MA540 Data Mining - Homework 3

Aaron Van De Brook - 2456908
Jose Gabriel Gonzales Nunez - 2584267
Justus Renkhoff - 2584202
Ozgur Ural - 2564455

Monday 11th April, 2022

1 Abstract

The implementation and improvement of Bayesian neural networks (BNN) have been the center and main topic of several studies for the last couple of years. While the general consensus is that this type of neural network offers a solution to the problem of overfitting among several other advantages, few of these research projects have focused on the interpretation and comparison of the data obtained using BNN and other approaches. In this work, several types of neural networks used to detect fake news in articles are implemented and compared to BNN. The result obtained will be analyzed by considering factors such as accuracy, loss, interpretability, complexity, training duration, and energy efficiency to assess the effectiveness of the model. This will determine the overall impact of identifying the uncertainty for the prediction and capacity that BNN has of avoiding making overconfident results compared to conventional approaches.

2 Problem Overview

Machine and deep learning are current topics of research. New methods are constantly being developed to train models for a wide variety of scenarios. These models are highly efficient in solving complex problems [14]. A widespread problem is the publication of so called fake news. This especially affects current political events and developments [1]. In the past, fake news has caused drastic damage. For example in 2013 the Dow Jones Industrial Average dropped 150 points when AP's Twitter account was hacked. The hackers posed the false message: "Breaking: Two Explosions in the White House and Barack Obama is injured." which caused confusion and led to turbulence on the stock market [1]. To prevent incidents like this in the future, models are trained to evaluate articles and assess whether the information being presented is true or false.
In this work, several approaches to fake news detection are implemented and compared. For this purpose, standard methods for natural language processing are implemented to evaluate the articles.

Furthermore, artificial neural networks and a Bayesian network are discussed.
While simpler models, such as support vector machines, can solve these types of problems well, even better results have been obtained in the past with various artificial neural networks [8]. The goal of this work is to compare these results with a Bayesian deep learning approach and achieve similar results. The advantage of Bayesian Neural Networks is that they not only give predictions, but also calculate the uncertainty of it [2]. This is especially interesting for a fake news detection problem, since it is often difficult to judge what is fake and what is not. This fluid boundary makes it easier to interpret the results, which is why this promising approach will be compared with fundamental and state-of-the-art methods. This comparison will take into account numerous factors. These include the accuracy of the methods, interpretability, as well as training duration and energy efficiency. It will be analyzed how the procedures change when the size of the training set is varied.

3 Background and Literature

Machine learning, especially neural networks, has been used to predict, classify, and model the spread of misinformation in social media and populations more generally. Specifically, Ajao et al. used a hybrid convolutional neural network to classify Tweets as fake or true (misinformation/disinformation vs. true information) and predict the spread of specific instances of misinformation throughout a community [3]. Bayesian neural networks have also been employed in an attempt to detect misinformation without the context of fact checking resources. In particular, Zhang et al. used a Bayesian neural network to verify the veracity of a claim based its replies, Tweets and replies, respectively in this case [20].
Machine learning methodologies have also been used to determine the sentiment of social media posts and especially in those posts relating to the COVID-19 pandemic [9, 10]. Imamah and Rachman used logistic regression to model the sentiment of social media posts and classify them as positive, negative, or neutral accordingly. Similarly, Li et al. used natural language processing and machine learning approaches (logistic regression, support vector machines, neural networks, etc.) to determine stress symptoms and sentiment in social media posts [10].
Bayesian neural networks are notable for their ability to measure the uncertainty of its results in addition to an output similar to that of a typical neural network [6, 18]. For example, in [20] the uncertainty of an output of the neural network is used to determine whether the information presented in the original claim is true or false based on the replies to the original claim. Goh and Chua used the uncertainty of a model output to create an error bounds for the model's nonlinear best fit to a given problem [6].
Creating numerical embeddings from text for input to machine algorithms has been addressed in the past. Different text embedding algorithms tend to work better with some machine learning algorithms than others e.g. algorithms that dynamic length output will not work very well with algorithms that required fixed-length input. Wang et al. found that text frequency-inverse document frequency (TF-IDF) worked best with decision trees/random forests and Sentence Bidirectional Encoder Repre-

sentation from Transformers (SBERT) worked best with neural networks [19, 15].
To provide the probabilistic programming interfaces necessary to implement Bayesian neural networks and Bayesian deep neural networks, two Python packages were considered that were published by TensorFlow and PyTorch called TensorFlow Distributions[1] and TyXe[2], respectively [5, 16]. We decided to use Tensorflow Distributions instead of TyXe partially due to the relative maturity of the frameworks. Our team also opted for the Tensorflow framework because the majority of the team is familiar with Tensorflow instead of PyTorch.

