Semantic Analysis of Soccer News for Automatic Game Event Classification

1st Aanund Jupskås Nordskog
Simula
University of Oslo
Oslo, Norway
aanundjn@ifi.uio.no

2nd Pål Halvorsen
SimulaMet
Oslo Metropolitan University
Oslo, Norway
paalh@simula.no

2nd Steven Hicks
SimulaMet
Oslo Metropolitan University
Oslo, Norway
steven@simula.no

3rd Håkon K. Stensland
Simula
Oslo, Norway
haakons@simula.no

3rd Hugo L. Hammer
Oslo Metropolitan University
Oslo, Norway
hugoh@oslomet.no

3rd Dag Johansen
University Tromso
Tromso, Norway
dag.johansen@uit.no

3rd Michael A. Riegler
SimulaMet
Oslo, Norway
michael@simula.no

***Abstract*—We are today overwhelmed with information, of which an important part is news. Sports news, in particular, has become very popular, where soccer makes up a big part of this coverage. For sports fans, it can be a time consuming and tedious to keep up with the news that they really care about. In this paper, we present different machine learning methods applied to soccer news from a Norwegian newspaper and a TV station's news site to summarize the content in a short and digestible manner. We present a system to collect, index, label, analyze, and present the collected news articles based on the content. We perform a thorough comparison between deep learning and traditional machine learning algorithms on text classification. Furthermore, we present a dataset of soccer news which was collected from two different Norwegian news sites and shared online.**

***Index Terms*—News, Soccer, Sport, Machine Learning, Natural Language Processing, Text Document Classification.**

I. Introduction

The world wide web is nearly an endless source of information, of which a lot comes in the form of text. From 2015 to 2018, the total number of websites have doubled, reaching a total number of 1.63 billion different websites in 2018 [1]. With this amount of data available, the need to categorize, index and label information is more critical than ever. Along with the increase in data, the popularity of machine learning has also increased. Based on Google search results, the search terms "machine learning" and "deep learning" have grown significantly over the past five years [2]. This rise in popularity is not without reason, as it has shown extraordinary results in numerous different use-cases. A few examples include image classification using convolutional neural networks (CNNs) [3], language translation using recurrent neural networks (RNNs) [4], and natural language processing where both RNNs and CNNs have seen much use and showed state-of-the-art performance [5].

The main purpose of this paper is to evaluate how deep learning and traditional machine learning algorithms perform on text classification. Furthermore, we explore how these methods perform when classifying paragraphs in soccer articles. The goal is to train a wide range of deep learning and traditional machine learning models in order to see how they compare against each other. The models will be compared on classification performance, training time and setup complexity. Two methods will be used to determine the classification performance of the different models. The first method will compare the performance during training, i.e., metrics will be calculated during training and analyzed. The second method uses the models created in method one and test how they perform in a real-world application on new and unseen data.

In addition to the study, this paper also presents a dataset and an application which were used to conduct the various experiments. The dataset consists of soccer articles extracted from VG.no and TV2.no (two large Norwegian newspapers). Paragraphs from these articles are labeled, stored in a database and used for all experiments. The application was used to make the labeling of the paragraphs easier. Furthermore, it also displays a visual representation of the paragraphs that have been classified by the different algorithms.

Deep learning based methods have mainly been applied to sentiment analysis. In this paper, we instead focus on news classification where the performance of deep learning methods has been far less studied. Therefore, the research question for this work is: How does deep learning compare to traditional machine learning on soccer news classification when it comes to classification performance, training time, and setup complexity? Thus, the main contributions of this paper to provide an answer to the research question are:

- A thorough comparison between deep learning and traditional machine learning algorithms on text classification.
- A dataset for soccer news is collected from two newspapers and is shared online for public use.
- An application that can help extend the dataset and provides a visual representation of how different models perform on the data.

