the clusters created after sentiment splitting the tweets with the clusters we created without splitting the sentiment.

### 5.4.1 The dataset

To be able to compare clustering results with and without two-step classification, the dataset used when testing the clustering algorithm after one-step classification would be reused. The dataset was presented in table 5.5.

### 5.4.2 Set-up

The same clustering algorithm, presented in section 4.3, would be used to cluster the tweets after two-step classification. Clustering was done on both the list of negative tweets as well as the list of positive tweets. Afterwards, we compared the clusters to the clusters achieved when these lists were combined. The important factors when comparing the two results were: The average size of the clusters, the number of clusters created and the precision, recall and F1 score of the clusters. All these factors would also be tested at different thresholds, because the clustering algorithm might require a lower threshold when the tweets have already been split on sentiment.

When calculating the recall and precision, an important factor is the total quantity of tweets about each movie. For this testing, the total amount of

tweet about each movie would be the same as before, only spread across two different lists, one negative and one positive. That is the reason that the total quantity of tweets in the recall column is not equal to previous tests. For example, "Doctor Strange" might have 20 negative tweets about it, which would result in the positive recall being "X/80" instead of "X/100". In this example, the clustering with the negative tweets would have had a recall of "X/20".

### 5.4.3 Testing

#### Cluster sizes

In table 5.9 we can see the amount of clusters divided between positive and negative tweets at different thresholds.

Threshold	Positive clusters	Negative clusters
0.15	33	8
0.2	22	9
0.25	13	6
0.3	9	5

Table 5.9: Amount of clusters at different thresholds with two-step classification

As this table shows, at the lower thresholds, the algorithm created too many clusters. The result of this is that every cluster was very small, usually with an average of 2 or 3 points in each cluster.

As shown in table 5.10, the number of outliers fell when the threshold was increased, as was expected.

Threshold	Positive tweets assigned to cluster	Negative tweets assigned to cluster	Outliers
0.15	179	74	268
0.2	259	109	153
0.25	290	127	104
0.3	296	131	94

Table 5.10: Amount of tweets assigned to clusters

#### Cluster quality

As was the case when we performed the clustering tests with one-step classification, the dataset only contained tweets about six different movies. Having one cluster for each movie makes determining the aggregated sentiment much simpler, and therefore it would be easier to use the information in a recommender system. We should therefore strive to have a number of clusters similar to the number of movies. For two-step classification, the focus would be on the same thresholds as cluster after one-step classification, 0.2 to 0.3. The

reason for this was that these thresholds created the most suitable quantity of clusters for the amount of movies in the dataset. The sentiment split of each movie made by the vivekn sentiment tool can be seen in 5.11. The quality scores for the best threshold in the two-step classification is presented in this section, and the results for the other thresholds can be seen in appendix B.1.

Movie	Positive entries	Negative Entries
Rogue One: A Star Wars Story	70	36
La La Land	72	30
Doctor Strange	80	20
Collateral Beauty	81	19
Nocturnal Animals	66	36
Assassin's Creed	8	3
<b>Total</b>	<b>377</b>	<b>144</b>

Table 5.11: Movies split into sentiment

Cluster	Size	Movie	Precision	Recall	F1 score
1	135	Rogue One: A Star Wars Story	31/135 = 0.23	31/70 = 0.44	0.30
2	47	La La Land	47/47 = 1	47/72 = 0.65	0.79
3	30	Doctor Strange	30/30 = 1	30/80 = 0.38	0.55
4	24	Nocturnal Animals	24/24 = 1	24/66 = 0.36	0.53
5	12	Rogue One: A Star Wars Story	12/12 = 1	12/70 = 0.17	0.29

Table 5.12: Top five clusters with threshold 0.25 with two-step classification, positive sentiment

Cluster	Size	Movie	Precision	Recall	F1 score
1	87	Rogue One: A Star Wars Story	28/87 = 0.32	28/36 = 0.78	0.45
2	17	Nocturnal Animals	17/17 = 1	17/36 = 0.47	0.64
3	15	La La Land	15/15 = 1	15/30 = 0.50	0.67
4	4	Rogue One: A Star Wars Story	4/4 = 1	4/36 = 0.11	0.53
5	2	Collateral Beauty	2/2 = 1	2/19 = 0.11	0.20

Table 5.13: Top five clusters with threshold 0.25 with two-step classification, negative sentiment

The average F1 score for the best threshold is presented here, the average F1 scores for the other thresholds can be seen in appendix B.2. F1 score for positive clusters at 0.25:

$$AveragePositive = \frac{0.30 + 0.79 + 0.55 + 0.53 + 0.29}{5} = 0.49 \quad (5.2)$$

F1 score for negative clusters at 0.25:

$$AveragePositive = \frac{0.45 + 0.64 + 0.67 + 0.53 + 0.20}{5} = 0.50 \quad (5.3)$$

Total F1 score for clusters at 0.25:

$$AveragePositive = \frac{0.49 + 0.50}{2} = 0.50 \quad (5.4)$$

### Assigning movie titles to clusters

The system successfully assigned the title of the movie to most clusters, and the result can be seen in appendix B.3.

