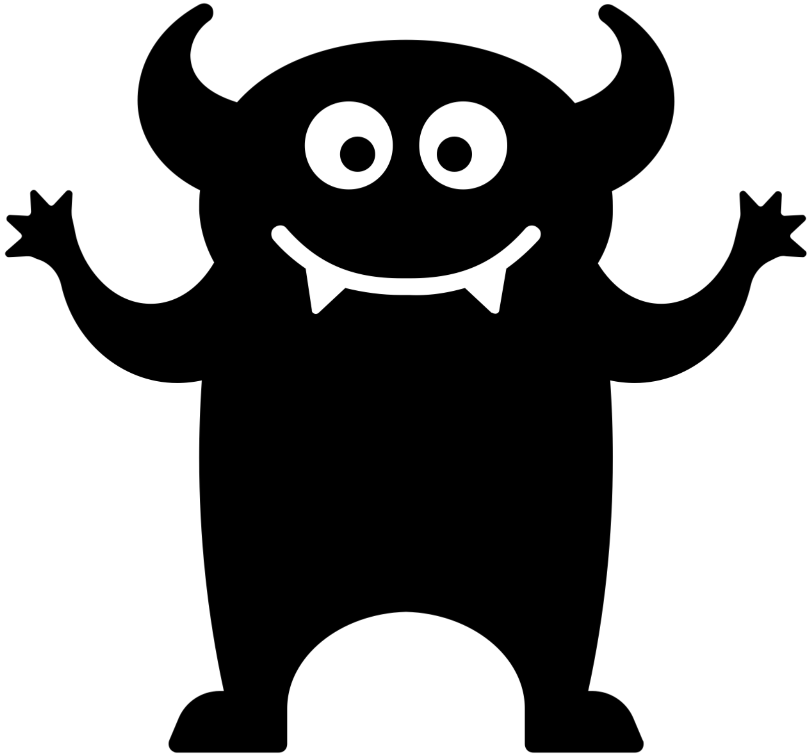
GAN-BERT: Generative Adversarial Learning for Robust Text Classification

Lecture: Seminar Klassifikation und Clustering

Lecturer: PD Dr. Stefan Langer

Presenters: Mengmeng Xu, Bolei Ma, Zheyu Zhang, Han Yang

Date: 23.01.2023



Agenda

1. Introduction
2. Basic Models
   2.1 BERT
   2.2 GANs
3. SS-GANs
4. GAN-BERT
5. Experimental Settings
6. Results & Analysis
7. Conclusion

Introduction

- **Goal:**

  1. Save time on data annotation (reduce the scale of labeled examples)
  2. Maintain good performances

- **Main Idea:**

  Use GAN to extend the fine-tuning of BERT-like architectures with unlabeled data in a generative adversarial setting

- **Main Contributions:**

  1. Extended the limits of Transformer-based architectures (i.e., BERT) in poor training conditions
  2. Systematically improved the robustness of such architectures, while not introducing additional costs to the inference

BERT – Bidirectional Encoder Representations from Transformers

- A stack of Transformer Encoder Layers that consists of multiple self-attention "heads"

- Workflow (crucial part):

  1. Pre-training: MLM (masked language modeling) & NSP (next sentence prediction)

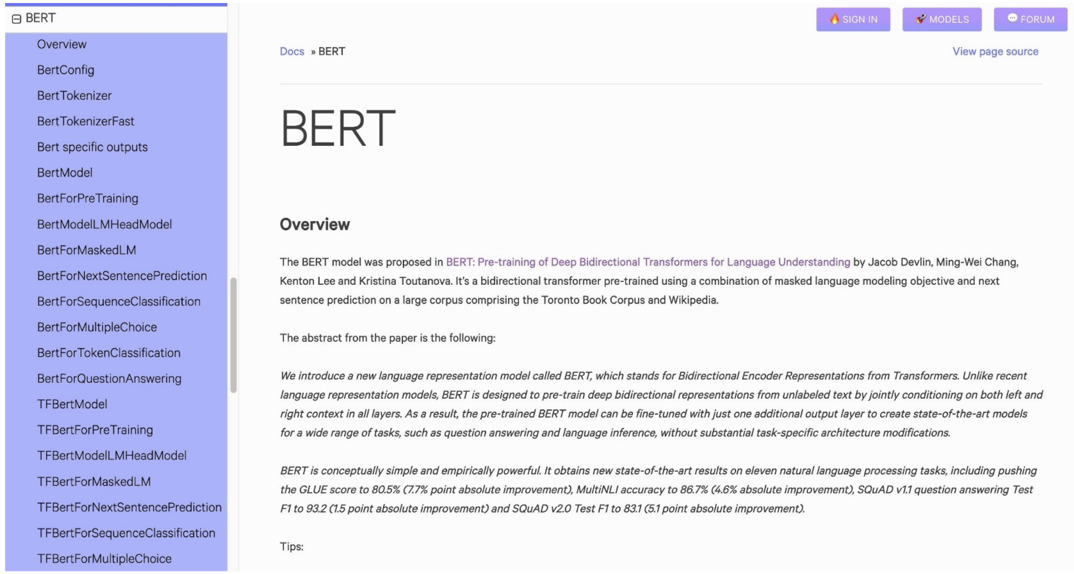
     Special tokens in NSP: *[CLS], [SEP]*

  2. Fine-tuning: (basically tells BERT what to ignore)
     - Presumably teach the model to rely more on the representations useful for the task at hand
     - Possibilities of improving the fine-tuning of BERT:
       1. Taking more layers into account: learning a complementary representation of the information in deep and output layers, using a weighted combination of all layers instead of the final one and layer dropout
       2. Two-stage fine tuning: introduces an intermediate supervised training stage between pre-training and fine-tuning. Ben-David et al. (2020) propose a pivot-based variant of MLM to fine-tune BERT for domain adaptation
       3. Adversarial token perturbations: improve the robustness of the model (Zhu et al., 2019)
       4. Adversarial regularization: helps alleviate pre-trained knowledge forgetting and therefore prevents BERT from overfitting to downstream tasks (Jiang et al., 2019a)
       5. Mixout regularization: improves the stability of BERT fine-tuning even for a small number of training examples (Lee et al., 2019)

BERT – Bidirectional Encoder Representations from Transformers

- Using different open-source models on the **Hugging Face** for fine-tuning

Bert Model Overview on Hugging Face



src: https://huggingface.co/transformers/v3.0.2/model_doc/bert.html

Bert Model Variations

Model	#params	Language
bert-base-uncased	110M	English
bert-large-uncased	340M	English
bert-base-cased	110M	English
bert-large-cased	340M	English
bert-base-chinese	110M	Chinese
bert-base-multilingual-cased	110M	Multiple
bert-large-uncased-whole-word-masking	340M	English
bert-large-cased-whole-word-masking	340M	English