4 Data Source

This project will initially focus on two different datasets, the IOST[3] fake news dataset and PHEME[4] dataset for Rumor Detection and Veracity Classification. These two datasets will serve as the initial baseline for our project. However, at later stages, and once more research is done, other datasets might be implemented to draw better results and a more accurate conclusion.
The IOST fake news dataset consists of types of articles fake and real news. Whiles the real news articles were all taken from Reuters.com, the fake news articles were obtained from multiple sources that were flagged by a fact-checking organization in the USA. While most of these articles focus on politics, some of them are also based on different topics.
The dataset is divided into two different files, a true file that contains 12,600 articles from Reuters.com and a fake file that also contains 12,600 articles from different sources. All the articles considered in this dataset were collected from the years 2016 to 2017. The information in these articles was classified in the following way: article title, text, type, and the date the article was published on. Although our team is aware of different features on this dataset that suggest bias towards fake news, we plan on using the fake news dataset just at the early stages of the project, so it does not affect our final results.
The PHEME dataset for Rumor Detection and Veracity Classification was created in 2016 and contains multiple tweets related to relevant news at the moment. These tweets were classified by the specific news they were related to and later on divided into two different folders rumors and non-rumors. The following points show the breakdown of this dataset:

- **Charlie Hebdo**: 458 rumours (22.0%) and 1,621 non-rumours (78.0%).
- **Ferguson**: 284 rumours (24.8%) and 859 non-rumours (75.2%).
- **Germanwings Crash**: 238 rumours (50.7%) and 231 non-rumours (49.3%).
- **Ottawa Shooting**: 470 rumours (52.8%) and 420 non-rumours (47.2%).
- **Sydney Siege**: 522 rumours (42.8%) and 699 non-rumours (57.2%).

[1] `https://github.com/tensorflow/probability`
[2] `https://github.com/TyXe-BDL/TyXe`
[3] `https://www.kaggle.com/clmentbisaillon/fake-and-real-news-dataset`
[4] `https://figshare.com/articles/dataset/PHEME_dataset_for_Rumour_Detection_and_Veracity_Classification/6392078`

We intend to use this dataset to train the algorithm we are going to create in this project to identify and classify specific attributes from short text entries. This dataset will allow us to compare the performance of the algorithm and the information it provides to the one acquired from the IOST fake news dataset which contains larger text entries in the form of complete articles. Even though this is a very small dataset it will be useful to quickly train and test the algorithm before using it for larger datasets.

5 Proposed Approach

Several algorithms are evaluated within an experiment. The previously mentioned data is pre-processed and divided into a training and a test data set. The algorithms will be implemented in Python within a Jupyter Notebook[5] using the libraries Tenserflow[6] and Keras[7]. The following algorithms will be considered:

- **Decision trees** are a fundamental algorithm to solve classification problems. This method is considered to provide a basis. It is not assumed that this method will give very good results, but it is interesting to consider this in order to compare the efficiency of decision trees with more complex methods.
- **Random forests** are capable of solving classification problems just like decision trees, however, they usually perform better than decision trees [13]. Random forests are also fundamental algorithms, which is why they are used for comparison with more complex methods.
- **Support Vector Machines** were developed by Cortes & Vapnik in 1995 and are still a very popular algorithm today. This method tries to find boundaries for data clusters through hyperplanes, so that new inputs can be clearly assigned to a certain cluster and thus class [11]. Various papers have shown that SVMs generally perform better than random forests. Thus, this method again increases the mathematical complexity, but also delivers correspondingly better results. Due to the popularity and good results, we include SVMs in our comparison [17].
- Keras presents a tutorial using the tensorflow-probability Python package[5] in which a **standard neural network**, a **Bayesian neural network (BNN)**, and a **probabilistic Bayesian neural network are implemented**. We use this tutorial to apply the algorithms to our preprocessed data and compare them.
- **Ensemble Models** are a method to improve the performance of machine learning algorithms with simple means. Here, several models are trained on different parts of the data. Afterwards, the output is determined from the average of the predictions. In general, this provides better results than simple classifiers, which is why we include them in the comparison.

