Claim Verification Using Generated Data by ChatGPT and Wikipedia

Leonard Persson Norblad
732A81 Text Mining
Linköping University
leope744@student.liu.se

Abstract

Claim verification is the task of predicting a text's truthfulness. Verification models are often trained on manually created datasets which are expensive and time consuming to create. In this project, a verification model was trained on generated data. The data was generated by feeding scraped Wikipedia articles together with instructions to generate a true or false statement in a prompt to ChatGPT 3 turbo (via API). The articles were used as evidence, providing a source of truth for the model. The ChatGPT response was used as claims, either a true statement meaning it aligns with the evidence, or a false statement, meaning it contradicts the evidence. The evidence and claim were then embedded by using BERT-small. A neural network containing bidirectional LSTM layers was trained to distinguish between false and true claims. The model was able to classify the validation samples with a macro average F1-score of 0.80. After evaluating the model using custom inputs it was found that it is sensitive to the term 'not'. The reason for this could be due to an overrepresentation of the term 'not' in the false claims.

1 Introduction

Fact verification is the task of verifying the truthfulness of a claim. It can help to detect when the *Large Language Model* (LLM) is hallucinating, meaning the model is producing an output containing false information. Which can otherwise be hard to detect since the model might sound convincing (Lee et al., 2022). The fact verification model could then be implemented as a step before the user receives the response (Semnani et al., 2023).

The verification task can be divided into three subtasks. *Document retrieval*, given a claim and a set of documents the model should provide a set of documents that can be used to verify it. *Sentence selection*, given a claim and a small set of documents the model should provide a set of sentences that can be used to verify the claim. *Claim verification*, given a claim and a set of sentences *evidence*, the model should determine if the claim aligns or contradicts the evidence (Nie et al., 2018). This project will only focus on the claim verification task, assuming the evidence is collected.

FEVER *Fact Extraction and VERification* is a popular dataset for claim verification. The data consists of manually written claims, created by altering Wikipedia articles. The claims are then annotated as 'Supported', 'Refuted', or 'NotEnoughInfo' (Thorne et al., 2018).

Nie et al.'s (2018) presented the *Neural Semantic Matching Network* (NSMN). The network consists of four layers. The *Encoding layer* uses bidirectional LSTM layers that encode the input tokens with their contexts. Then the *Alignment layer* performs an alignment between the two sequences based on the encoded tokens. The *matching layer* performs semantic matching between the two sequences, again by using bidirectional LSTM. Finally, in the *Output layer* the two sequences are projected onto two vectors by max-pooling. The vectors and the absolute difference between them are mapped to the final output. In this project, bidirectional LSTM layers will also be used for the verification model.

Manually creating datasets is both time consuming and costly (Jordon et al., 2022). This project aims to train a claim verification model solely based on generated data. When a reliable data generation process is in place, a large amount of data can be generated quickly. A larger dataset exposes the model to more variation, which improves its generalization capabilities (Géron, 2019).

2 Theory

2.1 BERT-small word embedding

Bidirectional Encoder Representations from Transformers (BERT) is a pre-trained deep bidirectional

transformer network used for contextualized word representations. BERT-small has fewer parameters than the original BERT model and represents the tokens in a 512 dimension space. Since the model is bidirectional it can capture the context of a word based on the words next to it. This is important since the meaning of a word can change dramatically based on its context (Devlin et al., 2018).

