Adaptive Twitter Classifier

Diana Cremarenco

4th Year Project Report
Computer Science
School of Informatics
University of Edinburgh

2018

Abstract

In this project, we build an adaptive Twitter classifier based on distant supervision using YouTube videos. Since each YouTube video always belongs to one category, we can assign that specific category to the tweet, thus obtaining a very large training set for topic classification of tweets. However, such a model is bound to worsen in time due to trending topics changes and the solution is to build a classifier based on monthly data, since it is most likely to mirror current trends and have a superior performance on tweets from that time period. This thesis presents the design, implementation and performance of this adaptive classifier in ten different languages, concluding that distant supervision is a viable technique in most languages and that indeed adaptive classifier outperforms the static one for recent data.

Acknowledgements

I would like to thank Prof. Walid Magdy for his ongoing support and brilliant ideas he had given me while working on this project.

Table of Contents

Chapter 1

Introduction

Twitter is the most popular micro-blogging platform where users post traditionally 140 characters messages to share with friends and followers. In 2016, it was estimated that around 6000 tweets are posted per second, with the number increasing every year. This is one of the main reasons why tweet classification became a new topic of interest in the world of academia, since information posted on Twitter can be of high value to companies (e.g. sentiment analysis). The main challenge in tweet classification is data labelling, since finding enough representative tweets for each targeted topic in multiple languages requires human resources which are expensive and time consuming. This is why classifiers based on such data are rather under-performing, since the quantity of training data required is much higher than what can realistically be obtained. In this thesis we build on the approach documented for the first time in Magdy et al., 2015 and namely distant supervision using YouTube video categories.

The main idea presented in Magdy et al., 2015 is that each YouTube video is assigned on upload one of 15 categories including (Autos&Vehicles, Gaming, Music, News&Politics etc.). Whenever a tweet is linked to a YouTube video, it should be safe to assume that the tweet can be classified as the according video. This method generates a very high amount of training data through daily, continuous Twitter mining by assigning to each tweet the category of the video it linked. The classifier can be then used for tweets that do not necessarily contain a linked YouTube video. The classifier based on this approach will be further referred to as Class Strength v1.

The problem with this approach is that the data collected at one point in time is not representative for future topic classification. This is most obvious in news, since they change daily and what a year ago was correctly classified as news, today is not and vice-versa. As the current model gets older, it is more likely to under-perform on today's data and thus the need for an adaptive model arises. This project focuses on the design of this adaptive system (ClassStrength v2), which is continuously mining Twitter using the Twitter API and storing tweets that contain a YouTube video, alongside their given category. Then, once per month this data is grouped by category, deduplicated, further pre-processed (we will refer to this as 'cleaning' the data) and fed to a classifier in the form of bag-of-words representation. The performance of the classifiers differs between languages, so we choose a 'best' one individually.

This classifier is built in 10 different languages, namely: English, French, German, Japanese, Chinese, Arabic, Russian, Spanish, Portuguese and Polish. While this method yields high volume of data in some languages (English, Portuguese, Japanese), the classifiers for some languages (e.g. Polish, Arabic) are still likely to suffer from training data shortage. We explore the particularities of each language, the average size of training data fed to each classifier and then try to reason about the performance of each. Suprinsingly, less/more data does not necessarily translate into better/worse performace (as we will see with English vs Polish). We also explore the potential impact of joining several months of data to increase the accuracy of the classifiers and discuss why it may or may not be beneficial.

Lastly, we present the online version of ClassStrength V2, which can be found at http://hawksworth.inf.ed.ac.uk:5000/twitter (only available within School of Informatics), which is based on a lightweight, efficient implementation of a multiclass Suport Vector Machine (SVMlight mutliclass). This is a service that runs continuously on the Informatics server and can be used by anyone within Informatics network to classify a tweet in any of the 10 languages with any classifier and classify a list of tweets submitted as a text file.

1.1 Previous Work

1.1.1 Tweet classification

The most popular classification task on tweets is sentiment analysis, with obvious impact on marketing strategies, improving products' success (Barbosa & Feng, 2010, Chen et al., 2012, Jiang et al., 2011). For instance, Thelwall et al., 2011 explores the sentiment strength in relation to the emergence of major news stories on Twitter, pointing to the fact that negative emotions are more likely to cause a story to spread. Research interest has also spiked lately in topic classification for tweets as to increase accuracy in information retrieval. For instance, Lee et al., 2011 performs topic classification into 18 categories for the trending topics of Twitter with two approaches: text-based and network-based classification achieving 70% accuracy. Becker et al., 2011 identifies real-world events on Twitter using an online clustering method, features being extracted from the formed clusters and used to determine whether it is a real-life event. While most papers presentexperiments where the dataset has been manually labelled, others try to use automatically labelled data or enhanced tweets. For example, Davidov et al., 2010 uses hashtags and emoticons for building training data. Pak & Paroubek, 2010 presents a method to automatically collect a corpus for sentiment analysis and opinion mining, then prove experimentally that the data can successfully be used for classification.

1.1.2 Distant supervision

Distant supervision is a method of generating training data by making use of an existent database to collect examples for a certain relation. Then we use these examples to extract the relation and construct a weakly labelled training set for our task. While the generated data is bound to be noisier than manually labelled data, the main advantage of this method is its capability of generating large amounts of data relatively fast. In literature, Go et al., 2009 uses distant supervision for tweet sentiment analysis by using the emoticons in the tweet to establish the sentiment. Husby & Barbosa (2012) uses distant supervision for blog topic classification, by using Freebase to obtain the domain of recognised objects in the text and using it as the assigned label. Similarly to our distant supervision approach, Zubiaga & Ji, 2013 uses the topic of the included URL in the tweet to determine the category of the tweet. One of the most recent papers doing similar work is Mohammed et al. (2017) where topic classification for tweets obtained via distant supervision is done. They identify some targeted Twitter accounts (e.g. ESPN for sports) and automatically label the tweets from these accounts with the main category. They do not opt for a multiclass classifier, but for multiple single class classifiers whose performance depends on the class (Sports very good, Entertainment not so much). Lastly, as noted in Magdy & Elsayed, 2014, the need for an often updated classifier is stringent when it comes to classifying social media texts, due to the dynamic nature of these mediums (what is described as news differs from day to day)

1.2 Outline and Goals

The main goal of this project is to build a stable system which mines Twitter continuously for tweets with linked YouTube videos, processes the data and trains a new classifier monthly in each of the above mentioned ten languages.

In order to complete this system, the following contributions were necessary:

- **Continuous data collection**: Twitter API is used for continuous mining on Twitter in the proposed languages, but the most effective method differs from language to language. In chapter 2 we discuss which languages are more popular on Twitter, the advantages/disadvantages of Search API vs Stream API, as well as the optimum behaviour that the system currently employs for each language.

- **Data preprocessing or cleaning**: in chapter 3 we discuss what steps are applied to the data before being able to use it as training data, including retrieving the category label from YouTube using YouTube API. We analyse how much this process decreases the size of our training set. Lastly, we analyse the relative category distribution in the considered languages, looking at the most common and rarest categories for each.

- **Data representation**: in chapter 4 we present the experiments conducted in terms of data representation for the classifiers. We examine the theoretical background of the bag-of-words (with or without tfidf) and present a set of experiments that determine the representation in all future ones.

- **Balancing methods**: in chapter 5 we present the imbalance problems that a dataset created this way faces and what has been done to remedy these problems.

- **Classification methods**: in chapter 6 we present the theoretical background in short for the attempted classifiers, which include: Naive Bayes, Maximum Entropy, Logistic Regression and present the experiments that have been performed. In the end, we choose a tuple of (data representation method, classifier) for each language.
- **Enrichment methods**: in chapter 7 we introduce tweet enrichment methods used in this paper(video title addition, use of hashtags) and a new one ('translating' into ascii). We have conducted experiments will all of them and present the results obtained with the classifiers described in the previous chapter. We obtain satisfactory results (especially in Spanish and Portuguese) when testing with the silver set (labelled tweets extracted using distant supervision).
- **Monthly-based classification**: in chapter 8 we discuss results of the classifier for the months of December, January and February. We motivate with results the fact that the classifier performs best when trained with data from the same month as the test set.
- **Online system**: in chapter 9 we discuss the online built system, how it is implemented and how it performs in day-to-day situations.
- **Conclusion and future work**: Lastly, in chapter 10 we present an overview of the results, potential sources of error affecting our system, as well as future work that could improve the performance of such a classifier.

1.3 Tools

The system was written in Python (version 2.7.5) for the Search API, and version 3.6.3 for the rest of the system. We used extensively the Natural Language Toolkit (NLTK)[1], which is an open-source Python based platform that is used to interact with human language data. It provides various text processing capabilities (e.g. tokenisation, parsing) with a very straightforward interface, making it easy to use. We also used for most of our experiments the scikit-learn package or also called Sklearn [2]. This package offers numerous off-the-shelf machine learning solutions for small-scale projects. We also use python-twitter pure Python interface for Twitter API, which was chosen for implementing the Streaming API. Some other used Python packages include: langid [3] used for language detection, unidecode [4] for translating text into ASCII characters, flask[5] for the online system.

[1] https://www.nltk.org/
[2] http://scikit-learn.org/stable/
[3] https://github.com/saffsd/langid.py
[4] https://pypi.python.org/pypi/Unidecode
[5] http://flask.pocoo.org/

Chapter 2

Data collection

Twitter is the most popular microblogging platform by letting people share with the world short online updates. A tweet is a short text of traditionally 140 characters, which is posted by a Twitter user and seen by other users who ’follow’ the posting user. A tweet can be replied to or ’retweeted’ by any user and thus the same tweet can spread very fast within the network. One can include media in the tweet (images, links, videos) and this is the type of tweets we are actively mining. A hashtag (e.g. #hashtag) when added to a tweet is a simple way of categorizing it, making it easier for other people to find tweets on the same topic. The use of hashtags facilitates communication with users that otherwise have nothing in common and popular hashtags often become a trending topic. *Note:* In November 2017 Twitter announced that it is doubling the character limit to 280 characters in all languages except Chinese, Japanese and Korean for a number of test users (Rosen, 2017)

Twitter API (Application programming interface) is the only way to communicate with Twitter and mine tweets and interact with them (i.e. extract information, post tweets, filter and search etc.). There are 2 ways in which Twitter API allows for tweet mining, namely Search and Stream API (first one conforms to the design principles of Representational State Transfer and can also be called REST API). For both APIs, one needs to create a user account and a Twitter application on the official website and thus obtain consumer and access tokens, which are then used to make authenticated API requests on the user account’s behalf. The tweets are retrieved in JSON format, each having assigned a unique id, thus allowing easy parsing and manipulation in python.

2.1 Search API

The Search API is used to search against a sample of recent tweets (published in the last 7 days). The standard, free search API focuses on relevance to the search query, rather than completeness (tweets may be missing from the response). Its main advantage is that it allows for better filtering than a streaming endpoint (i.e. we can search tweets in a specific language published since a specific day and including a YouTube video by specifying multiple parameters: *’q’:’url:youtube’, ’since’:’2018-01-12’, ’lang’:’en’,*

’tweet_mode’:’extended’). To avoid retrieving the same tweets more than once, we can replace *’since’* with, for example, *’since_id’:24012619984051000* in all subsequent searches.

The main disadvantage of this API is that it is rate-limited in that an application cannot make more than 15 calls in every 15-min window. These limits are problematic when trying to collect larger amounts of data or when repeatedly making calls to the API to retrieve a small number of tweets very often.

2.2 Streaming API

The Streaming API offers access to real-time tweets (only around 1%), with the cost of reducing filtering power. Establishing a connection to this API means maintaining a long-lived HTTP connection and incrementally read an infinite response. There is no rate limit in the number of retrieved tweets, but if the application reads the tweets too slow or otherwise fail to maintain the connection, it may be prevented from reconnecting too often. Recently, the API offers the capability to filter by search term and language, but earlier(when the system was built) it would only be possible to filter by the search term.

2.3 Implementation

The initial data mining system was built based on the Search API, with no intermediary python library (i.e. the authentication system, behaviour in case of rate limitation, reconnection behaviour were written from scratch). The choice for the Search API was motivated by the fact that we could start a search in each of the 10 different languages and thus obtain more tweets per language. An overview of average daily collected tweets can be analysed in 2.1 The rate limit was handled with frequent sleep time when the corresponding error code was received. The script is generally stable, being disrupted very rarely by connection errors, in which case an exception is raised. When the connection is disconnected, it is ensured that it is promptly restarted and tweets are appended to existing files.

The main issue with this method was that while it gathered more tweets in infrequent languages (such as Polish, Arabic) than a stream-based application, it failed to gather representative numbers of tweets in English due to the rate limit. Thus, we wrote a streaming version of the data mining algorithm to run in parallel with the search-based script. In order to obtain a quick overview of its potential, we used the python-twitter[1] library, a lightweight python wrapper for the Twitter API. Disruptions in the data collection due to python-twitter errors or connection errors were handled by simply restarting the python script. We connected to only one streaming endpoint, retrieving any tweet containing the term ’youtube’ and for each tweet, we would detect the

[1] https://github.com/bear/python-twitter

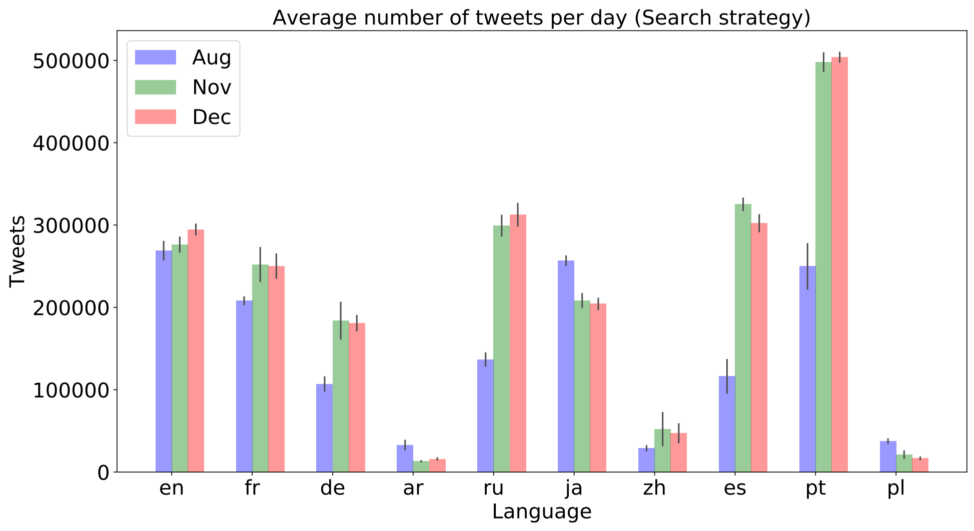


Figure 2.1: Average number of tweets per day collected with the search-based script (2017)

language with the licensed library langid, a standalone Language Identification tool. For each tweet, if the detected language was one of the 10 considered ones (with a probability of over 50%), we would append the tweet to our designed files.

In terms of storing the tweets, we store them in text files, one tweet per line, where we store the text of the tweet, the expanded YouTube URL and its id (only for tweets mined with Search API). The filename is easily parseable and representative for the data it stores (e.g. 2018-01-12_en.txt for Search API and 2018-01-12_en_stream.txt).