[5]https://jupyter.org/
[6]https://www.tensorflow.org/
[7]https://keras.io/

- **Probabilistic Classifiers:** These methods output not only a prediction but also an uncertainty. This has the advantage that the predictions are better interpretable. So these methods have the same advantage as BNN's, which is why we include them in the consideration.

- **Deep Learning Methods**

  - The foundation for **artificial neural networks** goes back to the 1940s with the McCulloch-Pitts Unit [7]. However, research in this area stagnated for several decades due to the lack of computing power. For example, it was not possible at that time to practically implement the concepts from the book 'Perceptrons: an Introduction to Computational Geometry' [12]. Today, with modern computing power, neural networks are used everywhere in industry and research. Many state-of-the-art techniques are based on different types of neural networks. In the last years it has been shown that excellent results can be achieved by artificial neural networks [4]. The goal of this project is to achieve similar or better results with Bayesian Neural Networks. Therefore, artificial neural networks are taken into account for this comparison.
  - **(Probabilistic) Bayesian Neural Networks** are neural networks with posterior inference to control overfitting. With this approach everything has a has a probability distribution attached to it, including the model parameters. The aforementioned standard artificial neural networks determine a class while Bayesian Neural Networks also determine the uncertainty for the prediction [2]. This is interesting for the problem at hand, as Fake News detection is not always unambiguous. Thus, the output of probabilities makes the interpretation of the results easier. The goal is to obtain similarly precise predictions with the Bayesian Network as with conventional neural networks.

To prepare the text in our dataset for input to our algorithms, Text Frequency-Inverse Document Frequency (TF-IDF) and Sentence BERT (SBERT) will be used to generate numerical word and sentence embeddings, respectively. Other works have used TF-IDF in tandem with Decision Trees and/or Random Forests and produced the best results among the non-neural network algorithms tested and SBERT embeddings used for neural network inputs produced the best results overall [19].
In order to compare the procedures, various criteria are being considered. Typical values are the loss and the accuracy. It is to be expected that the Bayesian Network performs worse on paper than a normal neural network. Therefore, the interpretability of the results should also be considered. However, the training duration and energy efficiency are also taken into account. The change of performance of the procedures is also analyzed when the size of the training set is varied. Through a contact at the Institute for Software Systems at the Trier University of Applied Sciences it is possible to perform such a test in a laboratory environment. At this stage, the metrics that will be compared are not yet finalized. This may be adjusted depending on the focus and requirements of the final paper.
In section 6, we shared our project Gantt-chart and the individuals' contributions. We prepared, approximately, a three month plan. For the first month, we plan to mostly focus on the literature

research, defining the goals and our approach, and architectural design of our project. For the second month, we plan to mostly focus on the development of the project. For the final month, we are planning to collect and analyze the project results.

6 Preliminary Results

6.1 Preprocessing

In any machine learning task, data preprocessing is as important as model building, especially for unstructured data such as text. Text preprocessing converts the textual data to a more consistent form which makes forming model inputs and numerical embeddings easier while preserving the original meaning and relationships present within the text.

Some of the common text preprocessing steps are:

- Lower casing
- Removal of Punctuations
- Removal of URLs
- Removal of HTML tags

Out[16]:

	title	text	subject	date
0	As U.S. budget fight looms, Republicans flip t...	WASHINGTON (Reuters) - The head of a conservat...	politicsNews	December 31, 2017
1	U.S. military to accept transgender recruits o...	WASHINGTON (Reuters) - Transgender people will...	politicsNews	December 29, 2017
2	Senior U.S. Republican senator: 'Let Mr. Muell...	WASHINGTON (Reuters) - The special counsel inv...	politicsNews	December 31, 2017
3	FBI Russia probe helped by Australian diplomat...	WASHINGTON (Reuters) - Trump campaign adviser ...	politicsNews	December 30, 2017
4	Trump wants Postal Service to charge 'much mor...	SEATTLE/WASHINGTON (Reuters) - President Donal...	politicsNews	December 29, 2017

Figure 1: The Raw Data

For our project, we are using two of the text preprocessing techniques. We may consider integrating additional preprocessing techniques such as URL and HTML tag removal at the later stage of our research if needed.