2.2 Bidriectional LSTM

Long Short-Term Memory (LSTM) is a type of recurrent network layer. It is especially good at tasks that require long-term dependencies, which is why it has been very successful in natural language process (NLP) tasks. The layer uses memory cells that allow for constant error flow during training to avoid the vanishing gradient problem. The memory cell can store and retrieve information over long sequences. The cell has an input gate, an output gate, and a forget gate. The forget gate allows the cell to decide what information to forget. The LSTM cell can be mathematically described as:

$$\boldsymbol{z}_t = g\left(\boldsymbol{W}_{\boldsymbol{z}}\boldsymbol{x}_t + \boldsymbol{U}_{\boldsymbol{z}}\boldsymbol{y}_{t-1} + \boldsymbol{b}_{\boldsymbol{z}}\right) \quad (1)$$

$$\boldsymbol{i}_t = \sigma\left(\boldsymbol{W}_{\boldsymbol{i}}\boldsymbol{x}_t + \boldsymbol{U}_{\boldsymbol{i}}\boldsymbol{y}_{t-1} + \boldsymbol{b}_{\boldsymbol{i}}\right) \quad (2)$$

$$\boldsymbol{f}_t = \sigma\left(\boldsymbol{W}_{\boldsymbol{f}}\boldsymbol{x}_t + \boldsymbol{U}_{\boldsymbol{f}}\boldsymbol{y}_{t-1} + \boldsymbol{b}_{\boldsymbol{f}}\right) \quad (3)$$

$$\boldsymbol{c}_t = \boldsymbol{i}_t \odot \boldsymbol{z}_t + \boldsymbol{f}_t \odot \boldsymbol{c}_{t-1} \quad (4)$$

$$\boldsymbol{o}_t = \sigma\left(\boldsymbol{W}_{\boldsymbol{o}}\boldsymbol{x}_t + \boldsymbol{U}_{\boldsymbol{o}}\boldsymbol{y}_{t-1} + \boldsymbol{b}_{\boldsymbol{o}}\right) \quad (5)$$

$$\boldsymbol{y}_t = \boldsymbol{o}_t \odot h\left(\boldsymbol{c}_t\right) \quad (6)$$

where $\boldsymbol{x_t}$ is the input vector at time t. $\boldsymbol{z}$ corresponds to the cell input activations, $\boldsymbol{f}$ to the forget gate activations, $\boldsymbol{i}$ to the input gate activations, $\boldsymbol{o}$ to the output gate activations, $\boldsymbol{y}$ to the cell output activations and $\boldsymbol{c}$ to the memory cell states. $\boldsymbol{W}$ is a weight matrix of the connections with the inputs, for example, $\boldsymbol{W_z}$ is the weight matrix of the connections between the cell input and the inputs. $\boldsymbol{U}$ is a weight matrix of the connections with the one-step delay cell output activations, for example, $\boldsymbol{U_z}$ is the weight matrix of the connections with the cell input and the one-step delay cell output activations. $\boldsymbol{b}$ is a bias vector. h is the activation function for the cell state, g is the activation for the cell input σ is the activation function for the gates (Arras et al., 2019)

To further utilize the LSTM layer, the sequence can be processed in both directions. By feeding the sequence to the LSTM layer in both directions we can extract features from both the forward states and the backward states, this is called a *bidirectional layer* (Huang et al., 2015)

3 Dense neural layer

Dense neural layers can be used for a large variety of machine learning tasks. By stacking multiple layers and connecting each node in one layer to all nodes in the next layer, forming a fully connected layer, the network maps the inputs $\mathbf{X}$ to the outputs $h_{\mathbf{W}}$ (Géron, 2019).

$$h_{\mathbf{W},\mathbf{b}}(\mathbf{X}) = \phi(\mathbf{X}\mathbf{W} + \mathbf{b})$$

where $\mathbf{W}$ is the weights, $\mathbf{b}$ is the bias vector and ϕ is the activation function.

4 Dropout

Dropout is a popular regularization technique. It works by temporarily excluding a set of units from the network during the training phase. The set of units is randomly selected and changes in every step. This helps the network to be more robust, generalize better, and not rely too heavily on a small set of nodes (Srivastava et al., 2014).