2.3.1 Performance

As expected, the streaming-based application outperformed the search-based one in terms of number of tweets collected in English and, surprisingly, in Arabic too (see table 2.1). In all the other languages, the search-based one is much stronger than streaming, collecting more tweets per day. The reason why the Streaming API is much better in English is because the probability that a processed tweet is in English is much higher than it is to be in other languages (we 'stumble' upon more English tweets). It is obvious that we do not process all the tweets we receive in real-time (we miss many of them due to processing time per tweet, i.e. extracting text and expanded URL, appending to collections of tweets), so this explains why the numbers in other languages decrease (probability of finding tweets in these languages is much lower). This is why the Search API excels in this case, as it actively searches for the tweets that streaming missed.

From now on, we refer to these tweets that are mined in the initial stage as being the 'raw' data, because these tweets contain duplicates, retweets and auto-generated tweets (standard text when sharing a YouTube video on Twitter is 'I liked this video ...') These tweets are not useful in our classification tasks and will be removed in later stages.

Language	Search	Stream
English	$\approx 300k$	$\approx$ **580k**
French	$\approx$ **250k**	$\approx 46k$
German	$\approx$ **180k**	$\approx 28k$
Arabic	$\approx 13k$	$\approx$ **75k**
Russian	$\approx$ **300k**	$\approx 30k$
Japanese	$\approx$ **200k**	$\approx 148k$
Chinese	$\approx$ **50k**	$\approx 20k$
Spanish	$\approx$ **300k**	$\approx 135k$
Portuguese	$\approx$ **500k**	$\approx 153k$
Polish	$\approx$ **21k**	$\approx 10k$

Table 2.1: Average number of tweets mined daily with Search API and Stream API in November

The chosen behaviour was running both scripts in parallel and each day choosing to preprocess the dataset with most tweets written to it. Combining them proved to be mostly useless, as the smaller set is roughly contained within the big one. This change increased the total number of raw tweets mined per month in English (especially) from < 800000 to > 1.4 million and Arabic (from ≈ 120000 to ≈ 160000), as it can be seen in 2.2. The variations in other languages in the total number are largely due to natural dynamic of tweet posts.

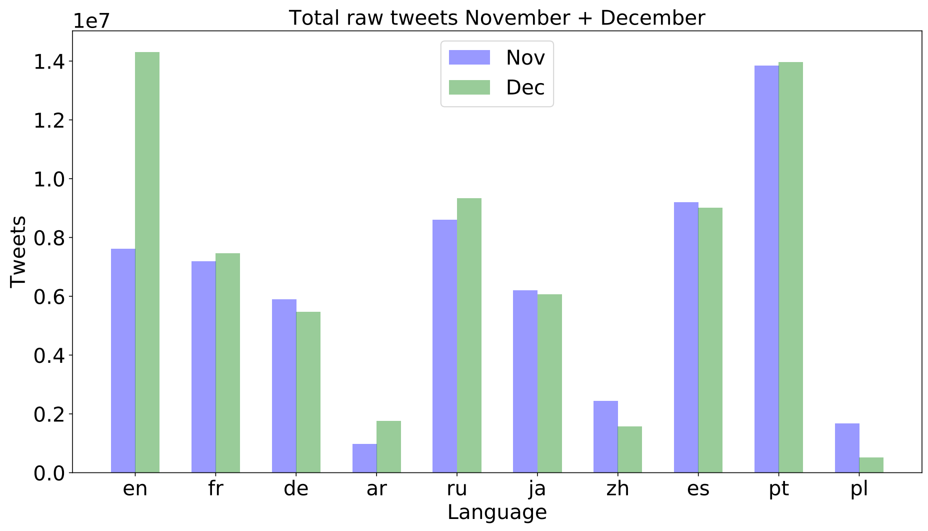


Figure 2.2: Total number of raw tweets in millions collected in November vs. December (2017)

We continue running both Search and Stream collection systems in parallel and every day we clean the bigger set for each language.

Chapter 3

Data preprocessing

3.1 Cleaning process

3.1.1 Tweet form

Given the ever-growing number of tweets posted daily, Twitter, as well as other platforms of social media, became a potential source of crowd wisdom. However, there are numerous challenges in using these bits of information for classifying data, especially due to the flexible form of a tweet, as well as the slang introduced by users either due to online trends, either to express as much information as possible given the character limit. As for the importance of doing this, as discussed in Singh & Kumari (2016), removing this noise, as well as using the proposed framework for dealing with slang words significantly improves the performance of the classifier, which is now capable to learn the important features.

A tweet is retrieved in a JSON format using the data collection script explained above and contains metadata beyond the text, such as: URLs, media, number of likes, user information, number of comments and retweets etc. Most of this information is not of any use for our purpose and must be discarded. Most tweets, especially those with a YouTube video associated do not give any unique information, being generated automatically (i.e. I liked this video https://youtu.be/...). For these tweets, the title of the video is usually automatically included as part of the text, hindering the process of cleaning. The number of raw tweets is much higher than what is actually fed into the classifier, because, once the noise is removed (retweets, automated generated tweets, tweets containing only emojis, tweets with less than two words, tweets containing only numbers etc.) the training set decreases in size. It is to be noted, that, even given this decrease, the training set is still larger than manually-obtained sets.

3.1.2 Cleaning process

The data gathered for each day is processed first by a daily running script, then secondly by the script that combines data for the whole month. The steps done for trying to remove the noise are the following:

- Discard tweets that do not have a YouTube video attached (perhaps only mention YouTube)
- Lower-case the text ('I LOVE THIS' will become 'i love this')
- Remove links, new-line symbols, sintagms such as 'via @youtube'
- Use NLTK TweetTokeniser to tokenise the text: this breaks the text into separate words (called tokens) and provides the option to standardize the length of some words. For example, the word 'waaaaaaaay' will be treated as 'waaay', which is what we intend for this task. In the end, we join all these tokens with space, such that both 'I love you!!!' and 'I love you !! !' are considered the same. We chose not to remove handles such as: #importance because hashtags may provide valuable information to the classifier.
- Remove mentions ('@Mary')
- Remove digits
- Remove titles (they will be appended to the text later as a form of tweet enrichment). This stage is particularly important, because many times the title of the video is the only one that makes a difference between the auto-generated tweets (e.g. until now 'I added a new video Beyonce-Single girls' and 'I added a new video Nelly-Bird' are considered different). We do this by preprocessing the title of the tweet (which we stored separately) in the same way as the tweet, then matching it within the tweet and deleting it.
- Deduplicate tweets, done lastly because the text has the highest probability to be recognized as identical now. However, this deduplication is not perfect. The tweets 'New brain research development appeared in the news' and 'Brain research development appeared in the news' are not identical in form, although they provide the same information.

Within the daily-cleaning script we also use the YouTube Data API to obtain the category of the attached video. Using the expanded URL (which is a field in the tweet JSON), we search for the YouTube video JSON, which contains the category ID (32 possible IDs). Sometimes the video has been removed or we cannot otherwise retrieve the category, which leads to discarding the tweet. It is crucial that this step is done daily because the YouTube Data API has strong rate-limitations (1 000 000 queries max per day) and we would definitely hit them if we do this category retrieval at the end of the month (only in English we have approximately 700 000 tweets per day to query the category for). We have a separate quota for each language, to ensure these limits are not hit. The decrease in training set size after cleaning can be analysed in 3.1.

Finally, tweets are deduplicated again when data for the whole month is combined, as

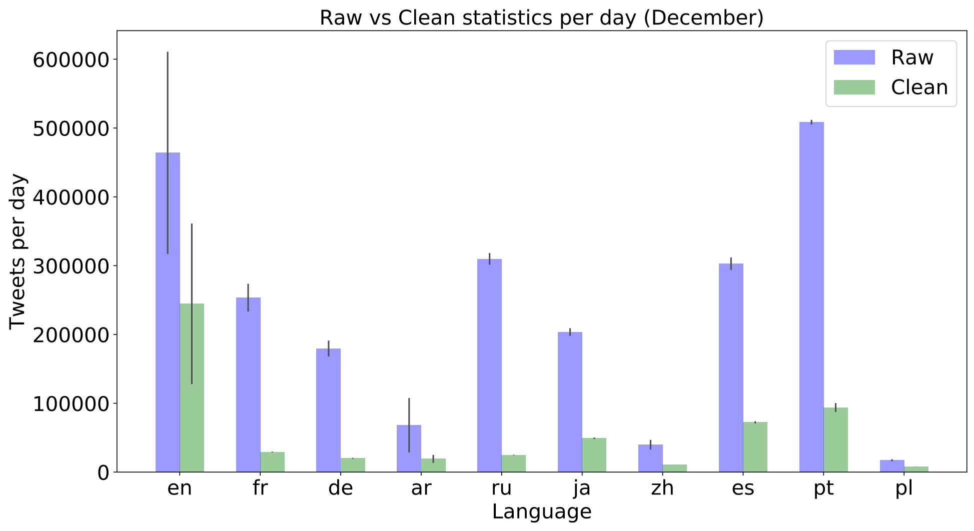


Figure 3.1: Before and after cleaning the data December (2017) Statistics per day

the same tweet can be posted on more days.

One can analyse a word cloud with the most common 200 words for the dataset of December (using appended title to the video) in the figure 3.2. We can notice the frequency of autogenerated tweets just by noticing the frequency of the word video, playlist, added. One can also notice the frequency of the word Christmas, which is unlikely to be seen in any other month. Also, we can notice the high frequency of words like de or ii or st which do not mean anything on their own.

3.2 Category statistics

As specified above, we are guaranteed to obtain one of the 32 possible IDS, but using them all as labels leads to sparsity of data in some categories, and may hinder the performance of the classifier especially in related categories (e.g. Movies vs. Action/Adventure vs. Thriller). Thus, we have merged the following categories under Film&Animation: Movies, Anime/Animation, Action/Adventure, Classics, Documentary, Drama, Family, Foreign, Comedy, Horror, Sci-fi/Fantasy, Thriller, Shorts, Shows, Short Movies. In the end, we work with 14 categories: Filme&Animation, Sports, Music, Gaming, How-to&Style, Education, Science&Technology, News&Politics, Pets&Animals, Nonprofit&Activism, People&Blogs, Autos&Vehicles, Entertainment (Comedy is assigned as Entertainment), Travel&Events. The category People&Blogs is the automatically assigned one for YouTube so we expect it to be noisy in terms of labels.

The classes are massively imbalanced in all languages, the 4 most common categories being assigned to over 75% of the data. For example, in English, we can see in figure 3.3 that the most common 5 categories make up 77% of the data. Generally, the most

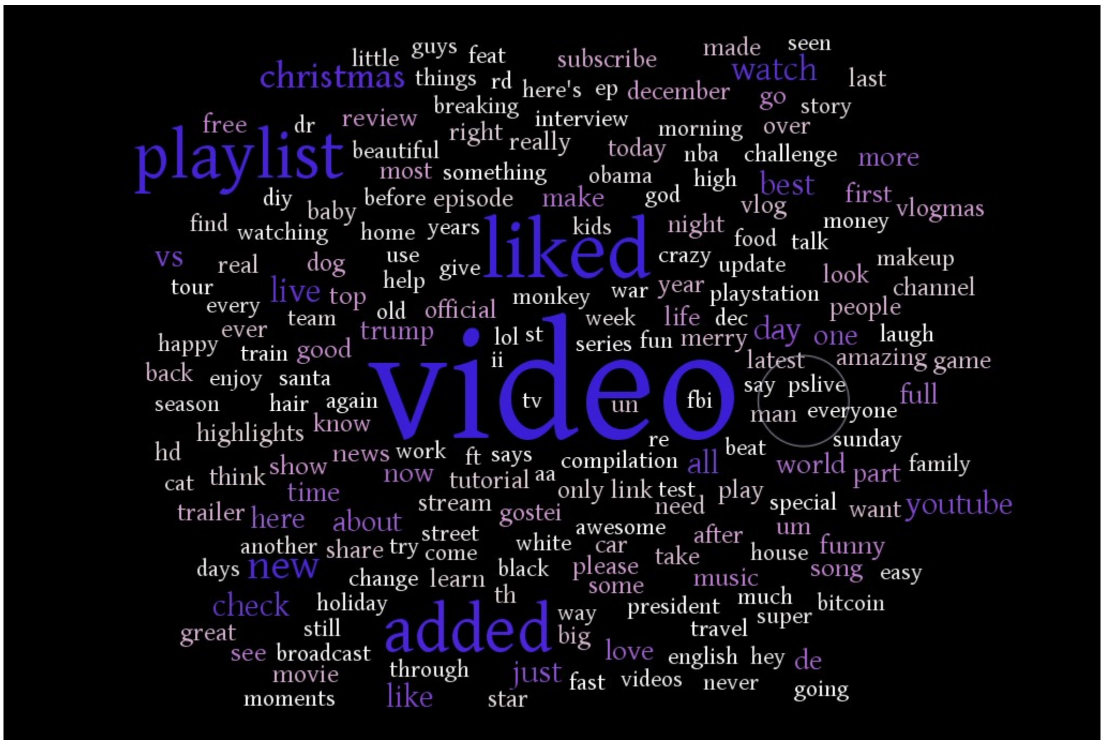


Figure 3.2: Word Cloud most frequent 200 words for December 2017

common categories are Music, Entertainment, People&Blogs and Gaming, the fifth most common one being either Education or News&Politics. In the table 3.1 we can analyse in more detail numbers in terms of category distribution for the most common and rarest one for each language for the month of December.