src: https://huggingface.co/bert-base-uncased

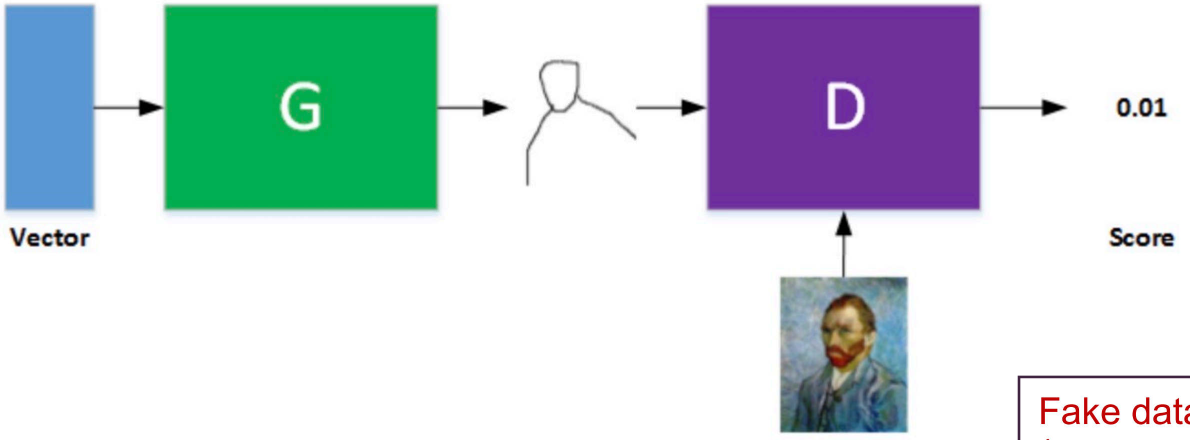
GANs – Generative Adversarial Networks

- An approach to generative modeling using deep learning methods, such as convolutional neural networks, applied in CV
- Automatically discovering and learning the regularities or patterns in input data
- Generate or output new examples that plausibly could have been drawn from the original dataset

Components of GANs:

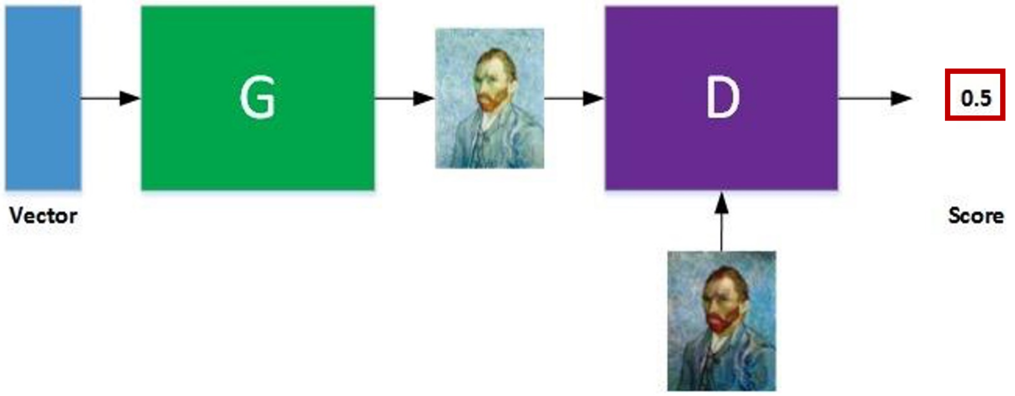
- Generator (G): be trained to generate new examples
- Discriminator (D): tries to classify examples as either real (from the domain) or fake (generated)
- Process: The two models are trained together in a zero-sum game, adversarial, until the discriminator model is fooled about half the time, meaning the generator model is generating plausible examples

GANs – Generative Adversarial Networks



Fake data
(generated, from the Generator)

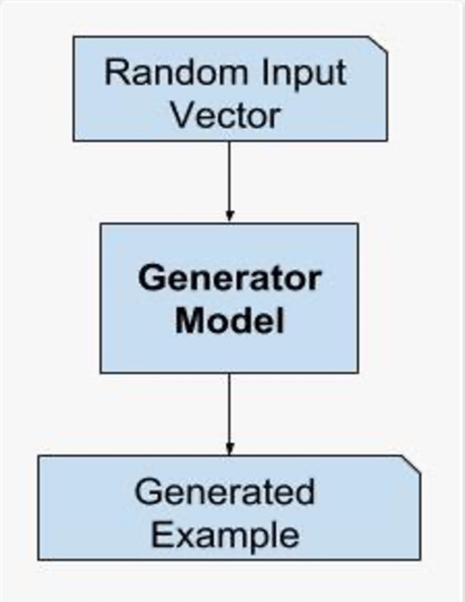
Real data
(from the training dataset/domain)



Endpoint of training

Detailed Illustration of GANs

Generator Model

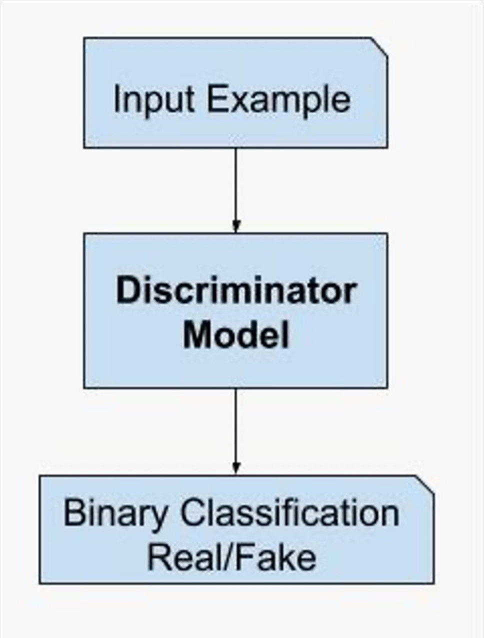


Example of the GAN Generator Model

1. **Takes a fixed-length random vector as input and generates a sample in the domain**
2. **The vector is drawn randomly from a Gaussian distribution, and the vector is used to seed the generative process**
3. **Points in this multidimensional vector space will correspond to points in the problem domain after training, forming a compressed representation of the data distribution**

Detailed Illustration of GANs

Discriminator Model

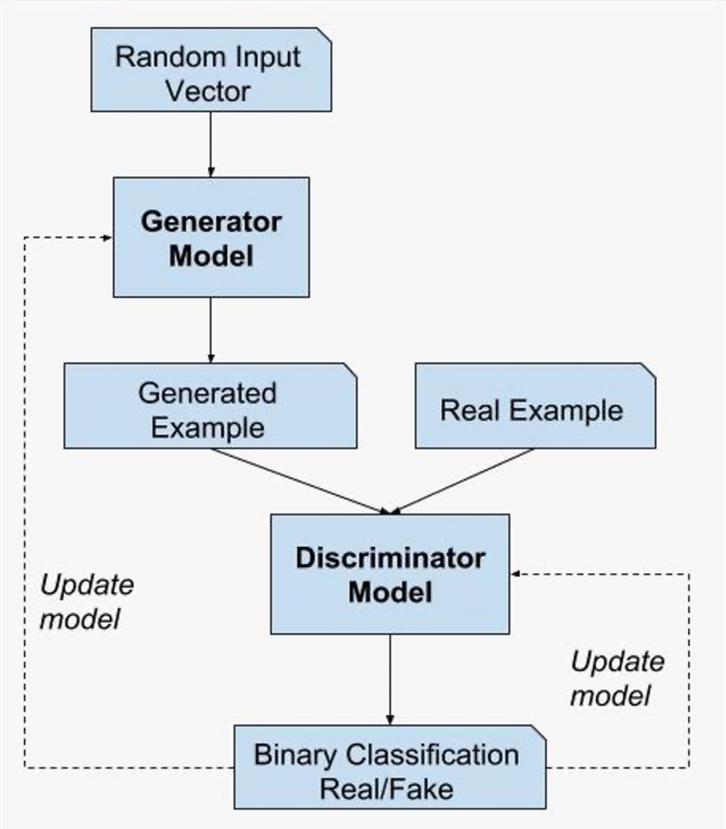


Example of the GAN Discriminator Model

1. The discriminator model takes an example from the domain as input (real or generated) and predicts a binary class label of real or fake (generated)