5 Batch normalization

A common problem when training neural networks is that the distribution of each layer's inputs changes as the parameters of the previous change during training, also referred as *internal covariance shift*. This makes the training process unstable and forces a low learning rate. Batch normalization can help to fight these issues. The layer works by zero-center and normalizing the layer inputs. This also has a regulating effect since it forces the network not rely to too much on specific features and encourages the network to learn more robust representations. The batch normalization layer learns two parameters, γ and β. Each input x_i of the mini-batch, size m, are normalized by the mini-batch metrics as:

$$\mu_{\mathcal{B}} \leftarrow \frac{1}{m}\sum_{i=1}^{m} x_i \quad (7)$$

$$\sigma^2_{\mathcal{B}} \leftarrow \frac{1}{m}\sum_{i=1}^{m}\left(x_i - \mu_{\mathcal{B}}\right)^2 \quad (8)$$

$$\widehat{x}_i \leftarrow \frac{x_i - \mu_{\mathcal{B}}}{\sqrt{\sigma^2_{\mathcal{B}} + \epsilon}} \quad (9)$$

$$y_i \leftarrow \gamma\widehat{x}_i + \beta \equiv \mathrm{BN}_{\gamma,\beta}\left(x_i\right) \quad (10)$$

where $\mu_{\mathcal{B}}$ is the mini-batch mean, $\sigma^2_{\mathcal{B}}$ is the mini-match variance, $\widehat{x}_i$ is the normalized inputs y_i is

the scale and shifted inputs. ϵ is a arbitrary small constant (Ioffe and Szegedy, 2015).

6 Early stopping

Early stopping is a regularization method that works by interrupting the training if a validation metric, such as validation loss, gets worse (loss increases). This directly prevents the model from overfitting as the training will be interrupted if the validation metric has not improved after a certain amount of epochs. The number of epochs without improvement is controlled by the patience parameter. The weights can then be rolled back to the state the model was in at the best validation metric (Géron, 2019).

7 L2 (ridge) regularization

L2 *ridge* regularization works by adding a penalty for large weights. This reduces overfitting as it keeps the model weights small. α controls the magnitude of the regularization, higher α means a higher penalty which means more regularization. The sum of the squared weights w is added to the loss. p is the number of weights. (Géron, 2019).

$$\alpha \sum_{j=1}^{p} w_j^2 \tag{11}$$

8 Data

An illustration of the project framework with links to the code can be found in appendix B.

8.1 Data generation

The data was generated in a series of steps. First, 2600 randomly selected Wikipedia articles were scraped (2600 were chosen to ensure 2000 samples after data cleaning). The articles were then shortened so that only the first complete sentences with less than 100 tokens and more than 20 tokens were extracted [1]. 20-100 tokens were chosen since sentences with this number of tokens represent a decent amount of information, representing the evidence extraction subtasks. 2000 samples with extracted summaries are then saved. 2000 samples were chosen due to limitations of time and cost expenses. To generate a statement from the summary, two custom prompts were constructed. One for true statements and one with false statements

[1] Tokens defined by ChatGPT 3.5 turbo tokenizer

True statement/claim aligns with evidence
Article title
Valentina Zenere
Statement/claim
Valentina Zenere plays Isadora Artiñán on Elite.
Summary/evidence
Valentina Zenere (born 15 January 1997) is an Argentine actress, model and singer known for her portrayal of Ámbar Smith in the Disney Channel series Soy Luna and Juacas, as well as for her portrayal of Alai Inchausti in the Argentine telenovela Casi Ángeles. Since 2022, she plays Isadora Artiñán on the Netflix series Elite.

False statement/claim contradicts the evidence
Article title
Wellington Square Baptist Church, Hastings
Statement/claim
Wellington Square Baptist Church is a Catholic church.
Summary/evidence
Wellington Square Baptist Church is a Baptist church in the centre of Hastings, a town and seaside resort in East Sussex, England. It was built in 1838 for a congregation which had previously been meeting for worship in hired premises, and it has been in continuous use since then.

