Kaunas University of Technology
Department of Applied Mathematics

Customer review classification in Lithuanian language for e-commerce business

Master's Final Degree Project

Kęstutis Daugėla
Project author

Assoc. prof. Vytautas Janilionis
Supervisor
Assoc. prof. Aušra Rūtelionė
Supervisor

Kaunas, 2018

Kaunas University of Technology
Department of Applied Mathematics

Customer review classification in Lithuanian language for e-commerce business

Master's Final Degree Project
Business Big Data Analytics (621G12002)

Kęstutis Daugėla
Project author

Assoc. prof. Vytautas Janilionis
Supervisor
Assoc. prof. Aušra Rūtelionė
Supervisor
Assoc. prof. Evaldas Vaičiukynas
Reviewer
Assoc. prof. Egidijus Rybakovas (signature)
Reviewer (date)

Kaunas, 2018

Kaunas University of Technology
Department of Applied Mathematics
Kęstutis Daugėla

Customer review classification in Lithuanian language for e-commerce business

Declaration of Academic Integrity

I confirm that the final project of mine, Kęstutis Daugėla, on the topic "Customer review classification in Lithuanian language for e-commerce business" is written completely by myself; all the provided data and research results are correct and have been obtained honestly. None of the parts of this thesis have been plagiarised from any printed, Internet-based or otherwise recorded sources. All direct and indirect quotations from external resources are indicated in the list of references. No monetary funds (unless required by law) have been paid to anyone for any contribution to this project.

I fully and completely understand that any discovery of any manifestations/case/facts of dishonesty inevitably results in me incurring a penalty according to the procedure(s) effective at Kaunas University of Technology.

(name and surname filled in by hand) (signature)

Table of contents

List of abbreviations

API – Application Programming Interface

CPU – Central Processing Unit

DT – Decision Tree

GPPL – General Purpose Programming Language

GPU – Graphics Processing Unit

JSON – JavaScript Object Notation

KPI – Key Performance Indicator

NN – Neural Network

NLP – Natural Language Processing

REST – Representational State Transfer

RF – Random Forests

SVM – Support Vector Machine

TDM – Term Document Matrix

TPU – Tensor Processing Unit

VM – Virtual Machine

List of figures

List of tables

Kęstutis Daugėla. Customer review classification in Lithuanian language for e-commerce business
Study field and area (study field group): Applied Mathematics, Informatics, Economics
Keywords: text mining, sentiment analysis, customer review classification, microservices, machine learning, deep learning
Kaunas, 2018.

Summary

The aim of the project is to create an application for customer review classification in Lithuanian language in the specific E-commerce business domain. Various Natural Language Processing techniques and machine learning algorithms were used for customer reviews evaluation and polarity categorization. Analyzing customer reviews in Lithuanian language makes the task more challenging due to the language specifics. Therefore, additional pre-processing steps were utilized in this work. Integration with external and internal Representational State Transfer Application Programming Interfaces were made in order to create upgradeable classification model and application for data transformation, both integral into an ecosystem based on microservices within any modern organization. From an analytical perspective, classic models such as Support Vector Machine and Random Forests were built and tested with differently pre-processed datasets. In addition to these methods, deep learning models were added for comparison. Best results were achieved using Support Vector Machine algorithm and Deep Neural Network with embedding and flatten layers. Using TensorFlow framework and additional R programming language libraries, the best model was deployed as a microservice, which classifies customer reviews written in Lithuanian in real-time. The final product could be integrated into other applications and services for a variety of text classification use cases.

Kęstutis Daugėla. E. verslo paslaugų vartotojų lietuviškų atsiliepimų klasifikavimas
Studijų kryptis ir sritis (studijų krypčių grupė): Taikomoji matematika, Informatika, Ekonomika
Reikšminiai žodžiai: teksto tyryba, sentimentų analizė, klientų atsiliepimų klasifikavimas, mikroservisai, mašininis mokymasis, gilus mokymasis
Kaunas, 2018.

Santrauka

Baigiamojo magistro projekto tikslas – sukurti mikroservisą klientų atsiliepimų apie lietuviškas elektronines parduotuves klasifikavimui. Darbe panaudoti įvairūs natūralios kalbos apdorojimo ir mašininio mokymosi metodai vartotojų atsiliepimų poliškumui nustatyti. Įvertinant lietuvių kalbos kompleksiškumą, šis klasifikavimo uždavinys tampa kiek sudėtingesnis, lyginant su anglų kalba. Dėl to tekstų apdorojimui buvo taikomos papildomos procedūros. Taip pat panaudoti išoriniai ir vidiniai "REST" aplikacijų programavimo sąsajų resursai duomenų paruošimo ir klasifikavimo uždavinių sprendimui bei sukurta analitinė aplikacija, skirta integracijai į mikroservisais paremtą architektūrą bet kurioje modernioje organizacijoje. Duomenų klasifikavimui buvo parinkti įvairūs metodai – pradedant klasikiniais (atraminių vektorių, atsitiktinių miškų metodai) ir baigiant gilaus mašininio mokymosi metodais. Geriausi klasifikavimo rezultatai buvo pasiekti taikant atraminių vektorių ir gilaus mašininio mokymosi metodus. Naudojant TensforFlow sistemą ir R programavimo kalbą, sukurta saityno tarnyba, galinti klasifikuoti lietuviškus klientų atsiliepimus realiuoju laiku. Galutinis produktas gali būti integruotas į kitas aplikacijas ar saityno tarnybas sprendžiant įvairias klientų atsiliepimų klasifikavimo problemas.

Introduction

Service quality is crucial part not only in terms of business performance. Higher service quality can help to decrease operational costs, boost customer satisfaction and loyalty. However, most of the businesses are struggling with service quality issues. In e-commerce sector this matter is even more critical, because it is harder to identify problematic products, service gaps or process issues in comparison with ordinary businesses. Customer reviews can be extremely useful and one of the cheapest resources in order to feel the pulse of the customer, analyze marketing decision or even get positive publicity. Therefore, negative customer sentiment identification and rapid reaction would present a competitive edge for every business, especially in e-commerce sector.

For that purpose, a microservice will be created. It will be able to cope with customer review classification task for any type of source – from an internal customer relationship management system to social media resources, such as Facebook or Twitter. Models for analytical application will be chosen according to the recommendations obtained by literature research and trained on real production data, which contains customer review texts. All the methodology which will be obtained during the research will be documented and implemented in a reproducible way, in order to provide universal text classification solution for e-commerce companies in Lithuania.

1. Literature review

1.1. Linking Online service quality and customer satisfaction

In currently changing business environment organizations face new challenges such as dealing with an exponentially increasing amount of data and rapidly spreading information via a number of various channels. Besides inventory management (Patil & Divekar, 2014), data and money transfer security (Reddy & Divekar, 2014), customer satisfaction plays a significant role in the success of a company. It is an almost intuitive thing that customer satisfaction has a huge impact on profits and there is big amount of research done to substantiate it (Zeithaml, 2000). E-commerce sector is an excellent example of a business field where customer satisfaction is crucial for maintaining successful business relationships. Comparing with traditional businesses, e-commerce companies face an additional problem when managing customer expectations, because usually customers can't try out the products before ordering them (Nisar & Prabhakar, 2017). It is also essential to know the customer and be able to fulfill changing needs of a client, have flexible and efficient service customization, introduce new products and make information easily accessible via Internet (Kumar, 2016).

In a survey where 194 senior marketing professionals where examined, 71 percent responded that they place customer satisfaction measure amongst one of the most useful metrics. It also got the third rank for businesses as a service (BAAS) and fourth rank for mixed type customers (both businesses and consumers) (Farris, Bendle, Pfeifer, & Reibstein, 2010).

Customer reviews can benefit significantly from this type of analysis, because they contain valuable insights for both positive and negative business activity and might bring additional metadata about the customer on the table. According to the best practice, most commonly used key performance indicator for measuring this type of activity are listed below:

- Response time on an issue;
- number of negative ratings received;
- percentage of resolved issues.

The survey made by Dimensional Research determined that customer service has a huge effect on the customer behavior, especially when making a decision to buy (Dimensional Research, 2013).

Another survey done by Podium suggests that 3.3 stars out of 5 are the minimum threshold of rating when a customer considers doing business with an approached company. 93% of the customers responded that online reviews written by others impact they decision (Podium, 2017).

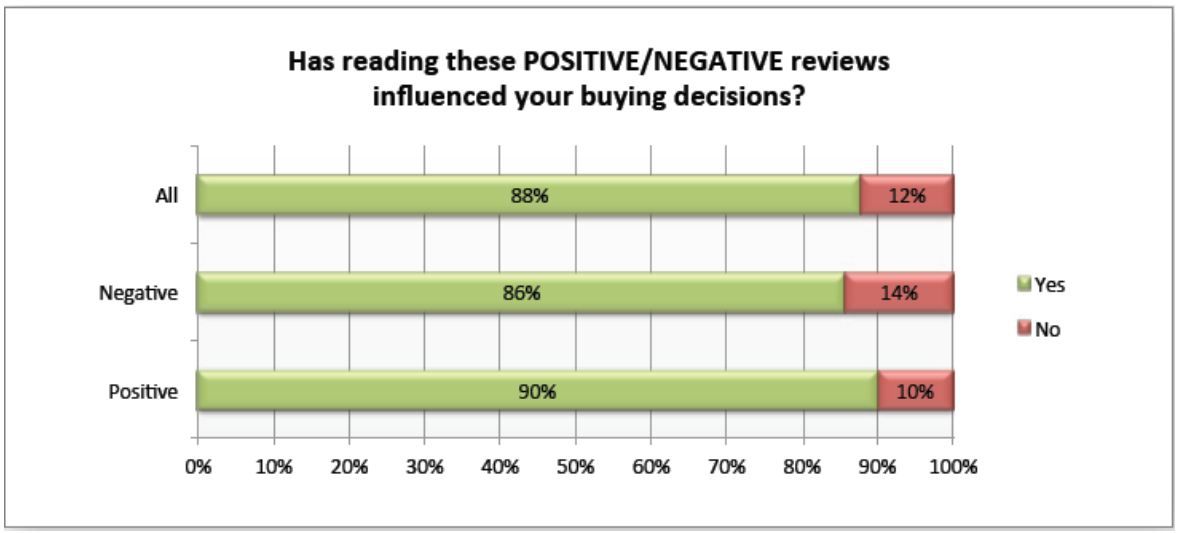

Figure 1: A Survey Of Customer Service From Mid-Size Companies. Dimensional Research, April 2013

While some studies suggest that negative reviews have a higher impact on customer's decision (Lee, 2016), it is clearly beneficial to track both types of feedback in order to get the pulse of the business.

The urgency is one of the key aspects for higher customer satisfaction, because these days there are lots of information sources (especially on the social media) where customers can leave their reviews. Therefore, a specific application which helps to deal with this kind of information could be beneficial, especially taking into the consideration low cost of this customer feedback type. Having such system implemented would help to take a proactive approach to whole customer relationship and reputation management.

1.2. Text analytics for customer review data

Online customer reviews bring a lot of useful information on the table concerning customer experience and online service quality in general. The drawback of analyzing such matter is that data is unstructured and requires a lot of pre-processing work. A total amount of reviews might differ depending on the size of business, but usually there are thousands of entries across not only on the

company's website, but specialized review websites and social media as well. Moreover, the text is usually written using informal manner and often contains some words from another language. Words also tend to be misspelled and written without the usage of native Lithuanian characters.

Text analytics comes into play when dealing with this kind of unstructured review data. With provided tools it is possible to do various kinds of analysis, such as clustering, classification and sentiment analysis. However, it is vital to have domain specific knowledge in order to be able to understand the clusters, categorize items or track trends. Sentiment analysis, on the other hand, provides a universal approach for extracting the most valuable attribute in the customer reviews – polarity level.

There are a variety of pre-built tools and libraries for sentiment analysis for English language or other popular tongues as French, Spanish or German. However, in Lithuanian language the support is almost nonexisting. This actually makes sense, taking into consideration complexity of Lithuanian language. According to professor Olegas Poliakovas, the difficult nature of Lithuanian language is mainly due to more advanced morphological structure comparing to other related languages. (Poliakovas, 2015). Even such service provider as Google does not support NLP applications for Lithuanian language (Google Cloud Natural Language API Documentation). Same goes for Lithuanian text stemming – officially there is no open source production-ready stemming library. On the other hand, there are several companies as Token Mill and other small Lithuanian start-ups working with Lithuanian language processing, but their work is not accessible for general public. However, there is NLP infrastructure yet created. "The first infrastructure that provides online access to open-source tools for managing, processing, annotating, and analyzing Lithuanian language texts" says one research team on the (Vitkutė-Adžgauskienė, Utka, Amilevičius, & Krilavičius, 2016). Performance of this framework is quite promising – precision for tasks as language filtering, spelling checker, lexical processing and morphosyntactic processing is way over 0.9, but scalability is questionable, as authors conclude. Since the solution is publicly available and working, it is a viable and robust alternative for processing Lithuanian words morphologically.

When it comes to modeling, deep learning frameworks are a clear winner in this area and became a trending tool for semantic data classification lately. In one particular use case, sentiment analysis was done in English language using the movies review data, which consisted of 11,855 single sentences. Recursive Neural Tensor Network gave tremendous results with this publicly available and labeled data (Socher et al., 2013).

Model	Fine-grained		Positive/Negative	
	All	Root	All	Root
NB	67.2	41.0	82.6	81.8
SVM	64.3	40.7	84.6	79.4
BiNB	71.0	41.9	82.7	83.1
VecAvg	73.3	32.7	85.1	80.1
RNN	79.0	43.2	86.1	82.4
MV-RNN	78.7	44.4	86.8	82.9
RNTN	**80.7**	**45.7**	**87.6**	**85.4**

Figure 2: Text classification accuracy at the sentence level. Socher, 2013

Another research indicated, that despite a good performance of various neural network models, Support Vector Machine classifier also performed remarkably strong (Tang, Qin, & Liu, 2015). On the contrary, the similar study points out Random Forests model superiority against Naive Bayes and SVM classifier (Fang & Zhan, 2015).

	Yelp 2013		Yelp 2014		Yelp 2015		IMDB	
	Accuracy	MSE	Accuracy	MSE	Accuracy	MSE	Accuracy	MSE
Majority	0.356	3.06	0.361	3.28	0.369	3.30	0.179	17.46
SVM + Unigrams	0.589	0.79	0.600	0.78	0.611	0.75	0.399	4.23
SVM + Bigrams	0.576	0.75	0.616	0.65	0.624	0.63	0.409	3.74
SVM + TextFeatures	0.598	0.68	0.618	0.63	0.624	0.60	0.405	3.56
SVM + AverageSG	0.543	1.11	0.557	1.08	0.568	1.04	0.319	5.57
SVM + SSWE	0.535	1.12	0.543	1.13	0.554	1.11	0.262	9.16
JMARS	N/A	–	N/A	–	N/A	–	N/A	4.97
Paragraph Vector	0.577	0.86	0.592	0.70	0.605	0.61	0.341	4.69
Convolutional NN	0.597	0.76	0.610	0.68	0.615	0.68	0.376	3.30
Conv-GRNN	0.637	0.56	0.655	0.51	0.660	0.50	0.425	**2.71**
LSTM-GRNN	**0.651**	**0.50**	**0.671**	**0.48**	**0.676**	**0.49**	**0.453**	3.00

Figure 3: Text classification results of YELP and IMDB datasets, Tang, Qin, & Liu, 2015

Despite that, there are not many pieces of research done for Lithuanian language. The accuracy of the best model for comments classifications was reached with Naive Bayes classifier and was around 0.68. SVM performed well as might be expected even without pre-processing phase and using only unigrams (accuracy – 0.611) (Kapočiūtė-Dzikienė, Krupavičius, & Krilavičius, 2013). SVM was also selected as a model of choice for legal text classification problem (Mickevičius, Krilavičius, & Morkevičius, 2015). Some results from a different type of text classification have provided below as an example.

Another interesting finding regarding Lithuanian texts preprocessing is: “English barely benefits from stemming and lemmatization, Lithuanian <...> benefit significantly from these pre-processing steps“ (Kapočiūtė-Dzikienė, Vaassen, Daelemans, & Krupavičius, 2012). Translation, on the other hand, does not provide cutting edge – research done by Vilnius University indicates that after translation procedure only half of the original lexicon was left (Okockis, 2016).

