Belgrade, March 2025.

Comparative Analysis of Different Formality Detection Approaches

Author: *Viktor Curcic*

Summary

In this paper, I have examined different formality detection approaches (e.g. libraries, fine-tuned models, LLMs) based on chosen metrics on a given dataset. The goal of the paper is to give a comparative analysis of these approaches, how they differ one from the other, to give my reasoning as to which approach is the best in which situation, and to prove my reasoning based on the aforementioned chosen metrics.

Contents

1 Collection and Processing of Data

1.1 Data Collection

For this part of the problem, I opted for a hybrid approach to forming datasets for evaluation. I decided to combine data from different sources. These include: *Grammarly's Yahoo Answers Formality Corpus*[1] (or GYAFC for short) - which represents only $\approx 2\%$ of the total dataset, generated data (generated by LLMs such as OpenAI's GPT-4 and DeepSeek) and my own ideas - which represent the rest of the dataset.

The idea behind this dataset was to give the models everything - from extremely casual, conversational sentences, to ambiguous sentences where only word choices reflect their formality. That way, models could better differentiate between formal and informal.

The dataset, consisting of approximately 5500 different formal and informal sentences written with appropriate formal/informal labels, is enclosed in the given repository under the name "*dataset.txt*".

1.2 Data Processing

Data processing is done in the enclosed files *"pair_generation.py"* and *"test_formation.py"*. The process goes as follows: data is forwarded to the *"pair_generation.py"* where it is split and labeled (1 for formal sentences and 0 for informal sentences). Afterwards, those processed sentences are passed to the *"synthetic_formality_data.csv"* file.

As the dataset is already cleaned and the classes (formal and informal samples) are balanced, there is no reason to do any further processing.

That file is then used to form training, testing and validation files. Dataset is split into training (70% of the original dataset, randomly chosen), validation (15% of the original dataset, randomly chosen) and testing sets (15% of the original dataset, randomly chosen), with their respective files named accordingly.

"train.csv" holds data used for training different models, for example fine-tuned models such as DistilBERT (more on that later). *"val.csv"* is used to tune hyperparameters during training - they can have an effect on the batch sizes and learning rate of the model, as well as on deciding when the model has finished training. *"test.csv"* has the final test data used to determine the necessary metrics for comparisons between models.

2 Metrics

When discussing metrics based on which we can compare different models, the most popular ones include **accuracy**, **precision**, **recall**, **F1-Score**, **Confusion Matrix**, **AUC-ROC** and **Spearman**. I will discuss which of these I have used and why in the following subsections.

2.1 Accuracy

Generally speaking, accuracy is regarded as the percentage of all correct predictions of a model.Mathematically, it is calculated as

$$accuracy = \frac{TP + TN}{Sum}$$

where TP is the number of all true positives, TN is the number of all true negatives and Sum is the sample size.

The downside of accuracy as a metric is that if the classes are more biased (for example, 90% formal compared to 10% informal sentences in a set), then accuracy loses on its relevance, as it will be skewed towards the more prominent class.

As my testing, training and validation data is selected randomly, there is always a chance that a very imbalanced set can be chosen. As such, I haven't taken accuracy into consideration as a metric. There are other metrics which take this imbalance into consideration.

2.2 Precision

Precision is the proportion of positive predictions that were correct. It is calculated as follows:

$$precision = \frac{TP}{TP + FP}$$

where TP is the number of all true positives and FP is the number of all false positives.

High precision means that there were very few false positives (i.e. informal texts mislabeled as formal). This is a very useful metric and will be used for my model comparisons.

2.3 Recall

Recall, also known as sensitivity or true positive rate (TPR), is the proportion of true positives that were correctly identified by the model. It is calculated as follows:

$$recall = \frac{TP}{TP + FN}$$

where TP is the number of all true positives and FN is the number of all false negatives.

High recall means that there were very few false negatives (i.e. formal texts mislabeled as informal). This is also a very useful metric and will be used for my model comparisons.

2.4 F1-Score

We already discussed how accuracy doesn't take the distribution of data into account and how that can be a serious issue. That is where F1-Score comes into play. F1-Score is the harmonic mean between precision and recall. It is calculated as follows:

$$F1 = 2 \cdot \frac{precision \cdot recall}{precision + recall}$$

Although a bit harder to interpret than accuracy, it is an important metric for analyzing models and their output.