## 5.4.4 Evaluation

### Cluster sizes

The amount of outliers created by the algorithm after two-step classification were very similar to the amount produced with one-step classification. At the higher end of the thresholds tested, about 20% of the tweets were classified as outliers. Since both methods produce close enough to the same amount of outliers, this would not be a major consideration when evaluating the two methods against each other.

### Cluster quality

The best cluster quality from the one-step classification occurred at threshold 0.3 and gave an average F1 score of 0.53. This was the F1 score the two-step classifier needed to beat to be considered better than one-step classification. After testing thresholds from 0.2 to 0.3, a threshold of 0.25 gave the best F1 score. Before discussing the result itself, it is worth discussing why the threshold that gave the best result differs between one-step classification and two-step classification. Because the tweets had been split between lists of positive and negative sentiment, there were less extreme cases in both lists and more inherent similarity because one large factor dividing the tweets, the sentiment, had already been accounted for. Therefore, there would be less difference in two tweets from the same list, which meant the threshold for considering two points part of the same cluster could be lowered.

At threshold 0.25 the average F1 score is 0.49 for positive clusters and 0.50 for negative clusters, giving an average F1 score of 0.50. In general, large positive clusters had a worse recall score than the largest negative clusters. However, the negative clusters scored very low in recall in the smallest clusters because they only contained 5 or less entries of a movie, therefore giving a lower recall score than the positive clusters.

### Assigning movie to cluster

There were no real difference in assigning titles to clusters between one-step classification and two-step classification. It suffered the same problems as before, with longer movie titles being harder to guess because they were often referred to by shortened and unofficial names when discussed on Twitter.



### 5.4.5 Conclusion

There were no difference when it came to assigning movie titles to cluster, so this was not be considered when evaluating one-step classification against two-step classification. The amount of clusters was also ignored, as both methods proved to create an amount of clusters usable by a recommender system. Therefore, the most important quality measure between the two methods is the highest achieved F1 score score.

As stated in section 4.5, the reason for testing two-step classification was that the clustering algorithm might struggle to cluster tweets based on movies and not based on sentiment. For two-step classification to perform better than one-step classification it needs to outperform in terms of cluster quality, measured in precision, recall and the combined score F1. In table 5.14, the highest F1 score achieved has been marked in bold. It occurs when the system only does one-step classification and uses a clustering threshold of 0.3.

Classification method	Threshold	F1 score
One-step	0.2	0.43
One-step	0.25	0.45
<b>One-step</b>	<b>0.3</b>	<b>0.53</b>
Two-step	0.2	0.35
Two-step	0.25	0.50
Two-step	0.3	0.42

Table 5.14: F1 score across one-step and two-step classification and different thresholds quality

Because of this result, the conclusion is that the best way to make tweets usable to a recommender system is to first cluster all tweets, and afterwards determine the sentiment of each cluster.

## 5.5 Evaluating the overall solution

This section discusses whether the result produced by our system is usable by a recommender system, and presents the final system configurations that gave the best results after testing.

### 5.5.1 Final configuration

#### Movie classifier

The movie classifier is an artificial neural network with a sequential model consisting of three layers. It is trained on movie related tweets with no stemming.

## Clustering

The clustering algorithm clusters all tweets found by the classifier with a cosine distance threshold of 0.3, with a system for finding a textual summary for each cluster based on word frequency. This summary is used to determine the movie titles being discussed in the cluster and also assigns a confidence score to the cluster to represent how likely the assigned movie title is to be correct. This system successfully assigns titles to the majority of clusters.

## Sentiment analysis

The sentiment analysis occurs after the clustering algorithm and utilizes the vivekn API. Each tweet assigned to a cluster is given a sentiment score between -2 and 2, and the system then aggregates these scores to create sentiment scores for each cluster.

## Final result

The sentiment scores, in addition to the names of the movies, can be used to improve recommendations of content-based filtering systems in particular, as a content-based filtering system would be able to gather information about the items attributes from external sources, in addition to using the sentiment gathered from the tweets. In figure 5.5, we see the results written to the database after running all parts of the system.

	id	size	sterSummarizat	movie	confidence	sentiment
	Filter	Filter	Filter	Filter	Filter	Filter
1	3	2	(2016): TV Ex...	Collateral Bea...	100	3
2	4	33	was @Cumbe...	Doctor Strange	100	5
3	6	55	of animals an...	Nocturnal Ani...	100	-4
4	5	68	RT in the see ...	La La Land	95	19
5	2	13	the Baranski ...	Collateral Bea...	92	8
6	0	229	is Rogue RT o...	Cezanne and I	0	51
7	1	3	<a href="https://t.co/a...">https://t.co/a...</a>	Assassin's Cr...	0	0
8	7	26	#RogueOne ...	Rogue One: A...	0	-9

Figure 5.5: Final result after clustering and assigning movie titles and sentiment to all clusters

## 5.5.2 Significance of results

### For usage in recommender systems

The focus of this thesis has been mitigating the cold-start problem by using data available on social media. Specifically, to help content-based filtering systems with cold items. This has been achieved by creating a system that is able to collect entries relevant for our domain from social media, categorize what items in this domain these entries are discussing and aggregating the sentiment for each item. By presenting a recommender system with a list of items and a sentiment belonging to those items, in addition to micro-reviews in the form of tweets for each item, the recommender system would no longer have to use a generic cold-start solution or simply guess. It will now have concrete data to use in its recommendation. This means that recommender systems can make an educated estimation when it comes to the popularity of brand new items. In addition, this system can in a majority of cases, determine the exact title of a movie in a cluster of tweets. This means that it can easily gather additional information about that movie from open movie databases, which can be used to further improve recommendations based on attributes.