2. The discriminator is a normal (and well understood) classification model

Adversarial Process

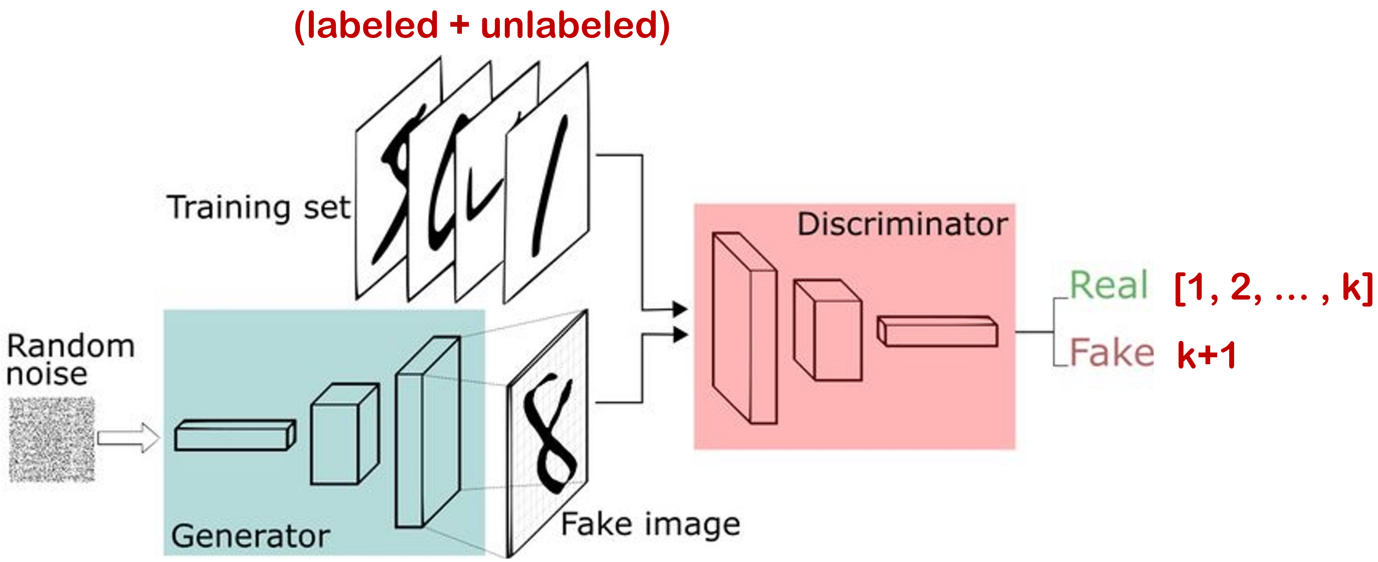


Example of the Generative Adversarial Network Model Architecture

SS-GANs

- **SS-GANs: semi-supervised learning in a GAN framework**
- **A (k + 1)-class objective:**
  - “real” examples are classified into target classes **[1, 2, … , k]**
  - generated / “fake” examples are classified into **k + 1**

- **Components:**
  - Generator (G)
  - Discriminator (D)



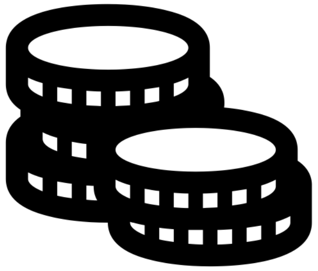
SS-GANs

- Loss of SS-GANs: L_D and L_G

- We will have two Loss functions, one for L(D) and one for L(G):

- $L_D = L_{D_{sup.}} + L_{D_{unsup.}}$

- $L_G = L_{G_{unsup.}} + L_{G_{feature\ matching}}$



SS-GANs

- **First Loss of SS-GANs:** $L_D = L_{D_{sup.}} + L_{D_{unsup.}}$

➢ $L_{D_{sup.}}$ measures the error in assigning the wrong class to a real example among the original k categories.

➢ $L_{D_{unsup.}}$ measures the error in incorrectly recognizing a real (unlabeled) example as fake and not recognizing a fake example.

$$L_{\mathcal{D}_{\text{sup.}}} = -\underbrace{\mathbb{E}_{x,y\sim p_d}}_{\text{real data}} \log \underbrace{[p_{\text{m}}(\hat{y} = y|x, y \in (1, ..., k))]}$$

Probs of associating real data in k classes

$$L_{\mathcal{D}_{\text{unsup.}}} = -\underbrace{\mathbb{E}_{x\sim p_d}}_{\text{real data}} \log \underbrace{[1 - p_{\text{m}}(\hat{y} = y|x, y = k+1)]}$$

$$- \underbrace{\mathbb{E}_{x\sim \mathcal{G}}}_{\text{fake data}} \log \underbrace{[p_{\text{m}}(\hat{y} = y|x, y = k+1)]}$$

Probs of associating fake data into k+1 class

SS-GANs

- **Second Loss of SS-GANs:** $L_G = L_{G_{unsup.}} + L_{G_{feature\ matching}}$

➢ The $L_{G_{unsup.}}$ considers the error induced by fake examples correctly identified by D

$$L_{\mathcal{G}_{unsup.}} = -\mathbb{E}_{x \sim \mathcal{G}} \log[1 - p_m(\hat{y} = y | x, y = k+1)]$$

fake data

Probs of associating real data in k classes

Probs of associating fake data into k+1 classes

SS-GANs

- **Second Loss of SS-GANs:** $L_G = L_{G_{unsup.}} + L_{G_{feature\ matching}}$

➢ **Reason for feature matching**: *G* is expected to generate examples that are similar to the ones sampled from the real distribution *P(d)*.

➢ **Feature matching loss calculation**: Let *f(x)* denote the activation on an intermediate layer of D; the generator should produce examples whose intermediate representations provided in input to *D* are very similar to the real ones.

fake data distribution in input to D

$$L_{G_{\text{feature matching}}} = \left\| \underbrace{\mathbb{E}_{x \sim p_d} f(x)} - \underbrace{\mathbb{E}_{x \sim g} f(x)} \right\|_2^2$$