6.1.1 Lower Casing

The idea is to convert the input text into the same casing format so that 'test', 'Test', and 'TEST' are treated the same way. This is more helpful for text featurization techniques like term frequency-inverse document frequency (TF-IDF) as it helps to combine the same words to ensure the correct frequency values for words in the document.

Out[20]:

	text	text_wo_punct	text_lower
0	WASHINGTON (Reuters) - The head of a conservat...	WASHINGTON Reuters The head of a conservative...	washington (reuters) - the head of a conservat...
1	WASHINGTON (Reuters) - Transgender people will...	WASHINGTON Reuters Transgender people will be...	washington (reuters) - transgender people will...
2	WASHINGTON (Reuters) - The special counsel inv...	WASHINGTON Reuters The special counsel invest...	washington (reuters) - the special counsel inv...
3	WASHINGTON (Reuters) - Trump campaign adviser ...	WASHINGTON Reuters Trump campaign adviser Geo...	washington (reuters) - trump campaign adviser ...
4	SEATTLE/WASHINGTON (Reuters) - President Donal...	SEATTLEWASHINGTON Reuters President Donald Tr...	seattle/washington (reuters) - president donal...

Figure 2: After Lower Casing the Data

6.1.2 Removal of Punctuation

It will help to treat 'test' and 'test!' in the same way. This is helpful for maintaining the overall meaning of our text while making it easier to keep an accurate frequency of words in the document/dataset while calculating numerical embeddings. Since our dataset it made entirely of English text, we use the punctuation corpus provided by the built-in Python string module[8].

Out[21]:

	text	text_wo_punct
0	WASHINGTON (Reuters) - The head of a conservat...	WASHINGTON Reuters The head of a conservative...
1	WASHINGTON (Reuters) - Transgender people will...	WASHINGTON Reuters Transgender people will be...
2	WASHINGTON (Reuters) - The special counsel inv...	WASHINGTON Reuters The special counsel invest...
3	WASHINGTON (Reuters) - Trump campaign adviser ...	WASHINGTON Reuters Trump campaign adviser Geo...
4	SEATTLE/WASHINGTON (Reuters) - President Donal...	SEATTLEWASHINGTON Reuters President Donald Tr...

Figure 3: After Removal of Punctuations of the Data

6.2 SBERT and TF-IDF

For input to our chosen algorithms the text in each data instance needs to be converted into a continuous numerical representation (word embedding/numerical embedding). To accomplish this we have chosen to use the Text Frequency-Inverse Document Frequency and Sentence Bidirectional Encoder Representations from Transformers (SBERT) algorithms to generate our numerical word and sentence embeddings, respecitvely. SBERT is an extension of Google's BERT language model[9] and produces fixed-length numerical embeddings for any given input, which makes it suitable for generating neural network inputs[15]. TF-IDF, on the other hand, produces one-to-one word embeddings with the given document i.e. for a document with 5 words, 5 word embeddings will be generated. To use the TF-IDF and SBERT algorithms we have chosen to use Sci-Kit Learn's TF-IDF vectorizer implementation[10]

[8] https://docs.python.org/3/library/string.html
[9] https://github.com/google-research/bert
[10] https://scikit-learn.org/stable/modules/feature_extraction.html#tfidf-term-weighting

and the SBERT sentence transformer python package[11].

6.3 Ensemble Models

We used the Random Forest, Gradient Boosting, and Ada Boosting ensemble methods/algorithms to establish a baseline accuracy for our more complex models (e.g. CNN(s) and BNN(s)) to beat in order to be considered useful. The Ensemble Models all deliver excellent results with both BoW and TF-IDF preprocessing. The accuracy is always above 95% depending on the train/test split. It can be assumed that these models provide such a good result, since the available data is very well separable (see section 6.6).

6.4 Hybrid Bayesian Neural Network

The Bayesian Neural Network (BNN) implemented thus far is a hybrid probabilistic network with two eight-unit deterministic layers and one probabilistic two-unit layer positioned at the end of the network. The two-unit probabilistic layer allows us to learn the mean and variance of the model's predictions. This combination of layers allows the BNN to be trained faster and produce more accurate predictions. A completely probabilistic network also rarely yields useful results relative to its complexity, so training a completely probabilistic network is not always worth the time and/or effort.
The BNN's prior model is modeled using a zero-mean, unit-variance multivariate Gaussian Normal distribution and the posterior is modeled similarly using a multivariate Gaussian distribution with learnable mean, variance, and covariances.