Language	Most common	#Tweets	Rarest	#Tweets
English	Music	$\approx 1.46M(22\%)$	Pets&Animals	$\approx 38k(0.58\%)$
French	Music	$\approx 175k(23\%)$	Pets&Animals	$\approx 3.4k(0.4\%)$
German	Gaming	$\approx 187k(33.4\%)$	Pets&Animals	$\approx 2.7k(0.5\%)$
Arabic	People&Blogs	$\approx 162k(\%34)$	Pets&Animals	$\approx 1.5k(0.33\%)$
Russian	People&Blogs	$\approx 210k(31\%)$	Pets&Animals	$\approx 6.1k(0.9\%)$
Japanese	Music	$\approx 320k(28\%)$	Nonprofits&Activism	$\approx 4.5k(0.4\%)$
Chinese	People&Blogs	$\approx 99k(31\%)$	Pets&Animals	$\approx 1.5k(0.5\%)$
Spanish	Music	$\approx 450k(24\%)$	Pets&Animals	$\approx 5.7k(0.3\%)$
Portuguese	People&Blogs	$\approx 465k(23\%)$	Pets&Animals	$\approx 11k(0.57\%)$
Polish	Music	$\approx 57k(26\%)$	Pets&Animals	$\approx 1.2k(0.55\%)$

Table 3.1: Number of training examples in the most common and rarest category for the month December 2017

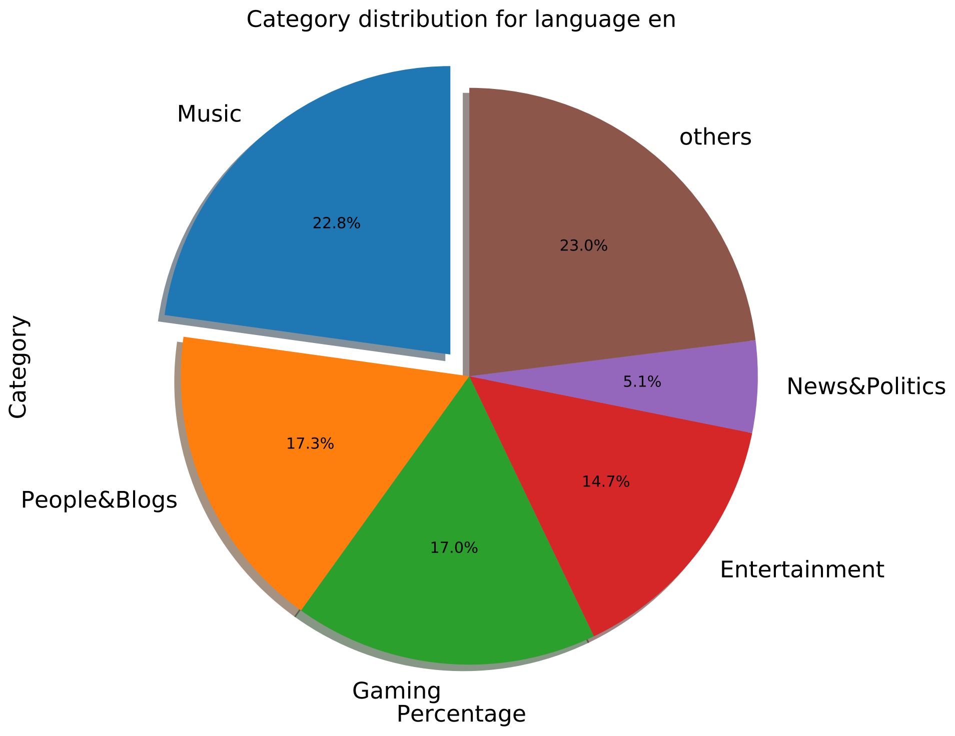


Figure 3.3: Most common categories for tweets in English for the month December 2017

Chapter 4

Data representation

Machine learning methods are widely used today for tasks ranging from image classification, natural language generation and many more variations are discovered every day. However, all these different types of inputs have to be represented in a numeric form that are then fed to the machine learning method. Our task is short text multiclass classification and in this section we will explore different ways of representing our data most efficiently.

For our machine learning models and data representation we have worked extensively with the Sklearn open source python library which offers free, off-the-shelf machine learning solutions for uncomplicated systems. Further on, we will detail the theoretical background of more bag-of-words models as they are named in the Sklearn library. We will continue to use these names to refer to the described models in the entire paper.

4.1 Experimental setup

For consistency of experiments, we will use the same training set in all experiments and test on a balanced test set (sizes can be analysed in 4.1). We will only use **data from November 2017 in all our experiments**, split into 90% training set and 10% validation set (once we undersampled the validation set in order to balance it, its size decreased considerably). Once we have chosen a data representation method, classifier and enrichment method for all languages, will we proceed to testing on data from December and January. Moreover, we will use as a measure of performance the F1 Macro measure, which is robust to class imbalance. F1 score for each class is computed as $f1\_score(c) = \frac{2*precision*recall}{precision+recall}$ and the macro F1 score is an average of these scores over all the 14 classes. In order to see which data representation has the highest potential, we will use the Sklearn default version of **SGDClassifier(max_iter=50)** which is an stochastic version of the SVM classfier with a maximum of 50 iterations over the training data. We are aware that each classification method may benefit from other data representations so we will always experiment with the ones that have the highest potential according to the following experiments. In all tables with results in this thesis we use a kind of colour heatmap ranging from red (the smallest values) to green (the higher values).

Language	#Tweets (training)	#Tweets(validation)
English	$\approx 756k$	$\approx 5.2k$
French	$\approx 160k$	$\approx 0.7k$
German	$\approx 169k$	$\approx 1.1k$
Arabic	$\approx 168k$	$\approx 0.9k$
Russian	$\approx 128k$	$\approx 1.4k$
Japanese	$\approx 590k$	$\approx 3.8k$
Chinese	$\approx 91k$	$\approx 0.6k$
Spanish	$\approx 595k$	$\approx 1.9k$
Portuguese	$\approx 1.15M$	$\approx 10k$
Polish	$\approx 43k$	$\approx 0.3k$

Table 4.1: Number of training/validation examples in an imbalanced training set and a undersampled, balanced, validation set

4.2 Bag-of-words representation

The bag-of-words model has been the most popular way of representing text documents until the appearance of competitive neural network models which use a different type of representation: word embeddings. In our project, however, we have experimented only with the classical machine learning classification methods such as: multinomial naive bayes, SVMs etc., all of which require a type of bag-of-words representation.

The bag-of-words model represents each document as a very high dimensional sparse document. In order to do this, we consider each word as a unique feature, thus creating a very large vocabulary. The size of this vocabulary defines our document vector size, since each document will be, in the simplest model, represented as a vector where we have either 0 (most of the terms) because the document does not contain the word, either the number of times the word appears. It is called a bag-of-words model because by doing this we disrespect the order of the words within the document completely. A visualisation of the simplest model can be seen in figure 4.1.

Such a representation of the document is faulty in many ways (most importantly by disregarding order of the words), including the fact that we will consider and count frequent words such as 'the', 'of', 'a' which provide no clue to the category. Such words are named stopwords and for many languages there are already available lists of such words that should be removed from our vocabulary. We can find such lists on nltk.corpus.stopwords and in all our text representation we use them if they are available (they are not available in Japanese, Chinese and Polish). For the excluded languages we use an empty stopword set.

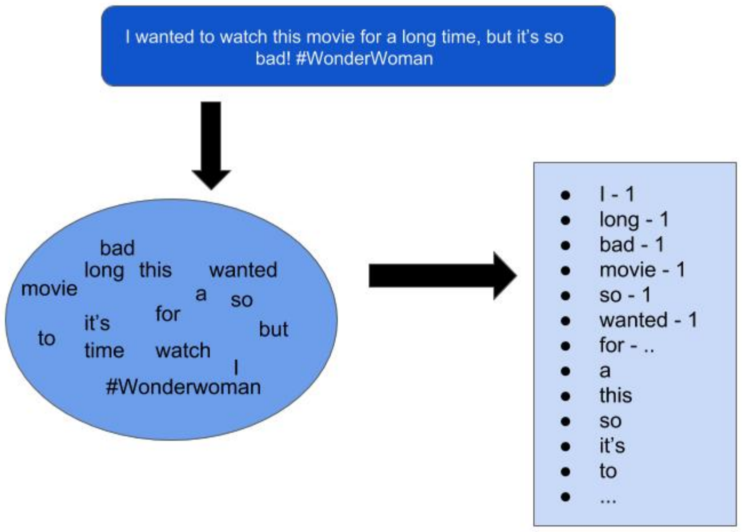

I wanted to watch this movie for a long time, but it's so bad! #WonderWoman
bad long this wanted movie a so for it's but to time watch I #Wonderwoman
I - 1
long - 1
bad - 1
movie - 1
so - 1
wanted - 1
for - ..
a
this
so
it's
to
...

Figure 4.1: Bag-of-words model

4.3 Count Vectorizer

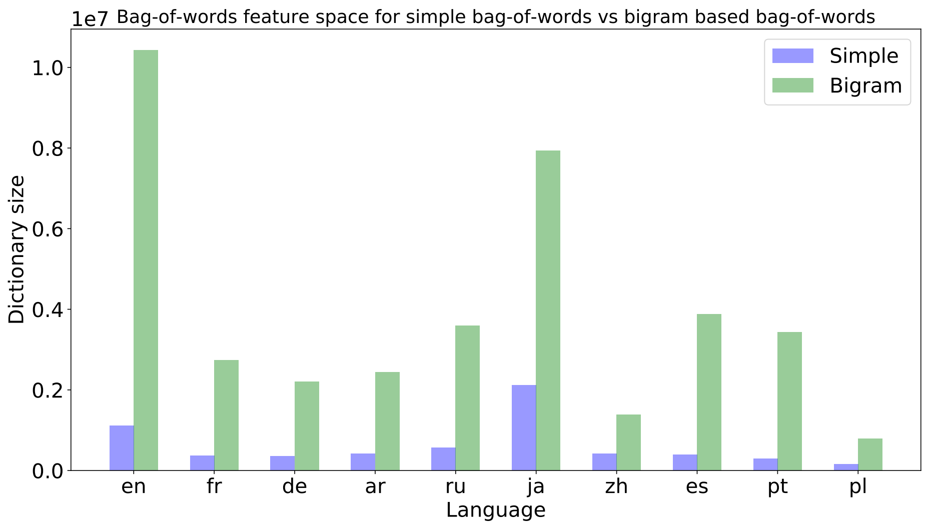


Figure 4.2: Vocabulary size for CountVectorizer simple vs bigram mode

The CountVectorizer class represents the simplest bag-of-words model described above. It stores a vocabulary in memory and each document is represented as a sparse vector with 0 or the number of times (#times) a word has appeared in the document.

All the vectorizers we will present and experiment with have multiple options, the most important one being the possibility of using bigrams. Since with the simple model the order is thrown away, we use bigrams, as well as unigrams as part of the vocabulary. This is done in order to preserve at least some of the local order of the words. Doing this more than doubles our vocabulary size in most languages, as seen in figure 4.2. One more interesting thing to note is that noticeably larger vocabulary size in Japanese than English. This may have consequences for classification that may not be present in other languages.

We also experimented with a binary form of the vectorizer, which instead of counting how many times a word was present, it simply marks if it has been found.

4.3.1 Experiments

Lang	Unigram	Bigram	Binary(Unigram)	Binary(Bigram)
en	0.381	0.424	0.378	0.423
fr	0.338	0.368	0.337	0.367
de	0.426	0.465	0.427	0.468
ja	0.404	0.413	0.406	0.413
zh	0.449	0.465	0.456	0.472
ar	0.41	0.45	0.411	0.444
ru	0.42	0.45	0.419	0.451
es	0.42	0.466	0.413	0.468
pt	0.392	0.458	0.395	0.456
pl	0.409	0.431	0.403	0.432

Table 4.2: Experiments with CountVectorizer (SGDClassifier(max_iter=50))

The results of these experiments can be analysed in 4.2. As one can see, using bigrams increases the F1 measure spectacularly compared to unigram based models. The results for binary/simple CountVectorizer are similar so far, so we will use both in further experiments.

4.4 TFIDF Vectorizer

TFIDF model is the most popular flavour of bag-of-words model in literature because it uses a numerical statistic to determine the importance of a word to a document. **TF** refers to term-frequency, while **IDF** to inverse document-frequency. Then **TFIDF** is computed as $tf-idf(t,d) = tf(t,d) * idf(t)$, where t refers to the considered term(word) and d to the document. While **TF** represents simply the count of the term in the document, **IDF** is computed as $idf(t) = log\frac{1+n_d}{1+df(d,t)}$ where n_d refers to the total number of documents and $df(d,t)$ to number of documents where t appears. Each resulting vector v=$(v_1, v_2, ..., v_n)$ is then normalized by the Euclidean form (divided by $\sqrt{v_1^2 + v_2^2 + ... + v_n^2}$). The Sklearn calculation differs slightly from the textbook implementation, but we believe the difference is not high enough to make a significant difference.

The result of applying this vectorizer to the collection of tweets is a matrix of floats, where each row represents a tweet where each word count is weighted by its corresponding tfidf value. Again, we will experiment with simple/bigram mode in all languages.

4.4.1 Experiments

The results for experiments with Tf-idf vectorizer can be analysed in table 4.3. Here, as previously, the best results are for the bigram version. The results, however, are lower than the bigram version of CountVectorizer, but this may be due to the chosen classifier.

Lang	Unigram	Bigram
en	0.354	0.386
fr	0.307	0.326
de	0.382	0.41
ja	0.384	0.392
zh	0.438	0.452
ar	0.392	0.431
ru	0.409	0.447
es	0.385	0.424
pt	0.371	0.435
pl	0.385	0.363

Table 4.3: Experiments with TfidfVectorizer SGDClassifier(max_iter=50)

4.5 Hashing Vectorizer

The hashing vectorizer is a bag-of-words model based on token occurence count (same as CountVectorizer), but instead of storing the vocabulary it computes a hashing integer value for each word and uses it as a feature_id. This model is less memory-consuming than the previous ones because the vocabulary is not stored, but it can also cause collisions between features.

4.5.1 Experiments

The experiments with this vectorizer are limited to unigram and bigram version as before. While the same trend continues, with better values for bigram, the values are worse than any vectorizer so far, so we will not experiment any more with this type of vectorizer. The results in detail can analysed in 4.4.

Lang	Unigram	Bigram
en	0.343	0.368
fr	0.311	0.312
de	0.363	0.373
ja	0.348	0.35
zh	0.44	0.417
ar	0.373	0.399
ru	0.398	0.421
es	0.366	0.401
pt	0.359	0.396
pl	0.339	0.354

Table 4.4: Experiments with HashingVectorizer SGDClassifier(max_iter=50)

4.6 Interim conclusions

There are already a few things that we would like to notice about our task:

- Bigram based data representation methods work best in all languages.
- While this classifier (SGDClassifier for 5 iterations) uses best the CountVectorizer(stopwords=stopwords, binary=True, ngram_range=(1,2)), it may not be true for other classifiers, so we will try CountVectorizer(stopwords=stopwords, ngram_range=(1,2)) and the binarized version of it, as well as the bigram version for Tfidf vectorizer.
- There is no correlation between dataset size and performance, since we have most data in English and the best performance in Chinese, where we have a reduced dataset.
- The training dataset is highly imbalanced and we have not applied any balancing techniques. These are likely to improve the performance of the classifier and are discussed in the next chapter.

Chapter 5

Balancing methods

As discussed in detail in 3.2 our datasets suffer greatly from class imbalance (the first 4 most common categories are over 70% of the data, while the least common ones are less than 1% of the data). This situation is very common in the real world, a popular example being disease detection (most patients are not sick, but it is crucial to determine the ones that are). In such cases, using accuracy as a performance measure is not ideal, at least not if the test set respects the same distribution as the training set, because accuracy places more weight on the common classes, thus giving a false indication of good performance.

As mentioned in Kotsiantis et al. (2006), there are 4 popular ways of dealing with imbalanced data:

- Dealing with imbalance at data level

  Oversample: We can 'artificially' create more examples from the under-represented classes. For instance for images we can rotate them, apply a filter, etc. For text, there is no easy way of adding data, except duplicating the documents.

  Undersample: We can reduce the overly represented classes to the size of the smallest class. The main disadvantage of doing this is that the data we suddenly discard may be valuable for the classification task.

- Dealing with imbalance at algorithm level:

  KNN: we can use a 'weighted' measure such that we take the closest neighbours into account more than the far away neighbours

  Cost-sensitive classifiers: we can modify the loss function such that every mistake the model does affects the model inversely proportionate to how frequent the respective class is.