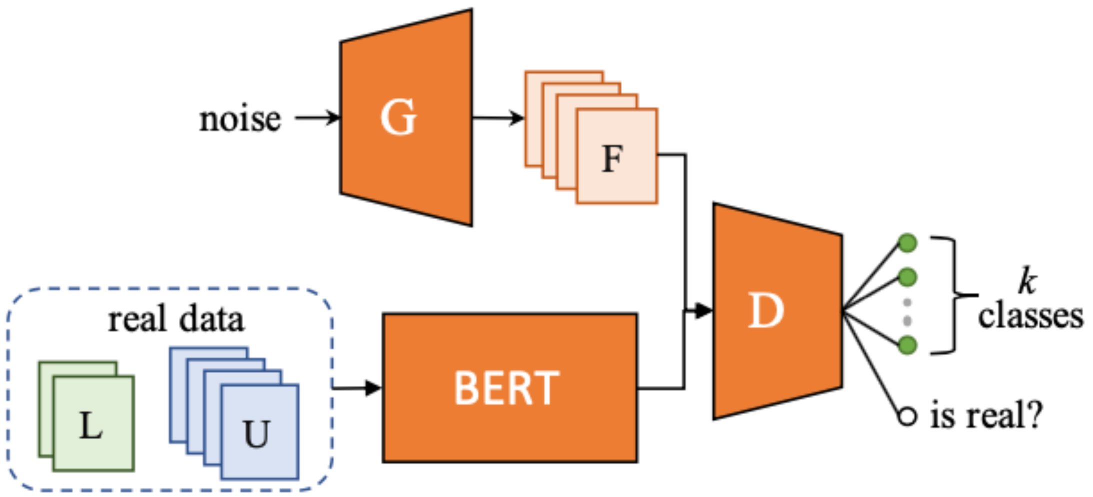
real data distribution in input to D

GAN-BERT

- **GAN-Bert**: an extension of pre-trained BERT model by using SS-GANs for the fine-tuning stage.
- **Two components:**

➢ task-specific layers, as in the usual BERT fine-tuning

➢ SS-GAN layers to enable semi-supervised learning

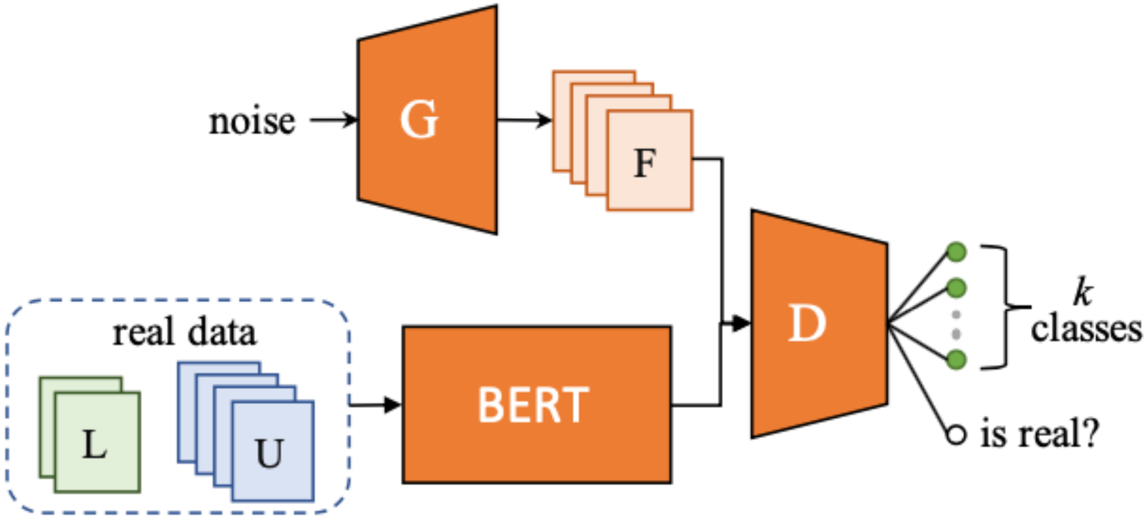


GAN-BERT

- **GAN-BERT architecture:**

➢ G generates a set of fake examples F given a random distribution.

➢ Fake examples F, along with unlabeled U and labeled L vector representations computed by BERT are used as input for the discriminator D.



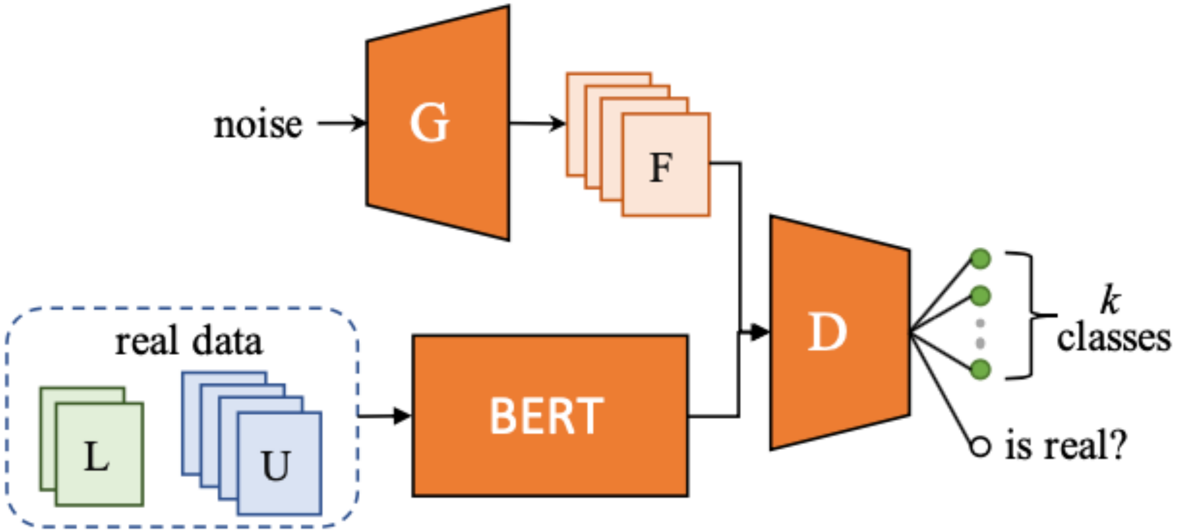
GAN-BERT

- **Loss of GAN-BERT (as of SS-GANs):**

➢ $L_D = L_{D_{sup.}} + L_{D_{unsup.}}$

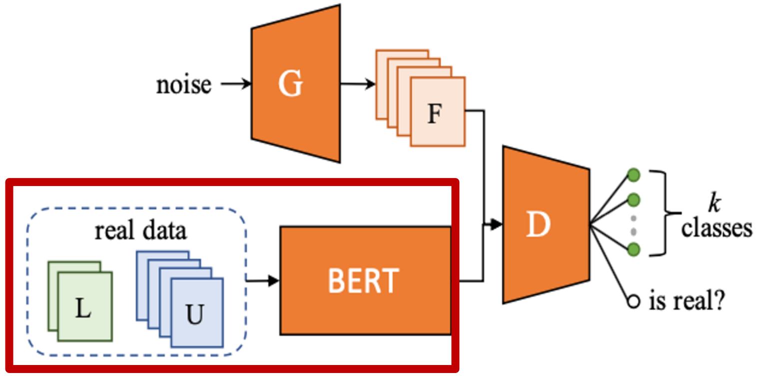
➢ $L_G = L_{G_{unsup.}} + L_{G_{feature\ matching}}$

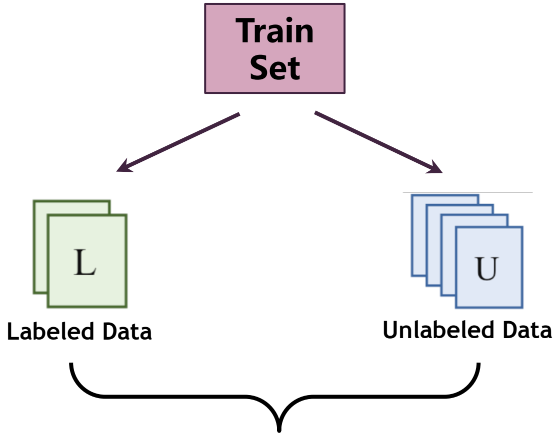
- **Weight update: G, D, BERT**



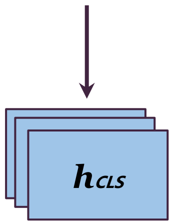
GAN-BERT

- **Pre-trained BERT model:**






```
from transformers import AutoModel, AutoTokenizer

model_name = 'bert-base-cased'  # for English tasks
# model_name = 'bert-base-multilingual-cased'   # for multilingual tasks
transformer = AutoModel.from_pretrained(model_name)
tokenizer = AutoTokenizer.from_pretrained(model_name)
```