To sum up, one thing is common from all of these researches – Lithuanian language requires additional data wrangling steps, which are the cornerstone for successful sentiment analysis. Some of these preparation steps will be included in this project in order to utilize and measure pure machine learning based text classification approach using most popular techniques: SVM, Random Forests and deep learning.

Table 1: Comparison of different models for text classification problems

Title	Author, year	Language	Classes	Best model accuracy	Comments
Recursive Deep Models for Semantic Compositionality Over a Sentiment Treebank	(Socher et al., 2013)	EN	5	RNTN, 0.807	Trained using sentiment treebank
Document Modeling with Gated Recurrent Neural Network for Sentiment Classification	(Tang et al., 2015)	EN	5	LSTM-GRNN, 0.676	Traditional RNN performance was weak
Sentiment analysis: Measuring opinions	(Bhadane, Dalal, & Doshi, 2015)	EN	2	SVM, 0.78	Additional preprocessing was done
An Extensive study of Sentiment Analysis tools and Binary Classification of tweets using Rapid Miner	(Vyas & Uma, 2018)	EN	2	SVM, 0.791	Rapid Miner software was used
Text Classification Improved by Integrating Bidirectional LSTM with Two-dimensional Max Pooling	(Zhou et al., 2016)	EN	2	BLSTM-2DCNN, 0.895	Neutral reviews removed
A Comparison of Approaches for Sentiment Classification on Lithuanian Internet Comments	(Kapočiūtė-Dzikienė et al., 2013)	LT	3	NB, 0.679	Unigrams and bigrams were used together
Opinion Mining in Latvian Text Using Semantic Polarity Analysis and Machine Learning Approach	(Špats & Birzniece, 2016)	LV	3	NB, 0.62	Lexicon based approach performed better (accuracy - 0.73)

1.3. Software architecture

One of the biggest most prominent, distinguishing the applications into two categories, is whether the application is build using monolithic or microservices architecture style. While applications designed in a monolithic manner clearly have their place, they lack agility and reliability. Implementation of change can consume a lot of time, because a different functionality of an application is firmly coupled together. Once developed, those applications have a barrier to newer technologies, since changing one thing will affect the whole application. When application adopts many new functionalities, the whole structure tends to be difficult to understand and change.

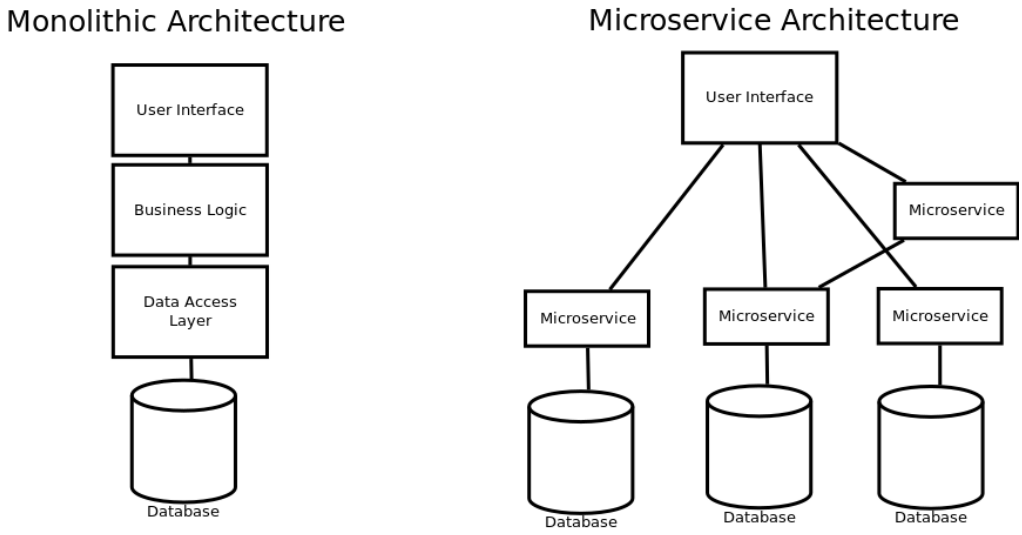

Figure 4: Monolithic vs microservice architecture overview

Most of the organization moved from monolithic application architecture to microservices. “There has been a gradual decrease in the size of monolithic applications by isolating these functional components into different deployment machines and allowing them to communicate to the main application and with each other using loose coupling interfaces“ according to (Katuwal, 2016) . These applications communicate using Application Programming Interfaces (API) and it became the standard approach for data integration and spreading functionality across the rich ecosystem of enterprise applications (Sturm et al., 2017). Most important features of microservices are independent deployment, scalability and fault isolation. In addition to that, one microservice is a relatively small application, easily maintainable by autonomous teams (Richardson, 2018)

Since the solution of the above mentioned classification problem perfectly fits under microservice definition and may require fast deployment, scalability and constant change, this data science application will be crafted for integration into the microservices ecosystem.

1.4. Choice of the programming language

The choice of programming language is also significant for the success of the project. General purpose programming languages dominate the market and are primary choices for ordinary applications, while building data science model is quite different. Most popular languages in this field are R and Python. The growth of popularity for both of these languages was remarkable during the last several years, but the growth of R community is even more impressing, keeping in mind, that the usage of Python is broader comparing to R. In TIOBE Programming Community index R reached 8th position in January, 2018 and showed consistent growth during the years.

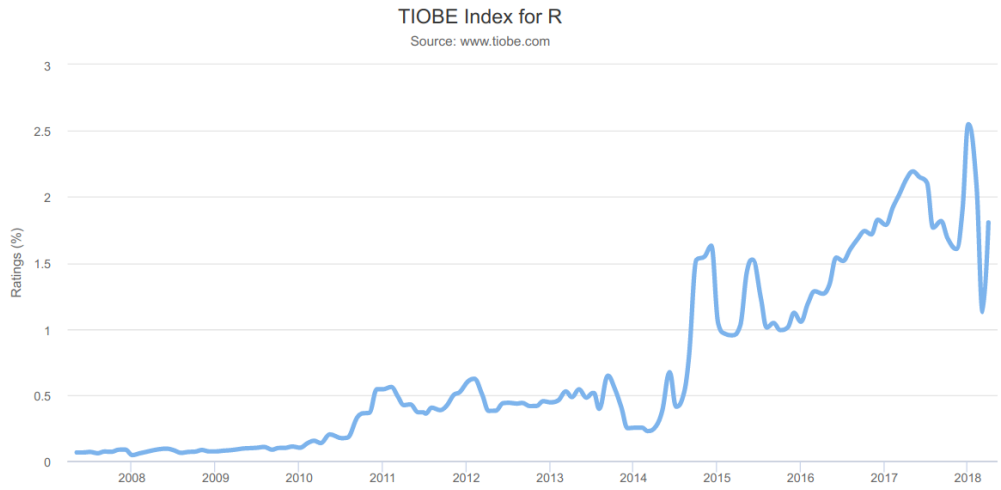

Figure 5: R programming language index ("TIOBE Index for R")

In addition to this, R community is growing exponentially and reached the level of C, in terms of Stack Overflow website traffic. Python, Java, C# and PHP are the only languages, which has more traffic at the moment.

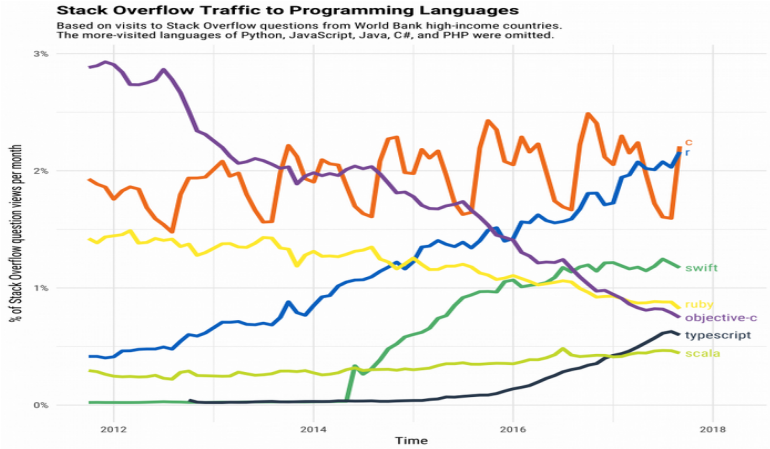

Figure 6: Stack Overflow traffic depending on the programming language

Another area, where R programming language can barely find an equal competition is convenient data preparation and transformation even for nondevelopers. It is arguably the most crucial part in any data related task. Various articles indicate that data preparation part consumes the most of any data analysis task (Endel & Piringer, 2015; Pérez et al., 2015). It might take up to 80% of total time spent on a current project, according to an even newer survey (CrowdFlower, 2016). Therefore, running R code in production will have two major advantages:

- Shorter development cycle;
- reduces the time for ad-hoc data preparation.

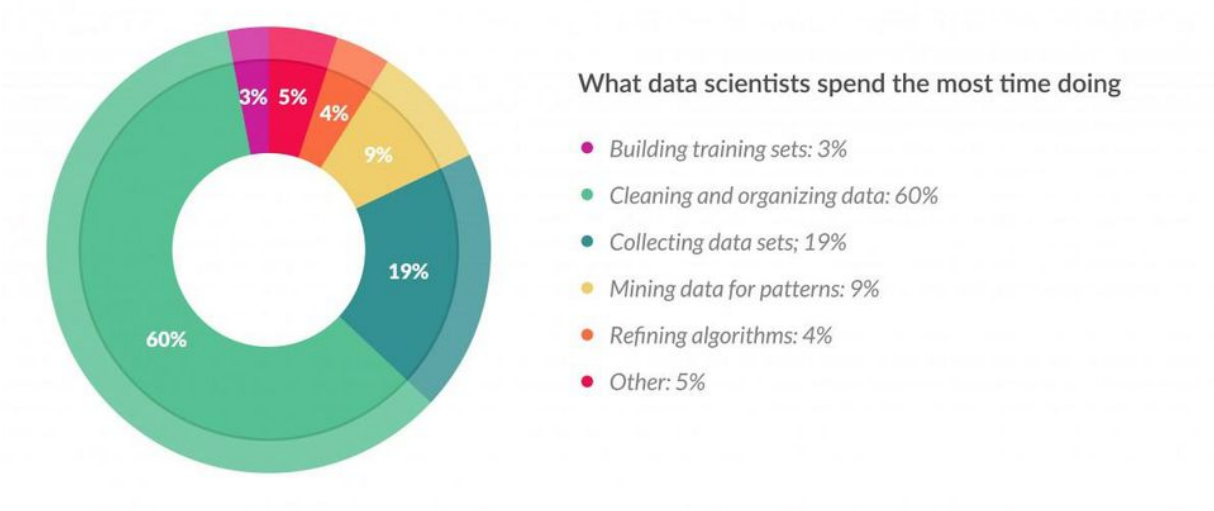

Figure 7: CorwdFlower Survey regarding data scientists time allocation

Another step is model implementation in the production environment. It is a common view that production applications should be written in general purpose programming languages (GPPL) such as C, Java or Scala, supposing that modeling with R is just for prototyping. According to professor Roger D. Peng, R programming language has some flaws in memory management, security and web services development. This programming language is also lacking performance when dealing with specific tasks. When doing the computation for economic related problem, R performance is far behind C++ and Java, but surprisingly better than Python (Aruoba & Fernández-Villaverde, 2015; BoraGan Aruoba & Fernández-Villaverde, 2018).

Table 2: R performance comparison with C++, Java and Python

	C++	Java	Python	R
Version/Compiler	GCC-7.3.0	9.04	CPython 3.6.4	3.4.3
Time	**1.60**	**3.20**	**145.27**	**57.06**

Different performance test done by NASA scientists showed that R performance is the slowest compared to Python, Scala and C. (Jules Kouatchou, 2016). However, this is hardly the case when R is used just as an interface for accessing distributed frameworks (Uskenbayeva et al., 2015).

On the other hand, imagine a completely different scenario of application development – let's say, that all models can be deployed to the production environment directly. This would dramatically reduce implementation cost and speed. Since R is capable deploying predictive models as a web service, gap between data analysis and deployment can be closed. This approach will eventually enable data scientist for managing projects, do experiments more efficiently and save human resources for application development. This innovative idea comes hand in hand with AGILE methodology and reduces the cost of change, which grows exponentially for monolithic applications, normally developed using waterfall method.

In combination with powerful machine learning algorithms, distributed frameworks ("SparkR (R on Spark)," 2018) and libraries for WEB service development, R puts high pressure on conventional approach of data science application deployment. Moreover, the most popular deep learning framework called "TensorFlow" recently became compatible with R by accessing Python frond-end API (Chollet & Allaire, 2018). The mentioned framework was created by "Google Brain" team and released at the end of 2015. There is also a proven record of applying "TensorFlow" for text classification, natural language processing, speech recognition and even Artificial Intelligence applications.

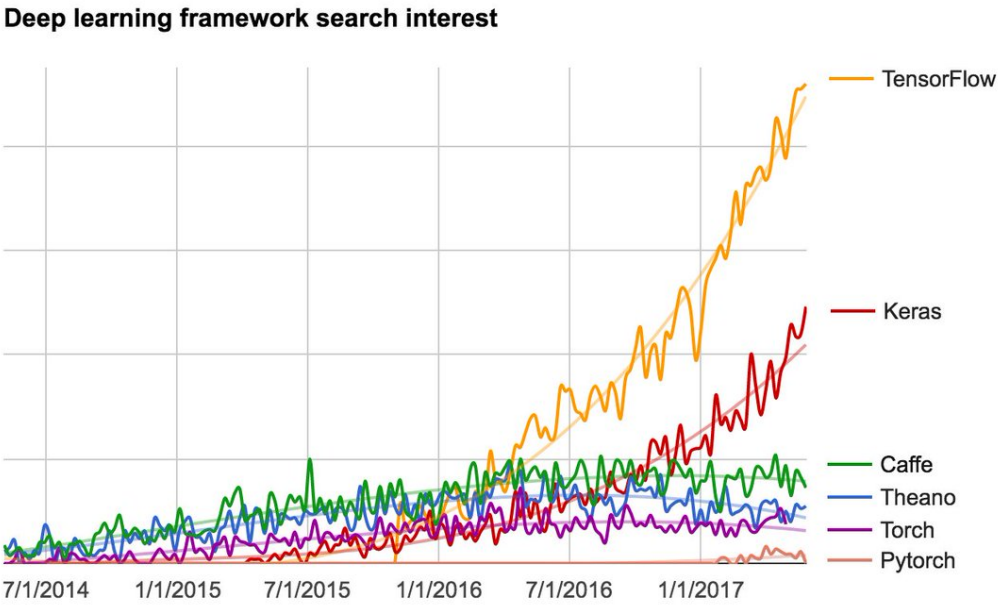

Figure 8: Deep learning framework search interest in Google

There are several ways of accessing TensorFlow deep learning framework. With R it is possible to use TensorFlow front-end directly or access Keras API, which is higher level implementation and have most of the parameters pre-configured according to the best data science practices. It is also a recommended way for Tensorflow usage in R environment.

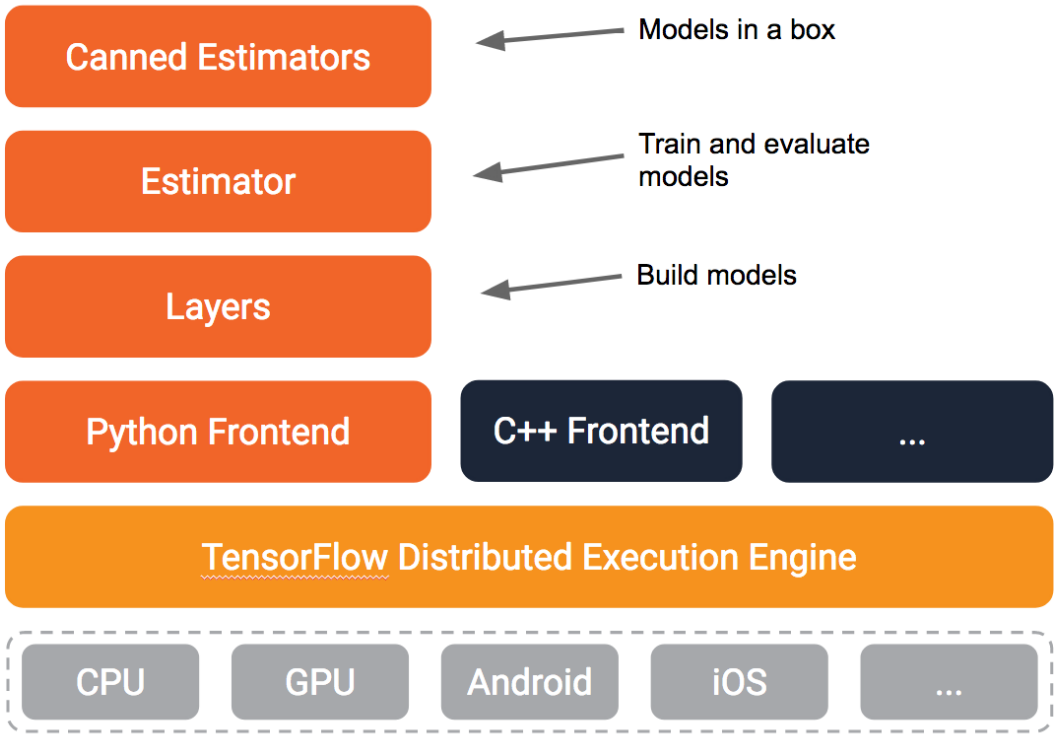

Figure 9: R interface to Tensorflow estimators (RStudio)

To sum up, both R and Python are viable options for this project, especially for the modeling part. Therefore, choice of a programming language comes to a personal preference. Due to the language expressiveness and versatility, R is chosen as the main programming language for this project.