6.5 Completed & Work in Progress Code

The code that has been submitted alongside this document consists of the main code and folders in the GitHub repository that we have set up for this project. The most important/significant contents are as follows:

- **Example algorithms** — This folder contains a Jupyter notebook with examples of at least some of the algorithms that we plan to implement in Tensorflow
- **Word embeddings** — This folder contains example code for two numerical embedding algorithms (TF-IDF and SBERT) that will be used to transform our preprocessed text data to numerical forms.
- **preprocessing.ipynb** — This Jupyter notebook contains the preprocessing code and descriptions of the code and processes therein
- **models.py** — This file processes the data with BoW and TF-IDF. Then it computes the accuracy of different ensemble models, probabilistic models and a neural network. The Ensemble methods deliver great results. The probabilistic models and the ANN are still work in progress.

[11] `https://www.sbert.net/`

- **pca.py** — This file does a principal component analysis of the bag of words (BOW), TF-IDF, and SBERT embeddings to help explain why our simple models are getting such good performance on our test data.
- **bnn.py** — This file contains the work in progress code of our hybrid Bayesian Neural Network (BNN). At the moment it trains correctly, but does not produce very good results.
- **preprocess.py** — Contains functions to simplify data downloading and preprocessing
- **main.py** — This will act as the main file for our project where all the models are created, trained, and tested

6.6 Principal Component Analysis of Word Embeddings

Considering the length of the input text to our models (think news article length text), we were not expecting the 'simple' machine learning models to perform particularly well, however, the results from our initial tests (as mentioned above) yielded extremely good performance. In an attempt to explain our results, we performed a principal component analysis (PCA)[12] on our BOW, TF-IDF, and SBERT numerical embeddings. Due to the size of the numerical embeddings in memory (typically around 40GB or more), it is extermely difficult to do a PCA of the entire dataset. Instead, we opted to take 6 random samples from our dataset without replacement (n = 3000), generate embeddings, perform the PCA, and plot the results.

The results from the PCA can be seen in figures 4, 5, and 6. With the slight exception of the numerical embeddings generated using BOW, the PCA of our vectors shows that the 2 principal components of our numerical embeddings are almost completely linearly separable which explains, to some extent, why the 'simple' models are performing so well. This also suggests that the complexity of models such as neural networks would not be necessary for this data since there is a clearly defined linear boundary. Lastly, these results indicate that our data is most likely extremely biased and/or poor quality considering the clear boundary between the two labels we are trying to classify. To mitigate this, we are going to analyze and choose some of the alternative datasets that we had previously identified at the beginning of the project.