Figure 1: Two samples from the generated dataset

see Appendix A. The prompts were then fed to the ChatGPT 3.5 turbo model using the `openai` API. The model's response is then saved as the generated claims together with a label based on which prompt was used. Half of the collected evidence data was used to generate true statements and the other half was used to generate false statements. Two samples from the generated dataset are shown in figure 1 and table 1 provides a description of the generated datset.

Varible	Description
Title	The title of the article.
Article	The content (text) of the article.
Summary	First few sentences of the article.
Statement	A one-sentence statement.
Label	Binary indicator of the truthfulness of the statement.

Table 1: Variable descriptions

8.2 Data usage

The two inputs to the claim verification model are evidence together with a claim. The statement variable is used for the claim and the summary variable is used for the evidence. This approach was chosen since this project does not include evidence extraction. However, the summary variable should represent collected evidence quite well since for true treatments the claim should align with the ev-

idence, and for false statements, the claim should contradict the evidence. This is exactly how the LLM has been prompted. It also contains more information than needed since the LLM is prompted to extract only one fact from the summary. This would also simulate how the evidence would look if it was extracted from an evidence extraction model. The data was divided into 70% training data (1400 samples) 20% validation data (400 samples) and 10% test data (200 samples). The training set was used to fit the parameters of the network. The validation set was used to monitor the network's generalization capabilities and to tune the hyperparameters/model architecture. The test set was used to examine the model on completely unseen data.

9 Method

The method is also a part of the project framework illustration (with code), found in appendix B.

9.1 Embedding

The claims and the evidence were tokenized using the BERT-small tokenizer. The claims were padded to a length of 27 tokens and the evidence was padded to a length of 127 tokens. The numbers were chosen based on the samples with the most tokens for each variable. The number of tokens differs a bit from the maximum number of tokens in the data generation since the BERT-small tokenizer differs from the ChatGPT 3.5 turbo tokenizer. Then both the claim and evidence were (separately) fed through the pre-trained network to obtain word embeddings. Each token receives a 512 long vector word embedding. This means that each claim is represented as a matrix with dimensions [27 $\times$ 512] matrix and each evidence is represented as a matrix with dimensions [127 $\times$ 512] (Bhargava et al., 2021) & (Turc et al., 2019).

9.2 Model architecture

Figure 2 shows the architecture of the claim verification model. The model starts with the BERT-small embeddings of the evidence and claim. After that, each input is passed to a bidirectional LSTM layer with 30 nodes, and 0.001 L2 regularization is applied to the kernel weight matrix, recurrent kernel weights, and to the output of the layer. The layer also contains 20% dropout of the units in the linear transformation of the recurrent state. The output of the layer is 20 activations (10 forward + 10 backward) since the layer is bidirectional and only the output for the last time step is returned (hidden states are also not used). The two outputs from the bidirectional LSTM layers are then concatenated, 20 from evidence and 20 from claim resulting in 40 features. After that, the dense block starts. Here, a dense block is defined as dropout + dense layer + batch normalization. The model has two dense blocks, both with 50% dropout, ReLu activation function, and 0.001 L2 regularization applied to both the kernel weight matrix and the output of the layer. The first block has 10 dense nodes and the second block has 5 dense nodes. Finally, the 5 outputs from the second dense block are inputted to the final dense layer with 1 node using the sigmoid activation function. The architecture was found by exploring different architectures and hyperparameters and then selecting the model with the highest validation F1 score. The idea with this architecture is that the LSTM layers should find features in both sequences that the dense layers can use to predict the truthfulness of the claim. Heavy regularization was implemented to reduce overfitting due to the small dataset and large model complexity.