1.5. Aim and objectives of the project

The purpose of the thesis is to build a microservice for customer review classification. In addition to this, implementation patterns will be created for successful integration into modern IT infrastructure as well as scalability for Big Data use cases. In order to reach these goals, following objectives need to be done:

1. Choose methods and programming implementation according to the provided literature review in order to solve text classification problem in Lithuanian language
2. Create a methodology for sentiment analysis in Lithuanian language
3. Implement best suited machine learning approaches for text classification task
4. Label review texts manually for training and testing samples
5. Create microservices for data related tasks:
    - Data flow process;
    - customer review classification.

2. Methodology

This section of the project will cover:

- High level application design;
- data preparation task;
- classification models;
- model implementation guidelines.

From a business perspective, requirements for data transformation and storage will be created and summarized for only one use case – customer reviews from one customer review website. However, it can be adjusted to store instances from multiple resources, such as streaming data, customer relationship management system, social networks and other, because input file format and patterns will remain the same.

Computational tasks will be done on a machine with Intel i7 CPU, 16 GB of RAM and SSD hard drive with Ubuntu 16.04 operating system running. The performance will be measured accordingly for all machine learning task in order to find the most optimal model to meet presumable business requirements in the production environment.

2.1. High level design of the application

The application will consist of two parts – data transformation application and analytical model application. Proof of concept phase for data transformation will be done in R environment. For data transformation application to be scalable, Spark framework can be used and accessed by using SparkR or sparklyr API later on. However, the amount of data is tiny, therefore it is not that efficient using distributed frameworks for processing only several megabytes of data.

Once data analysis is done, best classification model will be selected for the analytical model application. Most important factors for selecting the model are classification performance, scalability and maintenance related. “TensorFlow” comes together with serving functionality and also could be deployed on such services as CloudML, RStudio Connect or TensorFlow Serving. On the other hand, other classification models, such as Random Forests or Support Vector Machine also could be deployed as a web service using “plumber” R library or other alternatives. The final results

will be selected depending on the performance of each model and long term application maintenance factors.

High level application design is presented below. At this point, REST API resources will be created for transformation processes only inside the local environment. Analytical model REST API will also be created and reachable via HTTP protocol internally. For demonstration purposes, it will be configured to process only one instance of text.

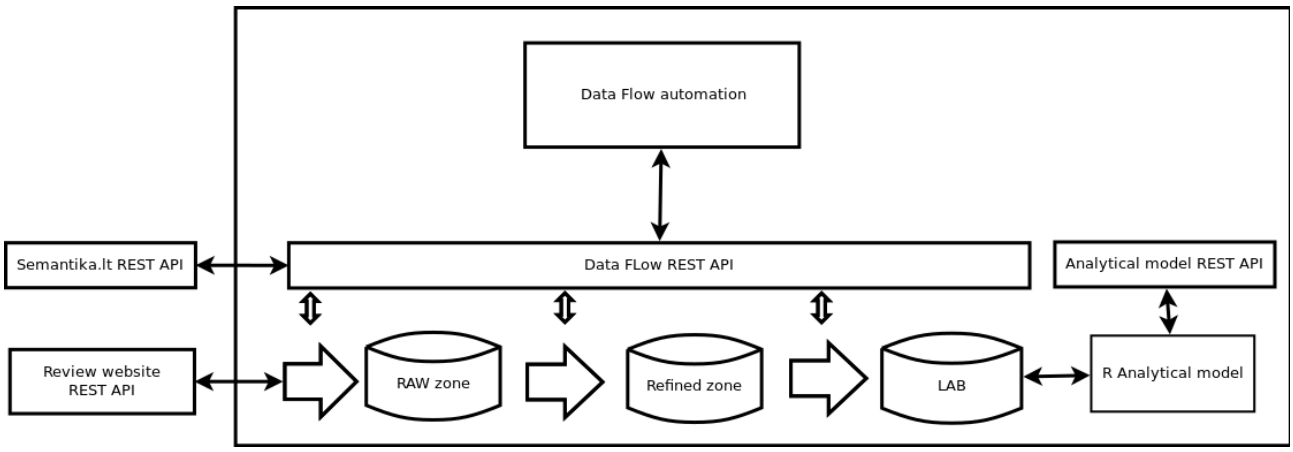

Figure 10: High level application design

2.2. Data extraction, loading and transformation

There is a massive debate between conventional Extract-Transform-Load pipeline versus Extract-Load-Transform, which is a modern alternative for Business Intelligence. The second approach is more suitable for this project, since the data is stored in a complex structure. It also requires additional transformations, which might not be so efficient since data is downloaded via HTTP protocol outside the enterprise network. Moreover, this approach could prevent data transformation mistakes later on, because data is stored in original format once retrieved from the server and could be traced back to it's original formation, in case some critical mistakes were made during the transformation phase. Transparency and data lineage also come as an advantage using current approach.

Raw zone. This is the area where data is stored in the original format. Data comes in JavaScript Object Notation (JSON) file format and the structure of this particular JSON file retrieved from customer review website might be quite complex – the depth is up to 8 levels in the current use case. Each response from the website will be stored as a separate file thus making a Raw zone in

Linux file system. In addition to this, when the data will grow bigger, there will be a viable option to distribute it within Hadoop cluster and process with Spark leaving all the logic and code in place.

Data will be stored in RDS file format, which creates a serialized version of data, but do not change original structure of data. This will save some storage space comparing with original JSON format and will serve as a more efficient way of reading files in R. R packages "httr" and "jsonlite" comes in handy for retrieving and reading the data via HTTP protocol from an external website and will be used for this application.

Refined zone. During this phase, all the files retrieved from the website will be merged together and stored as a table in SQL database. Unique id will be generated for each instance to serve as a primary key in the relational database schema. The refined zone will serve as a staging zone between Raw data and Lab databases. All attributes of the refined table are defined in the picture below.

MySQL server will be selected as storage technology in order to create a relational database, because the application is open source and provides good results regarding performance, scalability and reliability.

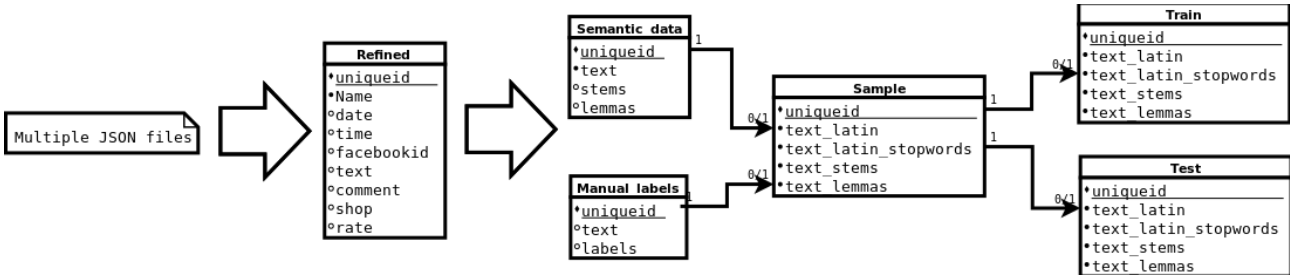

Figure 11: Relationship between Raw, Refined and Lab databases

Lab zone. Data preparation is one of the essential parts of any data science. In this case, research object is Lithuanian texts, which makes text preparation even more complicated. As previous research articles indicated, it is particularly important to do some specific preparation for data at the beginning in order to get better results from classification algorithms. Therefore, the first step will be getting rid of the items, which reviews texts may contain:

- Hypertext Markup Language notation (e.g. "<p>", "</br>");
- special symbols (e.g. "$", "@");
- numbers (e.g. "1", "213");
- stop words (e.g. "kur", "į").

Unique id, generated in staging step, will be assigned for each instance of text in order to create relationships between different tables. The second step includes preparation of table which contains:

- Unique id (primary key);
- original texts in Latin alphabet;
- original texts in Latin alphabet without stop words;
- texts containing word stems;
- texts containing word lemmas.

Data transformations will be done in a reproducible approach, meaning that the process can be initiated automatically for all upcoming datasets in the production environment.

The last step will be the preparation of data sample with correct labels assigned for analytical models. In order to do that, we need to make sure that correct labels are assigned to each text in the first place. The obvious way to identify the polarity of such texts is by setting custom rating intervals for each sentiment label. While this approach might work well for determining two classes, three or more classes requires additional effort. Moreover, people also tend to vote subjectively, have different scales and the rating might not always correspond the real customer's opinion.

For example, if we determine, that the review with the score of 8 is positive, we can face a huge problem in terms of data correctness, because there will definitely be some reviews which have neutral or even negative sentiments. In order to feed correct data to the classification model, manual reviews assessment will be made. Results will be stored in a separate table within the Lab area. These results will be used as final labels in the classification process.

Once the desired number of labels will be reached, sample data set can be created. Training and testing datasets will be the same for all classification algorithms. This will allow proper comparison between classification models, which operates on different objects – term document matrix and tensors, because the dictionary will be equal size.

2.3. Comparison of text classification algorithms

Once the data is prepared, two models from classic machine learning approach will be selected. In general, they provided the best results for majority of the classification problems in both – English

and Lithuanian languages according to the literature review done in the previous part. These algorithms are as following:

- Random Forests – ensemble based classification algorithm;
- Support Vector Machine – kernel based classification algorithm.

Decision tree model will be also added as a baseline model, since the model is faster concerning training speed in comparison with the models above. Also, it will be interesting to compare RF and DT models in terms of overfitting.

In order to utilize deep learning approach, TensorFlow framework will be used. Various techniques for neural network generation will be explored for finding best classifier. It will include Long Short Term Memory neural network layer, since the models which include this layer provided one of the best results in text classification.

Data for training and validation sets will be retrieved from the Analytical (Lab) database. Below mentioned datasets will be selected for training, testing and validation:

- Original texts in Latin alphabet
- original texts in Latin alphabet without Lithuanian stop words;
- texts containing word stems;
- texts containing word lemmas.

In order to have data equally distributed, it will be sampled from the whole database in nearly equal proportion for negative/neutral/positive labels respectively. Neutral review dataset, in this case, is a bit smaller, but it would not disturb the distribution in terms of polarity levels. The data will be split into train and test samples using the ratio 7:3. Both training and test data will be stratified, which keep the labels distributed proportionally amongst two datasets.

To test the model performance, various metrics will be used. In this particular use case, the most critical objective is to identify negative review and prevent negative review identification as positive or neutral. Therefore, the whole confusion matrix and most common metrics will be printed and added in the appendix part. However, results and interpretation will mostly rely on following metrics:

- Accuracy
- Recall

- Sensitivity for negative reviews
- Specificity for negative reviews

Considering the low amount of data (2800 items for training and 1200 for testing), 5-folds cross validation technique will be used for model performance evaluation, relying mostly on accuracy metric. It is proved technique for this type of situation and helps identify whether model performance is the same when introduced with a new validation dataset.

2.3.1. Classic approach

To engineer features for machine learning algorithms, texts should be transformed in a format which classification algorithms expect. The conventional approach to do feature engineering is by creating a term document matrix, as a representation of words in the reviews. It describes the frequency of terms used in the reviews – each row corresponds to the review text while column corresponds to each unique word used. As an example, let's take two sentences as an input:

1. This is the first example
2. The second example

Term document matrix, containing these words is listed below:

Table 3: An example of term document matrix

	This	Is	The	First	Example	Second
Text 1	1	1	1	1	1	0
Text 2	0	0	1	0	1	1

In order to assign word importance, term frequency-inverse document frequency approach will be used. This would help identify not very common words and assign bigger weight to them. Expression of term frequency–inverse document frequency calculation is following:

(1) $tfidf(t,d,D) = tf(t,d) \cdot idf(t,D)$

The output is a multiplication product of inverse document frequency and term frequency. Formulas for these calculations are provided below:

(2) $idf(t,D) = \log\left(\frac{N}{|\{d \in D : t \in d\}|}\right)$

(3) $tf = f_{t,d} / \sum (f_{t',d})$

Classification algorithms can deal with term document matrix since it is a representation of multidimensional space as opposed to raw text. While it has some drawbacks (such as high sparsity or not considering the primary order of words), it still remains one of the most popular approaches for solving text classification problems.

Modeling will be done using "RtextTools" library, which works as a wrapper for the variety of machine learning libraries, which are suited for text classification task. It allows to create joint training container as an object and has other useful features as a model ensemble or calculation of most common metrics. It is important to note, that this approach is suitable only for the prototyping or proof of concept phase. To deploy model in production, more efficient model implementation methods should be used for each algorithm.

2.3.1.1. Decision Tree method

In essence, tree model is based on a sequential division of classification problem into smaller problems in a recursive manner. The final solution is corresponded as tree-like graph, which shows all possible scenarios. The algorithm stops according to stopping conditions, e.g. when most of the points are assigned to the same class depending on the threshold or there are no remaining attributes left. Criteria for split evaluation is defined as an information gain, which is calculated the difference of an entropy for the whole dataset and weighted entropy for a current decision in the formula below.

(4) $Gain(D, D_L, D_R) = H(D) - D(D_L, D_R)$

The algorithm, in a way, mimics human level thinking, therefore it is easy to understand solution for most of the people, including business area specialists. Due to its performance and easy interpretation, it makes decision tree a viable option for solving a variety of classification problems, including text classification.

Decision tree algorithm will be implemented using "tree" R library using standard parameters. All hyperparameters are set to default.

2.3.1.2. Random Forests method

Random Forests algorithm is a decision tree based classifier, which consists of multiple decision trees. The key aspect of the algorithm is the randomness of the trees. Therefore it can cope with

overfitting issue more easily. The prediction of this algorithm is basically the average of individual tree weight:

$$(5) \quad \hat{y} = \frac{1}{m} \sum_{j=1}^{m} \sum_{i=1}^{n} W_j(x_i, x') y_i$$

Random forests have a proven record for text analytics cases. In most of the cases the algorithm is more accurate than the decision tree, because it contains an ensemble of such trees. An ensemble of 128 trees was used during the training phase, since the usage of more trees does not always leads to a higher accuracy of the model and uses more resources for computation (Oshiro, Perez, & Baranauskas, 2012).

An algorithm will be implemented in R using "randomForest" library.

2.3.1.3. Support Vector Machine method

It is kernel based method, which classifies samples by separating them with an optimal hyperplane, depending on the parameters. SVM remained one of most popular text classification algorithms due to its ability to classify texts accurately and cope with overfitting issue.

Algorithm will be implemented using "e1071" R library. Linear kernel and cost of 200 were set for the SVM model.

2.3.2. Deep learning using TensorFlow framework

Deep learning, as a subcategory of machine learning, is the trending area amongst all machine learning related activities. TensorFlow in a leading framework for utilizing these type of networks. One of the most beneficial features of TensforFlow is that training could be done on CPU, GPU or TPU. However, CPU will be used for computations in this work, since high-end GPU is not present on a current machine. Although, it is possible to use exactly the same for data processing on other machine or cloud and significantly increase the processing speed.