© 2019 IEEE. Personal use of this material is permitted. Permission from IEEE must be obtained for all other uses, in any current or future media, including reprinting/republishing this material for advertising or promotional purposes, creating new collective works, for resale or redistribution to servers or lists, or reuse of any copyrighted component of this work in other works. DOI: https://dx.doi.org/10.1109/CBMI.2019.8877417

II. Related Work

Over the last few years, natural language processing has seen state-of-the-art performance in many applications such as document classification and sentiment analysis [5]. Much research has been performed on different text-based datasets using different deep learning and traditional machine learning approaches. In this section, we present two datasets commonly used for text classification. Then, we discuss some previous works which used these datasets to evaluate their methods.

The Stanford Sentiment Treebank dataset (SST-1)[1] dataset is a collection of movie reviews where the main goal is to detect sentiment. There are five labels: very positive, positive, neutral, negative and very negative. One thing to note is that the training set is provided at the parse-level, meaning that it consists of phrases and sentences. The test set, however, consists only of sentences [6]. The Text REtrieval Conference question dataset (TREC)[2] is made up of a collection of questions. The goal of this dataset is to classify questions into different question types. For example, questions about a location belong to the location class. There are six labels: Abbrev., Entity, Description, Human, Location, and Numeric [7].

In 2014, Kim et al. [8] performed a comparison of different CNN models on a variety of text-based datasets (among them being SST-1 and TREC). The CNN-rand model is the only model which does not use pre-trained vectors with word2vec. The vectors are instead randomly initialized and modified during training. This model is chosen because word2vec is beyond the scope of this study and relayed to future work. Zhou et al. [9] presented a comparison between a Bi-LSTM model and the C-LSTM model evaluated on the SST-1 and TREC datasets. The Bi-LSTM is a one layered bidirectional recurrent neural network using the long short-term memory (LSTM) architecture.

Socher et al. [6] presents a support vector machine (SVM) and Naive Bayes based model used on the SST-1 dataset. They are both implemented with bag-of-words features. Zhang et al. [10] present an SVM and Naive Bayes model used on the TREC dataset for question classification. Both models are implemented with bag-of-words features and default values for the parameters (e.g., C value in SVM). The SVM model uses a linear kernel, but the same results were achieved for the polynomial, Radial Basis Function (RBF) and sigmoid kernel. Another work that needs to be mentioned is from Silva et. al. [11]. They managed to get 95% accuracy on the TREC dataset using an SVM. However, that implementation of the SVM model had highly engineered features.

As we can see from the results of the previous works (Table I), deep learning models outperform the traditional machine learning models by a good margin. The bidirectional LSTM has the highest performance closely followed by the CNN model; there is a small jump down to SVM and Naive Bayes. They perform about the same on the SST-1 dataset, but the SVM model has higher performance on the TREC dataset.

[1]http://nlp.stanford.edu/sentiment

[2]http://cogcomp.cs.illinois.edu/Data/QA/QC/

Model	SST-1 Accuracy	TREC Accuracy
Naive Bayes [6]	41.0	77.4
SVM [10]	40.7	85.8
Bi-LSTM [9]	47.8	93.0
CNN-rand [8]	45.0	91.2

TABLE I: Related work accuracy scores for the different models on the SST-1 and TREC datasets.

article_id	content_order	content	html_type	class
yvdVOJ	0	To mål av Espen Ruud - og Sarpsborg gikk på sitt fjerde strake tap	h1	Goal/Assist
yvdVOJ	1	(Sarpsborg-Odd 12) Odd-backen Espen Ruud (34) har laget fem mål på de	p	Goal/Assist

TABLE II: Example of how the data is stored in the database.

III. Data

A. Retrieving Data

To collect data from our selected news sources, we created a python program which fetches and processes raw HTML code from the web sites of VG and TV2. First, the program uses Request to fetch the HTML code from all the soccer articles on 'www.vg.no/fotball' and 'www.tv2.no/fotball'. Next, the relevant information from the HTML code is extracted using Beautiful Soup (python package for parsing HTML and XML files) and the result is stored in the database. There are four columns in the database table containing the extracted information and one containing the class label (see Table II):

- **article_id** - This is the id of the article given by TV2 and VG.
- **content_order** - This is the order of the paragraphs in the article. Zero is the headline, and the highest number is the last paragraph.
- **content** - The content in the paragraph.
- **html_type** - The type of the HTML tag used by VG and TV2. For example p for paragraph and h1 for the headline.
- **class** - the label of the paragraph.