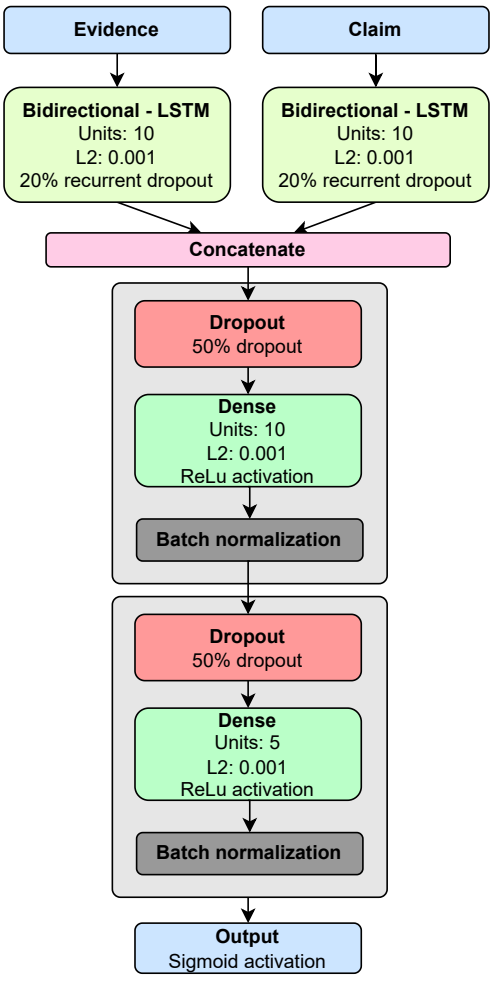

Figure 2: Architecture of the claim verification model

9.3 Model training

The model was fitted using adam optimizer, minimizing binary cross-entropy with a learning rate of 0.0005 and batch size 32. Early stopping with a patience of 10 epochs, monitoring the validation loss.

10 Results

10.1 Training history

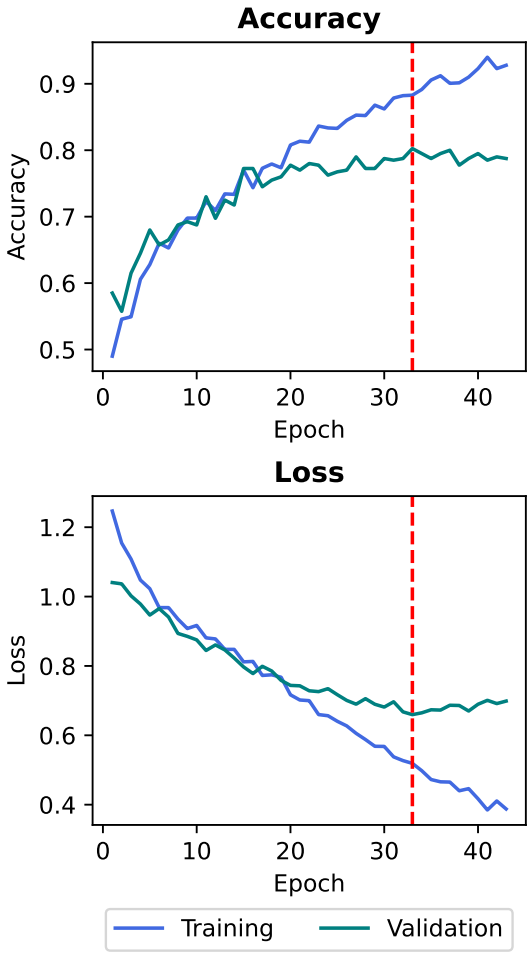

Figure 3: Training history

Figure 3 shows the accuracy and loss for training and validation data. The red curve shows the validation performance, the blue shows the training performance and the dashed vertical line shows the epoch with the lowest validation loss. Due to the lowest validation loss observed at epoch 33 and the patience of 10, the model stopped training after epoch 43. After the training was completed, the model weights were restored to the state they were in at epoch 33.