Before we begin, it is important to point out some concepts regarding TensorFlow. First of all – required input/output variables and data representation. Tensorflow operates on data objects which are called tensors. Tensors are a generalization of vectors and can be expressed mathematically as below. For instance, vector itself is a one dimensional tensor, matrix is a two dimensional tensor and so on. All tensor operations are based on tensor algebra. A mathematical representation of two dimensional tensor is provided below (7). In the current case, text sequences can be represented as 2D tensors, where one instance is the document id and other instance is word sequence.

(7) $T \in (V \otimes ... \otimes V) \otimes (V' \otimes ... \otimes V')$

1D, 2D and 3D tensors can also be easily represented graphically. A three dimensional tensor, in this case, could be an output of embedding layer – this way 2D layer could be embedded into 3D space using a selected number of dimensions.

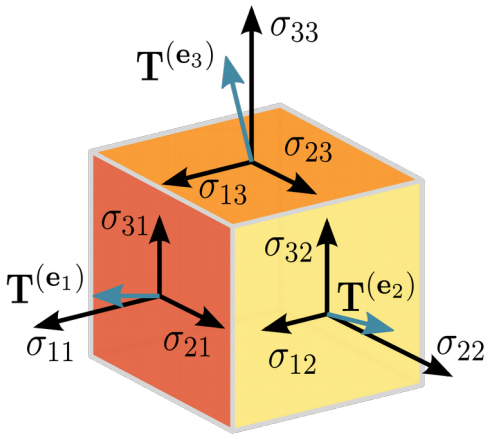

Figure 12: Three dimensional tensor example

Training is done in the following fashion: data is pre-formated to fit the requirements and converted to tensors. For this type of use case, data preparation is quite trivial – each review text is converted into a word vector. All the word vectors later are converted into tensors using one-hot-encoding. Targets also need to be set according to the dimensions of input tensors and target variables. During the training process, loss function defines learning feedback by comparing predictions with actual data, while optimizer determines how successfully learning process goes. Optimizer, on the other hand, updates neural network weights after every iteration. For multi-class classification problem, softmax activation function should be used. For optimization, Adam (adaptive moment estimation) optimization algorithm will be selected for all deep learning classification models in this project.

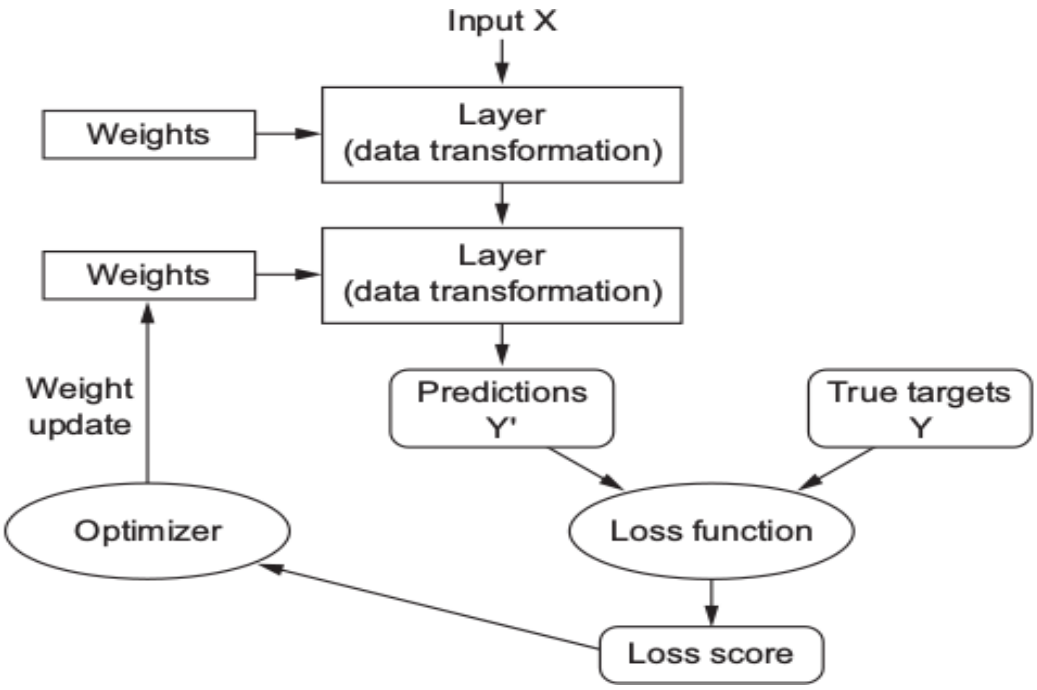

Figure 13: Relationships within Tensorflow framework, François Chollet 2018

Besides pre-defined structure (TensorFlow uses Static Computational Graph), "TensorFlow" model is easy to customize – there are a number of different activation functions, loss functions and optimizers. In this particular use case we will use two-dimentional tensor (text, labels). Model will be created using R language. With the pipe functionality it is possible to define the model in an intuitive way within R environment, although it is important to set input and output shapes in a right way in order to get correct results or even compile the model. Three following models will be implemented using TensorFlow with Keras API.

Neural network with embedding and flat layers. Embedding layer could be interpreted as a dictionary lookup, which maps words to dense vectors. Word vectors are adjusted using back propagation method. The second step flattens the output (3D tensor) to 2D tensors and feed the output to dense layer at the bottom.

Table 4: Neural network with embedding and flatten layers

Layer type	**Output shape**
Embedding	3D (None, max words, 128)
Flatten	2D (None, max words * 128)
Dense	2D (None, 3)

Neural network with Long Short Term Memory layer. LSTM layer is a variation of Recurrent Neural Network layer. As opposed to simple Recurrent Neural Network layer, LSTM is saving information for the future and keeps older signals, which were received in this layer. Required input for such layer is three dimensional tensor, therefore the first layer is embedded. In addition to this, simple dense layer with 128 outputs was added right before the output layer.

Table 5: Neural network with Long Short Term Memory layer

Layer type	**Output shape**
Embedding	3D (None, max words, 128)
LSTM	2D (None, 128)
Dense	2D (None, 128)
Dense	2D (None, 3)

Neural network with Bidirectional Long Short Term Memory and dropout layers. In this case, LSTM layer is duplicated by adding a bidirectional layer in it. Both two layers (original and duplicated) are getting input and reverse input sequence respectively. Dropout layer was added in order to prevent overfitting.

Table 6: Neural network with Bidirectional Long Short Term Memory and dropout layers

Layer type	**Output shape**
Embedding	3D (None, max words, 128)
Bidirectional LSTM	2D (None, 128)
Dropout	2D (None, 128)
Dense	(None, 3)

In the picture below visual comparison between all three neural networks is added. Other hyperparameters will be set as stated below:

- Number of training epochs – tuned depending on training/validation data;
- training optimizer – adaptive moment estimation;
- training loss function – categorical crossentropy;
- activation function – Softmax;
- batch size – 32.

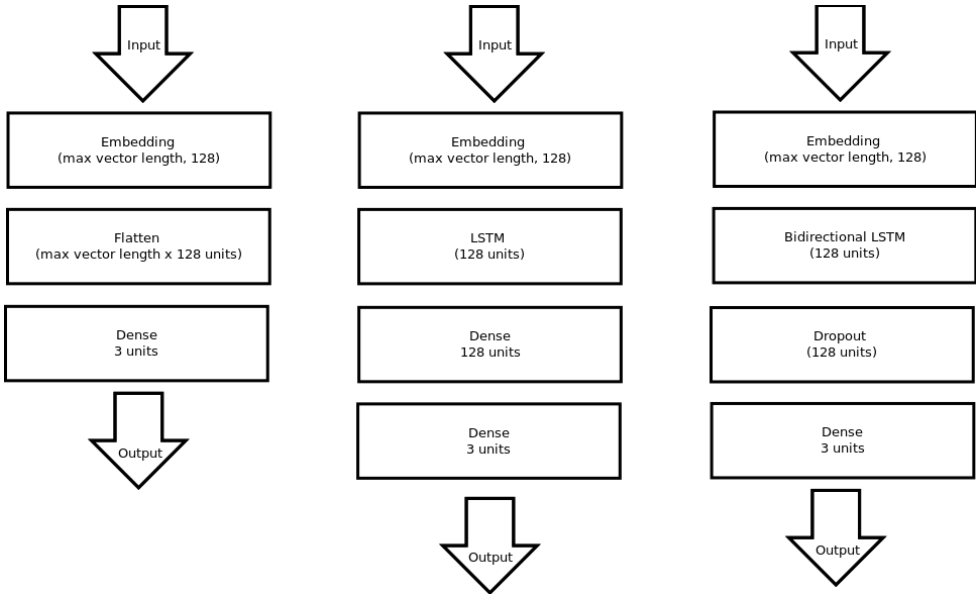

Figure 14: Implemented Deep learning models

2.4. Application development patterns

Before deploying application as a microservice, it is crucial to take into consideration following matters:

- Independent deployment of a service;
- scalability of a service;
- service isolation.

It means that microservices should be loosely coupled with other applications in the enterprise infrastructure. Therefore, they should have specific data storage for both structured and unstructured data and be reachable via HTTP communication channel. The process for data transformation will be written in R and deployed as microservice into the local (prototyping) environment. For deployment into the production environment, two most common approaches can be used:

- Service instance per Virtual Machine – an isolated instance of a server where VM determines resource allocation. It also encapsulates used technology, in this case – Web services which were developed in R – and thus becomes an equivalent of a black-box.

- Service instance per containers – a lightweight alternative to VM deployment. In this case, container image contains just a minimal amount of required technology – Linux kernel and all dependencies for current web service. On the other hand, the infrastructure for this technology is not that mature as in VM case.

Since all the work is done and tested in Linux operating system, there should not occur any issues in terms of compatibility and versioning. Both approaches will work, depending on the organizational structure of a company. A prototype application, however, will work on local Linux machine only – deployment won't be implemented, since requirements could not be defined without a concrete business use case. However, these additional matters should be considered and prepared before releasing application to any production environment:

- Logging – each application should log their action. R tends to optimize the logging experience for the user, since the language is created as an interactive statistical environment rather than GPPL. However, implementation of all the logging actions is relatively easy – it requires juts some additional effort.

- Error handling – each application should have some in-build logic for handling errors, as opposed to just plain scripts, which do only one purpose.

- Unit testing – helps to identify problems early in the development stage. In the beginning it would be enough to write unit tests for all edge-cases, but it can evolve during the time. There are several libraries in R built for that purpose, such as "Runit" or "testthat".

- Code version control – integration with git or other similar tool is a must for every development project.

Implementation of these matters is not in the scope of the project and these recommendations should serve just as guidelines for a concrete use case in production environment. Despite this, the prototypes will be created for data transformation process and analytical model implementation, which will be exposed as web services.

3. Results

Review data was retrieved from the website named www.evertink.lt. It is a central repository for most of the e-commerce shop reviews written in the Lithuanian language. Companies, which have an agreement with this service, ask their customers to fill in the form on this website. If the customer decides to leave a response for these e-commerce businesses, it's most likely to appear on this website first.

3.1. Data Preparation

For the initial phase, 700 instances of POST requests were sent to the resource. Each response contained up to 30 reviews and stored as RDS file on disk. Received data object is more suitable for passing the data from the back-end to the front-end application, in this case – web browser, rather than suited for data analysis. Since the relevant part of this project is only the review part, it will be stored on disk without any modifications. Structure of a file received from www.evertink.lt website in original JSON format is provided below:

Figure 15: Input JSON File Structure

During the second phase, all review instances were saved in MySQL database under “refined” schema. After a union, over 16 thousand reviews were stored in the database.

```
mysql> select count(*) from joined;
+----------+
| count(*) |
+----------+
|    16293 |
+----------+
```

Figure 16: Row count of combined review table

Third phase transformation consisted of three parts:

1. Removing HTML code and other symbols
2. Getting stems and lemmas for each word

In this case, two minor issues were found regarding Semantika.lt REST API. The first one is regarding lack of resource documentation. In this case, the resource “/chains/syntax” has a limited number of request per time interval. Using trial and error method, it was determined that more than 20 requests per minute give an error. Another issue occurs if the amount of Lithuanian words is less then 50% - the API responds with an error. The same result goes for Lithuanian text written in Latin alphabet – the system does not identify these words as Lithuanian at all. Despite these issues, text stemming and lemmatization went successfully – only 6% (233 reviews) failed. On the other hand, not all words have been processed due to spelling mistakes, Latin alphabet or barbarisms even for these successful entries.

3. Adding additional columns to the table for texts in Latin alphabet and texts without Lithuanian stop words.

Results of these transformations were stored in the table “semantic_data” under Lab database and prepared for manual label assignment process.

3.2. Label assignment process

During sentiment assignment process, three classes were chosen. The neutral class was the hardest to distinguish between other two and might be the most subjective. However, in this particular use case the main idea is to separate business critical reviews in order to react quickly to them rather than identifying customer feelings.

Table 7: Manually assigned sentiment labels

Sentiment	Label	Description
Negative	-1	The customer is not satisfied (usually comes with low rating (e.g. < 6))
Neutral	0	Can include both positive and negative comments, but are not business critical in terms of customer service
Positive	1	The customer is satisfied (usually comes with high rating (e.g. >9))

There are a couple of review examples provided below. As expected, labeling neutral reviews is challenging and mostly depends on a person who takes a decision to assign the value.

Table 8: Label examples of customer reviews texts

Average rating	Review text in Lithuanian	Review text in English	Label assigned
8	Likau labai patenkinta. Nuostabus aptarnavimas.	I was very pleased. Excellent service.	1
8	Esu labai patenkinta Jusu paslaugomis. Aciu, kad esate	I am very happy with your services. Thanks for being.	1
8	Per daznai siuntinejate naujienlaiskius. Visa kita puiku.	Newsletters are sent too often. Everything else is great.	0
8	Kilo problemų pristatant knygas (jo metu knygos buvo pažeistos), į nusiskundimą buvo greitai sureaguota, o problema operatyviai išspręsta. Dėkojame už puikų darbą!	Books were damaged during delivery phase. Reaction was great from your side and you managed to solve the problem quickly. Thank you for your great service!	0
8	Pirkta prekė neatitiko aprašymo. Esu labai nusivylusi.	Specifications were wrong for the item I bought. I am very disappointed.	-1

While inspection was done to quite a significant portion of data, some of the most significant areas of service quality issues were appointed as result of this analysis. Most of the negative reviews can rely on one or more categories listed below:

- Bad quality goods;
- broken goods during transportation;

- delivery timing issues;
- impolite consultants;
- missing goods;
- specification of goods differs in the website.

However, categorizing the review texts into broader categories is not within the scope of the project, but it might be useful to create a model for this type of categorization when implementing the application into production environment. This would allow to get more useful statistics and track key performance indicators for all these groups.

The initial aim of labeling was to mark at least one thousand reviews for each class. Since neutral reviews are quite rare, it was labeled slightly over 1000 reviews as neural, and over 1500 reviews as positive/negative respectively. To prevent selection of very short reviews, minimal amount of symbols in each review were set to 40.
At the end, a total of 4000 reviews were sampled randomly (1500/1000/1500 for each class respectively) and stored under “sample” data table in the Lab database.

3.3. Exploratory Analysis

Customers reviews for 41 E-commerce companies were retrieved from the website. Top three companies have the lion's share of reviews – it covers nearly 60% of total review count. On the other hand, reviews for different companies bring in more variety in the dataset in terms of products and customer opinions. Moreover, these reviews also bring more negative reviews on the table, since the average review score for top three companies is really high, thus balancing the labels in the sample a bit.

Table 9: Top 3 companies by review count

Shop name	**Number of reviews**	**Review average score**
Pigu.lt	6255	89
Knygos.lt	2041	92
Neriba.lt	2029	95

Review amount highly increases every December, since it is the most popular time to buy presents. Sales activities associated with this period are also high in most sectors. Because of such sharp peaks in terms of review count, the motivation for review classification software highly increases – there might not be enough resources to handle these reviews manually, especially during the most active periods.