In summary, the results from the tests performed in this thesis indicates that a recommender system can gather information about new items straight from social media. By doing this, the recommender system would be able to accurately arrange the items in order of sentiment and thereby get an even better recommendation to its users and would help users determine whether or not new items are well received by the general public.

## 5.5.3 Comparing to previous studies

### Recommendations based on tweets

In [3], a system was made which calculated trust between users on Twitter and they used this trust to adjust recommendations by weighing opinions from different users unequally. The benefit of the system presented in this thesis compared to the one presented in [3] is that our system does not require any external information about the user. All information is focused around the items, which is more easily gathered on social media. The system presented in this thesis focuses on improving recommendations on cold items for all users instead of improving recommendation for cold users.

### Categorizing and clustering data

One of the main challenges this thesis faced was gathering and classifying large quantities of *relevant* data. In chapter 3 some benchmarks were presented, although they were not directly comparable to this thesis. The accuracy scores

mentioned were an F1 score of 84% from [14] and 77% from [2]. The classifier presented in this thesis achieved 97.48% accuracy after being trained on tweets on specific movies, when tested on tweets discussing those same movies.

During the clustering tests, our solution performed worse than the one presented in [30], where they reported between 60% and 70% F1 score. The best F1 score achieved in this thesis was 0.53, or 53%. This lower score was largely due to worse recall and multiple clusters for a single movie, as the precision scores were usually always 100%.

### **Sentiment analysis**

For the sentiment analysis, this system uses the open API made by one of the authors of [22]. While the API achieved an accuracy of 90% in their report, our tests indicated a lower accuracy of 61%. The reason for this is most likely that the system in the report was trained on IMDB movie reviews, which means it is more suited for longer texts with more formal language. It still performed better than the sentiment analysis tool made for this thesis, which only achieved an accuracy of 51%.

### **Conclusion of comparison**

The results from the tests signifies that collecting data from Twitter about a specific domain is possible without the need for specific queries. No need for specific queries means the system is more versatile and less vulnerable to change in the recommender systems domain. For example, in movie recommendations, there is no need to update the query to ask for recently released movies, which requires an updated list on recently released movies. It also means that a classifier could be trained on automatically collected Twitter data and still produce a decent result, meaning that creating a dataset to train the classifier does not need to be as big of a challenge as it might seem. This in turn indicates that this system can be expanded to domains other than movie recommendations, as long as there is sufficient discussion about the domain on social media.

The fact that the clustering method presented in this thesis scored as high of an F1 score as it did indicates that an automated system would be able to determine what different items the reviews from social media were discussing. This is important because it otherwise will be very difficult to utilize the information the classifier gathers. Because the system is able to cluster the different reviews, determining sentiment and additional information about each item becomes easier. This especially helps content-based filtering systems, because accurately determining the name of an item allows the system to gather additional information through different tools about each item to further improve recommendations.

### 5.5.4 Known weak points of the tests

#### Movie tweet classification

The classification process' greatest challenge came from the fact that there was no perfectly fitting dataset to use to train the ANN classifier. This could cause problems, as there might not be enough variety in our dataset, both in terms of what movies and movie genres were represented. This manifested in an unusually high F1 score, most likely because of the repeated specific words in movie titles. That means that when the system is run on tweets discussing other movies that the five it was trained on, it would likely perform worse. However, after watching the classifier work on an unfiltered Twitter stream, it appeared to perform well, filtering out most tweets associated with movies.

#### Clustering

Clustering using the algorithm described in this thesis is dependent on a clustering threshold. During testing, several different thresholds were tested and the most effective one for our data was determined to be a cosine distance of 0.3. This threshold could vary depending on the types of data the algorithm is meant to cluster. For example, if the data is extremely similar, it will be harder for the algorithm to separate between different entries, meaning a lower threshold would have to be used to get smaller effective clusters, and not one large cluster. Similarly, larger differences in data would require a bigger clustering threshold.

Another problem with clustering was that the largest cluster always consisted of tweets about different movies. Both tweets that discussed more than one movie and tweets discussing different movies were grouped together in the largest cluster, and no solution was found for this problem. This problem persisted through all threshold tests and both with and without the "refresh clusters" method. These clusters whose data points discuss many different movies are hard to make useful for a recommender system. One possible solution for this problem is to ignore clusters without sufficient confidence in assigned movie title and base recommendations on the smaller and purer clusters.

#### Sentiment analysis

The sentiment analysis went through the most extensive testing, as determining an accurate sentiment is very important for a recommender system. The vivekn sentiment API was found to be the more accurate of the two methods tested, the other being the ANN trained on the SAR14 dataset. The problem with using an external API for something as important as sentiment analysis is that something might change in the API, causing our output to change.