- Given input sentence:

$$s = (t_1, ..., t_n)$$

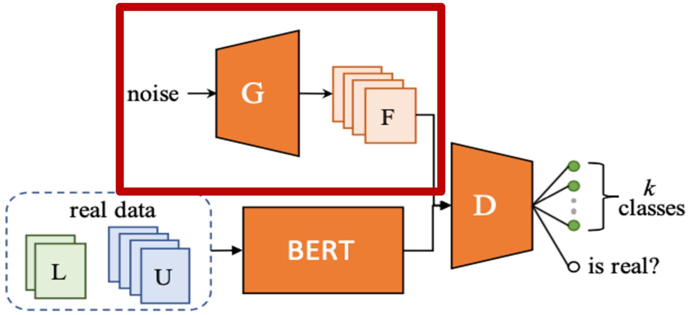
- **BERT** output n+2 vector representations:

$$(h_{CLS}, h_{t_1}, ..., h_{t_n}, h_{SEP})$$

- We adopt the h_{cls} representation as a sentence embedding

GAN-BERT

- **Generator G:**



- **MLP** with one hidden layer
- Activated by a **leaky-relu** function

Noise Vector

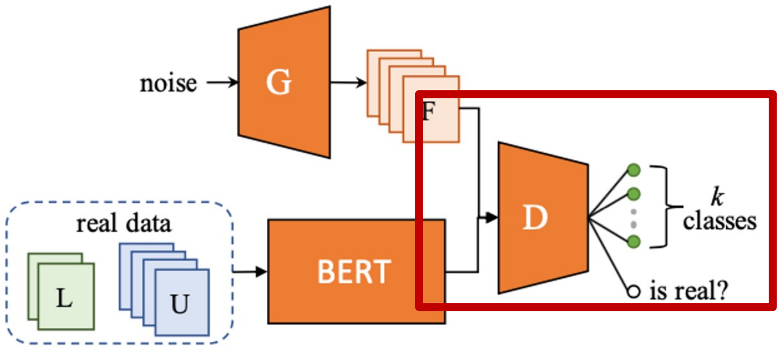
100-dimensional drawn from $N(\mu, \sigma^2)$

```
class Generator(nn.Module):
    def __init__(self, noise_size=100, output_size=512, hidden_sizes=None, dropout_rate=0.1):
        super(Generator, self).__init__()
        if hidden_sizes is None:
            hidden_sizes = [512]
        layers = []
        hidden_sizes = [noise_size] + hidden_sizes
        for i in range(len(hidden_sizes) - 1):
            layers.extend([nn.Linear(hidden_sizes[i], hidden_sizes[i + 1]), nn.LeakyReLU(0.2, inplace=True),
                           nn.Dropout(dropout_rate)])

        layers.append(nn.Linear(hidden_sizes[-1], output_size))
        self.layers = nn.Sequential(*layers)

    def forward(self, noise):
        output_rep = self.layers(noise)
        return output_rep
```

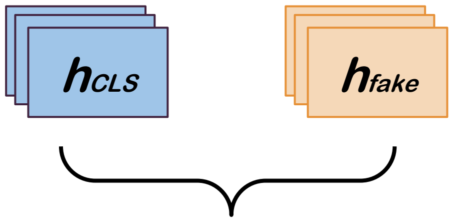
Fake Vector

GAN-BERT

- **Discriminator D:**



- **MLP** with one hidden layer
- Activated by a **leaky-relu** function
- Followed by a **softmax** layer for the final prediction



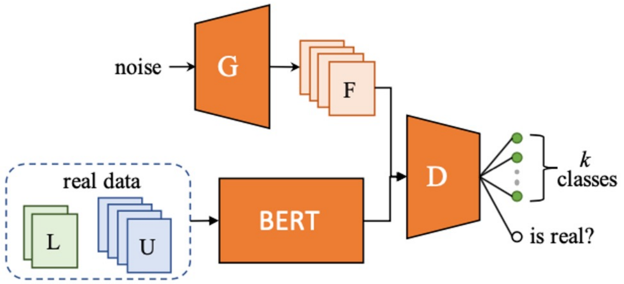

```python
class Discriminator(nn.Module):
    def __init__(self, input_size=512, hidden_sizes=None, num_labels=2, dropout_rate=0.1):
        super(Discriminator, self).__init__()
        if hidden_sizes is None:
            hidden_sizes = [512]
        self.input_dropout = nn.Dropout(p=dropout_rate)
        layers = []
        hidden_sizes = [input_size] + hidden_sizes
        for i in range(len(hidden_sizes) - 1):
            layers.extend([nn.Linear(hidden_sizes[i], hidden_sizes[i + 1]), nn.LeakyReLU(0.2, inplace=True),
                           nn.Dropout(dropout_rate)])

        self.layers = nn.Sequential(*layers)  # per il flatten
        self.logit = nn.Linear(hidden_sizes[-1],
                               num_labels + 1)  # +1 for the probability of this sample being fake/real.
        self.softmax = nn.Softmax(dim=-1)

    def forward(self, input_rep):
        input_rep = self.input_dropout(input_rep)
        last_rep = self.layers(input_rep)
        logits = self.logit(last_rep)
        probs = self.softmax(logits)
        return last_rep, logits, probs
```

Final Prediction

k+1 category

GAN-BERT

Training Procedure:



1. Encode real data in Transformer

```
# Encode real data in the Transformer
model_outputs = transformer(b_input_ids, attention_mask=b_input_mask)
hidden_states = model_outputs[-1]
```

2. Generate fake data

```
# Generate fake data that should have the same distribution of the ones
# encoded by the transformer.
# First noisy input are used in input to the Generator
noise = torch.zeros(real_batch_size, noise_size, device=device).uniform_(0, 1)
# Gnerate Fake data
gen_rep = generator(noise)
```

3. Put real and fake data into Discriminator

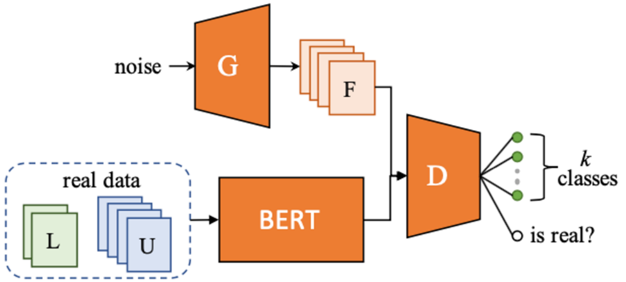
```
# Generate the output of the Discriminator for real and fake data.
# First, we put together the output of the tranformer and the generator
discriminator_input = torch.cat([hidden_states, gen_rep], dim=0)
# Then, we select the output of the discriminator
features, logits, probs = discriminator(discriminator_input)
```