10.2 Model metrics

	Precision	Recall	F1-score
False claim	0.95	0.97	0.96
True claim	0.97	0.95	0.96
Macro average	0.96	0.96	0.96

Table 2: Training data metrics

	Precision	Recall	F1-score
False claim	0.80	0.79	0.79
True claim	0.81	0.82	0.81
Macro average	0.80	0.80	0.80

Table 3: Validation data metrics

	Precision	Recall	F1-score
False claim	0.90	0.80	0.85
True claim	0.80	0.90	0.85
Macro average	0.85	0.85	0.85

Table 4: Test data metrics

Tables 2, 3, 4 shows the precision, recall and F1-score for the model, evaluated on train data (table 2), validation data (table 3) and test data (table 4). The model performs substantially better on training data than validation and test, this can be seen as all metrics are higher for training. This is expected since the model parameters have been fitted on this data. The gap between training and validation is quite large. This indicates that the model could be overfitted. The metric from the test data is calculated on a smaller set (200 samples) and is therefore not as reliable as the validation metrics in that sense. One could argue that since the model selection has been made using the validation data, this data is somewhat exposed in the training phase and may not be as trustworthy, however since the model performance is almost equally good on the test set the model does not seem to be overfitted to the validation data. It can also be seen in the tables that the model seems to perform equally good both classes, except for the training data which has a higher precision (0.90) and lower recall (0.80) for false claims and a lower precision (0.80) and a higher recall (0.90) for true claims.

10.3 Validation using edited claims

After manually reviewing the model's predictions on the test dataset, it was proven to be hard to make any conclusions about the model's classification capabilities. To get a better understanding of the model, the claims were modified to see how the model reacts to the different inputs, without changing the evidence. Since the sigmoid activation function is used for the output layer, we can interpret the output value as the model's confidence. Where a value close to 1 means it believes the claim agrees with the evidence and a value close to 0 means the model believes the claim contradicts the evidence.

Claim
Transportation Alternatives works to change transportation priorities.
Evidence
Transportation Alternatives (TransAlt, formerly T.A.) is a non-profit organization in New York City which works to change New York City\'s transportation priorities to encourage and increase non-polluting, quiet, city-friendly travel and decrease automobile use. TransAlt seeks a transportation system based on a "Green Transportation Hierarchy" giving preference to modes of travel based on their relative benefits and costs to society.
Prediction: 0.9579151 (correct=1)

Figure 4: Claim, evidence, and model prediction for one random sample from test data.

One random sample was chosen from the test dataset. Presented figure 4. The words that appear in both he claim and the evidence are marked with an underline in the claim and by a green background in evidence. The model correctly predicted the claim as true. This evidence is later used for predicting the claims in figures 5, 6, 7 and 8. *The edited claims presented here are a sample from many explored.*

Edited claim
Transportation Alternatives strives to shift the focus of transportation priorities.
Prediction: 0.95605856 (correct=1)

Figure 5: Edited claim, different wordning

Figure 5 presents a slight modification of the original claim. The wording is now edited to be more different from the evidence. However, the model still correctly classifies the claim as true. This shows good performance since it is of interest to have a model can can handle different wordnings.

Edited claim
Transportation Alternatives is a petroleum company.
Prediction: 0.18087105 (correct=0)

Figure 6: Edited claim, false

Figure 6 shows another edited claim. The claim was edited to not align with the evidence. The model correctly classified the sample as false.

Edited claim
The Transportation Alternatives does not work to change transportation priorities.
Prediction: 0.13681345 (correct=0)

Figure 7: Etided claim, added 'not' to a true claim

Figure 7 shows a claim, edited to be false by adding *not*. The model correctly classified the sample as false.

Edited claim
Transportation Alternatives does not support polluting.
Prediction: 0.12414804 (correct=1)

Figure 8: Edited claim, with 'not' in false claim

As a final edit, figure 8 shows an edit when the claim is true but uses the term 'not'. This managed to trick the model to believing the claim was false. This shows bad performance of the model.

11 Discussion

11.1 Related work

As the data in this project is generated and therefore of a unique character, comparing the result to related papers is challenging. However, to put it into some context, Nie et al.'s (2018) NSMN gained an F1 score of 72.6 (Supports), 70.4 (Refutes), and 56.3 (Not enough info). This is much lower than the models F1 score in this project 0.79 (False claim), 0.81 (True claim). Nevertheless, since the NSMN model combines these three subtasks (document retrieval, sentence selection, and claim verification) with three classes (supports, refutes, and not enough info), it is understandable why the performance is not as good.