2.5 Confusion Matrix

A Confusion Matrix is a 2x2 (in my case, though it can be larger) matrix showing the number of false and true positives and negatives. It looks like this:

	Predicted	
	Formal	**Informal**
Actual Formal	True Positives (TP)	False Negatives (FN)
Actual Informal	False Positives (FP)	True Negatives (TN)

It will be very useful when looking at the number of false positives and negatives and why the model interpreted them as such.

2.6 AUC-ROC

The AUC-ROC curve plots the True Positive Rate (TPR) against the False Positive Rate (FPR) at different thresholds showing how well a model can distinguish between two classes such as positive and negative outcomes. Generally, it is a graphical representation of a model's ability to distinguish between two classes.

AUC (Area Under the Curve) measures the area under the ROC curve. A higher AUC value indicates better model performance as it suggests a greater ability to distinguish between classes. An AUC value of 1.0 indicates perfect performance while 0.5 suggests it is random guessing.[2]

AUC-ROC is a useful metric when using binary classification and when the classes are balanced. It is good to use when there is a need for better precision or recall. We will take a look at AUC-ROC in some models. In those with a high F1-Score, there is no need for AUC-ROC.

2.7 Spearman

The most complex of all of these metrics is the Spearman correlation. Mathematically speaking, it is a statistical measure of the strength and direction of the monotonic relationship between two continuous variables. [3]

It measures if model's scores align with human rankings of formality in this instance. We will not be mentioning the Spearman correlation from this point onward. It is not useful when discussing binary classifications such as the one here, rather for continuous scores. If there were rankings of texts by formality, or there were human-rated continuous scores, it would have been useful.

3 Formality Detection Approaches

I will be discussing five different approaches which I have used. In this section, I will only describe them, how they work and why I chose them. In the next section, we will compare their results and functionalities.

3.1 DistilBERT

As the years go by, models become ever so larger. NVIDIA's MegatronLM has 8.3 billion parameters, while RoBERTa, Facebook AI's model, was trained on 160 GB of text. Many

of these models, although useful and efficient, lag behind in terms of speed and resources used. These large-scale models are especially hard to use on phones, which require light-weight, efficient, and responsive models.[4]

That is where DistilBERT comes into play. It is a much smaller and light-weight, general-purpose version of BERT - Bidirectional encoder representations from transformers designed by researchers at Google. It is simply a distilled version of BERT.

It reduces the size of BERT by 40%, while retaining 97% of its capabilities. DistilBERT will be very useful when doing formality detection. [5]

3.2 BART

The next model I will mention is Facebook's BART. BART is a transformer encoder-decoder (seq2seq) model with a bidirectional (BERT-like) encoder and an autoregressive (GPT-like) decoder. BART is pre-trained by corrupting text with an arbitrary noising function, and learning a model to reconstruct the original text. [6]

While analyzing different metrics, I will be using BART's BART-Large variant - a large-size version of BART. I will be discussing the performance of the zero-shot form - when the model relies on its pre-trained knowledge, and its fine-tuned form - curated to accomplish a specific task (like deduce the formality of my dataset). We will see how these forms compare to other models and libraries.

3.3 TextBlob

TextBlob is a Python library for processing textual data. It gives users the ability to perform common natural language processing (NLP) tasks such as part-of-speech tagging, noun phrase extraction, sentiment analysis, classification, and more. [7]

TextBlob is an interesting addition to this set of models, as it is, first and foremost, a library. I will be discussing TextBlob as a classifier. It gives the users an option to classify sentences as formal or informal based on the word length and word choices. Because of this, we can deduce that its effectiveness will be limited - TextBlob doesn't deep learn, it only does simple checks to see if a sentence is formal or informal.

3.4 DeBERTa-V3

The next addition to this collection of models and libraries is DeBERTa - Decoding Enhanced BERT with disentangled Attention. It is an improved model to BERT and RoBERTa because of its use of disentangled attention and enhanced mask decoder. [8]

DeBERTa is, like BART, also a large model, but because of its improvements, we will see how it compares to the other models in terms of time and other metrics. It will also be analyzed as a zero-shot model, but there is an option to fine-tune it if necessary.

3.5 Flan-T5

FLAN-T5 is an open-source, sequence-to-sequence, large language model that can be also used commercially. The model was published by Google researchers, and has been fine-tuned on multiple tasks. [9]

Its main appeal is in the field of translation, linguistic acceptability, sentence similarity and document summarization. In this paper I will analyze the Large variant, which has 780 milion parameters.