- Combining methods: one can use a 'mixture-of-experts'(multiple classifiers trained on different variants of the data) and take the most common prediction among these classifiers. Each classifier can be trained on an oversampled/undersampled version of the data. This has been shown to perform better than both undersampling and oversampling Estabrooks et al. (2004).

- Using appropriate performance measures: as mentioned above, accuracy is not suitable for this case, so instead we must use robust measures such as: ROC curves (area under the ROC curve as a performance measure) and average F1 score (the one that we use).

In Japkowicz & Stephen (2002) it is shown that a cost-sensitive algorithm usually outperforms random resampling of the data. In our experiments, we will experiment with undersampling vs. cost-sensitive classfiers vs. unbalanced data. We chose not to do any oversampling because it might increase the noise level in an already highly noisy dataset. Because of this, we have also not done any method combination. In all our experiments, we test on a **balanced validation set** constructed with undersampling in the validation data. We will focus on macro F1 as the primary performance measure, but we can also use accuracy because it is a balanced validation set.

In table 5.1 one can notice that undersampling drastically reduces the size of the training set, which is bound to affect the classification performance. For all the other experiments we will use the data described in 4.1.

The experiments have been performed with the same basic SGDClassifier(max_iter=50) with SVM loss function and using the CountVectorizer bigram and binary. In order to adjust the loss function to take into account class frequencies we set the property class_weight='balanced', which penalizes the model inversely proportionately to the frequency of the class. The results can be analysed in table5.2. As it can be noticed, undersampling is worse than even when the classifier is unaware of the imbalance. However, a cost-sensitive classifier improves the f1-score considerably over all classes. As such, from all experiments from now on we will use a cost-sensitive classifier (at least when possible: SVM, logistic regression etc.).

Language	#Tweets (training)	#Tweets(validation)
English	$\approx 52k$	$\approx 5.4k$
French	$\approx 8.2k$	$\approx 1k$
German	$\approx 10k$	$\approx 1.2k$
Arabic	$\approx 7.9k$	$\approx 0.9k$
Russian	$\approx 13k$	$\approx 1.4k$
Japanese	$\approx 23.5k$	$\approx 3k$
Chinese	$\approx 6k$	$\approx 0.6k$
Spanish	$\approx 21k$	$\approx 2.6k$
Portuguese	$\approx 90k$	$\approx 10k$
Polish	$\approx 3k$	$\approx 0.3k$

Table 5.1: Number of training/validation examples in an undersampled, balanced dataset

Lang	Imbalanced	Cost-sensitive	Undersample
en	0.423	0.485	0.421
fr	0.368	0.403	0.301
de	0.465	0.499	0.351
ja	0.413	0.419	0.389
zh	0.472	0.474	0.459
ar	0.45	0.514	0.364
ru	0.452	0.478	0.391
es	0.468	0.556	0.474
pt	0.456	0.56	0.515
pl	0.431	0.448	0.298

Table 5.2: Experiments with Balancing methods SGDClassifier(max_iter=50) with class_weight='balanced' set for Cost-sensitive

Chapter 6

Classification methods

As in the previous chapters, we will extensively use Sklearn python package to perform experiments with multiple methods of classification, including Multinomial Naive Bayes, Logistic Regression, SVMs and others. Based on findings so far, we will use bigram bag-of-words models for data representation(experiment with CountVectorizer, Count Vectorizer binarized and TfidfVectorizer) and we will use a modified loss function to account for class imbalance when possible. The sizes of the training/validation sets are detailed in 4.1

There are 2 main classes of classifiers for multi-class classification:

- Transformation to a binary problem (i.e. to a special case of 2 classes)

  One vs Rest (OVR): one class contains only samples from class C_k, the other class all the other samples. We train such a classifier for each class and in the end we choose the one for which the confidence score is the highest (that it belongs to that class)

  One vs One (OVO): we create $K(K-1)/2$ classifiers for each pair of classes and for predicting a voting mechanism is applied.

- Extending from binary: there are many machine learning methods that naturally adapt to multiclass problems, some of them being:

  KNNs: voting mechanism between the closest k neighbours in a |V| dimensional space, where |V| is the size of the vocabulary

  Naive Bayes: relatively simple probabilistic classifier that does not need any modification to accept multi-classes

  Decision Trees: they also incorporate multiclasses naturally, since each leaf can refer to one of K classes.

  Neural networks: perhaps the most powerful tool for classification these days, but not explored here due to time and resources constraints.

  etc.

6.1 Naive Bayes classifiers

Naive Bayes methods are one of the most powerful probabilistic, generative classifiers used in machine learning and perform very well, especially in text documents classification (McCallum et al. (1998), Lewis (1998), Pang et al. (2002), Ting et al. (2011)) despite their simplicity and strong independence assumption. As noted before, they can handle multiclass classification naturally, without any modification. The main 'naive' assumption is that the value of a particular feature is independent from the value of any other feature, given a particular class (also called independence assumption) and can be represented as follows: If each text document is of form $\mathbf{x} = (x_1, x_2...x_n)$ (sparse vector), then:

$$p(\mathbf{x}|C_k) = \prod_{i=1}^{n} p(x_i|C_k),$$

where C_k is one of the k classes (in our case 14). When the classifier predicts the category of an unseen document $\mathbf{x}$ it computes:

$$y \approx \underset{k\in 1..14}{\operatorname{argmax}}\, p(C_k) \prod_{i=1}^{n} p(x_i|C_k) \tag{6.1}$$

In (6.1), we use $\prod_{i=1}^{n} p(x_i|C_k)$ as a way of modelling document $\mathbf{x}$ using the independence assumption.

In the training phase, we compute $p(C_k)$ and $p(x_i|C_k)$. The former is the relative frequency of the class in the training set or is set to be equal over all classes (1/#classes). In order to compute $p(x_i|C_k)$ we must either assume a distribution or generate nonparametric models from the training set. For tasks like text classification the most common used distributions (or event models) are Multinomial Naive Bayes and Bernoulli Naive Bayes, mathematical details below:

- **Multinomial NB**: we only compute $p(x_i|C_k)$ for words appearing in the document, otherwise $p(x_i|C_k) = 1$ and we define $p(x_i|C_k)$ using a smoothed version of maximum likelihood. In the equation below, N_{ki} represents number of times feature x_i appears in class C_k in the training set and N_k represents the total number of features for class C_k. α is the smoothing parameter (used to account for unseen words without giving them 0 probability) and n is the vocabulary size.

$$p(x_i|C_k) = \frac{N_{ki} + \alpha}{N_k + \alpha n}$$

- **Bernoulli NB**: It is different from Multinomial NB in the sense that the non-occurence of a word is penalized, rather than ignored (we compute $p(x_i|C_k)$ for all words in the vocabulary). We define $p(x_i|C_k)$ as below, where $p(i|C_k)$ is defined as $p(x_i|C_k)$ for Multinomial NB. Bernoulli NB uses a binarized form of the text document vector (like CountVectorizer binarized form constructs).

$$p(x_i|C_k) = p(i|C_k)x_i + (1 - p(i|C_k))(1 - x_i)$$

McCallum et al. (1998) argues that BernoulliNB may perform better when the vocabulary size is smaller, while MultinomialNB performs better with large vocabularies. Lastly, despite its simplicity and impossible independence assumption, Naive Bayes is still largely used for text classification tasks even if only as a baseline due to its fast training and fairly good results.

6.1.1 Experiments

		Bernoulli NB			Multinomial NB		
Lang	Vectorizer	0.01	0.05	0.1	0.01	0.05	0.1
en	Count	0.483	0.413	0.337	0.495	0.472	0.441
	Tfidf	0.483	0.413	0.337	0.453	0.375	0.32
fr	Count	0.363	0.287	0.222	0.379	0.363	0.342
	Tfidf	0.363	0.287	0.222	0.342	0.299	0.268
de	Count	0.488	0.402	0.303	0.485	0.467	0.446
	Tfidf	0.488	0.402	0.303	0.473	0.403	0.345
ja	Count	0.423	0.308	0.234	0.449	0.424	0.414
	Tfidf	0.423	0.308	0.234	0.428	0.397	0.361
zh	Count	0.453	0.341	0.285	0.469	0.464	0.442
	Tfidf	0.453	0.341	0.285	0.468	0.43	0.374
ar	Count	0.485	0.376	0.319	0.496	0.489	0.455
	Tfidf	0.485	0.376	0.319	0.459	0.391	0.323
ru	Count	0.458	0.407	0.331	0.471	0.462	0.441
	Tfidf	0.458	0.407	0.331	0.45	0.41	0.375
es	Count	0.531	0.447	0.368	0.541	0.516	0.479
	Tfidf	0.531	0.447	0.368	0.504	0.43	0.374
pt	Count	0.573	0.494	0.406	0.578	0.541	0.502
	Tfidf	0.573	0.494	0.406	0.524	0.441	0.367
pl	Count	0.421	0.309	0.249	0.447	0.388	0.358
	Tfidf	0.421	0.309	0.249	0.438	0.35	0.315

Table 6.1: Experiments with Naive Bayes (no balancing method required) varying data representation, alpha parameter (values of 0.01, 0.05, 0.1) Bernoulli vs. Multinomial

We have experimented with BernoulliNB and MultinomialNB using various α values (0.01, 0.05, 0.1) and using CountVectorizer in the binarized form and TfidfVectorizer. The F1 macro scores can be analysed in table 6.1. As expected, the values for Bernoulli NB do not change depending on the used vectorizer because the features are transformed to 0/1s by default. We notice that **Multinomial NB performs better** in our case (except German) and that $\alpha = 0.01$ seems to be the best analysed value for α. The more we increase its value, the more the score decreases. We also point to the fact that CountVectorizer (binary form) performs better than Tfidf for Multinomial NB in all cases. Having taken this into consideration, we have also experimented with equal class priors ($p(C_k) = \frac{1}{K}$) instead of priors reflecting the distribution in the training set using MultinomialNB(alpha=0.01) and bigram, binary CountVec with results in table 6.2. In most cases, imposing equal prior probability increases the f1 score, with only a few exceptions (Japanese, Polish, Russian). With these results we end our experiments with Naive Bayes classifier, moving on to Logistic Regression.

Lang	Relative frequency	Equal probability
en	0.495	0.507
fr	0.379	0.399
de	0.485	0.493
ja	0.449	0.447
zh	0.469	0.444
ar	0.496	0.508
ru	0.471	0.459
es	0.541	0.556
pt	0.578	0.598
pl	0.447	0.44

Table 6.2: Experiments with MultinomialNB(alpha=0.01) and classes priors (i.e. setting class priors reflecting class imbalance vs all equal)

6.2 Logistic Regression

Logistic Regression (also known as MaximumEntropy/MaxEnt) is a type of log-linear discriminative classifier (despite the Regression term in its name). It has been applied for a long time to NLP tasks, one example of text classification being Nigam et al. (1999). This method directly computes $p(C_k|\mathbf{x})$ by extracting a set of features from the training data, combining them linearly and then applying the logistic function to transform the obtained value into a valid probability. Thus, the formula for predicting the class of a document is:

$$y = \operatorname*{argmax}_{k \in 1..14} p(C_k|\mathbf{x}) = \frac{1}{Z} exp(\sum_i w_i f_i(C_k, \mathbf{x}) + b),$$

where Z is a normalizing factor (sum of all exps over all classes). Since it does not contribute to differentiating between classes, it is discarded in the classification task.

The features of a logistic regression classifier are categorical (0/1 only) and are of form:

$$f_1(C_k, \mathbf{x}) = \begin{cases} 1 & \text{if 'pet' in } \mathbf{x} \text{ and } C_k = \text{Pets\&Animals} \\ 0 & \text{otherwise.} \end{cases} \tag{6.2}$$

These features are usually automatically created by applying a feature template (e.g. each word for each class etc.) To each feature we apply a weight that we keep updating through the training process. The whole process of training tries to update the weights such as the predicted labels for the training data match its labels. We do this using conditional maximum likelihood estimation (choosing $\mathbf{w}$ to maximize the log probability of y labels for training points x). The function for binary classification representing the error that we are trying to minimize (also called log loss) is presented below. For multinomial classification the formula is slightly different.

$$L(\mathbf{w}) = C\sum_{i=1}^{n} log(exp(-y_i(\mathbf{x}_i^T\mathbf{w}+b))+1)+R(\mathbf{w}), \tag{6.3}$$

where $R(\mathbf{w})$ is the regularization parameter. Regularization is usually necessary so that the model does not rely too much on strong features (e.g. the word 'pet' for its class) and is done by penalizing high weights. It comes in two flavours, which are also used in other classification methods:

- L1 regularization: $R(\mathbf{w}) = \sum_{i=1}^{n} |w_i|$
- L2 regularization: $R(\mathbf{w}) = \mathbf{w}^T\mathbf{w}$

Both of them are common in text classification, L2 usually leading to a lot of features with small weights and L1 leading to a sparser solution with a few features with high weights. Usually $L(\mathbf{w}) = \sum_j p(y^j|x^j) - \lambda R$, but the C parameter can accomplish the same idea by decreasing/increasing the importance of the weighted feature combination.

We maximize the function using gradient descent (use the derivative of the function to move into the direction of gradient). Its advantages over Naive Bayes are related to removing the independence assumption. If two features are indeed correlated, then Logistic Regression will assign a more accurate probability then Naive Bayes, even though the NB choice may still be correct. It is generally a slower model to train than NB, but not necessarily better.

Sklearn package offers 2 options for training Logistic Regression classifiers: multinomial and one vs the rest, the former one minimising loss over all classes and the latter fitting a binary classifier to each class (more expensive in terms of training time, but perhaps more accurate). We will explore both these options. The solver parameter is the optimization method used to find the minimum of the objective function and, according to their specifications, saga is the best fit for high-dimensional data in terms of speed of convergence and performance.

6.2.1 Experiments

In our experiments, first, we choose one model for each language using default settings (C=1, reg='l2') from multinomial and OvR types. Then, we explore the strength of regularization necessary (by varying the C parameter), then we vary the vectorizer that provides the best results. Since we prefer more features with smaller weights, we chose to use L2 regularization. First, in table 6.3 we can analyse results of OvR vs multinomial logistic regression. Since for all languages the multinomial version performs better, we choose it for all our further experiments. Secondly, we choose the best regularization

Lang	Multinomial	OvR
en	0.497	0.49
fr	0.358	0.355
de	0.499	0.499
ja	0.416	0.417
zh	0.459	0.452
ar	0.497	0.485
ru	0.468	0.452
es	0.559	0.557
pt	0.586	0.568
pl	0.438	0.438

Table 6.3: Experiments with LogisticRegression(C=1, class_weight='balanced', max_iter=1500, solver='saga') CountVec bigram binary varying multiclass strategy

Lang	$C : 0.7$	$C : 1$	$C : 2$
en	0.502	0.497	0.491
fr	0.361	0.358	0.356
de	0.503	0.499	0.49
ja	0.416	0.416	0.422
zh	0.454	0.459	0.47
ar	0.489	0.497	0.489
ru	0.467	0.468	0.459
es	0.562	0.559	0.554
pt	0.589	0.586	0.581
pl	0.444	0.438	0.44

Table 6.4: Experiments with LogisticRegression(multi_class='multinomial', C=0.7/2 class_weight='balanced',max_iter=1500,solver='saga'), CountVec bigram binary, varying C value

parameter C for l2 regularization in table 6.5. It seems that C=0.7 is the best value for all languages except Japanese and Chinese, where it is C=2. With these values for each language, we perform one last experiment in which we vary the vectorizer, with results in table 6.4.