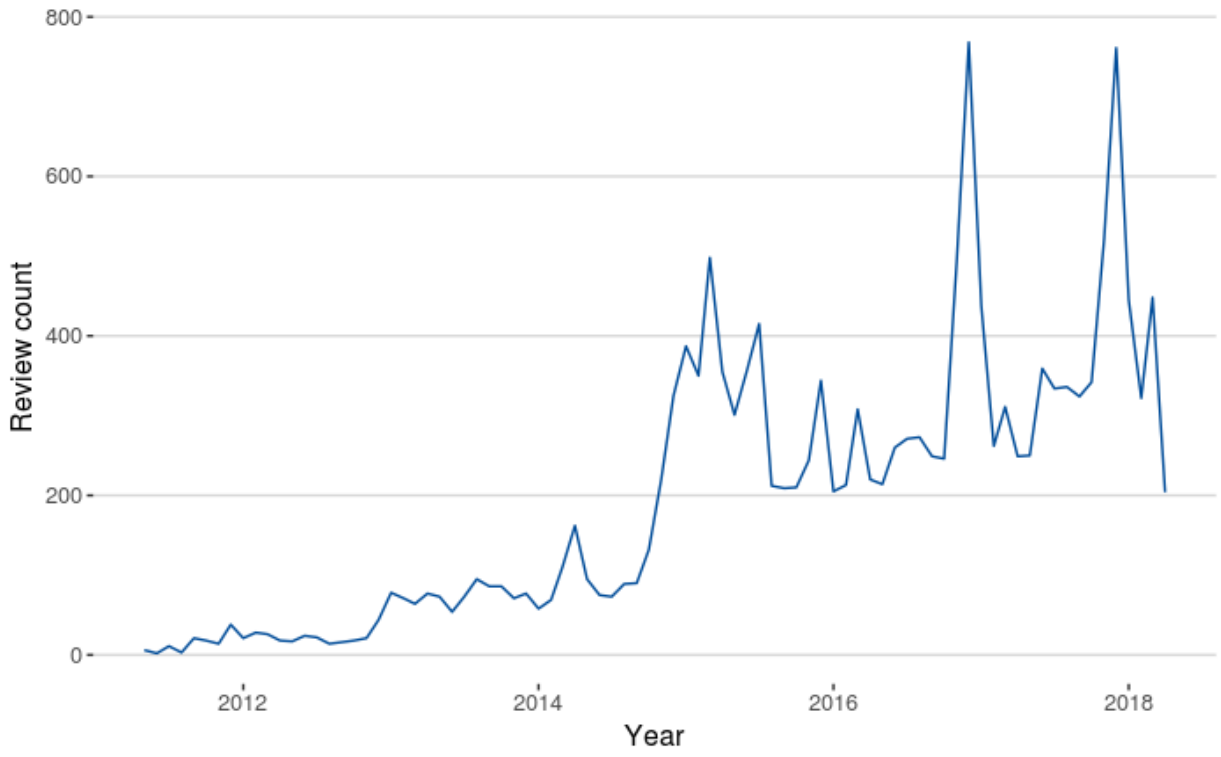

Figure 17: Review count by month

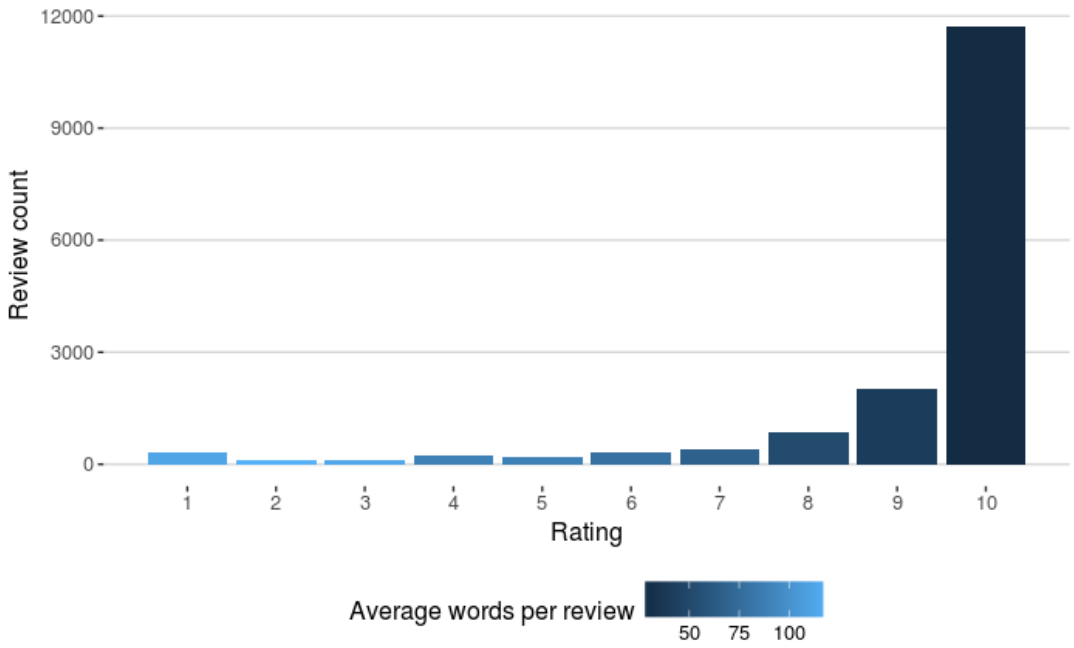

Figure 18: Review count per rating

The distribution of review rating is skewed to the right. However, the average word length is biggest for negative reviews. It shows that people tend to express negative emotions in detail, compared to reviews, which received the biggest score and are having positive or neutral sentiment. Most of the reviews were written in the morning, between 9 and 10 AM. Interesting facts from this type of analysis are following: longer reviews were written at midnight and shortest reviews were written at 4:00 AM.

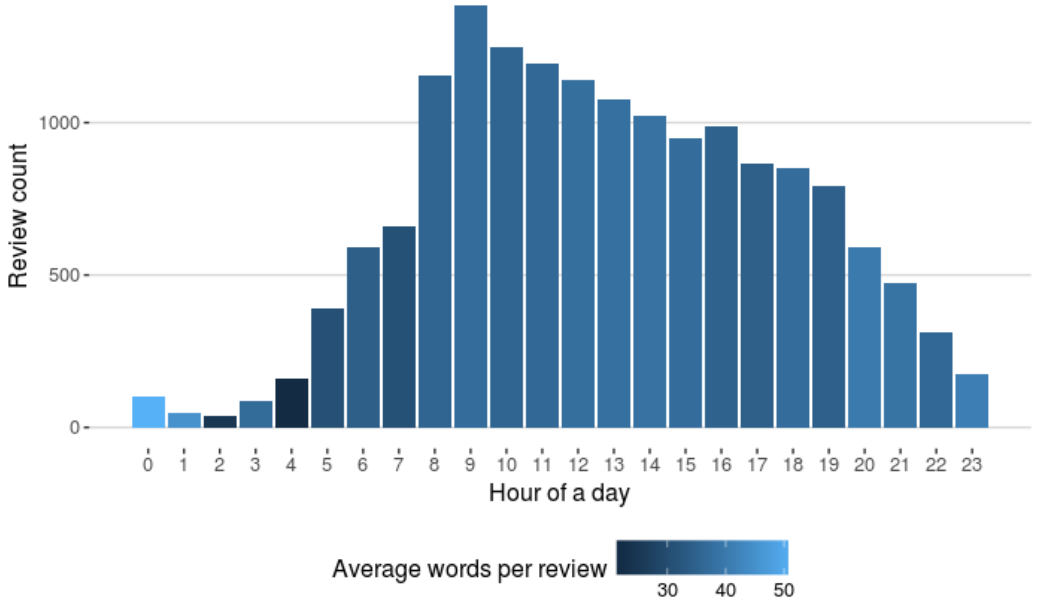

Figure 19: Review count per hour

As it could been expected, positive words which defines gratitude (e.g. “ačiū” (en. “thanks”), “greitai” (en. fast”) or patenkinta (en. “grateful”)) are more frequent amongst all review texts, since the reviews distribution is highly unbalanced in terms of ratings.

Figure 20: Wordcloud graph of all review texts

Most popular words for negative expression are provided below. The sample is lower and due to this reason it is more difficult to see patterns in words. However, it is easy to spot words, which indicates negative sentiment, such as “there is no” (lt.“nėra”), “haven’t received” (lt. “negavau”), or “wasn’t” (lt. “nebuvo”).

Figure 21: Wordcloud graph of negative review texts

Some reviews have comments, which are the replies written by customer service specialists. Amount of such comments is insignificant and mostly concerns reviews having very low rating, which is obvious. That is why detecting negative sentiment automatically and responding quickly to a negative review with a higher rating would be more impressive from the customer perspective.

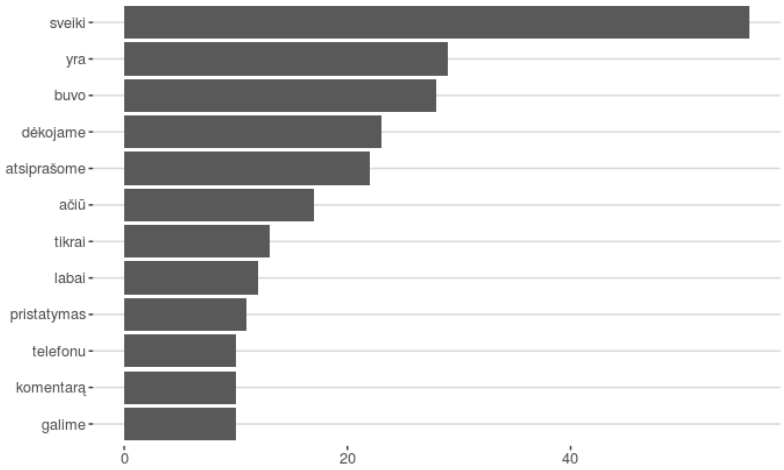

Figure 22: Most popular words in review comments

For testing purposes, most obvious lexicon based approach for text classification was introduced. The following analysis method is based on using bag of words. One publicly available lexicon for Lithuanian language contains 2190 words and was generated by a research group (Chen & Skiena, 2014). Unsurprisingly, results were not so good – when identifying only two classes (positive and negative) the accuracy was 0.735. More than a half of negatives were identified as positives. For three classes, accuracy went down to 0.53, leaving the same issue regarding false positives. For this reason, pure machine learning approach will be used in the further state of a project. Firstly, features were created using term document matrix. Below is provided statistics of four different matrices, depending on the data sample.

Table 10: Statistics of term document matrices

Dataset (n = 4000)	**Terms**	**Non-sparse entries**	**Sparse entries**
Latin encoding	15783	83922	63048078
Latin without stopwords	15607	70543	62357457
Only stems	10981	79324	43844676
Only lemmas	10684	80344	42655656

Another type of features called tensors were created using a similar approach for each dataset individually. When creating word indexes, a sequence of words have been generated for each review entry and then transformed to two dimensional tensors (matrices). In this project word indexes will not be shortened, which on the other hand is a common practice when optimizing the performance of deep learning networks. Because of a low amount of the data, those indexes were left with the same length as word index in matrices for each data sample respectively.

3.4. Results of implemented methodology

As expected, SVM was a clear winner amongst standard algorithms. Text transformations, on the other hand, didn't add any performance edge for SVM method. However, decision tree performance slightly increased due to these transformations – both accuracy and recall rose over 0.02 using word lemmas instead of ordinary words.

Table 11: Results of DT, RF and SVM models

Dataset	**Models**					
	Decision tree		**Random Forests**		**SVM**	
	Accuracy	Recall	Accuracy	Recall	Accuracy	Recall
Latin	0.594	0.533	0.715	0.663	***0.760***	***0.730***
Latin w/o stop-words	0.586	0.530	0.713	0.657	***0.767***	***0.737***
Stems	0.606	0.557	0.705	0.650	***0.738***	***0.703***
Lemmas	0.616	0.557	0.699	0.640	***0.738***	***0.700***

In terms of negative review classification, SVM managed to get highest specificity score for negative review class – it varies from 0.82 to 0.84 for all four datasets and is far more superior compared with other models. Sensitivity, however, is a bit lower and hardly reaches 0.89 even with the best dataset variation. On the other hand, SVM model is more stable and does not identify false negatives as much as other models. Another way to increase classification performance is by using ensemble which includes all three models. While two models does not lead to significant gain in performance, the combination of all three models increases recall up to 0.84 for 60% of text in Latin alphabet. However, in terms of accuracy, emsemble of these three models classifies reviews slightly worse than SVM by itself – it varies from 0.715 to 0.721 for all data samples listed below.

Table 12: Model ensemble performance based on consensus

Ensemble size	**Latin**		**Latin w/o stop words**		**Stems**		**Lemmas**	
	Coverage	Recall	Coverage	Recall	Coverage	Recall	Coverage	Recall
1	1	0.72	1	0.72	1	0.72	1	0.72
2	0.96	0.74	0.96	0.73	0.96	0.73	0.96	0.73
3	0.60	0.84	0.61	0.83	0.62	0.82	0.62	0.83

The average accuracy for SVM model using 5-Folds cross validation method is slightly lower than using the original train and test samples. It is a normal phenomenon, since hyperparameters are tuned using the response from the feedback from original validation. On the other hand, there are not much of difference and SVM still outperforms other models. The classification was the best from non-transformed words using only the Latin alphabet. Stability is excellent for RF algorithm as well. DT algorithm, on average, performed better than using pre-fixed training and validation datasets. Also, model performance slightly increased using linguistically pre-processed data, but still didn't reach SVM or RF level.

Table 13: 5-Fold cross validations results of DT, RF and SVM models

Dataset	**5-Folds cross validation average model accuracy**		
	Decision tree	**Random Forests**	**SVM**
Latin	0.599	0.717	***0.758***
Latin without stopwords	0.619	0.713	0.747
Stems	0.649	0.700	0.738
Lemmas	0.646	0.710	0.724

Regarding runtime, SVM performed the best here also. It ran even faster than DT algorithm for all datasets. RF was the slowest in terms of training speed and thus not entirely suitable for the current business case. On the other hand, in case the model would meet requirements for classification performance, there will always be a possibility to parallelize the model between multiple threads.

Table 14: DT, RF and SVM training times

Dataset	Decision tree	Random Forests	SVM
Latin	29.41 sec	1848.15 sec	10.29 sec
Latin without stopwords	24.14 sec	2286.66 sec	14.437 sec
Stems	16.29 sec	1128.98 sec	12.26 sec
Lemmas	14.76 sec	804.97 sec	11.74 sec

For TensorFlow models, best results were provided by a flat neural network with embedding layer. It is not that surprising – dataset volume is very low. Therefore, neural networks with more advanced structure do not get an additional performance boost and in some cases does not even reach significant higher than a naive score (0.375).

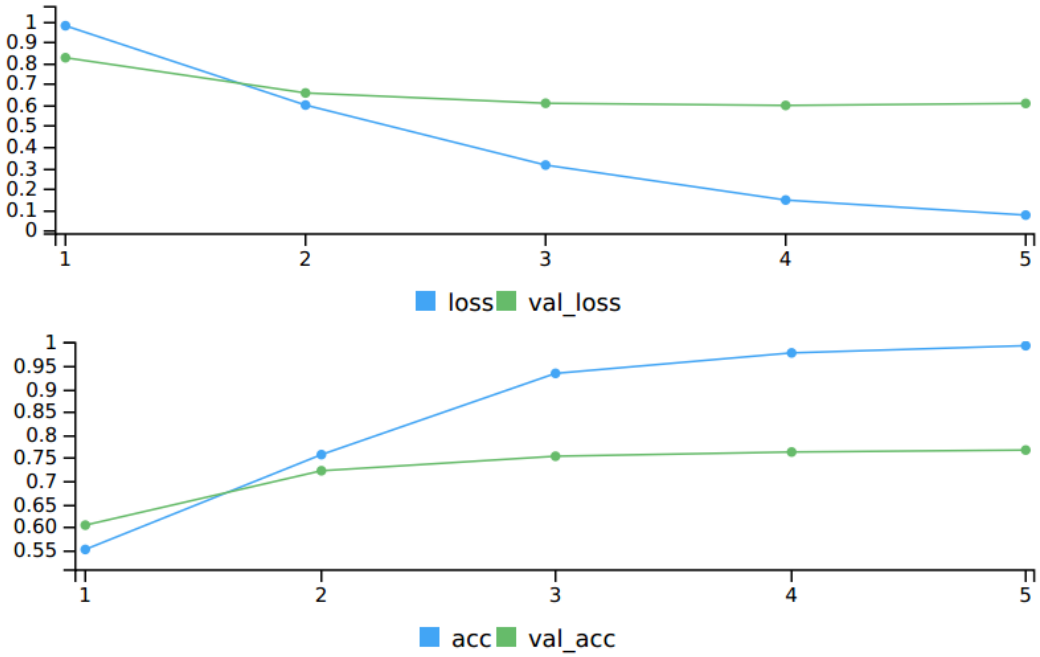

Figure 23: Learning curve of flat neural network with embedding layer (accuracy + loss metrics)

Of course, more advanced hyperparameters tunning needs to be done in order to increase the performance, especially for neural networks which contain LSTM layer. However, flat neural network easily reached the best performance level of SVM algorithm and thus became a model of choice for this project.