In total, the database consists of approximately $1,000$ articles, $20,000$ paragraphs and $1,000$ labeled paragraphs from each class.

B. Labeling Data

After the data was stored in the database, we created a web app called *Arx* to label the collected data. The purpose of Arx was to help with the labeling, making the process much quicker. Arx displays the content of each article in a structured way with a labeling option for each paragraph (see Figure 1). It is also possible to fetch all paragraphs or articles containing keywords (see Figure 2). For example, one can fetch all articles or paragraphs containing the word "målscorer" (goal scorer), "overgang" (attack after a break) or "vinnermålet" (winning goal). This option is beneficial since there are a lot of soccer

Class Label	Number of Samples
Goal/Assist	1117
Quotes	975
Transfer	887
Irrelevant	812
Ignore	663
Player Detail	340
Club Detail	315
Chances	300

TABLE III: Number of data samples per class from the dataset created for this paper.

Component	Model	Description
CPU	Intel i7-8700K [3]	Cores: 6 Clock spees: 3.7-4.7GHz Catch: 12MB
GPU	Nvidia GTX1070 Ti [4]	Memory: 8GB DDR5 Cuda cores: 2432 Boost clock: 1683 MHz
RAM	n/a	16 GB DDR4
OS	Ubuntu 18.04	n/a

TABLE IV: The computer hardware specifications of the machine used to train the machine learning algorithms.

articles that do not contain relevant paragraphs. Labeling is possible on both computers and mobile devices.

Arx was built using Django for the RESTful API server and React for the front-end web page. There are two possible HTTP request methods to the Django API; GET and PUT. React uses GET to fetch the articles and paragraphs from the database, and PUT to update a paragraph with the correct labels. There are in general only five labels, however, to make the labeling easier there are several more options to choose from than just five, e.g., club detail, player detail and chances, to name a few.

C. The Dataset

The dataset contains $5,526$ labeled data samples and is freely available online [12]. The labeling was performed by a person who is familiar with soccer. In Table III, there is an overview of how many data samples there are for each class. There are eight classes, but only five will be used because the last three classes have too few samples. The dataset is split with stratified 10-fold cross-validation during training.

IV. Experiments

The soccer articles are collected from Norwegian newspapers, and there are no libraries for pre-processing Norwegian text. As a result, there will be no pre-processing of the text in the dataset, except for removing special characters. In Table IV, there is an overview of the hardware used to train different machine learning algorithms.

The traditional machine learning algorithms Navie Bayes, support vector classification (SVC) and Linear SVC were all implemented using the python library Scikit-learn [13]. With Scikit-learn, the algorithms are imported and initiated using the relevant hyperparameters. All algorithms choose between two vectorizers to transform the paragraphs into vectors of numbers. The first method is term frequency-inverse document frequency (TFIDF), and the other method is a count vectorizer.

The deep learning models, RNN and CNN, are both created with the python library Keras. Keras [14] is a high-level neural network API written in python, that runs on top of TensorFlow [15], Theano [16] or CNTK [17]. We use the tokenizer from Keras to convert words into numbers. The input to the models is fixed, and the paragraph with the most words determine the input length. All other paragraphs are padded to match the max size.

The RNN model is a composition of the following layers: embedding layer, dropout layer, bidirectional LSTM layer, and a dense layer. To find the optimal RNN model, we run three different experiments. The first experiment will find the optimal embedding dimension for four different LSTM layers, with the other parameters set to the Keras' default values. For the second experiment, the optimal embedding dimension will be used to find the optimal number of LSTM neurons in the LSTM layer, with the other parameters set to the default value. The third experiments will use the optimal embedding dimension and the optimal number of LSTM neurons, to find the optimal dropout rate when the dropout layer is before and after the LSTM layer.

The CNN model is a composition of these seven layers: embedding layer, convolutional layer, dropout layer, pooling layer, flattening layer and two dense layers. To find the optimal CNN model four experiments will be run. The first experiment will find the optimal embedding dimension with different filter sizes, pooling sizes, and kernel sizes. The second experiment will use the optimal embedding dimension, kernel size, and pooling size to find the optimal filter size. The third experiment will use the optimal values found in experiment one and two to find the optimal number of neurons in the dense layer. The fourth experiment will find the optimal dropout rate when the optimal values are used for all parameters.