Note: Following our experiments in Chapter. 2, we will use class_weight='balanced' to penalise the model more for errors in infrequent classes. We will run all our experiments with a max_iter=1500 (i.e. maximum number of iterations for updating **w**).

6.3 Support Vector Machines

Support Vector Machines have been used successfully in text classification due to their effectiveness in high dimensional space (i.e. our sparse vectors). The paper from

Lang	CountVec	CountVec bigram	TfidfVec
en	0.499	0.502	0.48
fr	0.366	0.361	0.351
de	0.507	0.503	0.482
ja	0.414	0.422	0.411
zh	0.46	0.47	0.42
ar	0.496	0.489	0.467
ru	0.464	0.467	0.477
es	0.565	0.562	0.504
pt	0.589	0.589	0.547
pl	0.448	0.444	0.403

Table 6.5: Experiments with LogisticRegression(multi_class='multinomial', class_weight='balanced',max_iter=1500,solver='saga') varying data representation method

Joachims (1998) is one of the first to explore the benefits of this method, which is still largely used to this day. They have shown an increased performance of SVMs over other classification methods such as NB or KNN.

SVMs try to compute the decision boundary between the two classes as the line furthest from any training point. Usually, this boundary is a line (linear SVM), but it can take other more complex shapes if we plug in another kernel function (RBF, polynomial). The main reason for establishing a wide margin between classes is because the certainty of placing a point on one side or the other increases (slight miscalculations should not affect the final decision).

If we consider **w** to be perpendicular to the decision hyperplane and the intercept term b to define it among the infinite number of perpendicular planes, then all points **x** on the decision hyperplane have $\mathbf{w}^T\sigma(\mathbf{x})+b=0$. If we consider the binary classification task where the classes are +1/-1, then the prediction is computed as:

$$f(\mathbf{x}) = sign(\mathbf{w}^T\sigma(\mathbf{x})+b) \tag{6.4}$$

The decision hyperplane is defined only by the training points closest to each other (called support vectors), the other ones not being taken into consideration. One can see a visual representation of this idea in figure 6.1. Computing geometrically the distance from each training point to the hyperplane, we come to the standard formulation of an SVM as:

$$\min_{\mathbf{w},b} 1/2\mathbf{w}^T\mathbf{w} + C\sum_{i=1}^{n} l(y_i, f(\mathbf{x}_i)),$$

subject to $y_i f(\mathbf{x}_i) >= 1 - l(y_i, f(\mathbf{x}_i)),\ l(y_i, f(\mathbf{x}_i)) \geq 0,\ i = 1..n$. Function σ represents the kernel function chosen or identify function if the SVM is linear. Function $l(y_i, f(\mathbf{x}_i))$

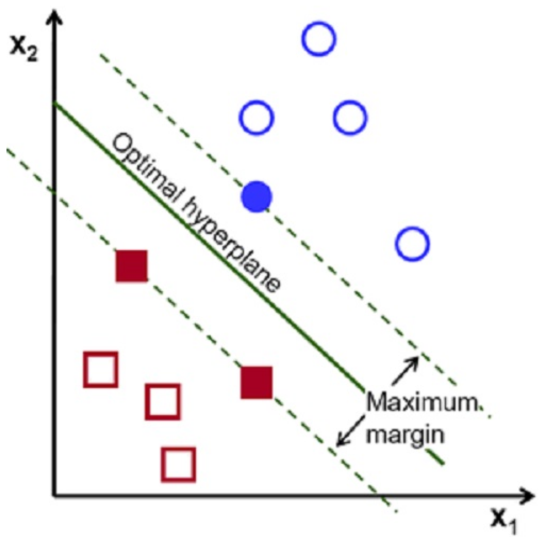


Figure 6.1: Illustration of SVM model from Eliot (2018)

represents the loss function and is used when the data is not clearly separable. For SVMs, the loss function, computing the current amount of training error is either:

- Hinge loss:
$$l(y_i, f(\mathbf{x}_i)) = max(0, 1 - y_i f(\mathbf{x}_i)) \tag{6.5}$$

- Squared hinge loss:
$$l(y_i, f(\mathbf{x}_i)) = (hingeloss)^2 \tag{6.6}$$

Parameter C controls this tolerable amount in training, such that for larger values, the margin will be smaller as to avoid any error, but for smaller values, the margin will enlarge and the classifier will make some accepted misclassifications.

In the multiclass case, both extensions from binary previously discussed are possible: OvR and OvO, even one more called structured SVM, but we choose to focus on the OvR method, basically constructing 14 different classifiers.

6.3.1 Experiments

In our experiments due to time constraints we have chosen to focus only on linear SVMs, represented in sklearn package as LinearSVC. Firstly, we experiment with squared-hinge vs normal hinge, then, having chosen a loss function we move on to experiment with the C parameter. Finally, we explore the best possible bag-of-words model from CountVectorizer bigram, Count Vectorizer bigram binarized and Tfidf model. Joachims (1998) suggests that using a Tfidf model increases the performance of the SVM so we expect to see the same in practice.

From 6.6 we learn that the two loss functions perform similarly, although the simple hinge function performs slightly better in most languages. As a result, we continue the

Lang	Hinge	Squared-hinge
en	0.454	0.456
fr	0.361	0.348
de	0.474	0.467
ja	0.427	0.415
zh	0.471	0.477
ar	0.481	0.464
ru	0.436	0.443
es	0.524	0.521
pt	0.556	0.55
pl	0.418	0.429

Table 6.6: Experiments with LinearSVC(class_weight='balanced',max_iter=1500, C=1), CountVectorizer bigram, binary and varying loss function

Lang	$C:0.7$	$C:1$	$C:2$	$C:5$
en	0.438	0.454	0.468	0.415
fr	0.346	0.361	0.367	0.331
de	0.457	0.474	0.479	0.441
ja	0.416	0.427	0.429	0.402
zh	0.47	0.477	0.473	0.473
ar	0.454	0.481	0.491	0.419
ru	0.429	0.436	0.441	0.41
es	0.509	0.524	0.527	0.487
pt	0.538	0.556	0.567	0.515
pl	0.415	0.418	0.413	0.406

Table 6.7: Experiments with LinearSVC(loss='hinge', class_weight='balanced', max_iter=1500) CountVectorizer bigram, binary and varyind C parameter

experiments with hinge function and we vary C to be 0.7, 1, 2 or 5 with results in table 6.7. We notice that results for C=1 and C=2 are very close, so we continue with these values in experiments with the 3 mentioned types of vectorizers (see table 6.8). It is indeed true that the tfidf data representation has the best results, regardless of the C value. We also notice that generally C=1 is the best value.

6.4 Stochastic gradient descent

These classifiers implement Stochastic Gradient Descent (SGD) when they are minimizing the loss function (regardless of its form). So far, we have analysed and experimented with (Batch) Gradient Descent, which is significantly slower, but perhaps more accurate than stochastic gradient based ones which are designed for large amounts of training data.

		Vectorizer		
Lang	C value	Count Vect	Count Vect binary	Tfidf Vect
en	1	0.455	0.454	0.5
	2	0.438	0.468	0.488
fr	1	0.362	0.361	0.399
	2	0.345	0.367	0.388
de	1	0.472	0.474	0.52
	2	0.456	0.479	0.506
ja	1	0.426	0.427	0.448
	2	0.415	0.429	0.442
zh	1	0.466	0.477	0.463
	2	0.47	0.473	0.475
ar	1	0.478	0.481	0.506
	2	0.447	0.491	0.503
ru	1	0.437	0.436	0.468
	2	0.43	0.441	0.467
es	1	0.522	0.524	0.581
	2	0.513	0.527	0.561
pt	1	0.557	0.556	0.601
	2	0.539	0.567	0.59
pl	1	0.427	0.418	0.45
	2	0.421	0.413	0.447

Table 6.8: Experiments with LinearSVC(class_weight='balanced', max_iter=1500) and varying data representation

A general form of the training error which we are trying to minimize is:

$$E(\mathbf{w}) = 1/n \sum_{i=1}^{n} L(y_i, f(\mathbf{x}_i)) + \alpha R(\mathbf{w}),$$

where $L(y_i, f(\mathbf{x}_i))$ can be any loss function (log loss for Logistic Regression, hinge loss/squared hinge loss for SVMs and others) and R($\mathbf{w}$) represents the regularization term. Then, using vanilla gradient descent, at each training step, for each dimension j of $\mathbf{w}$ we update w_j as follows:

$$w_j = w_j - \eta(\frac{\partial R(\mathbf{w})}{w_j} + \sum_{i=1}^{n} \frac{\partial L(\mathbf{w}^T\mathbf{x}_i + b, y_i)}{\partial w_j}),$$

meaning we have to store all the training points $(\mathbf{x}_i, y_i)$ and sum gradients for all of them. However, when we do SGD, at each training step, we compute the gradient for w_j only at one training point $\mathbf{x}_i$, instead of the whole training set:

$$w_j = w_j - \eta(\frac{\partial R(\mathbf{w})}{w_j} + \frac{\partial L(\mathbf{w}^T\mathbf{x}_i + b, y_i)}{\partial w_j})$$

The main advantage for doing this is that it converges much faster than batch gradient descent because it updates weights very frequently. While batch gradient descent is guaranteed to converge to local minimum (global minimum for convex surfaces), SGD has a more erratic behaviour and, should the learning rate (η) be too high, it might keep jumping over the minimum. However, a neat solution is to decrease the learning rate as we increase the epochs (done by sklearn by default with the parameter learning_rate=optimal).

This type of classifier was used in the initial experiments (Chapters 3,4) due to its fast training and relatively good results (the loss function is by default set to be hinge loss). Even if we went through the training data only 50 times (max_iter was set to 50), we still obtained comparable results to our SVM classifier from previous section (6.8).

Due to decreased training time, we can attempt all the loss functions so far: log loss (eq 6.3) (the stochastic variant of Logistic Regression) and hinge loss (eq 6.5), squared hinge loss (eq 6.6) and also with modified huber loss function, defined as follows:

$$L(\mathbf{w}) = \begin{cases} max(0, 1 - y_i f(\mathbf{x}_i))^2 & \text{for } y_i f(\mathbf{x}_i) \geq -1 \\ -4 y_i f(\mathbf{x}_i) & \text{otherwise,} \end{cases} \tag{6.7}$$

where $f(\mathbf{x}_i)$ is defined as in equation 6.4. This is proved to be less sensitive to outliers than others loss functions in certain cases.

6.4.1 Experiments

Lang	hinge	log	modified_huber	squared_hinge
en	0.49	0.449	0.509	0.389
fr	0.411	0.36	0.399	0.248
de	0.517	0.484	0.508	0.347
ja	0.434	0.34	0.439	0.352
zh	0.468	0.429	0.477	0.316
ar	0.506	0.456	0.517	0.3
ru	0.475	0.454	0.475	0.337
es	0.583	0.508	0.578	0.425
pt	0.571	0.499	0.586	0.387
pl	0.45	0.421	0.443	0.267

Table 6.9: Experiments with SGDClassifier(penalty='l2', alpha=0.0001, class_weight='balanced', max_iter=1500) CountVectorizer bigram, binary, varying loss function

We experiment with the above 4 mentioned loss functions, alongside the default L2 regularization with default parameter $\alpha = 0.0001$ with a Count Vectorizer bigram binary. The results can be analysed in table 6.9. We notice that log loss and especially squared_hinge loss are underperforming, but hinge and the modified_huber loss display some of the best results so far. We continued to experiment with hinge and modified_huber with the other two considered vectorizers with results in table 6.10. It seems that SGD classifier with modified_huber loss and Tfidf Vectorizer provide the best results so far in some languages (English, French, Chinese, Arabic, Russian and Polish):

		Vectorizer		
Lang	Loss	Count Vect	Count Vect binary	Tfidf Vect
en	hinge	0.493	0.49	0.471
	modified_huber	0.504	0.509	0.46
fr	hinge	0.411	0.411	0.397
	modified_huber	0.399	0.399	0.412
de	hinge	0.512	0.517	0.5
	modified_huber	0.509	0.508	0.515
ja	hinge	0.433	0.434	0.419
	modified_huber	0.439	0.439	0.398
zh	hinge	0.477	0.468	0.452
	modified_huber	0.477	0.477	0.484
ar	hinge	0.507	0.506	0.497
	modified_huber	0.517	0.517	0.518
ru	hinge	0.47	0.475	0.457
	modified_huber	0.475	0.475	0.486
es	hinge	0.579	0.583	0.553
	modified_huber	0.577	0.578	0.529
pt	hinge	0.571	0.571	0.559
	modified_huber	0.585	0.586	0.535
pl	hinge	0.449	0.45	0.454
	modified_huber	0.443	0.443	0.466

Table 6.10: Experiments with SGDClassifier(class_weight='balanced', max_iter=1500, penalty='l2', alpha=0.0001, loss='modified_huber' or 'hinge') and different vectorizers

6.5 K Nearest Neighbours

This classifier is the simplest and most intuitive classifier so far. The training phase consists of simply storing all the training data and then, when predicting, it computes the distances to all the training points and chooses the k nearest ones. The predicted class is chosen as a result of a vote between the k nearest neighbours. The performance depends on k: if it is too small, then we might take into consideration noisy points into the decision, but if it too high, it may consider points completely unrelated to the current point. We can deal with class imbalance by taking into account closer points than further ones from the closest k ones.

KNN is likely to underperform in high dimensional cases, due to the fact that many features may be irrelevant to the category choice and closeness to other similar points in terms of these features may hinder the decision. Thus, feature selection methods such as selecting p best features according to statistical measures such as chi2, mutual information etc., removing features with low variance, removing features with 0 weight after training a model with l1 regularization and even reducing data dimensionality with PCA may help the classifier.

In our experiments, we selected the best 70% of the features using the statistical measure chi squared. This test determines the likelihood that the feature and the class are indeed correlated by measuring coocurrences of them. By doing this feature selection we choose to ignore the most irrelevant statistically speaking features.

6.5.1 Experiments

The experiments with KNN are limited due to poor results. We believe that the feature space is too high (with bigrams even more), for KNN to perform well on our data. In table 6.11 one can analyse the performance of KNN with n=3/5/10 neighbours, with uniform importance of neighbours vs inversely proportional to their distance, with no feature selection vs. with the most important 70% features. The results are far lower than any other classifier so far, but we do notice that it seems to perform a little bit better with lower number of neighbours (3) and that considering close neighbours more has a positive impact. The results are unclear with regard to feature selection.

We performed only one last experiment to see if using a unigram bag-of-words model improves the performance significantly given n_neighbours=3. As proven in table 6.12, the performance improved a little bit with unigrams than bigrams, but not enough to compete with any other method so we stopped experimenting with KNN. There is possible that there is another feature selection method or dimensionality reduction that might help KNN, but it might also be that the data is too high dimensional and noisy for any configuration to provide competitive results.