4. Separate the Discriminator’s outpu for real and fake data

```
# Finally, we separate the discriminator's output for the real and fake
# data
features_list = torch.split(features, real_batch_size)
D_real_features = features_list[0]
D_fake_features = features_list[1]

logits_list = torch.split(logits, real_batch_size)
D_real_logits = logits_list[0]
D_fake_logits = logits_list[1]

probs_list = torch.split(probs, real_batch_size)
D_real_probs = probs_list[0]
D_fake_probs = probs_list[1]
```

GAN-BERT

Loss of GAN-BERT (as of SS-GANs):



$$L_G = L_{G_{unsup.}} + L_{G_{feature\ matching}}$$

```
# Generator's LOSS estimation
g_loss_d = -1 * torch.mean(torch.log(1 - D_fake_probs[:, -1] + self.epsilon))
g_feat_reg = torch.mean(
    torch.pow(torch.mean(D_real_features, dim=0) - torch.mean(D_fake_features, dim=0), 2))
g_loss = g_loss_d + g_feat_reg

# Discriminator's LOSS estimation
logits = D_real_logits[:, 0:-1]
log_probs = F.log_softmax(logits, dim=-1)
# The discriminator provides an output for labeled and unlabeled real data
# so the loss evaluated for unlabeled data is ignored (masked)
label2one_hot = torch.nn.functional.one_hot(b_labels, len(label_list))
per_example_loss = -torch.sum(label2one_hot * log_probs, dim=-1)
per_example_loss = torch.masked_select(per_example_loss, b_label_mask.to(self.device))
labeled_example_count = per_example_loss.type(torch.float32).numel()

# It may be the case that a batch does not contain labeled examples,
# so the "supervised loss" in this case is not evaluated
if labeled_example_count == 0:
    D_L_Supervised = 0
else:
    D_L_Supervised = torch.div(torch.sum(per_example_loss.to(self.device)), labeled_example_count)

D_L_unsupervised1U = -1 * torch.mean(torch.log(1 - D_real_probs[:, -1] + self.epsilon))
D_L_unsupervised2U = -1 * torch.mean(torch.log(D_fake_probs[:, -1] + self.epsilon))
d_loss = D_L_Supervised + D_L_unsupervised1U + D_L_unsupervised2U
```

$$L_D = L_{D_{sup.}} + L_{D_{unsup.}}$$

Weight update: G, D and BERT

Experimental Part

Goal:

- Assess the impact of **GAN-BERT** over sentence classification tasks characterized by different training conditions, i.e., number of examples and number of categories.
  - Binary Classification (Sentiment Analysis)
  - Multi-Class Classification (Author Identification)

Repeat the training of each model with an increasing set of annotated material (*L*)

Dataset:

- Sentiment (*40,000* texts for Train set / *10,000* texts for Test set)
- Letters (*39,077* texts for Train set / *4,881* texts for Test set)

Baseline:

- **BERT-base** model fine-tuned as described in [Devlin et al., 2019] on the available training material

Hyperparameters:

Transformer parameters:	GAN-BERT specific parameters:	Optimization parameters:
> Max Sequence Length = 64 > Batch Size = 64	> Number of Hidden Layers in G = 1 > Number of Hidden Layers in D = 1 > Size of Noisy Vector = 100 > Dropout Rate = 0.2	> Learning Rate of D = 5e-5 > Learning Rate of G = 5e-5 > Epsilon = 1e-8

Experiment 1: Sentiment Analysis

Dataset: Sentiment
Label: Sentiment (Negative / Positive)

40, 000 training data
10, 000 testing data

- Training set:

	Text	Sentiment
count	40000	40000
unique	39745	2

- Neg: 20015
- Pos: 19985

- Test set:

	Text	Sentiment
count	10000	10000
unique	9983	2

- Neg: 4985
- Pos: 5015

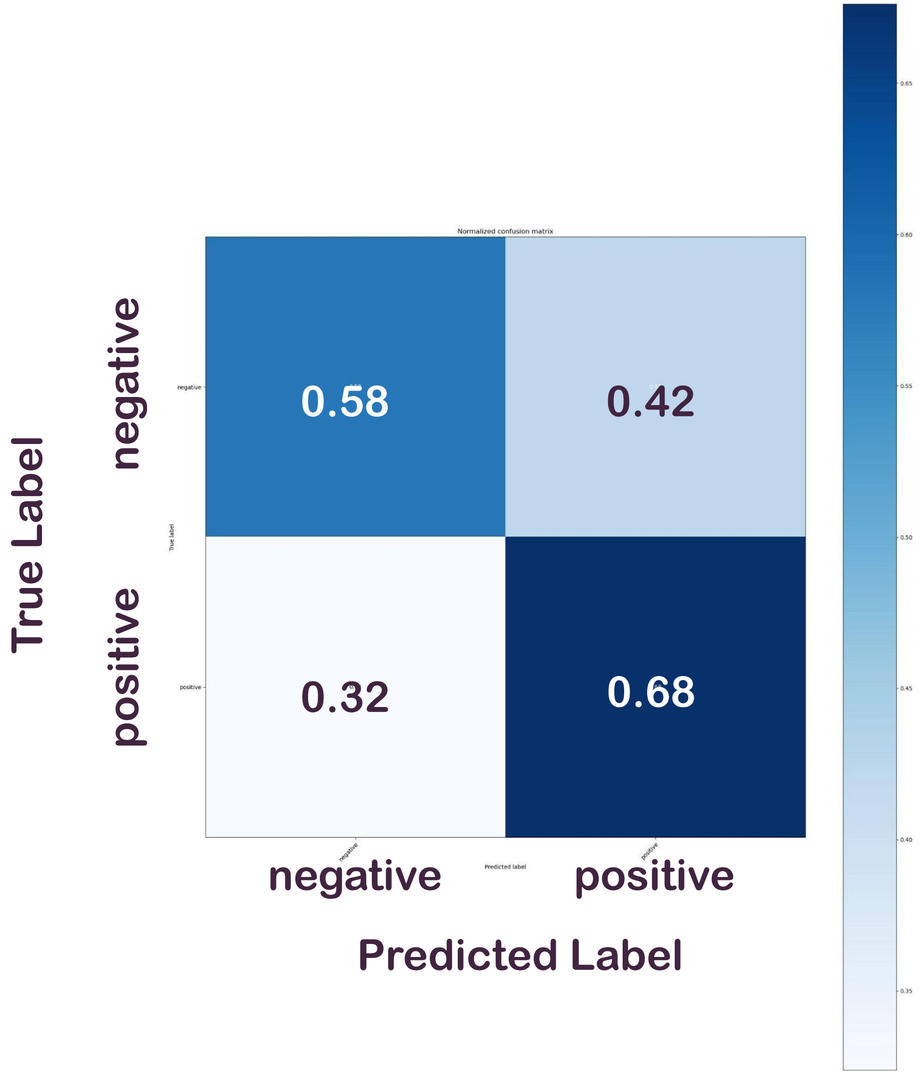
Experiment 1: Sentiment Analysis

Annotated Data: 0.1% (40 texts)

BERT: BERT-base

Epochs: 10

-	precision	recall	f1-score	support
negative	0.640	0.580	0.608	4985
positive	0.618	0.676	0.646	5015
accuracy			0.628	10000
macro avg	0.629	0.628	0.627	10000
weighted avg	0.629	0.628	0.627	10000