After the training experiments, each model will use the optimal parameters and train on the entire dataset. The models will then be used in the application to classify articles about soccer players. The result of the models will be compared and analyzed. The articles used in the application is collected in the period between December 2018 and February 2019, three months after the training set was made.

Figure 3 shows the user interface of the classification program. The application takes a player name and a machine learning type as input. It then searches the database for all articles where that player is mentioned. The paragraphs in these articles are run through the machine learning algorithm, and the result is presented in five different tags shown in Figure 3.

For the evaluation, each model will classify articles for five players. Two players will test the models on how well they classify the "Goal/Assist" paragraphs. The two players are "Lucas Moura" and "Marcus Rashford", i.e., both of these players scored goals and performed well in the period from December 2018 to February 2019. Two players will test the models on how well they classify "Transfer" paragraphs. The

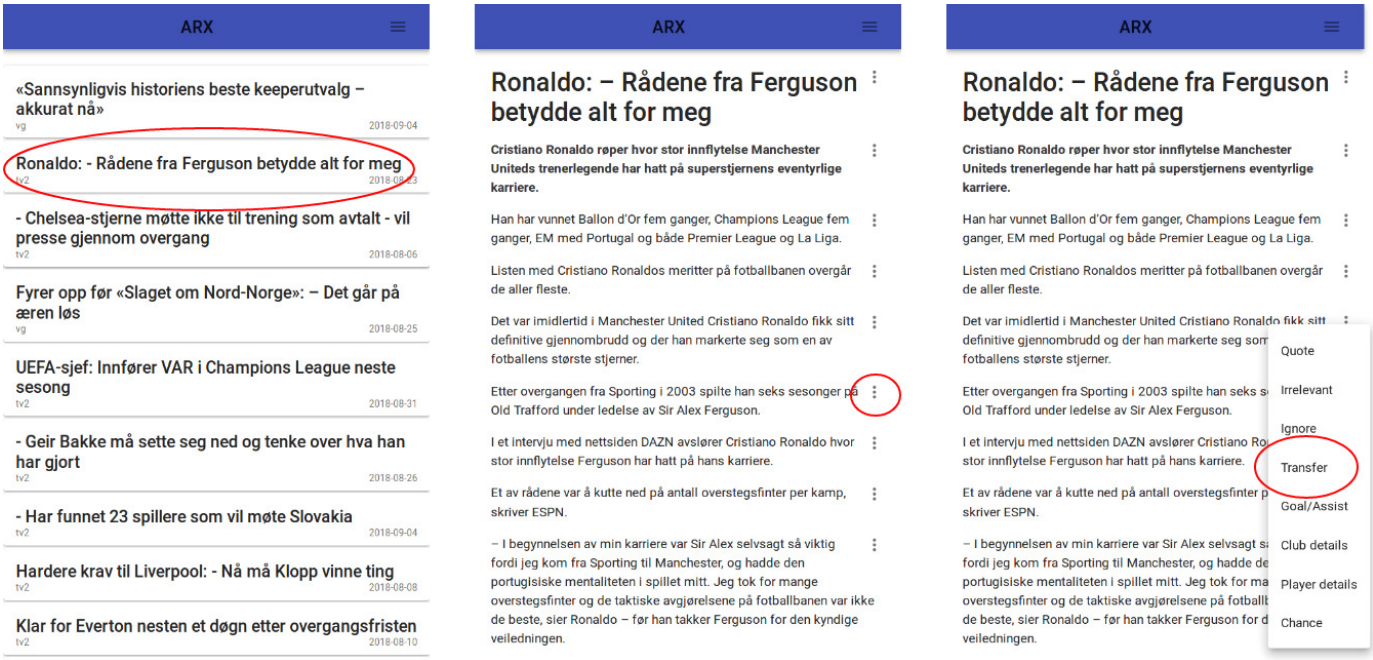

Fig. 1: Example of how the labelling is done with Arx. The first step is to click an article, the second step is to locate a paragraph, and the third step is to determine which label to give the chosen paragraph.