4 Analysis

4.1 Problems and Results: DistilBERT

DistilBERT was an interesting model in the sense that it was the first model I trained and also the first model I analyzed. I have already mentioned that DistilBERT is a lightweight model, and that really shows - its training, with the device set to CPU, took about 15 minutes to complete, and it didn't need to go through all 3902 training sentences, it went through about 1460 sentences.

Let's have a look at the results:

F1-Score = 0.987, **precision** = 0.987, **recall** = 0.987

Confusion Matrix =

$$\begin{bmatrix} 418 & 10 \\ 1 & 408 \end{bmatrix}$$

The learning rate was very high ($\approx 3 \cdot 10^{-5}$) and the training was done with the average speed of about $1.2\frac{it}{s}$. This clearly shows DistilBERT's advantages - it doesn't take a long time to produce good results.

As for a detailed look at the results, we can see a high F1-Score. Because of that, any other tweaking of precision/recall with AUC-ROC would be unnecessary. That is why I didn't include it with this model. The confusion matrix shows only 11 mistaken sentences. When we analyze the *"distilbert_predictions.csv"* file, we can look at some of these sentences. The sentence "I hope this email finds you well" was predicted as being informal, when in reality it is formal. That is also the only false positive. As for false negatives, DistilBERT mainly had trouble deducing some word choices. "Oversharing online can be risky." was predicted to be formal, but the word choices here are more informal. This is one of the ambiguous sentences and that is where DistilBERT only had trouble.

A detailed look at the code is given in the *"training_distilbert.py"* file.

4.2 Problems and Results: Zero-shot BART

The first zero-shot model will be BART. As I have already mentioned, when a model is zero-shot, it means that it's predictions will be done with no user training on them beforehand. As BART in this form is general purpose, only testing will be done. As such, the time it takes for the model to process the test examples will be significantly lower, and in this case is about 7 minutes.

The model is initialized through a HuggingFace zero-shot pipeline (from the Transformers library). It is used for classification through its approach to analyzing labels and provided text. In my model, the hypothesis template "This text is" necessitates a classification.

Let's look at the results:

F1-Score = 0.597, **precision** = 0.601, **recall** = 0.600

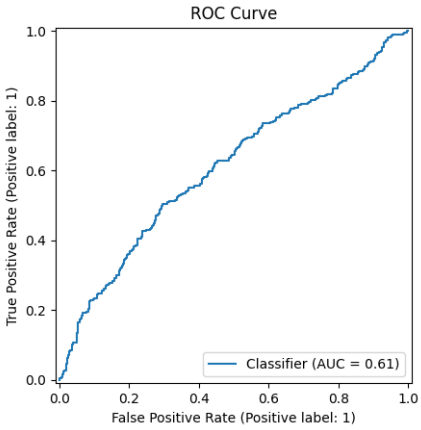
Confusion Matrix =

$$\begin{bmatrix} 293 & 135 \\ 200 & 209 \end{bmatrix}$$

We can clearly see a significant downgrade to DistilBERT's performance. Although the model sorts most of the sentences into their correct class, there are still a lot of them which are not sorted correctly. It is evident from the confusion matrix that the model has

problems with false positives. The problem here lies in this fact: how do we now know if the model is randomly guessing or if it just has a hard time with ambiguous sentences?

This is where AUC-ROC comes into play. Let's look at these results:



AUC is determined to equal 0.611, which mean that the model is not always randomly guessing, but rather has trouble with ambiguity. It seems that the model doesn't understand the differences between formal and informal in most cases.

We can see the same in the *"bart_zero_shot_predictions.csv"* file. Most of the predictions are not confident. It seems that for this dataset, fine-tuning the model should give better results.

A detailed look at the code is given in the *"zero_shot_bart.py"* file.

Let's look at how a fine-tuned BART compares to zero-shot BART.

4.3 Problems and Results: Fine-tuned BART

When talking about DistilBERT, I mentioned how it is a lightweight model. That would mean that it only has 6 layers - structures in a model's architecture. As for BART-Large, that number doubles to 12. It means that there are double the number of layers the information passes through when being processed. [10]

Fine-tuning BART was tricky to say the least. In terms of coding, not so much, but in terms of the actual learning of the model, it was a long process. The process of learning lasted $\approx$ 2 hours and 20 minutes. This is also the longest learning process out of all of the models mentioned in the paper. With the speed of $5.5\frac{it}{s}$, we can see why it was this slow.

One of the reasons for this was the large training dataset, but also because of hardware - the learning process was done through the CPU for all of the models. It could have also been done using CUDA, which would have significantly sped up the process, but I had trouble with having it work.