It is clear that the optimal solution would have been to be able to create a sentiment analysis tool ourself, but this was simply too comprehensive of a task considering that it was not the main focus of this thesis.

---

# Chapter 6

## Conclusion and Future Work

This chapter concludes the work conducted in this thesis and proposes future work. Section 6.1 presents the conclusion, the contributions of this thesis and answers the research questions set in section 1.5. In section 6.2 we propose what work could be done to further improve the system.

### 6.1 Conclusion

In this thesis, we have shown that social media can be used to improve the cold-start problem of recommender systems. We have shown that through Twitter, it is possible to gather enough accurate data to be able to determine not only the sentiment of the item being discussed, but also the name of the item. This is a great step towards mitigating the cold item problem in particular.

This has been demonstrated through training a classifier on relevant data with the correct media and domain, which in our case meant tweets about movies. Such a dataset could be easily created by automatically gathering data through the Twitter API with relevant search queries. After classifying entries relevant to our domain, this thesis has shown that a clustering algorithm based around cosine distance with updating cluster centroids can be effective enough for the system to be able to determine the exact titles of the items in a majority of cases. This information could easily be used by both collaborative filtering systems to discover new movies worth recommending, and perhaps even more suiting, by content-based filtering systems. Content-based filtering systems would be able to gather any relevant attributes of a movie using external sources, and use the sentiment determined by the tweets to weigh movies with the same attributes differently.

### 6.1.1 Contributions of the Thesis

#### Data representation and tweet classification

This thesis has shown that artificial neural networks receiving tweets represented by word2vec vectors can be used for binary classification of specific topics. The training set can be created by gathering data automatically from the media it will be working with. The data must come from the same media and not just contain discussions about the same topic. This was discovered when a classifier trained on topically relevant data such as movie reviews gave a worse accuracy compared to a classifier trained on actual movie tweets.

#### Clustering tweets with cosine distance to discover items

Clustering data points based on cosine distance can be used to effectively separate items in the data into different clusters. These clusters will generally score high in precision and lower in recall. To discover what items are being discussed, there was no need in our case to prepare for clustering in the form of sentiment splitting the data, as a sentiment sorted dataset scored a lower F1 score than the unsorted dataset.

#### Item detection

By using word frequency and a list of recently released movies it is possible to determine the title of a movie discussed in a cluster based on the text summary of that cluster in a majority of cases. Using this technique, the title will be more difficult to determine the longer the title is, or if it is otherwise discussed by another name than its official name.

### 6.1.2 Answering research questions

- **RQ1:** *How can social media be used to mitigate the cold-start problem?*

By classifying and collecting relevant tweets from Twitter, we were able to cluster movie specific tweets in such a way that allowed us to get usable information on specific items.

- **RQ1.1:** *How effectively can posts that relate to a recommender systems domain be filtered out on social media?*

By using an artificial neural network binary classifier trained on a dataset consisting of data relevant to our domain from the same social media we intend to gather information from represented by a word2vec vector, we were able to effectively sort relevant entries from the irrelevant ones. It was shown that the effect was greatest when the network was trained on



data from the same social media as the one it would be classifying data from.

- **RQ1.2:** *How can sentiment about specific items be extracted through these posts?*

By clustering all relevant entries, we were able to group most entries discussing the same item together. The system was usually able to determine the name of the item, meaning that sentiment values from multiple clusters discussing the same item would all be counted towards the final score for that item. Afterwards, we found the sentiment score of each cluster using an external API and could therefore see what items received the highest and lowest sentiment.

## 6.2 Future work

To improve and expand the solution, we present the following suggestions as possible future work.

### **Further improve accuracy of movie tweet classifier**

The classifier used to differentiate between tweets discussing movies and all other tweets were an important part of the system. Watching the classifier work through a Twitter stream made it clear that most of the filtered tweets were movie related, but far from all, despite the high F1 score achieved by the classifier. To improve the classifier, we suggest changing the size and content of the test set to include more tweets and more movies. This is hard to do during a short period because most movies discussed on Twitter are recently released titles. This makes it necessary to collect tweets over several months in order to cover as many movies as possible.

### **Improve title assigning to clusters**

Although our solution is able to gather movie related tweets without a list of recently released movie, such a list is still used to determine the title of the movie in each cluster. To improve item name assigning, we propose a solution that does not rely on a list of possible names, but instead looks for a single common phrase in all entries in a cluster. This is likely to be a very difficult task, but one that would give the system much more flexibility, as it would no longer require information from external sources to determine item names.

### **Create a better sentiment analysis tool**

The approach presented in this thesis ended up using an external API trained on movie reviews for sentiment classification. While this API was trained on data somewhat relevant to movie tweets, it would be better to create a classifier trained on actual movie tweets. If no such dataset can be made, it might be

possible to achieve better results using a technique other than an artificial neural network. By achieving a more accurate sentiment score for each item, the system can give recommendations that are more accurate. It would also remove any dependency on external APIs, allowing for a more independent system.

**Attribute extraction** To give users of recommender systems more information when they are presented with recommendations, we propose a way for the system to extract the most important attributes of the movie through tweets. This can be repeated comments from multiple tweets, such as "bad acting" or "great directing". This would give the users more information about the items, and they can prioritize what attributes are important to them, which in turn can give a recommender system implicit feedback about the user's taste.