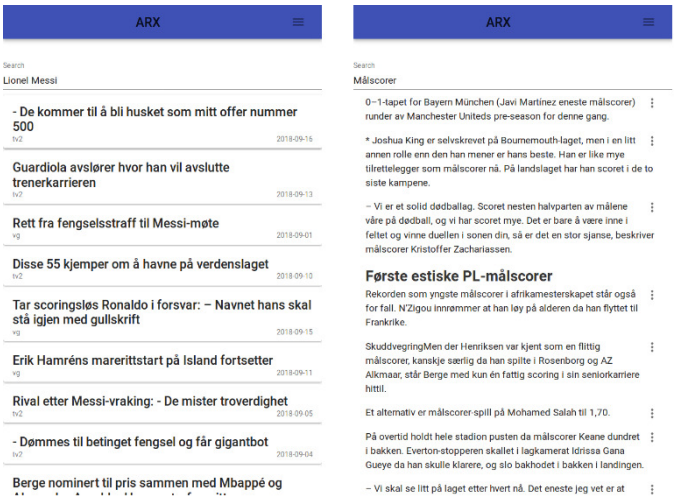

Fig. 2: Example of how to fetch articles and paragraphs with a keyword in Arx. The left figure finds all articles that contain ”Lionel Messi”. The right figure finds all paragraphs that contain the word ”Målscorer”.

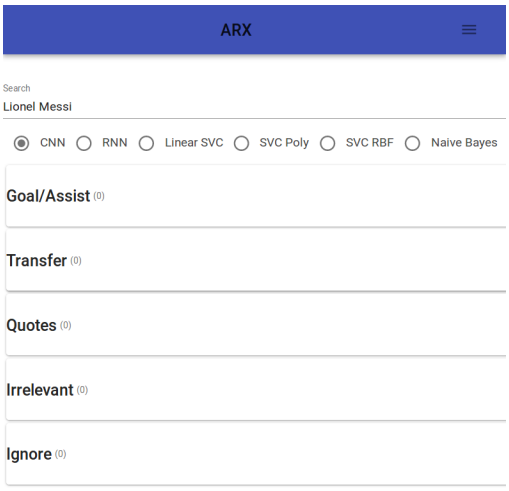

Fig. 3: An example of the classification of Lionel Messi with the CNN model. Each paragraph in articles about Lione Messi will be classified in one of the five tabs.

two players are ”Higuain” and ”Morata”. These players were sold in the 2019 January transfer window. The last player to test the models is ”Martin Ødegaard”, who performed well and was speculated in the transfer marked in the period December 2018 to February 2019.

Metric values and confusion matrix are calculated for each player on all the different models. In addition, for Lucas Moura and Morata, the paragraphs will be shown and the result will be analyzed. Five articles will be classified for Lucas Moura.

The following metrics are used to determine the performance of a classifier: Accuracy, recall, precision, f-measure, and matthews correlation coefficient (MCC). To get the result of the classifiers as general as possible, the metrics are calculated as an average over 10-fold cross-validation.

A. *Results*

The performance of Linear SVC, SVC RBF and SVC poly are almost identical during training, with an MCC of 82% and an accuracy of 85.5%. The Naive Bayes model performs 4% lower than the others. All the algorithms have the same pattern, where the f1 score of the ”Irrelevant” class has low performance, and the other classes have high performance. Especially for Naive Bayes the f1 score of the ”Irrelevant” class is 17% lower than the other algorithms, but for the other

Parameter	Metrics					f1 score for each class				
Model Name	Acc	MCC	Rec	F1	Prec	Go/As	Tr	Qu	Ir	Ig
RNN	0.887	0.859	0.887	0.885	0.892	0.907	0.884	0.982	0.740	0.912
CNN	0.879	0.849	0.879	0.876	0.882	0.900	0.879	0.979	0.732	0.888
SVM RBF	0.859	0.823	0.859	0.857	0.864	0.903	0.876	0.903	0.701	0.900
SVM Poly	0.859	0.822	0.859	0.855	0.861	0.904	0.882	0.903	0.694	0.891
Linear SVM	0.853	0.815	0.853	0.848	0.854	0.907	0.877	0.893	0.680	0.883
Naive Bayes	0.818	0.775	0.818	0.798	0.814	0.873	0.868	0.870	0.528	0.851