Let's look at the actual results:

F1-Score = 0.986, **precision** = 0.986, **recall** = 0.986

Confusion Matrix =

$$\begin{bmatrix} 422 & 6 \\ 6 & 403 \end{bmatrix}$$

A fine-tuned BART shows significant improvement on its zero-shot counterpart. In this sense, it is a highly functional model with a small error rate. It only had trouble

with ambiguous sentences such as "Quality dropped." More examples can be found in the *"bart_finetuned_predictions.csv"* file.

Because of its size and slow learning process, I would rather swap it with a lightweight model like DistilBERT, even though its results are on par with DistilBERT's.

A detailed look at the code is given in the *"finetuned_bart.py"* file.

4.4 Problems and Results: TextBlob

A more interesting addition to this batch is TextBlob. It works like this: TextBlob tokenizes the sentence; it then looks for any contractions ('s, 'll, ...), looks at the average word length in the sentence (generally, shorter words mean a bigger chance that a sentence is informal), and at any informal words ("gonna", "wanna", ...). Based on these metrics, it assigns a score and chooses formal or informal.

The problem with this is evident. Some sentences may lack contractions and informal words, but can still be short. Also, short formal words may be interpreted as informal. Because there is no deep learning, TextBlob will never really understand why it is doing what it is doing.

On the other hand, the testing process is done within seconds. Because of that, TextBlob is ultimately the fastest model/library in this paper.

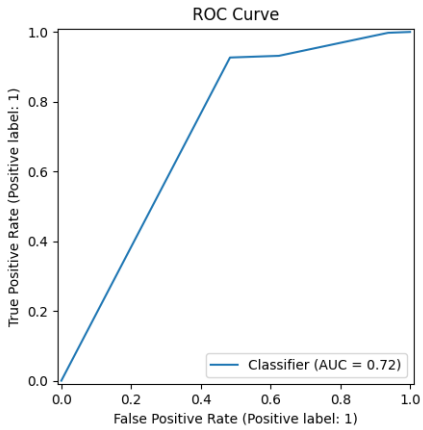
Let's look at the results:

F1-Score = 0.515, **precision** = 0.760, **recall** = 0.589

Confusion Matrix =

$$\begin{bmatrix} 87 & 341 \\ 3 & 406 \end{bmatrix}$$

The same question arises as with zero-short BART - is TextBlob randomly guessing? This answer can be given logically - as it is not a model which actively learns, but a model which works based on user-set metrics, it can't possibly do random guesses. We can also prove it mathematically using AUC-ROC:



With an AUC value of 0.719, TextBlob is even more confident than zero-shot BART. It's problem arises from the inability to differentiate false negatives from true positives, as I have mentioned above. In the *"textblob_predictions.csv"* file, we can see that practically every confidence score is 100%.

A detailed look at the code is given in the *"textblobb.py"* file.

As such, if a massive dataset has to be processed where there is no ambiguity and easily distinguishable informal words, and where there is a need for fast results, TextBlob is the best option out of all of these.

4.5 Problems and Results: DeBERTa-V3

Another zero-shot model is DeBERTa - an upgraded version of BERT. I decided on using the base version, as it is much smaller than the large version. Trained on 160GB of data, it can perform important zero-shot analyses, the same way BART can.

Like BART, DeBERTa doesn't need to learn, rather only to process the test samples. Also like BART, it takes about 7 minutes to give results.

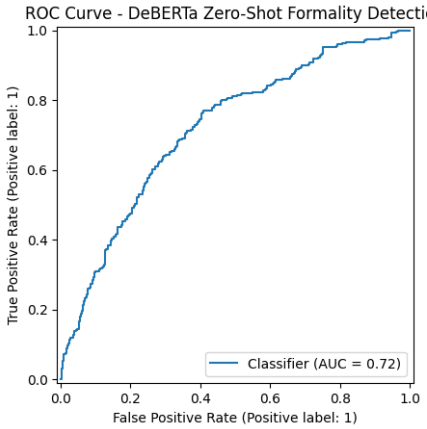
Let's look at them:

F1-Score = 0.622, **precision** = 0.655, **recall** = 0.637

Confusion Matrix =

$$\begin{bmatrix} 353 & 75 \\ 229 & 180 \end{bmatrix}$$

We can see that DeBERTa has trouble processing false positives. Let's look at its AUC-ROC score:



With an AUC score of 0.719, it's indicative that DeBERTa doesn't randomly guess. The curve is much better than both BART's and TextBlob's. We can conclude that DeBERTa has problems with ambiguous sentences, especially false positives.