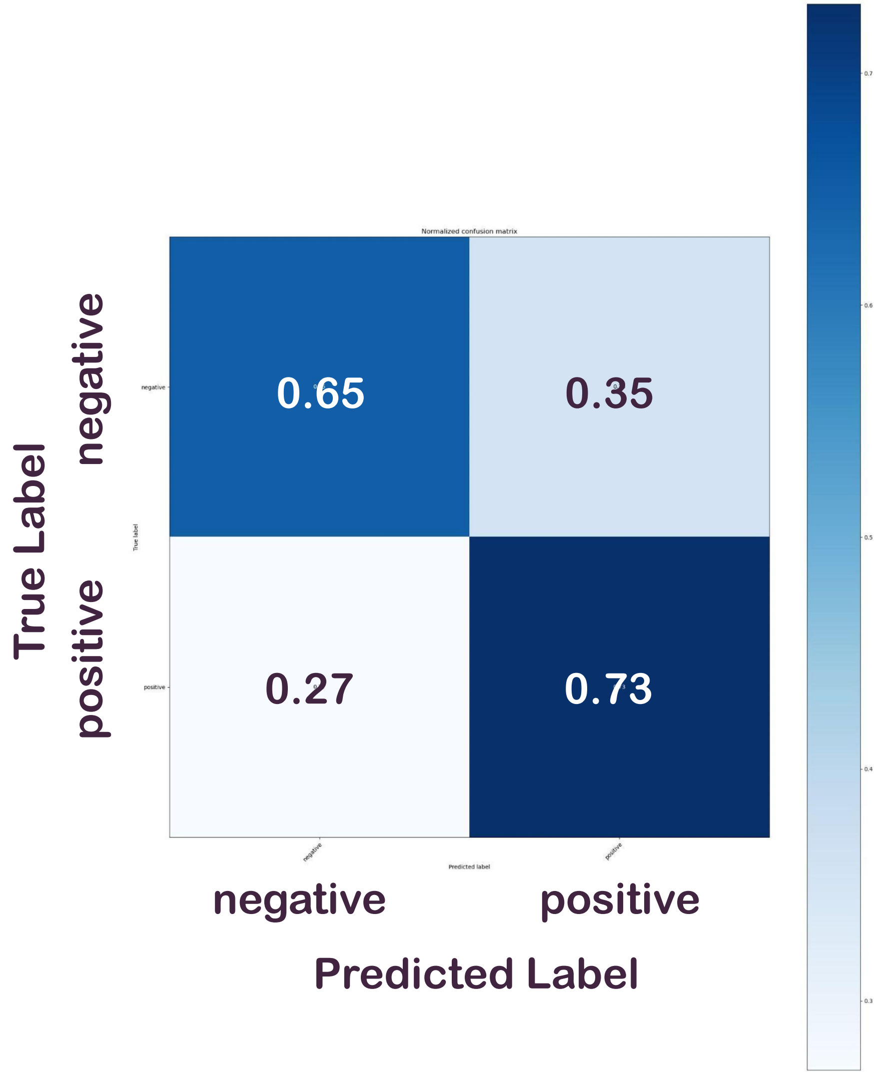
Experiment 1: Sentiment Analysis

Annotated Data: 0.25% (100 texts)

BERT: BERT-base

Epochs: 10

-	precision	recall	f1-score	support
negative	0.704	0.647	0.674	4985
positive	0.675	0.730	0.701	5015
accuracy			0.689	10000
macro avg	0.690	0.688	0.688	10000
weighted avg	0.690	0.689	0.688	10000



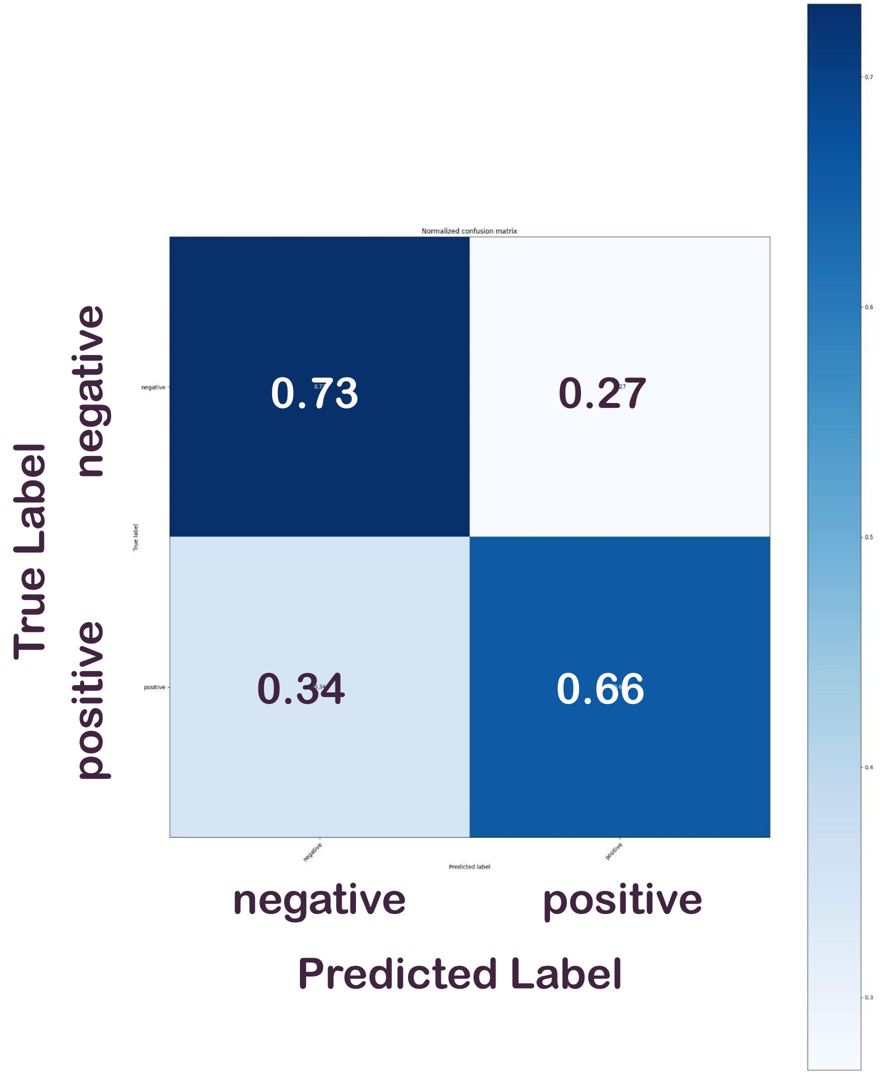
Experiment 1: Sentiment Analysis

Annotated Data: 0.5% (200 texts)

BERT: BERT-base

Epochs: 10

-	precision	recall	f1-score	support
negative	0.678	0.732	0.704	4985
positive	0.711	0.655	0.682	5015
accuracy			0.693	10000
macro avg	0.695	0.693	0.693	10000
weighted avg	0.695	0.693	0.693	10000