TABLE V: The result of the training experiments.

four classes, the performance is only 2% lower. The training time for SVC poly and SVC RBF is 2.7 seconds, that is 30 times higher than what it is for Naive Bayes and Linear SVC. And the prediction time is 0.4 ms per paragraph, and that is 25 times higher than what it is for Linear SVC and Naive Bayes. When running the optimal model for all the algorithms in the application, the results show a 20% drop in MCC for all the models. The SVC poly and SVC RBF have the best performance, with an MCC of 62%, beating Linear SVC with 2% and Naive Bayes with 6%. Then Naive Bayes classifier has a 5-7% higher f1 score on the "Transfer" class than the others. On the other hand, the f1 score of the "Ignore" class is 13-15% lower than the other models.

The performance of the RNN and CNN classifier is very similar during training. The RNN model has an MCC of 85.3% and an accuracy of 88.3%, which is 1% better than CNN. The two models have the same pattern, where the f1 score of the "Irrelevant" class is low, and the other classes f1 score is high. The training time for CNN is between 4 and 11 seconds, and for RNN it is between 7 and 15 seconds. The predictions time for CNN is 0.22 ms per paragraph, which is about half of what it is for RNN. When running the optimal model for RNN and CNN in the application, the results show a 20% drop in MCC score. The RNN classifier has the best performance, with an MCC of 64.7%, which is 2% higher than that of the CNN. The f1 score of the different classes is very similar between the two models.

V. Discussion

The performance of the algorithms during training was overall very high. There is an overview of the top results in Table V. Naive Bayes is at the low end with an MCC of 75% and RNN at the high end with an MCC of 85.3%. The MCC of the deep learning models was around 85%, while for the SVM models the MCC was around 82%. Thus, the performance is overall better for the deep learning algorithms, but not with a large margin. After training, each model was used in the application with the optimal parameters. The first thing to note from the results in the application is that all the algorithms suffered a $\sim 20\%$ drop in MCC. This drop might indicate that the models are overfitting, or that the training set is not representative of the entire domain. Since all the algorithms are affected the same, the problem most likely lies with the training set. The results from the application show that the deep learning models have a better performance than the traditional machine learning models; there is an overview of the results in table VI. As for training, an RNN has the highest MCC with 64.7%, and Naive Bayes has the lowest MCC with 56.4%. The SVM model with polynomial kernel was closest with an MCC of 62.4%. The difference in performance

Parameter	Metrics					f1 score for each class				
Model Name	Acc	MCC	Rec	F1	Prec	Go/As	Tr	Qu	Ir	Ig
RNN	0.733	0.647	0.733	0.717	0.778	0.611	0.672	0.936	0.718	0.650
CNN	0.703	0.625	0.703	0.696	0.776	0.558	0.684	0.881	0.699	0.659
SVM Poly	0.703	0.624	0.703	0.720	0.768	0.567	0.661	0.868	0.667	0.835
SVM RBF	0.697	0.618	0.697	0.711	0.767	0.563	0.650	0.872	0.663	0.809
Linear SVM	0.690	0.609	0.690	0.704	0.758	0.566	0.633	0.850	0.657	0.813
Naive Bayes	0.613	0.564	0.613	0.645	0.738	0.515	0.711	0.855	0.467	0.677

TABLE VI: The result of the application experiments.