A detailed look at the code is given in the *"deberta.py"* file.

4.6 Problems and Results: Flan T5

The last model I wanted to showcase is Google's Flan-T5. Flan-T5 is different from the other models in the sense that, like Claude or GPT, it requires prompts to determine the metrics. As a result, the quality of the prompts will determine the quality of the results.

There are a few limitations I should address. First, Flan-T5 is not normally zero-shot, but the prompting mechanism turns it into a model of that sort. Second, T5 wasn't designed for probabilistic classification. Because of that, the confidence scores should be taken with a grain of salt.

As I have mentioned, the quality of the prompt determines the quality of the results. My first prompt was: "Classify this text as formal or informal:" Let's see the results:

F1-Score = 0.321, **precision** = 0.239, **recall** = 0.489

Confusion Matrix =

$$\begin{bmatrix} 0 & 428 \\ 0 & 409 \end{bmatrix}$$

With this prompt, T5 was incapable of giving formal predictions. In this format, T5 is useless as a model. Because of that, I decided to tweak the prompt, as well as add a form of contrastive scoring and calibration. These additions and checks added significantly to the testing time - it took more than 50 minutes for the results to generate.

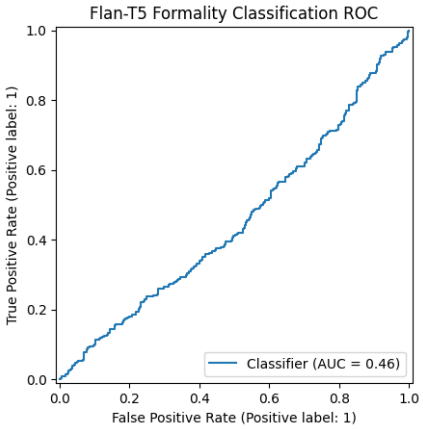
The prompt is: "Analyze this text's formality level. Choose between: A) Formal: Uses proper grammar, complex vocabulary, professional tone B) Informal: Casual language, contractions, colloquialisms The correct classification is:"

Let's see the results given with this prompt: **F1-Score** = 0.739, **precision** = 0.772, **recall** = 0.747

Confusion Matrix =

$$\begin{bmatrix} 388 & 40 \\ 172 & 237 \end{bmatrix}$$

But AUC-ROC paints a different picture:



An AUC value of 0.455 indicates complete random guessing. Looking at the probability values in the *"flant5_predictions.csv"* file, we can see that most of the predictions are given with a 50% confidence. If T5 were fine-tuned, these results would be much higher.

T5 shows that the quality of a prompt can significantly improve its output. Large language models should be given clearer instructions if we want quality results.

A detailed look at the code is given in the *"flant5.py"* file.

5 Conclusion

Let's look at the table of all important values:

Metric	DistilBERT	Zero-shot BART	Fine-tuned BART	TextBlob	roBERTa-v3	Flan-T5
Time	15min	7min	140min	2s	7min	50min
F1-score	0.987	0.597	0.986	0.515	0.622	0.739
Precision	0.987	0.601	0.986	0.760	0.655	0.772
Recall	0.987	0.600	0.986	0.589	0.637	0.747

"Time" in this table indicates the time it took to complete the testing and/or training and validation of the model. In this case, training and validation were only done with fine-tuned models.

The table clearly puts DistilBERT way above the other models. Fine-tuned BART, although having comparable results, lags behind in terms of time. If the dataset given was the size of GYAFC (160000 different sentences), fine-tuning would take days. With the dataset of that size even DistilBERT would take hours to parse through it. In that case, TextBlob with clearly defined conditions might be the best option.

This task had many challenges - some from the programming side, and others related to better prompting, ways to make the dataset larger and the time to train the models. These challenges taught me a lot about how models work, what their use can be in everyday life, and how I as a programmer can utilize them.

References

1. https://github.com/raosudha89/GYAFC-corpus
2. https://www.geeksforgeeks.org/auc-roc-curve/
3. https://www.geeksforgeeks.org/spearmans-rank-correlation/
4. https://medium.com/huggingface/distilbert-8cf3380435b5
5. https://huggingface.co/docs/transformers/model_doc/distilbert
6. https://medium.com/@nadirapovey/bart-model-architecture-8ac1cea0e877
7. https://textblob.readthedocs.io/en/dev/
8. https://github.com/microsoft/DeBERTa
9. https://huggingface.co/google/flan-t5-base
10. https://www.geeksforgeeks.org/layers-in-artificial-neural-networks-ann/