[12] `https://scikit-learn.org/stable/modules/decomposition.html#pca`

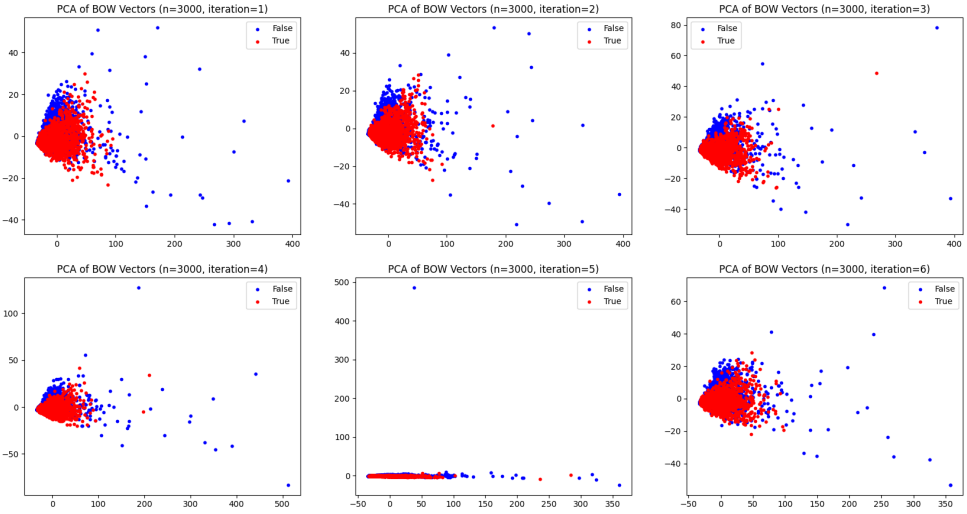

Figure 4: Principal Component Analysis of Bag of Words Vectors

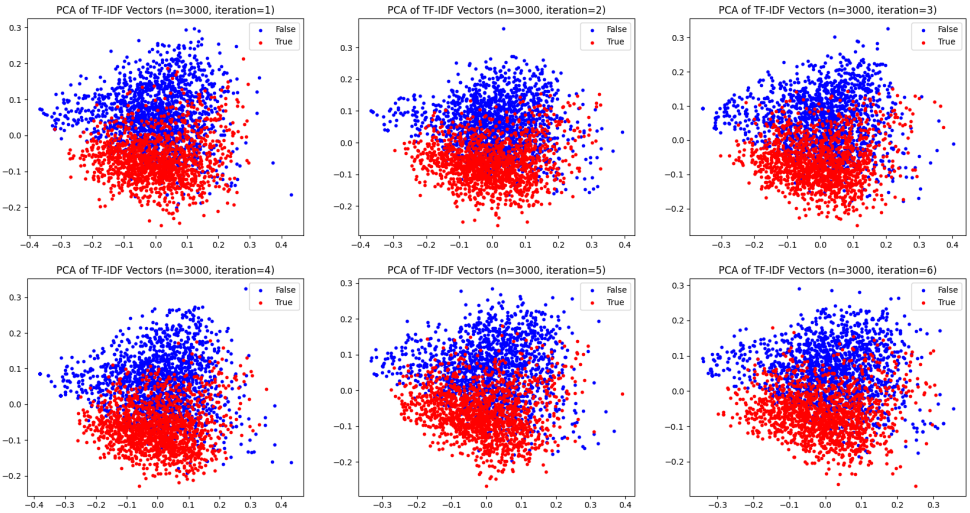

Figure 5: Principal Component Analysis of TF-IDF Vectors

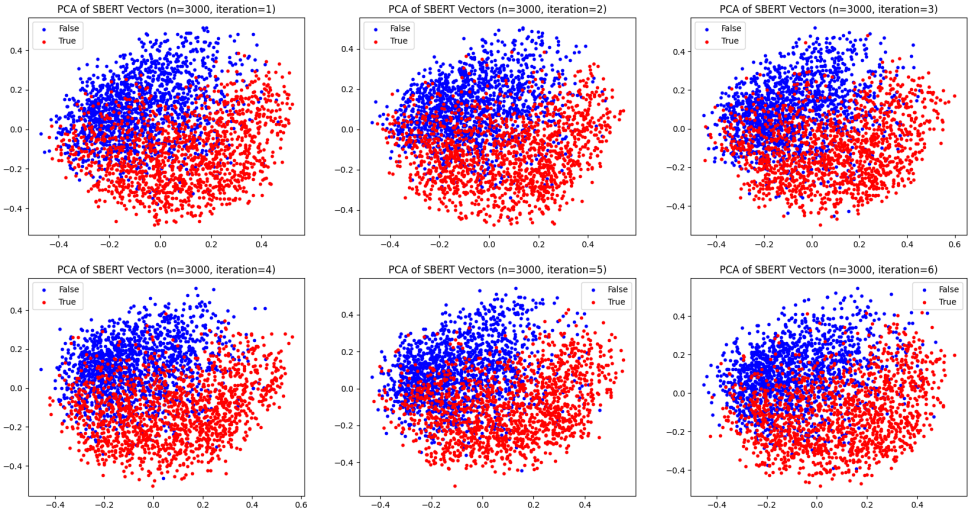

Figure 6: Principal Component Analysis of SBERT Vectors

7 Roadmap

7.1 Project Plan

Figure 4 shows a Gantt chart. This describes how we plan the flow of the project. Until mid-February, we are in the planning phase. In March, the primary implementation takes place, while we plan to run tests and document our results from the beginning of April. The results will then be summarized in a paper and possibly further tests will be performed from May and over the summer.

Between homeworks 2 and 3 we plan to start implementing algorithms and ensuring that the results from our text preprocessing and numerical embeddings are allowing us to achieve valid results. As we start getting results from the more basic models we can move on to more complex model such as convolutional neural networks and finally the Bayesian neural networks. Between homework 3 and the final submission we would ideally finalize our algorithms and results and send the final versions of our algorithms to Trier University so they can perform the energy consumption measurements on each of our algorithms.