---

# Appendices



---

# Appendix A

## One-step classification clustering tests

### A.1 Cluster quality

#### A.1.1 Threshold 0.2

Cluster	Size	Movie	Precision	Recall	F1 score
1	162	Rogue One: A Star Wars Story	$46/162 = 0,28$	$46/106 = 0,43$	0,34
2	47	La La Land	$47/47 = 1$	$47/102 = 0,46$	0,63
3	30	Nocturnal Animals	$30/30 = 1$	$30/102 = 0,29$	0,45
4	23	La La Land	$23/23 = 1$	$23/102 = 0,23$	0,37
5	21	Doctor Strange	$21/21 = 1$	$21/100 = 0,21$	0,35

Table A.1: Top five clusters with threshold 0.2

#### A.1.2 Threshold 0.25

Cluster	Size	Movie	Precision	Recall	F1 score
1	201	Rogue One: A Star Wars Story	$54/201 = 0,27$	$54/106 = 0,51$	0,35
2	45	La La Land	$45/45 = 1$	$45/102 = 0,44$	0,61
3	41	Nocturnal Animals	$41/41 = 1$	$41/102 = 0,40$	0,57
4	21	Doctor Strange	$21/21 = 1$	$21/100 = 0,21$	0,35
5	19	La La Land	$19/19 = 1$	$19/102 = 0,19$	0,32

Table A.2: Top five clusters with threshold 0.25

### A.1.3 Threshold 0.3

Cluster	Size	Movie	Precision	Recall	F1 score
1	238	Rogue One: A Star Wars Story	$71/238 = 0,30$	$71/106 = 0,67$	0,41
2	67	La La Land	$67/67 = 1$	$67/102 = 0,66$	0,80
3	42	Nocturnal Animals	$42/42 = 1$	$42/102 = 0,41$	0,58
4	37	Doctor Strange	$37/37 = 1$	$37/100 = 0,37$	0,54
5	26	Rogue One: A Star Wars Story	$26/26 = 1$	$19/106 = 0,18$	0,31

Table A.3: Top five clusters with threshold 0.3

## A.2 Average F1 scores

### A.2.1 Average F1 of top five clusters 0.2

The average F1 score score for the top five clusters when threshold was 0.2:

$$Average = \frac{0.34 + 0.63 + 0.45 + 0.37 + 0.35}{5} = 0.43 \quad (\text{A.1})$$

### A.2.2 Average F1 of top five clusters 0.25

The average F1 score score for the top five clusters when threshold was 0.25:

$$Average = \frac{0.35 + 0.61 + 0.57 + 0.35 + 0.32}{5} = 0.44 \quad (\text{A.2})$$

### A.2.3 Average F1 of top five clusters 0.3

The average F1 score score for the top five clusters when threshold was 0.3:

$$Average = \frac{0.41 + 0.80 + 0.58 + 0.54 + 0.31}{5} = 0.53 \quad (\text{A.3})$$

---

# Appendix B

## Two-step classification clustering tests

### B.1 Cluster quality

#### B.1.1 Threshold 0.2

Cluster	Size	Movie	Precision	Recall	F1 score
1	124	Nocturnal Animals	$33/124 = 0.27$	$33/66 = 0.50$	0.35
2	28	La La Land	$28/28 = 1$	$28/72 = 0.39$	0.56
3	22	Doctor Strange	$22/22 = 1$	$22/80 = 0.28$	0.44
4	14	Rogue One: A Star Wars Story	$14/14 = 1$	$14/70 = 0.20$	0.33
5	11	Nocturnal Animals	$11/11 = 1$	$11/66 = 0.17$	0.29

Table B.1: Top five clusters with threshold 0.2 with two-step classification, positive sentiment

Cluster	Size	Movie	Precision	Recall	F1 score
1	77	Rogue One: A Star Wars Story	$24/77 = 0.31$	$24/36 = 0.67$	0.42
2	12	Nocturnal Animals	$12/12 = 1$	$12/36 = 0.33$	0.50
3	5	Nocturnal Animals	$5/5 = 1$	$5/36 = 0.14$	0.25
4	4	La La Land	$4/4 = 1$	$4/30 = 0.13$	0.23
5	3	Rogue One: A Star Wars Story	$3/3 = 1$	$3/36 = 0.08$	0.15

Table B.2: Top five clusters with threshold 0.2 with two-step classification, negative sentiment

#### B.1.2 Threshold 0.25

Cluster	Size	Movie	Precision	Recall	F1 score
1	135	Rogue One: A Star Wars Story	$31/135 = 0.23$	$31/70 = 0.44$	0.30
2	47	La La Land	$47/47 = 1$	$47/72 = 0.65$	0.79
3	30	Doctor Strange	$30/30 = 1$	$30/80 = 0.38$	0.55
4	24	Nocturnal Animals	$24/24 = 1$	$24/66 = 0.36$	0.53
5	12	Rogue One: A Star Wars Story	$12/12 = 1$	$12/70 = 0.17$	0.29