Table 15: Results of implemented deep learning models

Dataset	Models					
	Embedded NN		LSTM NN		BLSTM NN	
	Accuracy	Recall	Accuracy	Recall	Accuracy	Recall
Latin	***0.766***	0.729	0.685	0.626	0.739	0.716
Latin without stop-words	0.751	0.715	0.678	0.656	***0.755***	0.735
Stems	***0.760***	0.727	0.376	0.333	0.737	0.716
Lemmas	***0.756***	0.721	0.687	0.624	0.735	0.710

When dealing with negative reviews, flat neural network with embedding layer performed slightly worse than SVM – best reached 0.874 sensitivity and 0.836 specificity level in comparison with 0.887 and 0.840 for SVM sensitivity and specificity score respectively.

Neural network with embedding and flatten layer also proved to be resistant to different data and training samples using cross validation method, while other models performed worse, comparing with original validation set. There were situations where more advanced model scored even lower than naive predictor (e.g. selecting only one sentiment for all inputs). This occurred due to lack of model tuning and the modest amount of training/validation entries.

Table 16: 5-Fold cross validations results of deep learning models

Dataset	5-Folds cross validation average model accuracy		
	Embedded NN	LSTM NN	BLSTM NN
Latin	***0.758***	0.654	0.303
Latin without stopwords	0.748	0.538	0.339
Stems	0.751	0.684	0.334
Lemmas	0.750	0.587	0.349

3.5. The final application

Application for ELT process is created and accessible via HTTP protocol Post functionality using the same architecture as in the figure no. 10. Data transformations can be managed via Data Flow managers such as Apache NIFI. Another way to manage ELT process is by scheduling in Apache Airflow or Luigi application, depending on current organization application stack. The most primitive way – schedule as in "cron" job, which is time-based job scheduler.

For the Analytical model application, neural network model with embedding and flatten layers was selected. The model was deployed locally as a Web service using "tfdeploy" package. It can be reached via the Swagger tool, because TensorFlow provides Swagger documentation by default.

Figure 24: TensorFlow model resource in Swagger

Since the model requires tensors as an input, another Web service was created for that purpose. R package "plumber" was used in order to perform that type of preparation. The main purpose of this web service is to transform the input using the "TensorFlow" tokenizer in order to create word vectors and tensors. Above mentioned microservice is exposed as a set of Get functions for Analytical application version check and text classification task. For text classification task input parameter is text, while the application returns additional metadata associated with the input, such as:

- Word vector;
- the richness of the vocabulary for particular text input;
- probabilities for each class and final sentiment label assigned.

An example of such response body is provided below.

Response body


```
{
  "status": [
    200
  ],
  "text": [
    "teigiamai vertinu sia elektronine parduotuve, rekomenduoju visiems cia pirkti."
  ],
  "tensor": [
    [
      3114,
      462,
      264,
      1458,
      62,
      211,
      354,
      87,
      95
    ]
  ],
  "vocabulary": [
    1
  ],
  "probs": [
    {
      "neg": 0,
      "neu": 0.0012,
      "pos": 0.9987
    }
  ],
  "sentiment": [
    "pos"
  ]
}
```


Figure 25: Response example of the Analytical application (microservice)

Analytical application was deployed as a web service and it can be integrated in a variety of use cases. For instance, it can classify streaming or batch data and be integrated into any front end or even used in mobile application.

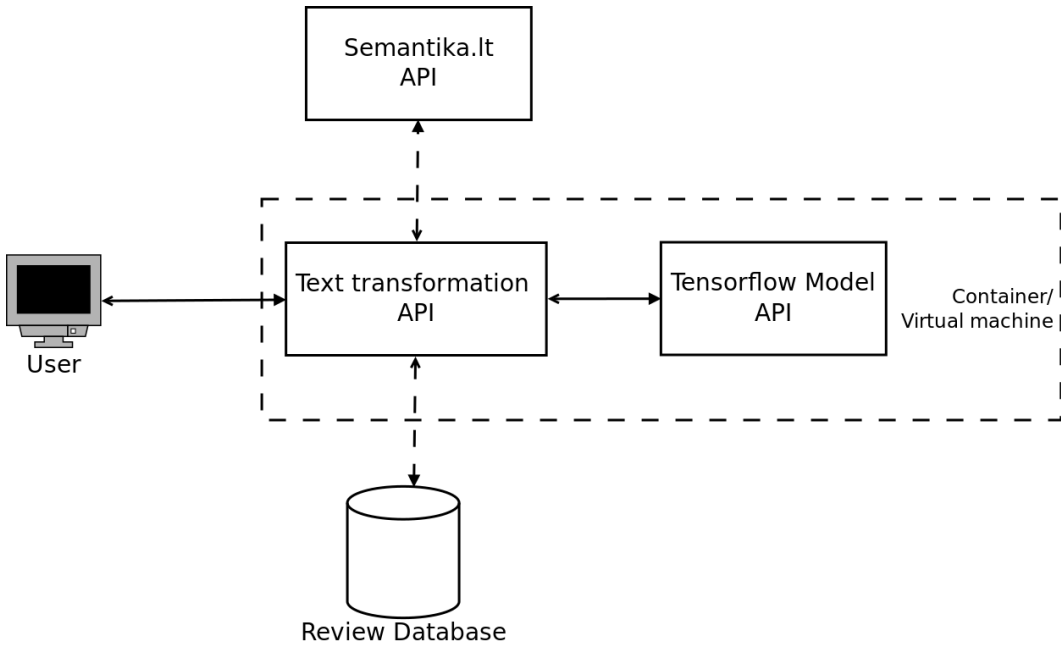

Figure 26: Structure of the Analytical application

Of course, the scalability question is still open, but it can be resolved by choosing a variety of techniques, such as deploying on different containers or using more performance efficient languages. The model of TensorFlow is separated by the design so it can be reused in other environments also.

Conclusions

- According to the obtained literature, SVM and Deep neural networks are the leading methods for customer review classification in the English language. For Lithuanian language research is mostly limited to conventional machine learning algorithms such as SVM, Naybe Bayes and Random Forests and includes complicated pre-processing techniques to cope with the limitations of these models.
- Methodology for data ingestion, transformation, sampling and review classification was created. It includes four different data samples and six different machine learning models. Data stemming and lemmatization were introduced as a part of sample preparation.
- Data stemming and lemmatization did not provide a significant increase in performance, primarily due to the limitations of semantika.lt Web service. Moreover, people tend to create low linguistic quality Lithuanian texts by doing grammar mistakes, using jargon or typing in Latin alphabet when writing reviews. As a result, above mentioned text pre-processing steps work just for part of the data.
- SVM classification model performed as good as expected (accuracy = 0.767) and could serve as a valid alternative in comparison with deep learning models developed in this project.
- Best deep learning model was achieved using embedding and flatten layers (accuracy = 0.766). The rest of the models did not performed as good, mostly due to lack of training data and hyperparameters tuning. Pre-trained word embedding layer could be another option for getting the performance up. From a sustainability perspective, deep learning approach is better when comparing to classical models in terms of application maintenance.
- Reproducible data transformations were implemented using only R language. Data transformation application was deployed as a microservice, but further adjustments as error handling or unit tests need to be added when building fully functional production application, as suggested in the guidelines.
- Tensorflow model was successfully implemented as a microservice and classifies raw text input into three categories – negative, neutral and positive. All tests in local Unix

environment were successful. It could be used in a variety of use cases for text classification within any modern organization environment.

- The usage of R programming language for such use cases boosts the development process significantly and allows Data Scientists to take full ownership of microservices they create.

With a few minor modifications, this application for sentiment analysis can be used in variety of business sectors, such as banking, entertainment and customer service. It can also be adjusted for such use cases, where implementation is not so trivial: evaluating customer experience when chatting with a customer serving specialist or a bot. When done in automated fashion, it can provide valuable metrics and insights about customer experience in the real time environment.

List of references

Aruoba, S. B., & Fernández-Villaverde, J. (2015). A comparison of programming languages in macroeconomics. *Journal of Economic Dynamics and Control*, *58*, 265–273. https://doi.org/10.1016/J.JEDC.2015.05.009

Bhadane, C., Dalal, H., & Doshi, H. (2015). ScienceDirect Sentiment analysis: Measuring opinions. *Procedia - Procedia Computer Science*, *45*, 808–814. https://doi.org/10.1016/j.procs.2015.03.159

BoraGan Aruoba, S., & Fernández-Villaverde, J. (2018). A Comparison of Programming Languages in Economics: An Update. Retrieved from https://www.sas.upenn.edu/~jesusfv/Update_March_23_2018.pdf

Chen, Y., & Skiena, S. (2014). Building Sentiment Lexicons for All Major Languages, 383–389. Retrieved from https://aclanthology.info/pdf/P/P14/P14-2063.pdf

Chollet, F., & Allaire, J. J. (2018). *Deep learning with R*.

CrowdFlower. (2016). CrowdFlower Data Science Report, 8–9.

Dimensional Research. (2013). *Customer Service and Business Results*. *Dimensional Research*. Retrieved from www.dimensionalresearch.com

Endel, F., & Piringer, H. (2015). Data Wrangling: Making data useful again. *IFAC-PapersOnLine*, *48*(1), 111–112. https://doi.org/10.1016/j.ifacol.2015.05.197

Fang, X., & Zhan, J. (2015). Sentiment analysis using product review data. *Journal of Big Data*, *2*(1), 5. https://doi.org/10.1186/s40537-015-0015-2

Farris, P., Bendle, N., Pfeifer, P., & Reibstein, D. (2010). *MARKETING METRICS SECOND EDITION*. Retrieved from http://ptgmedia.pearsoncmg.com/images/9780137058297/samplepages/9780137058297.pdf

Google Cloud Natural Language API Documentation | Google Cloud Natural Language API | Google Cloud. (n.d.). Retrieved April 16, 2018, from https://cloud.google.com/natural-language/docs/

Jules Kouatchou. (2016). Basic Comparison of Python, Julia, R, Matlab and IDL. Retrieved from https://modelingguru.nasa.gov/docs/DOC-2625

Kapočiūtė-Dzikienė, J., Krupavičius, A., & Krilavičius, T. (2013). A Comparison of Approaches for Sentiment Classification on Lithuanian Internet Comments, 2–11. Retrieved from http://www.aclweb.org/anthology/W13-2402

Kapočiūtė-Dzikienė, J., Vaassen, F., Daelemans, W., & Krupavičius, A. (2012). Improving Topic Classification for Highly Inflective Languages. Retrieved from https://pdfs.semanticscholar.org/bddc/986580ac78be02a3b4be1928a13ff24133c6.pdf

Katuwal, K. (2016). Microservices : A Flexible Architecture for the Digital Age Version 1 . 1, *3*(4), 23–28.

Kumar, G. (2016). Identify The Need for Developing a New Service Quality Model in Today's Scenario: A Review of Service Quality Models. Retrieved from https://www.omicsonline.org/open-access/identify-the-need-for-developing-a-new-service-quality-model-in-todaysscenario-a-review-of-service-quality-models-2223-5833-1000193.pdf

Lee, I. (2016). *Encyclopedia of e-commerce development, implementation, and management*.

Mickevičius, V., Krilavičius, T., & Morkevičius, V. (2015). Classification of Short Legal Lithuanian Texts, 106–111. Retrieved from http://bpti.lt/wp-content/uploads/2016/02/bsnlp2015.pdf

Nisar, T. M., & Prabhakar, G. (2017). What factors determine e-satisfaction and consumer spending in e-commerce retailing? *Journal of Retailing and Consumer Services*, *39*, 135–144. https://doi.org/10.1016/J.JRETCONSER.2017.07.010

Okockis, V. (2016). RESEARCH OF SENTIMENT ANALYSIS METHODS FOR DIGITAL CONTENT. Retrieved from http://old.mii.lt/files/09p_okockis_ataskaita2016.pdf

Oshiro, T. M., Perez, P. S., & Baranauskas, J. A. (2012). How Many Trees in a Random Forest? (pp. 154–168). Springer, Berlin, Heidelberg. https://doi.org/10.1007/978-3-642-31537-4_13

Patil, H., & Divekar, B. R. (2014). Inventory Management Challenges for B2C E-commerce Retailers. *Procedia Economics and Finance*, *11*, 561–571. https://doi.org/10.1016/S2212-5671(14)00221-4

Pérez, J., Iturbide, E., Olivares, V., Hidalgo, M., Martínez, A., & Almanza, N. (2015). A Data Preparation Methodology in Data Mining Applied to Mortality Population Databases. *J Med Syst*, *39*. https://doi.org/10.1007/s10916-015-0312-5

Podium. (2017). Online Review Stats. Retrieved from http://learn.podium.com/rs/841-BRM-380/images/Podium-2017-State-of-Online-Reviews.pdf?mkt_tok=eyJpIjoiTUdRM04ySTBOR1ZqTURNNSIsInQiOiJVTktEOXNtTXlpZGFhM29YQUFyNXJZWXpNRGhLTUpYVk5nSWdcL0RPMmcwcWdjaFRlazRiMlU5ZDlcL01DMVJBNVdLVHNLYUs0eEM5Uko1dkRCdVZoRHFVbzNDM

Poliakovas, O. (2015). *The marvel of Indo-European cultures and languages : the Lithuanian bridge to Indo-European*. Vilniaus Univ. Publ. House. Retrieved from https://books.google.lt/books/about/The_Marvel_of_Indo_European_Cultures_and.html?id=zlXWjgEACAAJ&redir_esc=y

Reddy, N. A., & Divekar, B. R. (2014). A Study of Challenges Faced By E-commerce Companies in India and Methods Employed to Overcome Them. *Procedia Economics and Finance*, *11*, 553–560. https://doi.org/10.1016/S2212-5671(14)00220-2

Richardson, C. (2018). *Microservice Patterns.* Manning Pubns Co. Retrieved from https://www.manning.com/books/microservices-patterns

Socher, R., Perelygin, A., Wu, J. Y., Chuang, J., Manning, C. D., Ng, A. Y., & Potts, C. (2013). Recursive Deep Models for Semantic Compositionality Over a Sentiment Treebank. Retrieved from https://nlp.stanford.edu/~socherr/EMNLP2013_RNTN.pdf

SparkR (R on Spark). (2018). Retrieved April 19, 2018, from https://spark.apache.org/docs/latest/sparkr.html

Špats, G., & Birzniece, I. (2016). Opinion Mining in Latvian Text Using Semantic Polarity Analysis and Machine Learning Approach. *Complex Systems Informatics and Modeling Quarterly*, *0*(7), 51–59. https://doi.org/10.7250/csimq.2016-7.03

Sturm, R., Pollard, C., Craig, J., Sturm, R., Pollard, C., & Craig, J. (2017). Application Programming Interfaces and Connected Systems. In *Application Performance Management (APM) in the Digital Enterprise* (pp. 137–150). Elsevier. https://doi.org/10.1016/B978-0-12-804018-8.00011-5

Tang, D., Qin, B., & Liu, T. (2015). Document Modeling with Gated Recurrent Neural Network for Sentiment Classification, 1422–1432. Retrieved from http://www.emnlp2015.org/proceedings/EMNLP/pdf/EMNLP167.pdf

Uskenbayeva, R., Kuandykov, A., Cho, Y. I., Temirbolatova, T., Amanzholova, S., & Kozhamzharova, D. (2015). ScienceDirect Integrating of data using the Hadoop and R. *Procedia Computer Science*, *56*, 145–149. https://doi.org/10.1016/j.procs.2015.07.187

Vitkutė-Adžgauskienė, D., Utka, A., Amilevičius, D., & Krilavičius, T. (2016). NLP Infrastructure for the Lithuanian Language. Retrieved from https://eltalpykla.vdu.lt/bitstream/handle/1/33093/ISBN9782951740891.PG_2539-2542.pdf?sequence=1&isAllowed=y

Vyas, V., & Uma, V. (2018). ScienceDirect An Extensive study of Sentiment Analysis tools and Binary Classification of tweets using Rapid Miner. *Procedia Computer Science*, *125*(00), 329–335. https://doi.org/10.1016/j.procs.2017.12.044