6.6 Final chosen classifier

In this chapter we have attempted to explore which classification method may have best results on our balanced validation datasets. We explored with Multinomial Naive Bayes, Logistic Regression, SVMs, Stochastic classifiers etc. and we noticed that each language seems to be best modeled by a different type of classifier. Among the most successful ones we can name SGD with loss='modified_huber', as well as the linearSVC. We are aware that there may be even better performing methods, but we have chosen to continue experimenting with other techniques to increase the performance, as discussed in the next chapter.

In table 6.13 we present the best and second best results for each language, together with the achieved maximum f1 macro score on the basic dataset. We mention that the bigram version of each vectorizer is used. We notice that French/Japanese/Polish

		No feat sel.		Chi2 70%	
Lang	N neighbours	Uniform	Distance	Uniform	Distance
en	3	0.167	0.236	0.17	0.232
	5	0.146	0.191	0.158	0.201
	10	0.123	0.165	0.132	0.172
fr	3	0.145	0.194	0.139	0.19
	5	0.163	0.208	0.144	0.192
	10	0.126	0.176	0.08	0.158
de	3	0.171	0.251	0.172	0.247
	5	0.165	0.23	0.127	0.189
	10	0.134	0.19	0.09	0.148
ja	3	0.214	0.266	0.185	0.236
	5	0.184	0.233	0.159	0.219
	10	0.141	0.226	0.1	0.186
zh	3	0.176	0.211	0.156	0.167
	5	0.152	0.183	0.153	0.18
	10	0.165	0.196	0.114	0.14
ar	3	0.147	0.212	0.128	0.179
	5	0.134	0.183	0.106	0.135
	10	0.103	0.145	0.05	0.086
ru	3	0.194	0.248	0.199	0.254
	5	0.171	0.219	0.166	0.228
	10	0.191	0.104	0.199	0.145
es	3	0.252	0.293	0.244	0.289
	5	0.215	0.237	0.214	0.254
	10	0.168	0.216	0.174	0.223
pt	3	0.214	0.317	0.24	0.308
	5	0.182	0.275	0.21	0.273
	10	0.161	0.234	0.138	0.23
pl	3	0.163	0.21	0.159	0.21
	5	0.143	0.194	0.138	0.192
	10	0.117	0.182	0.1	0.179

Table 6.11: Experiments with KNN using CountVect bigram, binary version varying number of neighbours and voting mechanism (Uniform vs. based on distance to the test point)

classifiers seem to perform the worst so far. It is interesting to see how even though in Japanese we have a very large training set compared to Chinese(where the performance is the third best) (590k vs 91k tweets from table 3.1), Japanese classifier is much worse than the Chinese one. Thus we prove that the classification performance is influenced not only by the training set size, but also by the particularities of the language, the noise level in the training set. Our next set of experiments is going to explore some techniques designed to increase the classifier performance, but this time, they are applied to the data.

Lang	No feat sel. Unigram	No feat sel. Bigram	Chi2 70% Unigram	Chi2 70% Bigram
en	0.246	0.236	0.241	0.232
fr	0.219	0.194	0.209	0.19
de	0.261	0.251	0.252	0.247
ja	0.28	0.266	0.249	0.236
zh	0.237	0.211	0.199	0.167
ar	0.216	0.212	0.192	0.179
ru	0.26	0.248	0.256	0.254
es	0.311	0.293	0.31	0.289
pt	0.341	0.317	0.329	0.308
pl	0.199	0.21	0.201	0.21

Table 6.12: Experiments with KNN(n_neighbours=3, weight='distance') CountVectorizer binary unigram vs bigram

Lang	Classification method Vectorizer	Classification method Classifier	Value
en	CountVec(Bin)	SGD(modified_huber)	0.509
	CountVec(Bin)	MNB(alpha=0.01, f_p=False)	0.507
fr	TfidfVec	SGD(modified_huber)	0.412
	CountVec	SGD(hinge)	0.411
de	TfidfVec	LinearSVC(C=1)	0.52
	TfidfVec	SGD(modified_huber)	0.515
ja	CountVec(Bin)	MNB(alpha=0.01)	0.449
	CountVec(Bin)	MNB(alpha=0.01, f_p=False)	0.447
zh	TfidfVec	SGD(modified_huber)	0.484
	CountVec	SGD(hinge)	0.477
ar	TfidfVec	SGD(modified_huber)	0.518
	CountVec	SGD(hinge)	0.517
ru	TfidfVec	SGD(modified_huber)	0.486
	CountVec	SGD(hinge)	0.475
es	CountVec(Bin)	SGD(hinge)	0.583
	TfidfVec	LinearSVC(C=1)	0.581
pt	TfidfVec	LinearSVC(C=1)	0.601
	CountVec(Bin)	MNB(alpha=0.01, f_p=False)	0.598
pl	TfidfVec	SGD(modified_huber)	0.486
	TfidfVec	LinearSVC(C=1)	0.45

Table 6.13: First 2 best classification methods for each language

Chapter 7

Enrichment methods

In the original paper, Magdy et al. (2015), there were 2 enrichment methods used to improve the performance of the classifier in English. In this chapter, we will explain what the 2 techniques are and even add a new one that proved to have the highest potential of all so far.

Based on the results so far, we have established the first two classification methods for each language (i.e. the ones that gave the highest potential on a basic dataset. The methods are summarized in table 6.13. We are going to use these two classifiers for each language and test them on the new, improved datasets to examine the potential of each technique.

First, we discuss the original 2 enrichment techniques:

- **Adding the title of the video to the tweet**: Since all our training data has a YouTube video attached to it, we can append its title to the text of the tweet. As such, even though the tweet may not be useful to the classifier ('I love this video'), the added title may teach the classifier about other things (e.g. artists, bands, actors etc.). Such an example is 'I love this video: Beyonce - Single girls' paired with the Music category. The classifier will thus associate Beyonce to Music, which is correct. Since most of the videos come with it already appended, one of our initial cleaning steps was to remove it to simulate a real-life tweet. Now we choose to skip this cleaning step and make sure the title is part of the tweet.

  Note: When we removed the title of the video, the remaining text was usually automatically generated text (e.g. 'I love this video'). These tweets got deduplicated so the training set size decreased significantly. When we add the video title, these automatically generated tweets are not deduplicated anymore(e.g. 'I love this video Beyonce - Single girls' is different from 'I love this video Eminem - Mockingbird'). One can analyse the impact of doing this in figure 7.1, which also displays the size of the new training set size when this enrichment technique is applied.

- **Adding the hashtags items as normal words**: So far our dataset kept the hashtag words as hashtags (i.e. '#football' is different than the word 'football'). The proposed enrichment technique is to add the hashtag terms as normal words at the end of the tweet (e.g. 'I love #football #fifa' -> 'I love #football #fifa football fifa').

Doing this does not impact the size of the training set, since it does not affect the deduplicating behaviour.

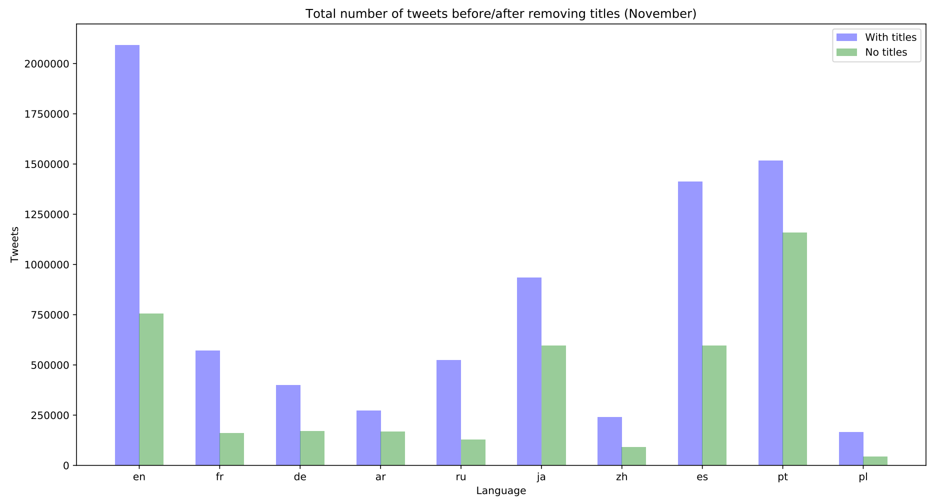


Figure 7.1: Size of the training set with titles removed / titles kept (or added) for December

We chose to experiment with 4 different datasets:

- **train00**: the basic tweets set that has been used so far (no titles added, hashtags considered hashtags)
- **train01**: tweets with hashtags added as normal words as well.
- **train10**: tweets with video titles appended to the texts.
- **train11**: tweets with both video titles and hashtag terms as normal words appended.

Each has a balanced validation set constructed, but the validation set tries to mimic the real-life tweets as much as possible, so we apply none of these enrichment methods (it continues to look like train00 dataset).

The experiment results can be analysed in tables 7.1 and 7.2 respectively. We notice that adding hashtags as normal words does not bring improvements in most languages (only exception French, where the best dataset is train11). All the languages benefit from adding the title of the video to the text tweet, regardless of the classifier that we use. An interesting result is that it is not guaranteed that if the classifier gave the best result on train00 it gives the best result on other datasets. Perhaps one of the best examples is English, where the best classifier gives a 0.51 score on train10 and the second best classifier gives a 0.535 on the same dataset.

Lang	train00	train01	train10	train11
en	0.509	0.507	0.51	0.507
fr	0.412	0.412	0.396	0.397
de	0.52	0.516	0.545	0.544
ja	0.449	0.445	0.467	0.461
zh	0.484	0.472	0.504	0.501
ar	0.516	0.516	0.516	0.511
ru	0.486	0.484	0.487	0.487
es	0.583	0.577	0.579	0.572
pt	0.601	0.6	0.624	0.622
pl	0.466	0.463	0.503	0.504

Table 7.1: Experiments with enrichment techniques on the best classification methods as specified in table 6.13 (first method)

Lang	train00	train01	train10	train11
en	0.507	0.504	0.535	0.532
fr	0.411	0.407	0.429	0.432
de	0.515	0.514	0.515	0.513
ja	0.447	0.442	0.464	0.457
zh	0.477	0.461	0.558	0.554
ar	0.516	0.499	0.511	0.514
ru	0.475	0.464	0.495	0.496
es	0.581	0.577	0.589	0.582
pt	0.598	0.596	0.616	0.614
pl	0.45	0.453	0.504	0.504

Table 7.2: Experiments with enrichment techniques on the second best classification methods as specified in 6.13 second method

The last enrichment method that was not discussed in Magdy et al. (2015) is transforming the text to ASCII characters (or as close as possible). So far the tweet text was represented in utf-8 encoding, because almost all languages discussed use special characters (think Chinese, Japanese, Russian). We initially tested this on French only, thinking that some people write with accents, some without, and the classifier considers them as separate words. Removing the accents would equalize the two versions. Warning: it may also be the case that this way we consider 2 words to be the same even though they are not (only difference in spelling being the accents e.g. in German 'schon' - already vs 'schön' - beautiful). To do this we have used the python open-source package unidecode. As specified in the documentation, it tries to transform the text as well as possible into what a native speaker would spell with an American keyboard. While it is expected to give near perfect results on languages close to English (e.g. French, Portuguese etc.), it is not great in languages such as Chinese. Doing this also removes a lot of noise automatically, such as detected emoticons that cannot be translated.

As seen in 7.3 doing this improves the performance of the classifier more than any other method. With the exception of Arabic, the f1 macro score increased in all languages, less spectacularly only in languages close to English, where the use of accents in limited (Spanish, Portuguese).

Lang	utf-8	ASCII
en	0.535	0.556
fr	0.432	0.502
de	0.545	0.559
ja	0.467	0.592
zh	0.558	0.57
ar	0.516	0.505
ru	0.495	0.519
es	0.589	0.589
pt	0.624	0.627
pl	0.504	0.506

Table 7.3: Experiment with utf-8 encoding vs ASCII translation using the best training set and best classification method from the previous experiments

We conclude this chapter with a summary of the best classification methods, best enrichment technique that should be applied for each language. We have finished experimenting with data collected in one month only and the system will use the datasets, enrichment techniques and classifiers as described in table 7.4. We will translate to ASCII in all languages except Arabic, where the results showed a decreased performance.

	Best classification method				
Lang	Enrichment	utf-8/ascii	Vectorizer	Classifier	F1 score
en	titles	ascii	CountVec(bin)	MNB(α=0.01, f_p=False)	0.556
fr	titles+hashtags	ascii	CountVec	SGD(hinge)	0.509
de	titles	ascii	TfidfVec	LinearSVC(C=1)	0.559
ja	titles	ascii	CountVec(bin)	MNB(α=0.01)	0.592
zh	titles	ascii	CountVec	SGD(hinge)	0.57
ar	titles	utf8	TfidfVec	SGD(modified_huber)	0.516
ru	titles	ascii	CountVec	SGD(hinge)	0.535
es	titles	ascii	TfidfVec	LinearSVC(C=1)	0.589
pt	titles	ascii	TfidfVec	LinearSVC(C=1)	0.624
pl	titles	ascii	TfidfVec	SGD(modified_huber)	0.506

Table 7.4: Final classification methods chosen for the entire system for the month of November 2017

Chapter 8

Monthly classification

In this chapter we explore results using data from other months than November. In particular, we construct a classifier for each month December, January and February and use it to test not only data from that respective month, but data from other months as well. Moreover, we explore if it might be better to combine data from different months to achieve a better performance.

We notice that the storing the trained models in Sklearn is very costly in terms of memory, so we explore SVMlight multiclass version, which creates a very compact and fast model based on an SVM classification. The original implementation is based on models described in Joachims et al. (2009) and Tsochantaridis et al. (2004). This is also the method that was used in the original paper for Class Strength v1 (Magdy et al. (2015))

8.1 Analysis of results for December, January

For data from December and all the other months we performed the best chosen course of action as detailed in the last chapter. We created a large, imbalanced dataset and a rather small balanced test set for each month. To motivate the choice for an adaptive classifier, we tried to use a classifier for a month to classify the test set from another month. The results (Table 8.1 for classifier trained on December data and table 8.2 for classifier trained on January data) are interesting to analyse as follows:

- The bigger time distance between when the training data was collected vs. when the test data was collected, the lower the performance. There are only a few exceptions to this (e.g. in Japanese), most likely due to natural dynamics of tweets.
- We also notice a similarity of results between all months for all languages (English in the 0.5-0.55 range, Spanish and Portuguese over 0.55 etc.) This is interesting to notice because the raw data collected, at least in English and Arabic, increased by 2 times from November to December (remember figure 2.2, but the F1 scores are stable. This only strengthens the conclusion that, although more data means

more training points, it also means more noise, apparently enough to cancel the expected positive effect of an increase in training data.

- Moreover, we notice that even though we have more data collected in languages such as French than Polish, this does not guarantee a better performance in French. Apparently the French results are the worst from all languages. This may well be due to French tweeting style, perhaps mixing with other languages or other factors.