Table B.3: Top five clusters with threshold 0.25 with two-step classification, positive sentiment

Cluster	Size	Movie	Precision	Recall	F1 score
1	87	Rogue One: A Star Wars Story	$28/87 = 0.32$	$28/36 = 0.78$	0.45
2	17	Nocturnal Animals	$17/17 = 1$	$17/36 = 0.47$	0.64
3	15	La La Land	$15/15 = 1$	$15/30 = 0.50$	0.67
4	4	Rogue One: A Star Wars Story	$4/4 = 1$	$4/36 = 0.11$	0.53
5	2	Collateral Beauty	$2/2 = 1$	$2/19 = 0.11$	0.20

Table B.4: Top five clusters with threshold 0.25 with two-step classification, negative sentiment

### B.1.3 Threshold 0.3

Cluster	Size	Movie	Precision	Recall	F1 score
1	165	Rogue One: A Star Wars Story	$50/165 = 0,30$	$50/70 = 0,71$	0,42
2	51	La La Land	$51/51 = 1$	$51/72 = 0,71$	0,83
3	31	Doctor Strange	$31/31 = 1$	$31/80 = 0,39$	0,56
4	25	Nocturnal Animals	$25/25 = 1$	$25/66 = 0,38$	0,55
5	9	Rogue One: A Star Wars Story	$9/9 = 1$	$9/70 = 0,13$	0,23

Table B.5: Top five clusters with threshold 0.3 with two-step classification, positive sentiment

Cluster	Size	Movie	Precision	Recall	F1 score
1	109	Rogue One: A Star Wars Story	$30/109 = 0,28$	$30/36 = 0,83$	0,42
2	15	La La Land	$15/15 = 1$	$15/30 = 0,50$	0,67
3	3	Rogue One: A Star Wars Story	$3/3 = 1$	$3/36 = 0,08$	0,15
4	2	Collateral Beauty	$2/2 = 1$	$2/19 = 0,11$	0,20
5	2	Rogue One: A Star Wars Story	$2/2 = 1$	$2/36 = 0,05$	0,10

Table B.6: Top five clusters with threshold 0.3 with two-step classification, negative sentiment

## B.2 Average F1 scores

### B.2.1 Threshold 0.2

F1 score for positive clusters at 0.2:

$$AveragePositive = \frac{0.35 + 0.56 + 0.44 + 0.33 + 0.29}{5} = 0.39 \quad (B.1)$$

F1 score for negative clusters at 0.2:

$$AveragePositive = \frac{0.42 + 0.50 + 0.25 + 0.23 + 0.15}{5} = 0.31 \quad (B.2)$$


---



Total F1 score for clusters at 0.2:

$$AveragePositive = \frac{0.39 + 0.31}{2} = 0.35 \quad (B.3)$$

### B.2.2 Threshold 0.25

F1 score for positive clusters at 0.25:

$$AveragePositive = \frac{0.30 + 0.79 + 0.55 + 0.53 + 0.29}{5} = 0.49 \quad (B.4)$$

F1 score for negative clusters at 0.25:

$$AveragePositive = \frac{0.45 + 0.64 + 0.67 + 0.53 + 0.20}{5} = 0.50 \quad (B.5)$$

Total F1 score for clusters at 0.25:

$$AveragePositive = \frac{0.49 + 0.50}{2} = 0.50 \quad (B.6)$$

### B.2.3 Threshold 0.3

F1 score for positive clusters at 0.3:

$$AveragePositive = \frac{0.42 + 0.83 + 0.56 + 0.55 + 0.23}{5} = 0.52 \quad (B.7)$$

F1 score for negative clusters at 0.3:

$$AverageNegative = \frac{0.42 + 0.67 + 0.15 + 0.20 + 0.10}{5} = 0.31 \quad (B.8)$$

Total F1 score for clusters at 0.3:

$$AverageTotal = \frac{0.52 + 0.31}{2} = 0.42 \quad (B.9)$$

## B.3 Movie title detection

APPENDIX B. TWO-STEP CLASSIFICATION CLUSTERING TESTS

	id	size	sterSummarizat	movie	confidence
	Filter	Filter	Filter	Filter	Filter
1	0	109	Rogue La is N...	La La Land	0
2	2	15	Vimeo Going ...	La La Land	93
3	3	3	watch @AyaL...	One Night	0
4	1	2	beauty' I beau...	Collateral Bea...	100
5	4	2	The begins... ...	The Accountant	0

Figure B.1: Titles and confidence scores given to clusters made with a threshold of 0.3, positive sentiment

	id	size	sterSummarizat	movie	confidence
	Filter	Filter	Filter	Filter	Filter
1	0	165	I is Rogue an...	Cezanne and I	0
2	5	51	to LA RT I Lan...	La La Land	95
3	4	31	was a #FavGi...	Doctor Strange	100
4	6	25	animals 'Noct...	Nocturnal Animals	100
5	7	9	WAS One #ro...	One Night	0
6	3	7	@YouTube htt...	Collateral Beauty	86
7	1	3	https://t.co/P...	Assassin's Creed	0
8	8	3	#rogueoneast...	Rogue One: A St...	0
9	2	2	(2016) Spot h...	Collateral Beauty	100