Zeithaml, V. A. (2000). Service Quality, Profitability, and the Economic Worth of Customers: What We Know and What We Need to Learn. *Journal of the Academy of Marketing Science*, *28*(1), 67–85. https://doi.org/10.1177/0092070300281007

Zhou, P., Qi, Z., Zheng, S., Xu, J., Bao, H., & Xu, B. (2016). Text Classification Improved by Integrating Bidirectional LSTM with Two-dimensional Max Pooling, 3485–3495. Retrieved from http://www.aclweb.org/anthology/C16-1329

Appendices

Appendix 1

DT, RF, SVM and model ensemble results

```
<<DocumentTermMatrix (documents: 4000, terms: 15783)>>
Non-/sparse entries: 83922/63048078
Sparsity           : 100%
Maximal term length: 24
Weighting          : term frequency - inverse document frequency (normalized) (tf-idf)
Training models, phase textlatin: 2210.973 sec elapsed
          Reference
Prediction  -1   0   1
        -1 426 227 174
        0    8  19   8
        1   17  54 268
Overall Statistics

               Accuracy : 0.5937
                 95% CI : (0.5653, 0.6216)
    No Information Rate : 0.3755
    P-Value [Acc > NIR] : < 2.2e-16

                  Kappa : 0.3534
 Mcnemar's Test P-Value : < 2.2e-16

Statistics by Class:
                     Class: -1 Class: 0 Class: 1
Sensitivity             0.9446  0.06333   0.5956
Specificity             0.4653  0.98224   0.9055
Pos Pred Value          0.5151  0.54286   0.7906
Neg Pred Value          0.9332  0.75901   0.7889
Precision               0.5151  0.54286   0.7906
Recall                  0.9446  0.06333   0.5956
F1                      0.6667  0.11343   0.6793
Prevalence              0.3755  0.24979   0.3747
Detection Rate          0.3547  0.01582   0.2231
Detection Prevalence    0.6886  0.02914   0.2823
Balanced Accuracy       0.7050  0.52279   0.7505
          Reference
Prediction  -1   0   1
        -1 390  98  27
        0   50 152  52
        1   11  50 371
Overall Statistics

               Accuracy : 0.7602
                 95% CI : (0.735, 0.7841)
    No Information Rate : 0.3755
    P-Value [Acc > NIR] : < 2.2e-16

                  Kappa : 0.6319
 Mcnemar's Test P-Value : 5.533e-05
Statistics by Class:
                     Class: -1 Class: 0 Class: 1
Sensitivity             0.8647   0.5067   0.8244
Specificity             0.8333   0.8868   0.9188
Pos Pred Value          0.7573   0.5984   0.8588
Neg Pred Value          0.9111   0.8437   0.8973
Precision               0.7573   0.5984   0.8588
Recall                  0.8647   0.5067   0.8244
F1                      0.8075   0.5487   0.8413
Prevalence              0.3755   0.2498   0.3747
Detection Rate          0.3247   0.1266   0.3089
```

```
Detection Prevalence    0.4288   0.2115   0.3597
Balanced Accuracy       0.8490   0.6967   0.8716
          Reference
Prediction  -1   0   1
        -1 412 173  65
        0   12  75  13
        1   27  52 372
Overall Statistics

               Accuracy : 0.7152
                 95% CI : (0.6888, 0.7406)
    No Information Rate : 0.3755
    P-Value [Acc > NIR] : < 2.2e-16

                  Kappa : 0.5517
 Mcnemar's Test P-Value : < 2.2e-16
Statistics by Class:
                     Class: -1 Class: 0 Class: 1
Sensitivity             0.9135  0.25000   0.8267
Specificity             0.6827  0.97225   0.8948
Pos Pred Value          0.6338  0.75000   0.8248
Neg Pred Value          0.9292  0.79564   0.8960
Precision               0.6338  0.75000   0.8248
Recall                  0.9135  0.25000   0.8267
F1                      0.7484  0.37500   0.8257
Prevalence              0.3755  0.24979   0.3747
Detection Rate          0.3430  0.06245   0.3097
Detection Prevalence    0.5412  0.08326   0.3755
Balanced Accuracy       0.7981  0.61113   0.8607
ENSEMBLE SUMMARY

       n-ENSEMBLE COVERAGE n-ENSEMBLE RECALL
n >= 1                1.00              0.72
n >= 2                0.96              0.74
n >= 3                0.60              0.84


ALGORITHM PERFORMANCE
    SVM_PRECISION        SVM_RECALL        SVM_FSCORE FORESTS_PRECISION
        0.7400000         0.7300000         0.7333333         0.7333333
   FORESTS_RECALL    FORESTS_FSCORE    TREE_PRECISION       TREE_RECALL
        0.6633333         0.6466667         0.6166667         0.5333333
      TREE_FSCORE
        0.4866667
<<DocumentTermMatrix (documents: 4000, terms: 15607)>>
Non-/sparse entries: 70543/62357457
Sparsity           : 100%
Maximal term length: 24
Weighting          : term frequency - inverse document frequency (normalized) (tf-idf)
Training models, phase textstop: 2386.74 sec elapsed
          Reference
Prediction  -1   0   1
        -1 408 214 169
        0   11  28  13
        1   32  58 268
Overall Statistics

               Accuracy : 0.5862
                 95% CI : (0.5577, 0.6142)
    No Information Rate : 0.3755
    P-Value [Acc > NIR] : < 2.2e-16

                  Kappa : 0.3433
 Mcnemar's Test P-Value : < 2.2e-16
Statistics by Class:
                     Class: -1 Class: 0 Class: 1
Sensitivity             0.9047  0.09333   0.5956
Specificity             0.4893  0.97336   0.8802
Pos Pred Value          0.5158  0.53846   0.7486
```

```
Neg Pred Value          0.8951  0.76327   0.7841
Precision               0.5158  0.53846   0.7486
Recall                  0.9047  0.09333   0.5956
F1                      0.6570  0.15909   0.6634
Prevalence              0.3755  0.24979   0.3747
Detection Rate          0.3397  0.02331   0.2231
Detection Prevalence    0.6586  0.04330   0.2981
Balanced Accuracy       0.6970  0.53335   0.7379
          Reference
Prediction  -1   0   1
        -1 400  96  24
        0   42 146  51
        1    9  58 375

Overall Statistics

               Accuracy : 0.7669
                 95% CI : (0.7419, 0.7905)
    No Information Rate : 0.3755
    P-Value [Acc > NIR] : < 2.2e-16

                  Kappa : 0.6412
 Mcnemar's Test P-Value : 2.996e-06
Statistics by Class:
                     Class: -1 Class: 0 Class: 1
Sensitivity             0.8869   0.4867   0.8333
Specificity             0.8400   0.8968   0.9108
Pos Pred Value          0.7692   0.6109   0.8484
Neg Pred Value          0.9251   0.8399   0.9012
Precision               0.7692   0.6109   0.8484
Recall                  0.8869   0.4867   0.8333
F1                      0.8239   0.5417   0.8408
Prevalence              0.3755   0.2498   0.3747
Detection Rate          0.3331   0.1216   0.3122
Detection Prevalence    0.4330   0.1990   0.3680
Balanced Accuracy       0.8635   0.6917   0.8721

          Reference
Prediction  -1   0   1
        -1 410 166  61
        0    9  66   9
        1   32  68 380
Overall Statistics

               Accuracy : 0.7127
                 95% CI : (0.6862, 0.7382)
    No Information Rate : 0.3755
    P-Value [Acc > NIR] : < 2.2e-16

                  Kappa : 0.5466
 Mcnemar's Test P-Value : < 2.2e-16
Statistics by Class:
                     Class: -1 Class: 0 Class: 1
Sensitivity             0.9091  0.22000   0.8444
Specificity             0.6973  0.98002   0.8668
Pos Pred Value          0.6436  0.78571   0.7917
Neg Pred Value          0.9273  0.79051   0.9029
Precision               0.6436  0.78571   0.7917
Recall                  0.9091  0.22000   0.8444
F1                      0.7537  0.34375   0.8172
Prevalence              0.3755  0.24979   0.3747
Detection Rate          0.3414  0.05495   0.3164
Detection Prevalence    0.5304  0.06994   0.3997
Balanced Accuracy       0.8032  0.60001   0.8556
ENSEMBLE SUMMARY

       n-ENSEMBLE COVERAGE n-ENSEMBLE RECALL
n >= 1               1.00             0.72
n >= 2               0.96             0.73
```

```
n >= 3               0.61             0.83


ALGORITHM PERFORMANCE

    SVM_PRECISION        SVM_RECALL        SVM_FSCORE FORESTS_PRECISION
        0.7433333         0.7366667         0.7366667         0.7400000
   FORESTS_RECALL    FORESTS_FSCORE    TREE_PRECISION       TREE_RECALL
        0.6566667         0.6333333         0.6033333         0.5300000
      TREE_FSCORE
        0.4933333
<<DocumentTermMatrix (documents: 4000, terms: 10981)>>
Non-/sparse entries: 79324/43844676
Sparsity           : 100%
Maximal term length: 20
Weighting          : term frequency - inverse document frequency (normalized) (tf-idf)
Training models, phase stems: 1044.385 sec elapsed

          Reference
Prediction  -1   0   1
        -1 421 207 133
        0   18  48  58
        1   12  45 259
Overall Statistics

               Accuracy : 0.6062
                 95% CI : (0.5779, 0.6339)
    No Information Rate : 0.3755
    P-Value [Acc > NIR] : < 2.2e-16

                  Kappa : 0.3824
 Mcnemar's Test P-Value : < 2.2e-16
Statistics by Class:
                     Class: -1 Class: 0 Class: 1
Sensitivity             0.9335  0.16000   0.5756
Specificity             0.5467  0.91565   0.9241
Pos Pred Value          0.5532  0.38710   0.8196
Neg Pred Value          0.9318  0.76602   0.7842
Precision               0.5532  0.38710   0.8196
Recall                  0.9335  0.16000   0.5756
F1                      0.6947  0.22642   0.6762
Prevalence              0.3755  0.24979   0.3747
Detection Rate          0.3505  0.03997   0.2157
Detection Prevalence    0.6336  0.10325   0.2631
Balanced Accuracy       0.7401  0.53782   0.7498
          Reference
Prediction  -1   0   1
        -1 391 108  26
        0   41 127  56
        1   19  65 368

Overall Statistics

               Accuracy : 0.7377
                 95% CI : (0.7119, 0.7624)
    No Information Rate : 0.3755
    P-Value [Acc > NIR] : < 2.2e-16

                  Kappa : 0.5954
 Mcnemar's Test P-Value : 5.532e-07
Statistics by Class:
                     Class: -1 Class: 0 Class: 1
Sensitivity             0.8670   0.4233   0.8178
Specificity             0.8213   0.8923   0.8881
Pos Pred Value          0.7448   0.5670   0.8142
Neg Pred Value          0.9112   0.8229   0.8905
Precision               0.7448   0.5670   0.8142
Recall                  0.8670   0.4233   0.8178
F1                      0.8012   0.4847   0.8160
```

```
Prevalence              0.3755   0.2498   0.3747
Detection Rate          0.3256   0.1057   0.3064
Detection Prevalence    0.4371   0.1865   0.3764
Balanced Accuracy       0.8441   0.6578   0.8530
          Reference
Prediction  -1   0   1
        -1 411 169  63
        0    7  60  11
        1   33  71 376
Overall Statistics

               Accuracy : 0.7052
                 95% CI : (0.6786, 0.7309)
    No Information Rate : 0.3755
    P-Value [Acc > NIR] : < 2.2e-16

                  Kappa : 0.5343
 Mcnemar's Test P-Value : < 2.2e-16

Statistics by Class:

                     Class: -1 Class: 0 Class: 1
Sensitivity             0.9113  0.20000   0.8356
Specificity             0.6907  0.98002   0.8615
Pos Pred Value          0.6392  0.76923   0.7833
Neg Pred Value          0.9283  0.78629   0.8974
Precision               0.6392  0.76923   0.7833
Recall                  0.9113  0.20000   0.8356
F1                      0.7514  0.31746   0.8086
Prevalence              0.3755  0.24979   0.3747
Detection Rate          0.3422  0.04996   0.3131
Detection Prevalence    0.5354  0.06495   0.3997
Balanced Accuracy       0.8010  0.59001   0.8485
ENSEMBLE SUMMARY

       n-ENSEMBLE COVERAGE n-ENSEMBLE RECALL
n >= 1                1.00              0.72
n >= 2                0.96              0.73
n >= 3                0.62              0.82

ALGORITHM PERFORMANCE
    SVM_PRECISION        SVM_RECALL        SVM_FSCORE FORESTS_PRECISION
        0.7066667         0.7033333         0.6966667         0.7300000
   FORESTS_RECALL    FORESTS_FSCORE    TREE_PRECISION       TREE_RECALL
        0.6500000         0.6266667         0.5866667         0.5566667
      TREE_FSCORE
        0.5333333
<<DocumentTermMatrix (documents: 4000, terms: 10684)>>
Non-/sparse entries: 80344/42655656
Sparsity           : 100%
Maximal term length: 24
Weighting          : term frequency - inverse document frequency (normalized) (tf-idf)
Training models, phase lemmas: 788.05 sec elapsed
          Reference
Prediction  -1   0   1
        -1 408 214 130
        0   16  33  21
        1   27  53 299
Overall Statistics

               Accuracy : 0.6162
                 95% CI : (0.588, 0.6438)
    No Information Rate : 0.3755
    P-Value [Acc > NIR] : < 2.2e-16

                  Kappa : 0.3927
 Mcnemar's Test P-Value : < 2.2e-16
Statistics by Class:
                     Class: -1 Class: 0 Class: 1
```

```
Sensitivity             0.9047  0.11000   0.6644
Specificity             0.5413  0.95893   0.8935
Pos Pred Value          0.5426  0.47143   0.7889
Neg Pred Value          0.9042  0.76393   0.8163
Precision               0.5426  0.47143   0.7889
Recall                  0.9047  0.11000   0.6644
F1                      0.6783  0.17838   0.7214
Prevalence              0.3755  0.24979   0.3747
Detection Rate          0.3397  0.02748   0.2490
Detection Prevalence    0.6261  0.05828   0.3156
Balanced Accuracy       0.7230  0.53447   0.7790
```

```
          Reference
Prediction  -1   0   1
        -1 392 105  25
        0   46 130  64
        1   13  65 361
Overall Statistics

               Accuracy : 0.7352
                 95% CI : (0.7093, 0.76)
    No Information Rate : 0.3755
    P-Value [Acc > NIR] : < 2.2e-16

                  Kappa : 0.5926
 Mcnemar's Test P-Value : 6.329e-06

Statistics by Class:

                     Class: -1 Class: 0 Class: 1
Sensitivity             0.8692   0.4333   0.8022
Specificity             0.8267   0.8779   0.8961
Pos Pred Value          0.7510   0.5417   0.8223
Neg Pred Value          0.9131   0.8231   0.8832
Precision               0.7510   0.5417   0.8223
Recall                  0.8692   0.4333   0.8022
F1                      0.8058   0.4815   0.8121
Prevalence              0.3755   0.2498   0.3747
Detection Rate          0.3264   0.1082   0.3006
Detection Prevalence    0.4346   0.1998   0.3655
Balanced Accuracy       0.8479   0.6556   0.8492

          Reference
Prediction  -1   0   1
        -1 403 163  54
        0    5  53  13
        1   43  84 383
Overall Statistics

               Accuracy : 0.6986
                 95% CI : (0.6718, 0.7244)
    No Information Rate : 0.3755
    P-Value [Acc > NIR] : < 2.2e-16

                  Kappa : 0.5233
 Mcnemar's Test P-Value : < 2.2e-16

Statistics by Class:

                     Class: -1 Class: 0 Class: 1
Sensitivity             0.8936  0.17667   0.8511
Specificity             0.7107  0.98002   0.8309
Pos Pred Value          0.6500  0.74648   0.7510
Neg Pred Value          0.9174  0.78142   0.9030
Precision               0.6500  0.74648   0.7510
Recall                  0.8936  0.17667   0.8511
```

```
F1                     0.7526  0.28571   0.7979
Prevalence             0.3755  0.24979   0.3747
Detection Rate         0.3356  0.04413   0.3189
Detection Prevalence   0.5162  0.05912   0.4246
Balanced Accuracy      0.8021  0.57834   0.8410
ENSEMBLE SUMMARY

       n-ENSEMBLE COVERAGE n-ENSEMBLE RECALL
n >= 1                1.00              0.72
n >= 2                0.96              0.73
n >= 3                0.62              0.83


ALGORITHM PERFORMANCE

    SVM_PRECISION        SVM_RECALL        SVM_FSCORE FORESTS_PRECISION
        0.7033333         0.7000000         0.7000000         0.7166667
   FORESTS_RECALL    FORESTS_FSCORE    TREE_PRECISION       TREE_RECALL
        0.6400000         0.6133333         0.6000000         0.5566667
      TREE_FSCORE
        0.5266667
```