Model Name	Expression	Total Training Time (minutes)
Naive Bayes	$4 * 80 * 0.063 * 10$	3
Linear SVM	$3 * 80 * 0.084 * 10$	3.36
SVM RBF	$4 * 80 * 2.7 * 10$	144
SVM Poly	$6 * 80 * 2.6 * 10$	207.6
RNN	$7 * 80 * 11 * 10$	1020
CNN	$27 * 80 * 8 * 10$	2880

TABLE VII: The total training time for each algorithm calculated in minutes.

is marginal, with RNN performing slightly better than SVM models. The training time can be divided into three brackets, where Naive Bayes and linear SVM have a very low training time, SVM with polynomial and RBF kernel have a high training time, and RNN and CNN have a very high training time. In Table VII, there is an overview of the approximate time for the different models. Where the time is calculated with the function 1 and where the different parameters are as follow:

- Experiments (exp) - Number of experiments that were executed.
- Parameters per experiment (ppe) - Number of values that were tested for each experiment.
- Average training time (att) - The average training time for each model.
- Number of folds in cross-validation (fcv) - Number of folds that were used during training.

$$\text{Total training time} = \text{exp} \cdot \text{ppe} \cdot \text{att} \cdot \text{fcv} \tag{1}$$

SVM with RBF and polynomial kernel have a lower training time compared to what is shown in Table VII. The implementation from Scikit-learn does not support multi-threading, so each of them only ran on one processor core. However, during training, the task was split up to use the six cores available. Therefore, the time shown in the table can roughly be divided by six. The linear SVM model has a lower training time than the other SVM models because Scikit-learn uses a different multi-class implementation. For linear SVM, Scikit-learn uses one-vs-rest, and for the other two, it uses one-vs-one, which is more computationally expensive. The total training time is much higher for the deep learning algorithms than it is for the traditional machine learning models. One reason for the high training time is that there are more parameters to optimize for the deep learning models, combined with higher training time in general. Despite the high training time compared to the other models, it is not unreasonably high and still very manageable.

A. Failure Analysis

Based on our results, we see that the "Irrelevant" class has a low f1 score compared to the other classes. Furthermore,

”Goal/Assist” paragraphs are often mistaken for ”Irrelevant” paragraphs. There is also a significant drop in performance when applying the models to the application. The ”Irrelevant” class contains many different types of paragraphs, for example, goal chances, team lineups, and other trivial information. Thus, the ”Irrelevant” class is many classes put into one which makes it difficult for the models to generalize what belongs to this class. One solution to this problem may be to change the ”Irrelevant” class. For example, one of the main issues in the application is that paragraphs about goal chances get classified to the ”Goal/Assist” class instead of to the ”Irrelevant” class. Renaming the ”Goal/Assist” class to ”Situation” class, and changing the content from being about goals and assists to goals, assists and chances would simplify the ”Irrelevant” class. Another problem in the application is that paragraphs about team line-ups get classified to the ”Ignore” class instead of the ”Irrelevant” class. Moving these paragraphs to the ”Ignore” class makes sense, maybe that is where they should have been from the start. These two measures would reduce the size of the ”Irrelevant” class without complicating the ”Goal/Assist” and ”Ignore” class.

One possible reason for the big difference between the training experiment result and the application result can be the labeling method used on the training set and the test set. There are three ways to label data samples (1) label paragraphs in random articles, (2) label paragraphs in articles that contain a keyword, for example, articles that contain a certain player name, or (3) label paragraphs from a list of paragraphs containing a keyword, for example, a list of all paragraphs that contain the word ”Målscorer”. When making the training set, all three methods were used, but the third method was used the most. On the other hand, when making the test set, only the second method was used, which means that the training set has a less diverse set of paragraphs compared to the test set.

RNN and CNN are data-hungry algorithms compared to Naive Bayes and SVM. The training set used in this research contains approximately 5,500 data samples, which is considered to be few when it comes to deep learning. However, even with a limited training set, the deep learning algorithms outperformed the traditional machine learning algorithms. Such datasets are a work in progress, meaning that they will grow over time. It is, therefore, reasonable to think that CNNs and RNNs will benefit more from this than SVM and Naive Bayes.