Figure B.2: Titles and confidence scores given to clusters made with a threshold of 0.3, negative sentiment

## Bibliography

- [1] G. Adomavicius and A. Tuzhilin. “Toward the next generation of recommender systems: a survey of the state-of-the-art and possible extensions”. In: *IEEE Transactions on Knowledge and Data Engineering* 17.6 (2005), pp. 734–749. ISSN: 1041-4347. DOI: 10.1109/TKDE.2005.99.
- [2] M. Ahmed et al. “Using Crowd-Source Based Features from Social Media and Conventional Features to Predict the Movies Popularity”. In: *2015 IEEE International Conference on Smart City/SocialCom/SustainCom (SmartCity)*. Dec. 2015, pp. 273–278. DOI: 10.1109/SmartCity.2015.83.
- [3] D. H. Alahmadi and X. J. Zeng. “Twitter-Based Recommender System to Address Cold-Start: A Genetic Algorithm Based Trust Modelling and Probabilistic Sentiment Analysis”. In: *Tools with Artificial Intelligence (ICTAI), 2015 IEEE 27th International Conference on*, pp. 1045–1052. ISBN: 1082-3409. DOI: 10.1109/ICTAI.2015.149.
- [4] James Allan et al. “Detections, bounds, and timelines: Umass and tdt-3”. In: *Proceedings of topic detection and tracking workshop*. 2000, pp. 167–174.
- [5] Stefano Baccianella, Andrea Esuli, and Fabrizio Sebastiani. “SENTIWORDNET 3.0: An Enhanced Lexical Resource for Sentiment Analysis and Opinion Mining”. In: (2006).
- [6] Yoshua Bengio et al. “A neural probabilistic language model”. In: *journal of machine learning research* 3.Feb (2003), pp. 1137–1155.
- [7] Yubo Chen and Jinhong Xie. “Online consumer review: Word-of-mouth as a new element of marketing communication mix”. In: *Management science* 54.3 (2008), pp. 477–491.
- [8] Paolo Cremonesi et al. “TV Program Detection in Tweets”. In: *Proceedings of the 11th European Conference on Interactive TV and Video*. EuroITV '13. Como, Italy: ACM, 2013, pp. 45–54. ISBN: 978-1-4503-1951-5. DOI: 10.1145/2465958.2465960. URL: <http://doi.acm.org/10.1145/2465958.2465960>.

- [9] Simon Doods, Toon De Pessemier, and Luc Martens. “Movietweetings: a movie rating dataset collected from twitter”. In: *Workshop on Crowdsourcing and human computation for recommender systems, CrowdRec at RecSys*. Vol. 2013. 2013, p. 43.
- [10] John R. Firth. “A synopsis of linguistic theory, 1930-1955”. In: *Studies in Linguistic Analysis*. Special volume of the Philological Society. Oxford: Blackwell, 1957, pp. 1–32.
- [11] Alec Go, Richa Bhayani, and Lei Huang. “Twitter sentiment classification using distant supervision”. In: *CS224N Project Report, Stanford 1.12* (2009).
- [12] Frédéric Godin et al. “Multimedia Lab@ ACL W-NUT NER Shared Task: Named Entity Recognition for Twitter Microposts using Distributed Word Representations”. In: *ACL-IJCNLP 2015* (2015), p. 146.
- [13] Thorsten Hennig-Thurau, Caroline Wiertz, and Fabian Feldhaus. “Does Twitter matter? The impact of microblogging word of mouth on consumers’ adoption of new movies”. In: *Journal of the Academy of Marketing Science* 43.3 (2015), pp. 375–394. ISSN: 1552-7824. DOI: 10.1007/s11747-014-0388-3. URL: <http://dx.doi.org/10.1007/s11747-014-0388-3>.
- [14] U. R. Hodeghatta. “Sentiment analysis of Hollywood movies on Twitter”. In: *Advances in Social Networks Analysis and Mining (ASONAM), 2013 IEEE/ACM International Conference on*. Aug. 2013, pp. 1401–1404. DOI: 10.1145/2492517.2500290.
- [15] Vasu Jain. “Prediction of movie success using sentiment analysis of tweets”. In: *The International Journal of Soft Computing and Software Engineering* 3.3 (2013), pp. 308–313.
- [16] Daniel Jurafsky and James H. Martin. *Speech and Language Processing (2Nd Edition)*. Upper Saddle River, NJ, USA: Prentice-Hall, Inc., 2009. ISBN: 0131873210.
- [17] S. Lai et al. “How to Generate a Good Word Embedding?” In: *IEEE Intelligent Systems* PP.99 (2016), pp. 1–1. ISSN: 1541-1672. DOI: 10.1109/MIS.2016.45.
- [18] Jovian Lin et al. “Addressing Cold-start in App Recommendation: Latent User Models Constructed from Twitter Followers”. In: *Proceedings of the 36th International ACM SIGIR Conference on Research and Development in Information Retrieval*. SIGIR ’13. Dublin, Ireland: ACM, 2013, pp. 283–292. ISBN: 978-1-4503-2034-4. DOI: 10.1145/2484028.2484035. URL: <http://doi.acm.org/10.1145/2484028.2484035>.