Deep learning model summary (one data sample)

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
embedding_8 (Embedding)      (None, 344, 128)          2014336
_________________________________________________________________
flatten_4 (Flatten)          (None, 44032)             0
_________________________________________________________________
dense_10 (Dense)             (None, 3)                 132099
=================================================================
Total params: 2,146,435
Trainable params: 2,146,435
Non-trainable params: 0
_________________________________________________________________
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
embedding_9 (Embedding)      (None, None, 128)         2014336
_________________________________________________________________
lstm_5 (LSTM)                (None, 128)               131584
_________________________________________________________________
dense_11 (Dense)             (None, 128)               16512
_________________________________________________________________
dense_12 (Dense)             (None, 3)                 387
=================================================================
Total params: 2,162,819
Trainable params: 2,162,819
Non-trainable params: 0
_________________________________________________________________
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
embedding_10 (Embedding)     (None, None, 128)         2014336
_________________________________________________________________
bidirectional_3 (Bidirecti (None, 128)               98816
_________________________________________________________________
dropout_3 (Dropout)          (None, 128)               0
_________________________________________________________________
dense_13 (Dense)             (None, 3)                 387
=================================================================
Total params: 2,113,539
Trainable params: 2,113,539
Non-trainable params: 0
```

Results of Deep learning models

1st model – Embedded NN

2nd model – LSTM NN

3rd model – BLSTM NN

```
[1] "Deep learning model: lemmas1"
Confusion Matrix and Statistics

     Reference
Prediction  -1   0   1
    -1 387 104  18
    0   42 134  46
    1   22  61 386

Overall Statistics

       Accuracy : 0.7558
        95% CI : (0.7305, 0.7799)
  No Information Rate : 0.3758
  P-Value [Acc > NIR] : < 2.2e-16

         Kappa : 0.6232
 Mcnemar's Test P-Value : 2.43e-06

Statistics by Class:

          Class: -1 Class: 0 Class: 1
Sensitivity             0.8581   0.4482   0.8578
Specificity             0.8371   0.9023   0.8893
Pos Pred Value          0.7603   0.6036   0.8230
Neg Pred Value          0.9074   0.8313   0.9124
Precision               0.7603   0.6036   0.8230
Recall                  0.8581   0.4482   0.8578
F1                      0.8062   0.5144   0.8400
Prevalence              0.3758   0.2492   0.3750
Detection Rate          0.3225   0.1117   0.3217
Detection Prevalence    0.4242   0.1850   0.3908
Balanced Accuracy       0.8476   0.6752   0.8736
[1] "Deep learning model: lemmas2"
Confusion Matrix and Statistics

     Reference
Prediction  -1   0   1
    -1 402 190  42
    0   14  37  23
    1   35  72 385

Overall Statistics

       Accuracy : 0.6867
        95% CI : (0.6596, 0.7128)
  No Information Rate : 0.3758
  P-Value [Acc > NIR] : < 2.2e-16

         Kappa : 0.5045
 Mcnemar's Test P-Value : < 2.2e-16

Statistics by Class:

          Class: -1 Class: 0 Class: 1
```

```
Sensitivity             0.8914  0.12375   0.8556
Specificity             0.6903  0.95893   0.8573
Pos Pred Value          0.6341  0.50000   0.7825
Neg Pred Value          0.9134  0.76732   0.9082
Precision               0.6341  0.50000   0.7825
Recall                  0.8914  0.12375   0.8556
F1                      0.7410  0.19839   0.8174
Prevalence              0.3758  0.24917   0.3750
Detection Rate          0.3350  0.03083   0.3208
Detection Prevalence    0.5283  0.06167   0.4100
Balanced Accuracy       0.7908  0.54134   0.8564
[1] "Deep learning model: lemmas3"
Confusion Matrix and Statistics

     Reference
Prediction  -1   0   1
    -1 376  98  12
    0   63 154  86
    1   12  47 352

Overall Statistics

       Accuracy : 0.735
         95% CI : (0.7091, 0.7598)
  No Information Rate : 0.3758
  P-Value [Acc > NIR] : < 2.2e-16

         Kappa : 0.5963
 Mcnemar's Test P-Value : 0.0002676

Statistics by Class:

          Class: -1 Class: 0 Class: 1
Sensitivity             0.8337   0.5151   0.7822
Specificity             0.8531   0.8346   0.9213
Pos Pred Value          0.7737   0.5083   0.8564
Neg Pred Value          0.8950   0.8384   0.8758
Precision               0.7737   0.5083   0.8564
Recall                  0.8337   0.5151   0.7822
F1                      0.8026   0.5116   0.8177
Prevalence              0.3758   0.2492   0.3750
Detection Rate          0.3133   0.1283   0.2933
Detection Prevalence    0.4050   0.2525   0.3425
Balanced Accuracy       0.8434   0.6748   0.8518
[1] "Deep learning model: textlatin1"
Confusion Matrix and Statistics

     Reference
Prediction  -1   0   1
    -1 394 104  19
    0   31 131  37
    1   26  64 394

Overall Statistics

       Accuracy : 0.7658
         95% CI : (0.7408, 0.7895)
  No Information Rate : 0.3758
  P-Value [Acc > NIR] : < 2.2e-16

         Kappa : 0.6372
 Mcnemar's Test P-Value : 2.371e-10

Statistics by Class:

          Class: -1 Class: 0 Class: 1
Sensitivity             0.8736   0.4381   0.8756
```

```
Specificity             0.8358   0.9245   0.8800
Pos Pred Value          0.7621   0.6583   0.8140
Neg Pred Value          0.9165   0.8322   0.9218
Precision               0.7621   0.6583   0.8140
Recall                  0.8736   0.4381   0.8756
F1                      0.8140   0.5261   0.8437
Prevalence              0.3758   0.2492   0.3750
Detection Rate          0.3283   0.1092   0.3283
Detection Prevalence    0.4308   0.1658   0.4033
Balanced Accuracy       0.8547   0.6813   0.8778
[1] "Deep learning model: textlatin2"
Confusion Matrix and Statistics

     Reference
Prediction  -1   0   1
    -1 390 189  48
    0   31  47  17
    1   30  63 385

Overall Statistics

       Accuracy : 0.685
         95% CI : (0.6579, 0.7112)
  No Information Rate : 0.3758
  P-Value [Acc > NIR] : < 2.2e-16

         Kappa : 0.5036
 Mcnemar's Test P-Value : < 2.2e-16

Statistics by Class:

          Class: -1 Class: 0 Class: 1
Sensitivity             0.8647  0.15719   0.8556
Specificity             0.6836  0.94673   0.8760
Pos Pred Value          0.6220  0.49474   0.8054
Neg Pred Value          0.8935  0.77195   0.9100
Precision               0.6220  0.49474   0.8054
Recall                  0.8647  0.15719   0.8556
F1                      0.7236  0.23858   0.8297
Prevalence              0.3758  0.24917   0.3750
Detection Rate          0.3250  0.03917   0.3208
Detection Prevalence    0.5225  0.07917   0.3983
Balanced Accuracy       0.7742  0.55196   0.8658
[1] "Deep learning model: textlatin3"
Confusion Matrix and Statistics

     Reference
Prediction  -1   0   1
    -1 376  96  19
    0   63 160  80
    1   12  43 351

Overall Statistics

       Accuracy : 0.7392
         95% CI : (0.7133, 0.7638)
  No Information Rate : 0.3758
  P-Value [Acc > NIR] : < 2.2e-16

         Kappa : 0.6026
 Mcnemar's Test P-Value : 0.0002094

Statistics by Class:

          Class: -1 Class: 0 Class: 1
Sensitivity             0.8337   0.5351   0.7800
Specificity             0.8465   0.8413   0.9267
```

```
Pos Pred Value          0.7658   0.5281   0.8645
Neg Pred Value          0.8942   0.8450   0.8753
Precision               0.7658   0.5281   0.8645
Recall                  0.8337   0.5351   0.7800
F1                      0.7983   0.5316   0.8201
Prevalence              0.3758   0.2492   0.3750
Detection Rate          0.3133   0.1333   0.2925
Detection Prevalence    0.4092   0.2525   0.3383
Balanced Accuracy       0.8401   0.6882   0.8533
[1] "Deep learning model: textstop1"
Confusion Matrix and Statistics

     Reference
Prediction  -1   0   1
    -1 387 103  18
    0   44 129  47
    1   20  67 385

Overall Statistics

       Accuracy : 0.7508
         95% CI : (0.7253, 0.7751)
  No Information Rate : 0.3758
  P-Value [Acc > NIR] : < 2.2e-16

         Kappa : 0.6153
 Mcnemar's Test P-Value : 5.108e-06

Statistics by Class:

           Class: -1 Class: 0 Class: 1
Sensitivity             0.8581   0.4314   0.8556
Specificity             0.8385   0.8990   0.8840
Pos Pred Value          0.7618   0.5864   0.8157
Neg Pred Value          0.9075   0.8265   0.9107
Precision               0.7618   0.5864   0.8157
Recall                  0.8581   0.4314   0.8556
F1                      0.8071   0.4971   0.8351
Prevalence              0.3758   0.2492   0.3750
Detection Rate          0.3225   0.1075   0.3208
Detection Prevalence    0.4233   0.1833   0.3933
Balanced Accuracy       0.8483   0.6652   0.8698
[1] "Deep learning model: textstop2"
Confusion Matrix and Statistics

     Reference
Prediction  -1   0   1
    -1 323 109  13
    0  113 144  91
    1   15  46 346

Overall Statistics

       Accuracy : 0.6775
         95% CI : (0.6502, 0.7039)
  No Information Rate : 0.3758
  P-Value [Acc > NIR] : < 2e-16

         Kappa : 0.5122
 Mcnemar's Test P-Value : 0.00182

Statistics by Class:

           Class: -1 Class: 0 Class: 1
Sensitivity             0.7162   0.4816   0.7689
Specificity             0.8371   0.7736   0.9187
Pos Pred Value          0.7258   0.4138   0.8501
```

```
Neg Pred Value          0.8305   0.8181   0.8689
Precision               0.7258   0.4138   0.8501
Recall                  0.7162   0.4816   0.7689
F1                      0.7210   0.4451   0.8075
Prevalence              0.3758   0.2492   0.3750
Detection Rate          0.2692   0.1200   0.2883
Detection Prevalence    0.3708   0.2900   0.3392
Balanced Accuracy       0.7767   0.6276   0.8438
[1] "Deep learning model: textstop3"
Confusion Matrix and Statistics

     Reference
Prediction  -1   0   1
    -1 359  69  10
    0   79 173  66
    1   13  57 374

Overall Statistics

       Accuracy : 0.755
         95% CI : (0.7296, 0.7791)
  No Information Rate : 0.3758
  P-Value [Acc > NIR] : <2e-16

         Kappa : 0.6277
 Mcnemar's Test P-Value : 0.6313

Statistics by Class:

          Class: -1 Class: 0 Class: 1
Sensitivity             0.7960   0.5786   0.8311
Specificity             0.8945   0.8391   0.9067
Pos Pred Value          0.8196   0.5440   0.8423
Neg Pred Value          0.8793   0.8571   0.8995
Precision               0.8196   0.5440   0.8423
Recall                  0.7960   0.5786   0.8311
F1                      0.8076   0.5608   0.8367
Prevalence              0.3758   0.2492   0.3750
Detection Rate          0.2992   0.1442   0.3117
Detection Prevalence    0.3650   0.2650   0.3700
Balanced Accuracy       0.8453   0.7088   0.8689
[1] "Deep learning model: stems1"
Confusion Matrix and Statistics

     Reference
Prediction  -1   0   1
    -1 388  97  23
    0   40 140  43
    1   23  62 384

Overall Statistics

       Accuracy : 0.76
         95% CI : (0.7348, 0.7839)
  No Information Rate : 0.3758
  P-Value [Acc > NIR] : < 2.2e-16

         Kappa : 0.6296
 Mcnemar's Test P-Value : 5.467e-06

Statistics by Class:

          Class: -1 Class: 0 Class: 1
Sensitivity             0.8603   0.4682   0.8533
Specificity             0.8398   0.9079   0.8867
Pos Pred Value          0.7638   0.6278   0.8188
Neg Pred Value          0.9090   0.8373   0.9097
```

```
Precision               0.7638   0.6278   0.8188
Recall                  0.8603   0.4682   0.8533
F1                      0.8092   0.5364   0.8357
Prevalence              0.3758   0.2492   0.3750
Detection Rate          0.3233   0.1167   0.3200
Detection Prevalence    0.4233   0.1858   0.3908
Balanced Accuracy       0.8500   0.6881   0.8700
[1] "Deep learning model: stems2"

     Reference
Prediction  -1   0   1
    -1 451 299 450
    0    0   0   0
    1    0   0   0

Overall Statistics

       Accuracy : 0.3758
         95% CI : (0.3483, 0.4039)
  No Information Rate : 0.3758
  P-Value [Acc > NIR] : 0.5109

        Kappa : 0
 Mcnemar's Test P-Value : NA

Statistics by Class:

          Class: -1 Class: 0 Class: 1
Sensitivity             1.0000   0.0000    0.000
Specificity             0.0000   1.0000    1.000
Pos Pred Value          0.3758      NaN      NaN
Neg Pred Value             NaN   0.7508    0.625
Precision               0.3758       NA       NA
Recall                  1.0000   0.0000    0.000
F1                      0.5463       NA       NA
Prevalence              0.3758   0.2492    0.375
Detection Rate          0.3758   0.0000    0.000
Detection Prevalence    1.0000   0.0000    0.000
Balanced Accuracy       0.5000   0.5000    0.500
[1] "Deep learning model: stems3"
Confusion Matrix and Statistics

     Reference
Prediction  -1   0   1
    -1 364  90  15
    0   78 164  79
    1    9  45 356

Overall Statistics

       Accuracy : 0.7367
         95% CI : (0.7108, 0.7614)
  No Information Rate : 0.3758
  P-Value [Acc > NIR] : < 2.2e-16

        Kappa : 0.6
 Mcnemar's Test P-Value : 0.008565

Statistics by Class:

          Class: -1 Class: 0 Class: 1
Sensitivity             0.8071   0.5485   0.7911
Specificity             0.8598   0.8257   0.9280
Pos Pred Value          0.7761   0.5109   0.8683
Neg Pred Value          0.8810   0.8464   0.8810
Precision               0.7761   0.5109   0.8683
Recall                  0.8071   0.5485   0.7911
```

```
F1                     0.7913   0.5290   0.8279
Prevalence             0.3758   0.2492   0.3750
Detection Rate         0.3033   0.1367   0.2967
Detection Prevalence   0.3908   0.2675   0.3417
Balanced Accuracy      0.8335   0.6871   0.8596
```

Source code

BitBucket repository:

https://bitbucket.org/KestutisD/pr00m132_final_project/

Structure of the project directory:

```
├── PR00M132_Final_Project
│   ├── 1_Transform
│   │   ├── connection.R
│   │   ├── Data
│   │   ├── server.R
│   │   └── transform.R
│   ├── 2_TF_model
│   │   ├── final_model
│   │   │   ├── saved_model.pb
│   │   │   └── variables
│   │   │       ├── variables.data-00000-of-00001
│   │   │       └── variables.index
│   │   └── server.R
│   └── 3_Analytical_model
│       ├── AM.R
│       ├── server.R
│       └── tokenizer
├── Notebooks
│   ├── 1_Download_reviews_from_evertink_lt.Rmd
│   ├── 2_Store_reviews_in_sql_database.Rmd
│   ├── 3_Integration_with_semantika_lt_api.Rmd
│   ├── 4_Prepare_training_and_test_sets.Rmd
│   ├── 5_Tidy_text_notebook.Rmd
│   ├── 6_DT_RF_SVM_models.Rmd
│   └── 7_Deep_learning_models.Rmd
└── README.md
```