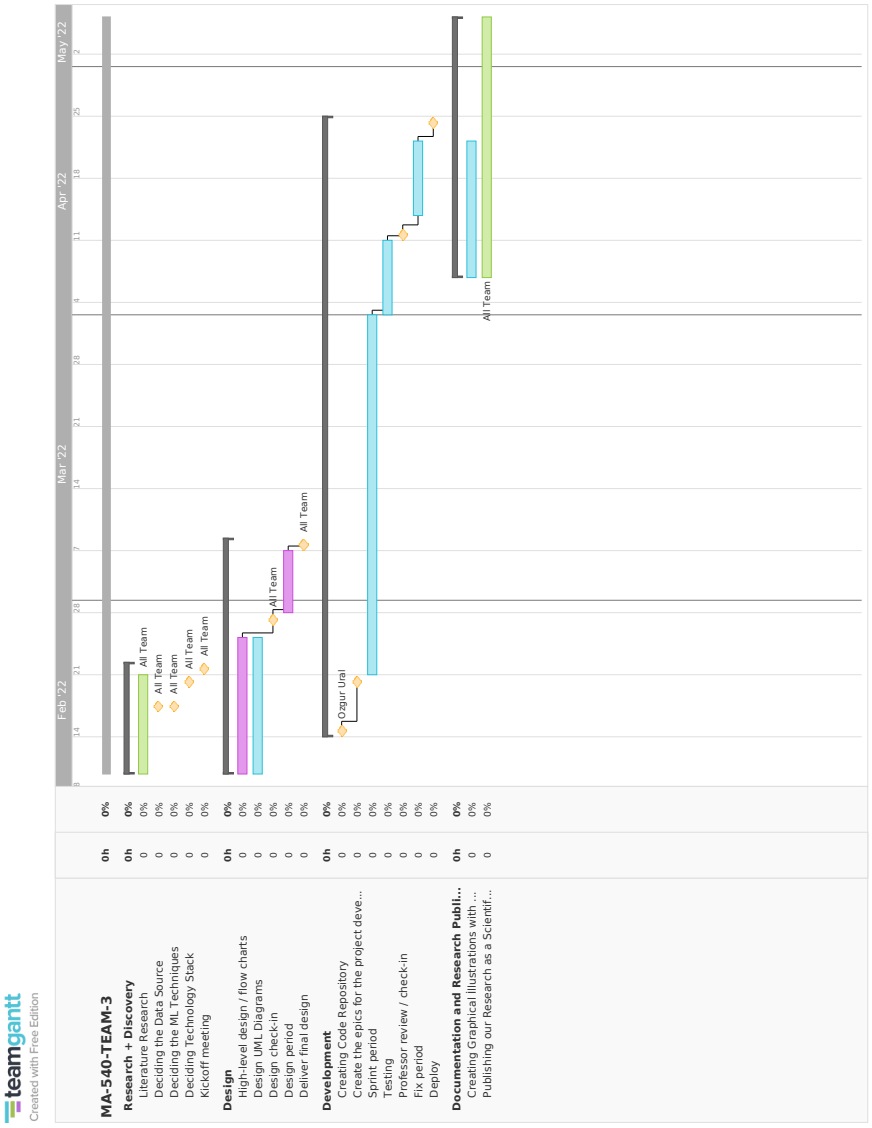

Figure 7: The Gantt Chart of the Project

7.2 Individual Contributions

7.3 Aaron Van De Brook

- Implement at least one machine learning/deep learning algorithm
- Contribute to Bayesian deep learning model
- Find relevant material on Bayesian deep learning and Bayesian neural networks
- Background and literature review section of homework 1
- Added new papers since homework 1 to literature review
- Set up TF-IDF and SBERT code for word embeddings

7.3.1 Jose Gabriel Gonzalez

- Implementation of at least one deep learning algorithm
- Partly implementing the Bayesian Network
- Mathematical interpretation of results
- Interpretation of datasets and their implementation to different algorithms
- Writing reports

7.3.2 Justus Renkhoff

- Implementation of ensemble methods
  (RandomForestClassifier, GradientBoostingClassifier, AdaBoostClassifier)
- Implementation of probabilistic classifiers
- Implementation of artificial neural network
- Implementation of BoW and TF-IDF preprocessing
- Measuring the energy consumption of the algorithms with the help of Trier University of Applied Sciences.
- Test and comparison of different models and network architectures

7.3.3 Ozgur Ural

- Designing and developing the infrastructure of the project.
- Designing high-level flow charts.
- Participating in the implementation of the deep learning algorithm.
- Partly participating in implementing the Bayesian Network.
- Creating graphical illustrations with the results.
- Writing the research results as a scientific paper.