- [19] Bing Liu. “Sentiment analysis and opinion mining”. In: *Synthesis lectures on human language technologies* 5.1 (2012), pp. 1–167.
- [20] Pasquale Lops, Marco de Gemmis, and Giovanni Semeraro. “Content-based Recommender Systems: State of the Art and Trends”. In: *Recommender Systems Handbook*. Ed. by Francesco Ricci et al. Boston, MA: Springer US, 2011, pp. 73–105. ISBN: 978-0-387-85820-3. DOI: 10.1007/978-0-387-85820-3\_3. URL: [http://dx.doi.org/10.1007/978-0-387-85820-3\\_3](http://dx.doi.org/10.1007/978-0-387-85820-3_3).
- [21] Tomas Mikolov et al. “Efficient Estimation of Word Representations in Vector Space”. In: *CoRR* abs/1301.3781 (2013). URL: <http://arxiv.org/abs/1301.3781>.
- [22] Vivek Narayanan, Ishan Arora, and Arjun Bhatia. “Fast and accurate sentiment classification using an enhanced Naive Bayes model”. In: *CoRR* abs/1305.6143 (2013). URL: <http://arxiv.org/abs/1305.6143>.
- [23] Dai Quoc Nguyen et al. “Sentiment classification on polarity reviews: an empirical study using rating-based features”. In: (2014).
- [24] Seung-Taek Park et al. “NaïVe Filterbots for Robust Cold-start Recommendations”. In: *Proceedings of the 12th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. KDD '06. Philadelphia, PA, USA: ACM, 2006, pp. 699–705. ISBN: 1-59593-339-5. DOI: 10.1145/1150402.1150490. URL: <http://doi.acm.org/10.1145/1150402.1150490>.
- [25] Jeffrey Pennington, Richard Socher, and Christopher D Manning. “Glove: Global Vectors for Word Representation.” In: *EMNLP*. Vol. 14. 2014, pp. 1532–1543.
- [26] Saša Petrović, Miles Osborne, and Victor Lavrenko. “Streaming first story detection with application to twitter”. In: *Human Language Technologies: The 2010 Annual Conference of the North American Chapter of the Association for Computational Linguistics*. Association for Computational Linguistics. 2010, pp. 181–189.
- [27] Øystein Repp. “Event Detection in Social Media ”. MA thesis. Norway: Norwegian University of Science and Technology, 2016.
- [28] Berthier Ribeiro and Ricardo Baeza. *Modern information retrieval*. Upper Saddle River, NJ: Pearson Higher Education, 2010. ISBN: 978-0-321-41691-9.

- [29] Francesco Ricci, Lior Rokach, and Bracha Shapira. *Recommender Systems: Introduction and Challenges*. Recommender Systems Handbook. Boston, MA: Springer US, 2015, pp. 1–34. ISBN: 978-1-4899-7637-6. DOI: 10.1007/978-1-4899-7637-6\_1. URL: [http://dx.doi.org/10.1007/978-1-4899-7637-6\\_1](http://dx.doi.org/10.1007/978-1-4899-7637-6_1).
- [30] Kevin Dela Rosa et al. “Topical clustering of tweets”. In: *Proceedings of the ACM SIGIR: SWSM* (2011).
- [31] Badrul Sarwar et al. “Item-based Collaborative Filtering Recommendation Algorithms”. In: *Proceedings of the 10th International Conference on World Wide Web. WWW '01*. Hong Kong, Hong Kong: ACM, 2001, pp. 285–295. ISBN: 1-58113-348-0. DOI: 10.1145/371920.372071. URL: <http://doi.acm.org/10.1145/371920.372071>.
- [32] Shai Shalev-Shwartz and Shai Ben-David. *Understanding machine learning: From theory to algorithms*. Cambridge University Press, 2014.
- [33] *Statistics twitter user data*. <https://www.statista.com/statistics/282087/number-of-monthly-active-twitter-users/>. Accessed: 2016-03-10.
- [34] L. Venkata Subramaniam et al. “A Survey of Types of Text Noise and Techniques to Handle Noisy Text”. In: *Proceedings of The Third Workshop on Analytics for Noisy Unstructured Text Data. AND '09*. Barcelona, Spain: ACM, 2009, pp. 115–122. ISBN: 978-1-60558-496-6. DOI: 10.1145/1568296.1568315. URL: <http://doi.acm.org/10.1145/1568296.1568315>.
- [35] *Twitter is used for real time events*. <https://brand.twitter.com/>. Accessed: 2016-25-10.
- [36] Katia Vila et al. “Noise-tolerance feasibility for restricted-domain Information Retrieval systems”. In: *Data and Knowledge Engineering* 86 (2013), pp. 276–294. ISSN: 0169-023X. DOI: <http://dx.doi.org/10.1016/j.datak.2013.02.002>. URL: <http://www.sciencedirect.com/science/article/pii/S0169023X13000219>.
- [37] *Word2vec usage and explanation*. <http://www.netbase.com/blog/understanding-beliebers-word2vec-twitter/>. Accessed: 2016-15-11.
- [38] B Yegnanarayana. *Artificial neural networks*. PHI Learning Pvt. Ltd., 2009.