11.2 Limitations and improvements

As it is not possible to manually check all generated statements, there is a risk that the data does

not match expectations. Incorrectly generated statements could lead to labels and statements not corresponding. For example, if the LLM is instructed to generate a true statement but generates a false statement unintentionally.

The prompts strictly asked for answers shorter than 8 words, this requirement was not always fulfilled by the LLM, however, when it was added the responses were shortened and it was therefore kept. This shows a limitation of the data generation as the LLM does not always act as instructed.

By evaluating the model with edited claims, it was found that the model tends to predict a claim is false if it contains the term 'not'. This is probably due to how the data is generated. Since the true statements are found by picking one fact from the summarized article, it is unlikely that it will contain the term 'not' while when the LLM is instructed to generate false statements, it tends to add terms like 'not' to make the statement false. The model seems to have picked up this pattern and tends to falsely predict true statements as false when they contain the term 'not'. There are several ways to avid this. The false samples with 'not' could be reduced before feeding them to the model. It is also possible to further instruct the LLM or ensure articles containing the term 'not' is used, this might lead to more true samples containing 'not'.

The data generation performed in this project assumes the document retrieval and sentence selection is always correct. However, in a real word scenario this is not a realistic assumption. To further improve the quality of the generated data some samples should have not sufficient amount of evidence to provide a truthfulness estimation and be labeled as *NotEnoughInfo*. This gives the model the option to determine whether the evidence is insufficient, which will happen if no evidence exists or if the evidence extraction is incorrect. This was implemented in the FEVER dataset (Thorne et al., 2018).

The dataset is relatively small and it is therefore challenging to make a model that generalizes well. Since the data can be easily generated, it would be simple to further extend the dataset to reduce the effect of sampling noise.

The LSTM layer for the evidence is applied to the whole sequence. The 20 features produced by the layer will therefore be fitted on the whole sequence. This might not be the optimal network architecture since we know only parts of the evidence is relevant to prove the claim. To further improve the model an attention mechanism could be used to weight the evidence with the claim to highlight important parts of the evidence. This could help the model extract useful information from the evidence which could improve the quality of the extracted features.

The model is missing a method to directly compare the claim and the evidence. The claim and the evidence are presented to the dense blocks as 20 claim features and 20 evidence features. The dense layers then have to map the 40 features to figure out if the claim is true or false. In the NSMN model, the absolute difference between the two vectors representing the claim and the evidence was computed in the final layer (Nie et al., 2018). An architecture like this might be more suitable for the task since the truthfulness of the claim is directly dependent on the evidence.

Since the model is only evaluated on generated data, the performance of human written text is not known. To run the model on data that is not generated one would need to develop the document retrieval and sentence selection processes. Another option would be to use the data from the FEVER dataset. However, the shape of the input was not specifically designed for this dataset which might lead to a large part of the data being truncated.

12 Conclusion

In this project, a novel method to generate data to train a fact verification model was explored. It was found that the generation process needs to be further tuned to be more robust. The main flaw of the generation process is the overrepresented term 'not' in the false statements. A claim verification model was fitted and had a validation F1 macro average of 0.80.

References

Leila Arras, José Arjona-Medina, Michael Widrich, Grégoire Montavon, Michael Gillhofer, Klaus-Robert Müller, Sepp Hochreiter, and Wojciech Samek. 2019. *Explaining and Interpreting LSTMs*, page 211–238. Springer International Publishing.

Prajjwal Bhargava, Aleksandr Drozd, and Anna Rogers. 2021. Generalization in nli: Ways (not) to go beyond simple heuristics.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2018. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*.