The results for February and November are highly similar to these so they are not attached.

Lang	November	December	January	February	March
en	0.552	0.547	0.531	0.492	0.491
fr	0.48	0.504	0.451	0.413	0.47
de	0.505	0.556	0.532	0.46	0.467
ja	0.649	0.534	0.615	0.559	0.575
zh	0.539	0.636	0.541	0.469	0.498
ar	0.48	0.479	0.473	0.429	0.438
ru	0.483	0.507	0.499	0.46	0.452
es	0.543	0.596	0.537	0.527	0.505
pt	0.624	0.661	0.557	0.533	0.523
pl	0.466	0.578	0.515	0.409	0.389

Table 8.1: Experiment with classifiers based on data from December

Lang	November	December	January	February	March
en	0.535	0.517	0.547	0.5	0.503
fr	0.446	0.444	0.477	0.421	0.475
de	0.5	0.526	0.57	0.464	0.477
ja	0.615	0.595	0.582	0.567	0.582
zh	0.521	0.536	0.586	0.49	0.473
ar	0.447	0.45	0.49	0.448	0.452
ru	0.456	0.46	0.527	0.458	0.462
es	0.488	0.515	0.6	0.557	0.539
pt	0.587	0.604	0.624	0.592	0.565
pl	0.419	0.385	0.558	0.471	0.438

Table 8.2: Experiment with classifiers based on data from January

8.1.1 Combining data

So far we explored with using data from only one month, but now we will explore using data from December in particular if the performance increases if we combine data with other months. More particularly, we could use two strategies in the long term:

- Sliding window of 2 (2m-window): We could add data from previous month to current data (First month of November would be on its own)
- Sliding window of 3 (3m-window): We could add data from previous and next month corresponding to the current one. If there is no next month, we could just use the previous one and when data collection for one month completes we also update the previous month's classifier.

The results for the two described strategies for the month of December are presented in table 8.3. It seems that the performance depends massively on the quality of the data from those months. For example, in Spanish and Polish, the quality even decreased when we combined with poor data from November or December. The main issue, however, appeared when we tried to save these models. In order to save the trained classifiers we used the pickle python module, which saves any python object in memory. The models for some languages, especially English and others too, are particularly large (>2GB) in occupied space, which is more than the storage space required even for the training data. Combining 3 months in this format, using Sklearn + Pickle, presents extreme memory issues. Even saving a classifier based on a single month in English is more than 1.5Gb of memory. These issues were unknown in the beginning of the project, when we first used Sklearn, and it quickly became obvious that it might be difficult to use these classifiers for the online version due to 2 issues: high loading time for each and impossibility of keeping so many of them loaded in memory at all times (think 10 languages x at least 5 months). The single-month based classifiers in all languages are continously computed and stored on the server, but for the online service, we decided to also store a lighter SVMlight Multiclass model.

Lang	Single	2m-window	3m-window
en	0.538	0.565	0.573
fr	0.496	0.519	0.52
de	0.566	0.595	0.624
ja	0.534	0.568	0.667
zh	0.628	0.654	0.668
ar	0.483	0.491	0.501
ru	0.525	0.559	0.561
es	0.64	0.616	0.63
pt	0.688	0.694	0.717
pl	0.578	0.485	0.505

Table 8.3: Experiment with combining classifiers for the month of December

8.2 SVMlight Multiclass

SVMlight Multiclass [1] is a program written in C that creates very compact models based on linear support vector machines in the multiclass case. This program has been used

[1] https://www.cs.cornell.edu/people/tj/svm_light/svm_multiclass.html

in the initial paper Magdy et al. (2015) successfully and the trained model occupies around a third or less of the space required by Sklearn (around 0.5-0.6 Gb for the large dataset type languages such as English, Portuguese, Spanish).

The main issue for using it is that is absolutely requires a balanced training set (it has no option for a modified loss function). The way we dealt with this situation is by experimenting with 3 possible solutions:

- Undersampled dataset (U case): Undersample each category to match the size of the smallest category
- Combined undersampled datasets (C case): Undersample each category as previously and then join data from 3 months like the sliding window option.
- Oversampled dataset (O case): Oversample each category to match the maximum number of items in one category (up to a max of 30000 items in each category)

We used Sklearn to construct the Tfidf bigram representation of these datasets (Tfidf because both Joachims (1998) and previous experimental results show a better performance for SVMs) and then we fed specially formatted inputs to the SVMlight classifier. Based on previous experiments, we also used the enrichment methods that proved successful for the other classifiers. The main experimentation parameter for SVMlight is C(described as trade-off between training error and margin). This parameter had to be increased greatly (unlike the Sklearn LinearSVC), up to more than 1000000 in some cases (it depends on the number of training points available) until at some point increasing it did not increase the f1 score (it stabilized).

	U case			C case			O case			Nov
Lang	Nov	Dec	C	Nov	Dec	C	Nov	Dec	C	BestSK
en	0.45	0.436	75k	0.501	0.486	6M	0.407	0	err	0.556
fr	0.396	0.396	50k	0.461	0.453	4M	0.472	0.487	3M	0.509
de	0.454	0.443	50k	0.527	0.516	3M	0.529	0.538	1.5M	0.559
ja	0.493	0.466	75k	0.612	0.599	4.5M	0.468	0	err	0.592
zh	0.508	0.522	75k	0.585	0.622	2M	0.585	0.642	2M	0.57
ar	0.411	0.414	20k	0.446	0.46	1.5M	0.501	0.482	0.1M	0.505
ru	0.454	0.447	75k	0.49	0.516	3M	0.518	0.508	1M	0.519
es	0.522	0.513	75k	0.562	0.589	4M	0.535	0.488	3.5M	0.589
pt	0.539	0.532	75k	0.596	0.597	4.5M	0.512	0.5	4M	0.624
pl	0.477	0.423	10k	0.504	0.48	500k	0.526	0.529	200k	0.506

Table 8.4: Experiments with SVMlight Multiclass

The results for experimentation for the month of November and December in the three cases described above are presented in table 8.4, alongside the experimentally found optimum value of C. It is interesting to see that in some cases (Japanese, Chinese, Polish) we obtain better results with SVMlight Multiclass than our best classifier so far. Moreover, we note that the oversample case suffers from memory issues in languages with many number of tokens (English and Japanese), so much so that we get errors when we tried with C values >50k (and the accuracies then were around 0.2). It seems that in some languages (French, German, Arabic, Russian and Polish) the oversample results is higher than the Combined case (especially so in Polish and Arabic), but this happens at a memory cost (the oversampled model is at least 5x as large as the combined model). As such, as best options in terms of SVMlight we chose the combined model (taking into account the memory usage) for all languages except Arabic and Polish, where the difference in performance is significant.

8.3 Gold set

All the experiments so far have been tested on the validation set which was obtained the same way as the training data. This presents obvious disadvantages because it is just as noisy as the training set. A low f1 score may be due to poor classification performance or very noisy validation set. To solve this a gold set is needed (a set of tweets manually labeled by people), just as it was used in Magdy et al. (2015). We used the same gold set in English (almost balanced with 84 to 128 tweets in each category) and tested with the best Sklearn stored classifier from January 2018 and obtained an f1 score of 0.159 and an accuracy of 0.21. While the figure is very low, it is explicable by the fact that the gold set is about 3 years old and we cannot expect our classifier to know about news or artists from 3 years ago. This is the perfect motivation for creating an adaptive version of the Twitter Classifier.

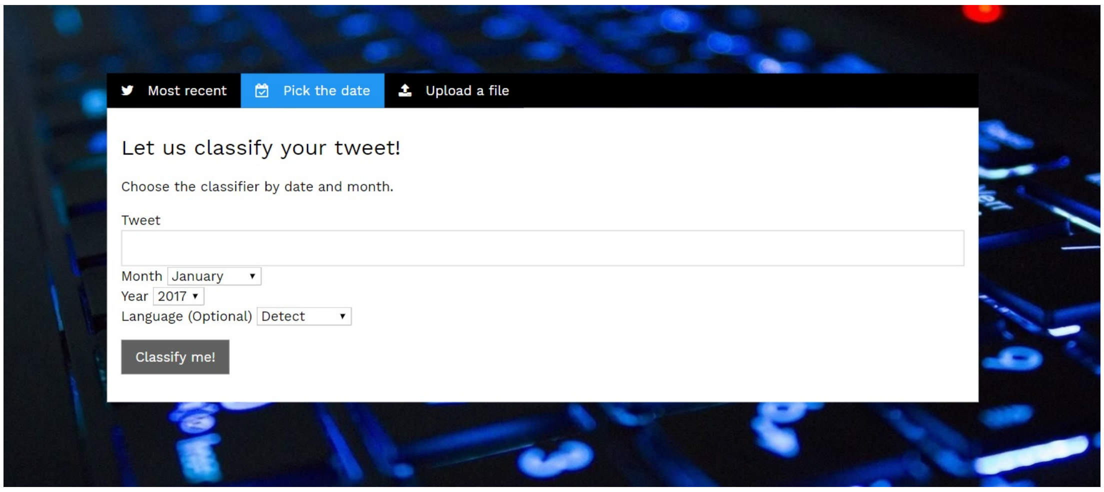
Substantial efforts were made to create a current, new gold set in English, French and German. Using the same technique described in the original paper, we collected 3000 tweets in each language by searching tweets with hashtags related to the assigned category (e.g. #truck for Autos&Vehicles) and we managed to obtain a natural, balanced set of tweets (i.e. without YouTube videos). We created multiple projects on the crowdflower platform [2] , but the efforts were cut short due to lack of funds for this endeavour (300$ at minimum for one language).

[2] https://www.crowdflower.com/

Chapter 9

Online System

In this chapter we present the online systems that are visible within the School of Informatics at the links http://hawksworth.inf.ed.ac.uk:5000/twitter for SVMlight based and http://hawksworth.inf.ed.ac.uk:5001/twitter for the Sklearn based one. This classifier is running continuously and has 3 main options: choose the most recent classifier, choose the date of the classifier you want to use or submit a file with tweets to be classified. Some screenshots from the currently online system can be seen in figures 9.1 and 9.2. If the date provided is unavailable (in the future or before Nov 2017) or the language detected is either not supported on unknown, we display an error screen with a link to Try again (see figure 9.3).



Let us classify your tweet!

Choose the classifier by date and month.

Tweet

Month January

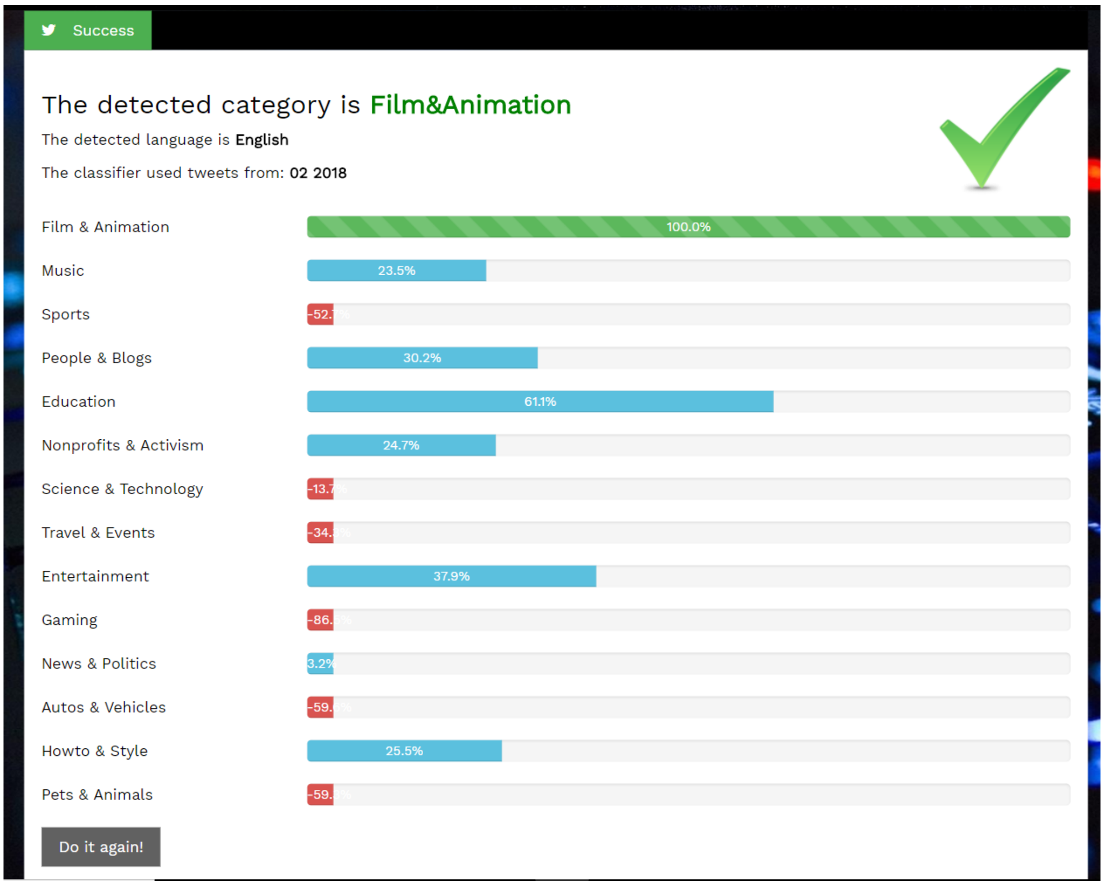
Year 2017

Language (Optional) Detect

Figure 9.1: First page, second tab: Choose your own classifier

The monthly update of the classifier is done effortlessly. If a user selects a certain date, we try to select that particular classifier. If the classifier does not exist, we show an error page, but with time passing, more and more classifiers will exist and be available to the user automatically.

We have constructed two such systems, one based on the Sklearn classifier, and the default one based on the SVMlight package. With the Sklearn based one, the classifiers



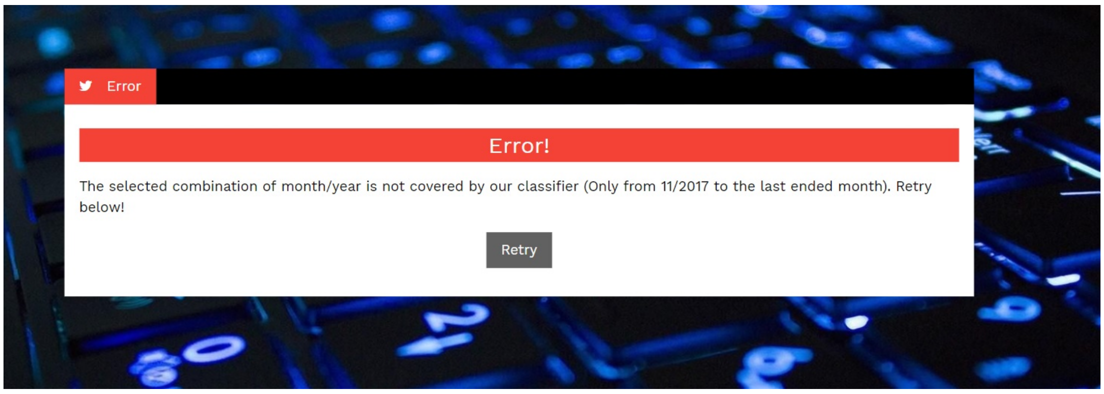
The detected category is **Film&Animation**

The detected language is **English**

The classifier used tweets from: **02 2018**

Film & Animation	100.0%
Music	23.5%
Sports	-52.7%
People & Blogs	30.2%
Education	61.1%
Nonprofits & Activism	24.7%
Science & Technology	-13.7%
Travel & Events	-34.3%
Entertainment	37.9%
Gaming	-86.5%
News & Politics	3.2%
Autos & Vehicles	-59.6%
Howto & Style	25.5%
Pets & Animals	-59.3%

Figure 9.2: Success page after classification with class strengths for SVMlight based one. Classified tweet: "It took J.K. Rowling, 6 years to write Harry Potter and the Sorcerer's Stone. Never give up. #amwriting #amediting Write ...Keep Writing."