References

[1] Ap twitter account hacked in fake 'white house blasts' post, Apr 2013.

[2] What is a bayesian neural network?, Dec 2021.

[3] Oluwaseun Ajao, Deepayan Bhowmik, and Shahrzad Zargari. Fake news identification on twitter with hybrid cnn and rnn models. In *Proceedings of the 9th International Conference on Social Media and Society*, SMSociety '18, page 226–230, New York, NY, USA, 2018. Association for Computing Machinery.

[4] Wissam Antoun, Fady Baly, Rim Achour, Amir Hussein, and Hazem Hajj. State of the art models for fake news detection tasks. 04 2020.

[5] Joshua V. Dillon, Ian Langmore, Dustin Tran, Eugene Brevdo, Srinivas Vasudevan, Dave Moore, Brian Patton, Alex Alemi, Matt Hoffman, and Rif A. Saurous. Tensorflow distributions, 2017.

[6] A. T. C. Goh and C. G. Chua. Bayesian neural networks.

[7] S. Hayman. The mcculloch-pitts model. In *IJCNN'99. International Joint Conference on Neural Networks. Proceedings (Cat. No.99CH36339)*, volume 6, pages 4438–4439 vol.6, 1999.

[8] Chaitra K Hiramath and G. C Deshpande. Fake news detection using deep learning techniques. In *2019 1st International Conference on Advances in Information Technology (ICAIT)*, pages 411–415, 2019.

[9] Imamah and Fika Hastarita Rachman. Twitter sentiment analysis of covid-19 using term weighting tf-idf and logistic regresion. In *2020 6th Information Technology International Seminar (ITIS)*, pages 238–242, 2020.

[10] Diya Li, Harshita Chaudhary, and Zhe Zhang. Modeling spatiotemporal pattern of depressive symptoms caused by covid-19 using social media data mining. *International Journal of Environmental Research and Public Health*, 17(14), 2020.

[11] David Meyer and FH Technikum Wien. Support vector machines. *R News*, 1(3):23–26, 2001.

[12] Marvin Minsky and Seymour Papert. *Perceptrons: An Introduction to Computational Geometry*. MIT Press, Cambridge, MA, USA, 1969.

[13] S Madeh Piryonesi and Tamer El-Diraby. Role of data analytics in infrastructure asset management: Overcoming data size and quality problems. *Journal of Transportation Engineering, Part B: Pavements*, 146:04020022, 06 2020.

[14] Samira Pouyanfar, Saad Sadiq, Yilin Yan, Haiman Tian, Yudong Tao, Maria Presa Reyes, Mei-Ling Shyu, Shu-Ching Chen, and S. S. Iyengar. A survey on deep learning: Algorithms, techniques, and applications. *ACM Comput. Surv.*, 51(5), sep 2018.

[15] Nils Reimers and Iryna Gurevych. Sentence-bert: Sentence embeddings using siamese bert-networks. *arXiv preprint arXiv:1908.10084*, 2019.

[16] Hippolyt Ritter and Theofanis Karaletsos. Tyxe: Pyro-based bayesian neural nets for pytorch. *arXiv preprint arXiv:2110.00276*, 2021.

[17] Phan Thanh Noi and Martin Kappas. Comparison of random forest, k-nearest neighbor, and support vector machine classifiers for land cover classification using sentinel-2 imagery. *Sensors*, 18(1):18, 2018.

[18] Hao Wang and Dit-Yan Yeung. A survey on bayesian deep learning. *ACM Comput. Surv.*, 53(5), sep 2020.

[19] Jason Wang, Kaiqun Fu, and Chang-Tien Lu. Sosnet: A graph convolutional network approach to fine-grained cyberbullying detection. In *2020 IEEE International Conference on Big Data (Big Data)*, pages 1699–1708, 2020.

[20] Qiang Zhang, Aldo Lipani, Shangsong Liang, and Emine Yilmaz. Reply-aided detection of misinformation via bayesian deep learning. In *The World Wide Web Conference*, WWW '19, page 2333–2343, New York, NY, USA, 2019. Association for Computing Machinery.