VI. Conclusion and Future Work

The research question raised in Section I was: ”**How does deep learning compare to traditional machine learning on text classification when it comes to classification performance, training time and setup complexity?**”. To answer this question, different machine learning and deep learning models were created and then compared against each other. Based on the research done in this paper, there is no clear answer to whether deep learning or traditional machine learning algorithms are preferred for text classification. On the one hand, the deep learning models did perform slightly better overall compared to the traditional machine learning algorithms. However, on the other hand, the training time for the deep learning models is much higher, and the setup is more complex. Moreover, if the dataset is limited as it is in this work, the traditional machine learning models are the better choice. They are easy setup, low training time and limited maintenance outweigh the small performance gain given by deep learning. However, if the dataset is growing over time and has the potential of becoming much bigger than it is now, the deep learning models are the better choice. The overhead would then be worth the potential performance gain that is given by deep learning.

For future work, we plan the following improvements. First, to get a more thorough comparison between deep learning and traditional machine learning, more algorithms can be tested, e.g., decision trees and k-nearest neighbours. Additionally, we aim to test different architectures for RNNs and CNNs. Furthermore, pre-processing of the dataset was almost non-existent in this research; only special characters were removed. Therefore, more advanced pre-processing could be tested on the dataset. One way to do this is to use the Google Translate API, and translate the dataset into English for so to apply pre-processing tools available in English.

References

[1] Total number of websites - internet live stats. [Online]. Available: http://www.internetlivestats.com/total-number-of-websites/

[2] Google trends. [Online]. Available: https://trends.google.com/trends/explore?date=today

[3] A. Krizhevsky, I. Sutskever, and G. E. Hinton, “Imagenet classification with deep convolutional neural networks,” in *Advances in neural information processing systems*, 2012, pp. 1097–1105.

[4] G. Lewis-Kraus, “The great ai awakening,” *NYT*, vol. 14, 2016.

[5] T. Young, D. Hazarika, S. Poria, and E. Cambria, “Recent trends in deep learning based natural language processing,” *ieee Computational intelligenCe magazine*, vol. 13, no. 3, pp. 55–75, 2018.

[6] R. Socher, A. Perelygin, J. Wu, J. Chuang, C. D. Manning, A. Ng, and C. Potts, “Recursive deep models for semantic compositionality over a sentiment treebank,” in *Proc. of EMNL*, 2013, pp. 1631–1642.

[7] X. Li and D. Roth, “Learning question classifiers,” in *Proc. of CL*. Association for Computational Linguistics, 2002, pp. 1–7.

[8] Y. Kim, “Convolutional neural networks for sentence classification,” *arXiv preprint arXiv:1408.5882*, 2014.

[9] C. Zhou, C. Sun, Z. Liu, and F. Lau, “A c-lstm neural network for text classification,” *arXiv preprint arXiv:1511.08630*, 2015.

[10] D. Zhang and W. S. Lee, “Question classification using support vector machines,” in *Proc. of RDIR*. ACM, 2003, pp. 26–32.

[11] J. Silva, L. Coheur, A. C. Mendes, and A. Wichert, “From symbolic to sub-symbolic information in question classification,” *Artificial Intelligence Review*, vol. 35, no. 2, pp. 137–154, 2011.

[12] A. Nordskog, “Text classification project,” original-date: 2019-05-16T06:12:33Z. [Online]. Available: https://github.com/Halflingen/Text-Classification-Project

[13] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg *et al.*, “Scikit-learn: Machine learning in python,” *Journal of machine learning research*, vol. 12, no. Oct, pp. 2825–2830, 2011.

[14] F. Chollet *et al.*, “Keras,” 2015. [Online]. Available: https://keras.io/

[15] M. Abadi, P. Barham, J. Chen, Z. Chen, A. Davis, J. Dean, M. Devin, S. Ghemawat, G. Irving, M. Isard *et al.*, “Tensorflow: a system for large-scale machine learning.” in *OSDI*, vol. 16, 2016, pp. 265–283.

[16] J. Bergstra, O. Breuleux, F. Bastien, P. Lamblin, R. Pascanu, G. Desjardins, J. Turian, D. Warde-Farley, and Y. Bengio, “Theano: A cpu and gpu math compiler in python,” in *Proc. of PIS*, vol. 1, 2010.

[17] F. Seide and A. Agarwal, “Cntk: Microsoft’s open-source deep-learning toolkit,” in *Proc. of SIGKDD*. ACM, 2016, pp. 2135–2135.