Error!

The selected combination of month/year is not covered by our classifier (Only from 11/2017 to the last ended month). Retry below!

Figure 9.3: Date error page

are kept in memory indefinitely if at least a user has used that particular classifier (language, month, year). With the SVMlight system we are reading every time the model to classify the tweet, which, although more sustainable in the long term, is slower for individual tweets than the Sklearn based classifier. The SVMlight based application can also provide class strengths for each class (given the discriminant value for each class, we normalize them such that the highest value is scaled to 100% and the others scaled with the same ratio) and is intuitively more helpful, as well as having a more informative interface. This is not possible to do in the other classifier because some chosen classifiers do not have any class strengths (e.g. multinomial naive bayes)

Both systems are currently running (port 5000 for SVMlight and port 5001 for Sklearn), but this is unlikely to happen indefinitely. Given that enough people try both applications we can obtain some useful feedback as to which one we should keep running.

Chapter 10

Conclusion

In this project we presented an Adaptive Twitter Classifier in 10 languages: English, French, German, Japanese, Chinese, Arabic, Russian, Spanish, Portuguese and Polish, which is built based on multiple conceptual blocks: data collection, data preprocessing, classifier training and online system. Our main task is multiclass classification, having 14 categories such as: People&Blogs, Music, Sports, Pets&Animals, Education etc. We summarize the project by recapitulating the chosen implementations for each such block:

- **Data collection**: We collect tweets in all 10 languages that have a YouTube video included. We explored both Search API and Streaming API for Twitter and found that in English and Arabic using the Streaming API is beneficial, while for all the other languages Search API provides more results. The system currently running is using both APIs to collect tweets in all languages and we only pass to preprocessing the larger set of the two.

- **Data preprocessing**: Each day, we preprocess each Tweet from the previous day (e.g. remove links, lowercase, tokenise etc.) and obtain the YouTube video category from the attached link (which will later serve as label for the tweet). At the end of each month, we group the tweets by categories and construct training and validation sets.

- **Classifier training**: We build 2 distinct classifiers each month (Sklearn based and SVMlight Multiclass based), second one born out of memory issues with the first type. Experimentally, we have found that the best data representation methods, classifiers and enrichments methods for each language to be the ones in table 10.1 (the same one as 7.4). For the SVMlight based classifier, we usually combine undersampled datasets from 3 months (except in Arabic and Polish where we used oversampled datasets for a month) to build a classifier based only on linear SVMs and tfidf bigram representation.

- **Online sytem**: We have an online system running continuously at http://hawksworth.inf.ed.ac.uk:5000/twitter within the School of Informatics network. Every time a month's classifier is trained and stored, it is automatically incorporated in the online version. It allows the user to classify a tweet/file containing tweets in any of the 10 languages either using the most recent classifier or a dated one.

Lang	Best classification method				
	Enrichment	utf-8/ascii	Vectorizer	Classifier	F1 score
en	titles	ascii	CountVec(bin)	MNB(α=0.01, f_p=False)	0.556
fr	titles+hashtags	ascii	CountVec	SGD(hinge)	0.509
de	titles	ascii	TfidfVec	LinearSVC(C=1)	0.559
ja	titles	ascii	CountVec(bin)	MNB(α=0.01)	0.592
zh	titles	ascii	CountVec	SGD(hinge)	0.57
ar	titles	utf8	TfidfVec	SGD(modified_huber)	0.516
ru	titles	ascii	CountVec	SGD(hinge)	0.535
es	titles	ascii	TfidfVec	LinearSVC(C=1)	0.589
pt	titles	ascii	TfidfVec	LinearSVC(C=1)	0.624
pl	titles	ascii	TfidfVec	SGD(modified_huber)	0.506

Table 10.1: Final classification methods chosen for the entire system for the month of November

This adaptive system performs much better than a static system (i.e. based on a sample of data from some time ago), as seen from testing on the only available gold set in English (same one as in Magdy et al. 2015) which was collected 3 years ago. Proof to this conceptual drift in social media, as detailed in Magdy & Elsayed 2014 stands the fact that classifiers from period A perform worse on test sets from period B, the larger the gap between A and B is. Our current system achieves a low score on the gold set (0.159 f1 score), even though it performs well on data from early 2018.

10.1 Sources of error

We would like to discuss some of the sources of error that hinder the performance of this classifier. While some of them are beyond our control, some of the others may be amended in a future version of the classifier.

- Wrongly detected language by Twitter: we currently use the Twitter language detector (for the languages in which we collect using Search API) which provides some wrong results, especially for latin languages, which are very similar. One should use a more powerful language detector (currently used only by the Streaming API data collection block)
- Video title in another language: we currently append all the video titles in their initial form to the text of the tweet. A significant amount of times, this title is not in the same language as the tweet and this hinders the performance of the classifier. Possible solutions could include translating the video title or discarding the tweet entirely if this is the case.
- Too noisy data is the main issue for datasets obtained through distant supervision. It is not only the language that we are referring to (i.e. slang or words written

differently such as 'i l o v e y o u'), but also unmeaningful or wrong pairings. Manual work even with the cleaned sets reveal tweets such as: 'u s' paired to News&Politics or 'check out my portofolio' to Entertainment. Such pairings hinder the performance of the classifier the most.

- Too many automatically generated tweets: Most of the tweets that come with a YouTube video associated also come with some form of automated text associated. Attempts were made to discard it, but there are many forms of this type of text and in many languages.
- Lack of a gold set: Testing on data obtained in the same way as the training data can be misleading due to its noise. The validation set also has a lot of wrong pairings or impossible to categorise tweets even by humans. The performance of the classifier on such tweets is not meaningful and can even be misleading (classifier could be actually better or worse than expected)
- General categories: categories such as People&Blogs (default category of YouTube), Entertainment or News&Politics do not have strict boundaries. It depends on individual Twitter users/YouTube video authors what consider to be each of them and such differences impact the performance of our classifier massively. One potential solution is to build a classifier taking into consideration only categories with harder boundaries such as Sports or Pets&Animals.

10.2 Future work

We finish this project with an overview of potential future work in order to improve this classification system. There are two main directions in which we could do experimentation to improve the quality of the system:

- **Work on the dataset**: The field of distant supervision is still relatively new so there are bound to exist better ways to collect data that is less noisy and closer to what a human annotator would collect. Papers such as Mohammed et al. 2017 (curated streams), Magdy et al. 2015(YouTube labels), Go et al. 2009(emoticons) explore different ways of collecting Tweets through distant supervision, but since the importance of Twitter topic classification is growing, there is bound to more research effort allocated to finding such methods. One could also focus on better preprocessing methods, perhaps to develop a method detecting the usefulness of a tweet for our classifier or better filtering methods for automatically generated ones or those too short or too long. An important part of future work that is essential is to develop a reliable gold set, such that any improvements could be measured against performance on a correct test set.
- **Work on the classifier**: While I personally believe that work on the dataset has a higher chance of making a significant difference for this type of classifier, one could also focus further on finding a better classification method or feature selection methods. There are many techniques which were not attempted in the experiments, most important being, of course, using neural networks for

classification. Using deep learning for text classification is an area of high research interest lately, with many papers such as Mikolov et al. 2013, Kim 2014 or Joulin et al. 2016 that explore the effectiveness of word embeddings layers and convolutional neural networks in text classification. Experimenting with this is likely to take a significant amount of time, not only due to the size of training data that we have, but also due to a very high number of potential architectures and hyperparameters to vary.

Bibliography

Barbosa, Luciano and Feng, Junlan. Robust sentiment detection on twitter from biased and noisy data. In *Proceedings of the 23rd International Conference on Computational Linguistics: Posters*, pp. 36–44. Association for Computational Linguistics, 2010.

Becker, Hila, Naaman, Mor, and Gravano, Luis. Beyond trending topics: Real-world event identification on twitter. *ICWSM*, 11(2011):438–441, 2011.

Chen, Lu, Wang, Wenbo, Nagarajan, Meenakshi, Wang, Shaojun, and Sheth, Amit P. Extracting diverse sentiment expressions with target-dependent polarity from twitter. *ICWSM*, 2(3):50–57, 2012.

Davidov, Dmitry, Tsur, Oren, and Rappoport, Ari. Enhanced sentiment learning using twitter hashtags and smileys. In *Proceedings of the 23rd international conference on computational linguistics: posters*, pp. 241–249. Association for Computational Linguistics, 2010.

Eliot, Dr. Lance. Support vector machines (svm) for ai self-driving cars, 2018. URL https://aitrends.com/wp-content/uploads/2018/01/1-19SVM-2.jpg. [Online; accessed April 05, 2018].

Estabrooks, Andrew, Jo, Taeho, and Japkowicz, Nathalie. A multiple resampling method for learning from imbalanced data sets. *Computational intelligence*, 20(1):18–36, 2004.

Go, Alec, Bhayani, Richa, and Huang, Lei. Twitter sentiment classification using distant supervision. *CS224N Project Report, Stanford*, 1(12), 2009.

Husby, Stephanie D and Barbosa, Denilson. Topic classification of blog posts using distant supervision. In *Proceedings of the Workshop on Semantic Analysis in Social Media*, pp. 28–36. Association for Computational Linguistics, 2012.

Japkowicz, Nathalie and Stephen, Shaju. The class imbalance problem: A systematic study. *Intelligent data analysis*, 6(5):429–449, 2002.

Jiang, Long, Yu, Mo, Zhou, Ming, Liu, Xiaohua, and Zhao, Tiejun. Target-dependent twitter sentiment classification. In *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies-Volume 1*, pp. 151–160. Association for Computational Linguistics, 2011.

Joachims, Thorsten. Text categorization with support vector machines: Learning with many relevant features. In *European conference on machine learning*, pp. 137–142. Springer, 1998.

Joachims, Thorsten, Finley, Thomas, and Yu, Chun-Nam John. Cutting-plane training of structural svms. *Machine Learning*, 77(1):27–59, 2009.

Joulin, Armand, Grave, Edouard, Bojanowski, Piotr, and Mikolov, Tomas. Bag of tricks for efficient text classification. *arXiv preprint arXiv:1607.01759*, 2016.

Kim, Yoon. Convolutional neural networks for sentence classification. *CoRR*, abs/1408.5882, 2014. URL http://arxiv.org/abs/1408.5882.

Kotsiantis, Sotiris, Kanellopoulos, Dimitris, Pintelas, Panayiotis, et al. Handling imbalanced datasets: A review. *GESTS International Transactions on Computer Science and Engineering*, 30(1):25–36, 2006.

Lee, Kathy, Palsetia, Diana, Narayanan, Ramanathan, Patwary, Md Mostofa Ali, Agrawal, Ankit, and Choudhary, Alok. Twitter trending topic classification. In *Data Mining Workshops (ICDMW), 2011 IEEE 11th International Conference on*, pp. 251–258. IEEE, 2011.

Lewis, David D. Naive (bayes) at forty: The independence assumption in information retrieval. In *European conference on machine learning*, pp. 4–15. Springer, 1998.

Magdy, Walid and Elsayed, Tamer. Adaptive method for following dynamic topics on twitter. In *ICWSM*, 2014.

Magdy, Walid, Sajjad, Hassan, El-Ganainy, Tarek, and Sebastiani, Fabrizio. Distant supervision for tweet classification using youtube labels. In *ICWSM*, pp. 638–641, 2015.

McCallum, Andrew, Nigam, Kamal, et al. A comparison of event models for naive bayes text classification. In *AAAI-98 workshop on learning for text categorization*, volume 752, pp. 41–48. Citeseer, 1998.

Mikolov, Tomas, Chen, Kai, Corrado, Greg, and Dean, Jeffrey. Efficient estimation of word representations in vector space. *arXiv preprint arXiv:1301.3781*, 2013.

Mohammed, Salman, Ghelani, Nimesh, and Lin, Jimmy. Distant supervision for topic classification of tweets in curated streams. *arXiv preprint arXiv:1704.06726*, 2017.

Nigam, Kamal, Lafferty, John, and McCallum, Andrew. Using maximum entropy for text classification. In *IJCAI-99 workshop on machine learning for information filtering*, volume 1, pp. 61–67, 1999.

Pak, Alexander and Paroubek, Patrick. Twitter as a corpus for sentiment analysis and opinion mining. In *LREc*, volume 10, 2010.

Pang, Bo, Lee, Lillian, and Vaithyanathan, Shivakumar. Thumbs up?: sentiment classification using machine learning techniques. In *Proceedings of the ACL-02 conference on Empirical methods in natural language processing-Volume 10*, pp. 79–86. Association for Computational Linguistics, 2002.

Rosen, Aliza. Tweeting made easier. https://blog.twitter.com/official/en_us/topics/product/2017/tweetingmadeeasier.html, November 2017. Accessed on 2018-01-12.

Singh, Tajinder and Kumari, Madhu. Role of text pre-processing in twitter sentiment analysis. *Procedia Computer Science*, 89:549 – 554, 2016. ISSN 1877-0509. doi: https://doi.org/10.1016/j.procs.2016.06.095. URL http://www.sciencedirect.com/science/article/pii/S1877050916311607. Twelfth International Conference on Communication Networks, ICCN 2016, August 19âĂŞ 21, 2016, Bangalore, India Twelfth International Conference on Data Mining and Warehousing, ICDMW 2016, August 19-21, 2016, Bangalore, India Twelfth International Conference on Image and Signal Processing, ICISP 2016, August 19-21, 2016, Bangalore, India.

Thelwall, Mike, Buckley, Kevan, and Paltoglou, Georgios. Sentiment in twitter events. *Journal of the Association for Information Science and Technology*, 62(2):406–418, 2011.

Ting, SL, Ip, WH, and Tsang, Albert HC. Is naive bayes a good classifier for document classification. *International Journal of Software Engineering and Its Applications*, 5 (3):37–46, 2011.

Tsochantaridis, Ioannis, Hofmann, Thomas, Joachims, Thorsten, and Altun, Yasemin. Support vector machine learning for interdependent and structured output spaces. In *Proceedings of the twenty-first international conference on Machine learning*, pp. 104. ACM, 2004.

Zubiaga, Arkaitz and Ji, Heng. Harnessing web page directories for large-scale classification of tweets. In *Proceedings of the 22nd International Conference on World Wide Web*, pp. 225–226. ACM, 2013.