Aurélien Géron. 2019. *Hands-on machine learning with Scikit-Learn, Keras, and TensorFlow: Concepts, tools, and techniques to build intelligent systems*. " O'Reilly Media, Inc.".

Zhiheng Huang, Wei Xu, and Kai Yu. 2015. Bidirectional LSTM-CRF models for sequence tagging. *CoRR*, abs/1508.01991.

Sergey Ioffe and Christian Szegedy. 2015. Batch normalization: Accelerating deep network training by reducing internal covariate shift. *arXiv preprint arXiv:1502.03167*.

James Jordon, Lukasz Szpruch, Florimond Houssiau, Mirko Bottarelli, Giovanni Cherubin, Carsten Maple, Samuel N. Cohen, and Adrian Weller. 2022. Synthetic data – what, why and how?

Nayeon Lee, Wei Ping, Peng Xu, Mostofa Patwary, Mohammad Shoeybi, and Bryan Catanzaro. 2022. Factuality enhanced language models for open-ended text generation. *ArXiv*, abs/2206.04624.

Yixin Nie, Haonan Chen, and Mohit Bansal. 2018. Combining fact extraction and verification with neural semantic matching networks. *arXiv preprint arXiv:1811.07039*.

Sina Semnani, Violet Yao, Heidi Zhang, and Monica Lam. 2023. Wikichat: Stopping the hallucination of large language model chatbots by few-shot grounding on wikipedia. In *Findings of the Association for Computational Linguistics: EMNLP 2023*, pages 2387–2413.

Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. 2014. Dropout: A simple way to prevent neural networks from overfitting. *Journal of Machine Learning Research*, 15(56):1929–1958.

James Thorne, Andreas Vlachos, Christos Christodoulopoulos, and Arpit Mittal. 2018. FEVER: a large-scale dataset for fact extraction and VERification. In *NAACL-HLT*.

Iulia Turc, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2019. Well-read students learn better: The impact of student initialization on knowledge distillation. *CoRR*, abs/1908.08962.

13 Appendices

A Promts to generate statements

Templates used to create prompts to generate statements. article_name is the Wikipedia article name and article is the Wikipedia article, summarized.

A.1 True statements

```
Answers must be written in a way that can be understood without context. Your response must consist of a single sentence shorter than 8 words. Pick one single detail in this text about article_name and write it as a standalone statement: 'article'
```

A.2 False statements

```
Answers must be written in a way that can be understood without context. Your response must consist of a single sentence shorter than 8 words and be a false statement. Change one single detail in this text about article_name and write it as a standalone false statement: 'article'
```

B Project framework

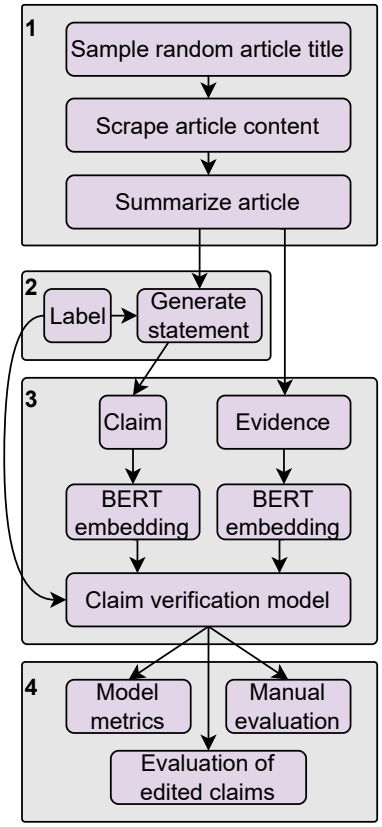

Figure 9: Project framework

Figure 9 visualizes the framework of the repository fact-verification. Each number is a notebook, each notebook has a `.py` script with helper functions.

1. Data Scraping
2. Statement generation
3. Claim verification model
4. Evaluation

To simplify the graph, the datasplit done in the verification model notebook is not shown.