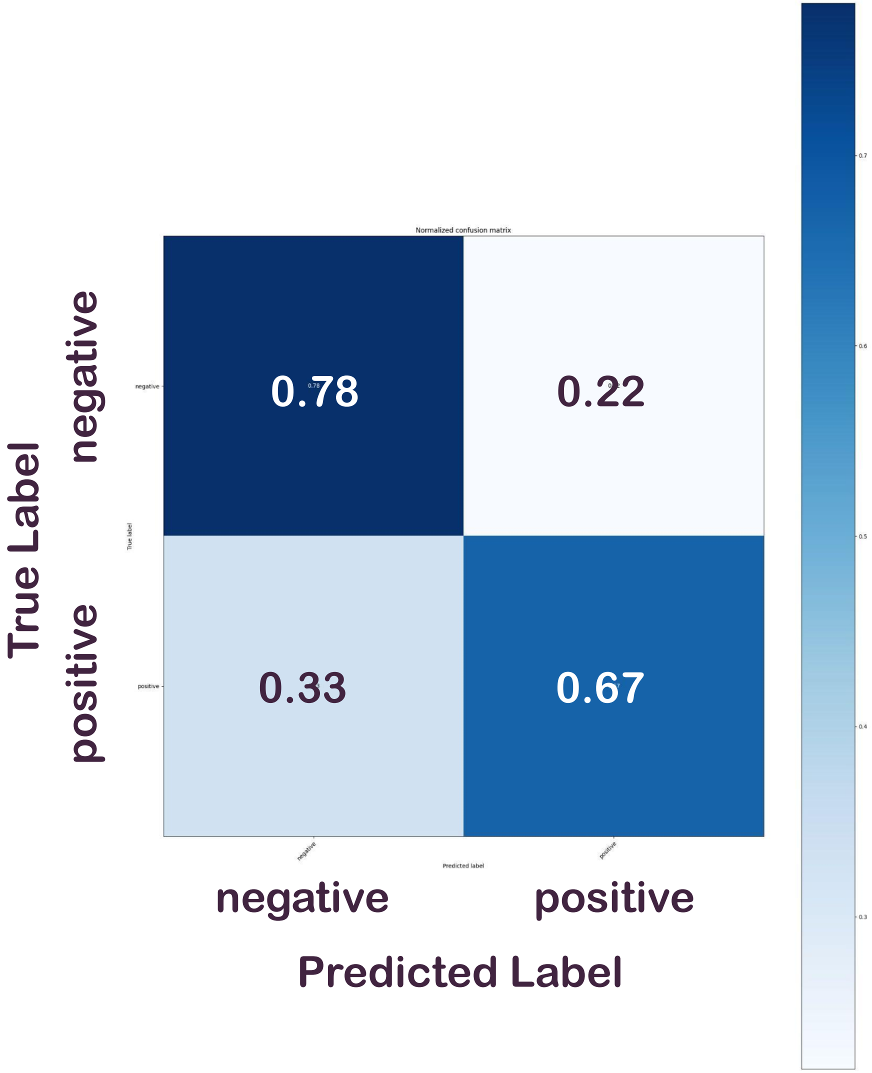
Experiment 1: Sentiment Analysis

Annotated Data: 1% (400 texts)

BERT: BERT-base

Epochs: 10

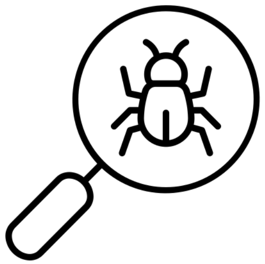
-	precision	recall	f1-score	support
negative	0.701	0.780	0.739	4985
positive	0.754	0.670	0.709	5015
accuracy			0.725	10000
macro avg	0.728	0.725	0.724	10000
weighted avg	0.728	0.725	0.724	10000

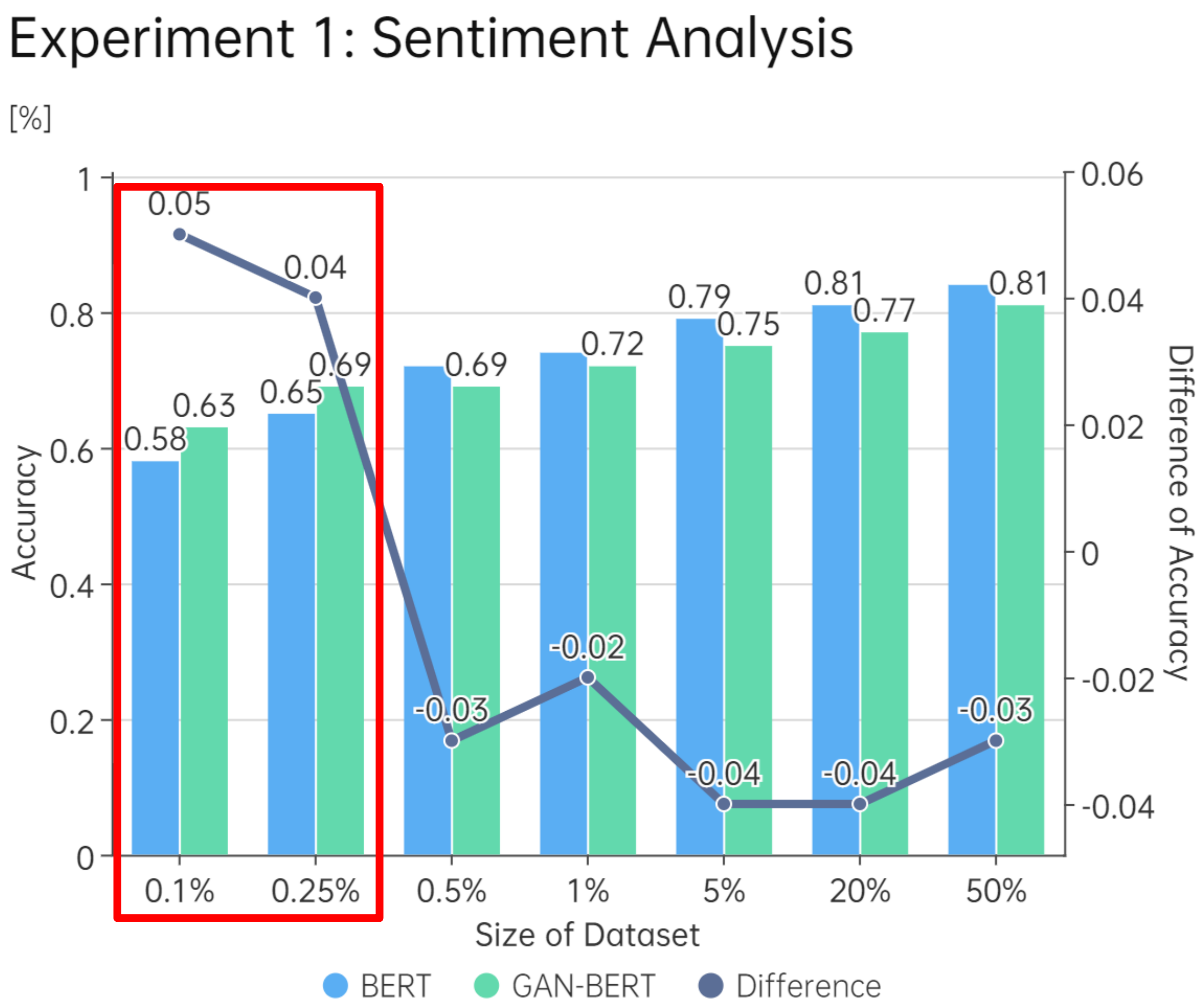


Results

GAN-BERT performs better than BERT:

with 0.1% and 0.25% of labeled training data



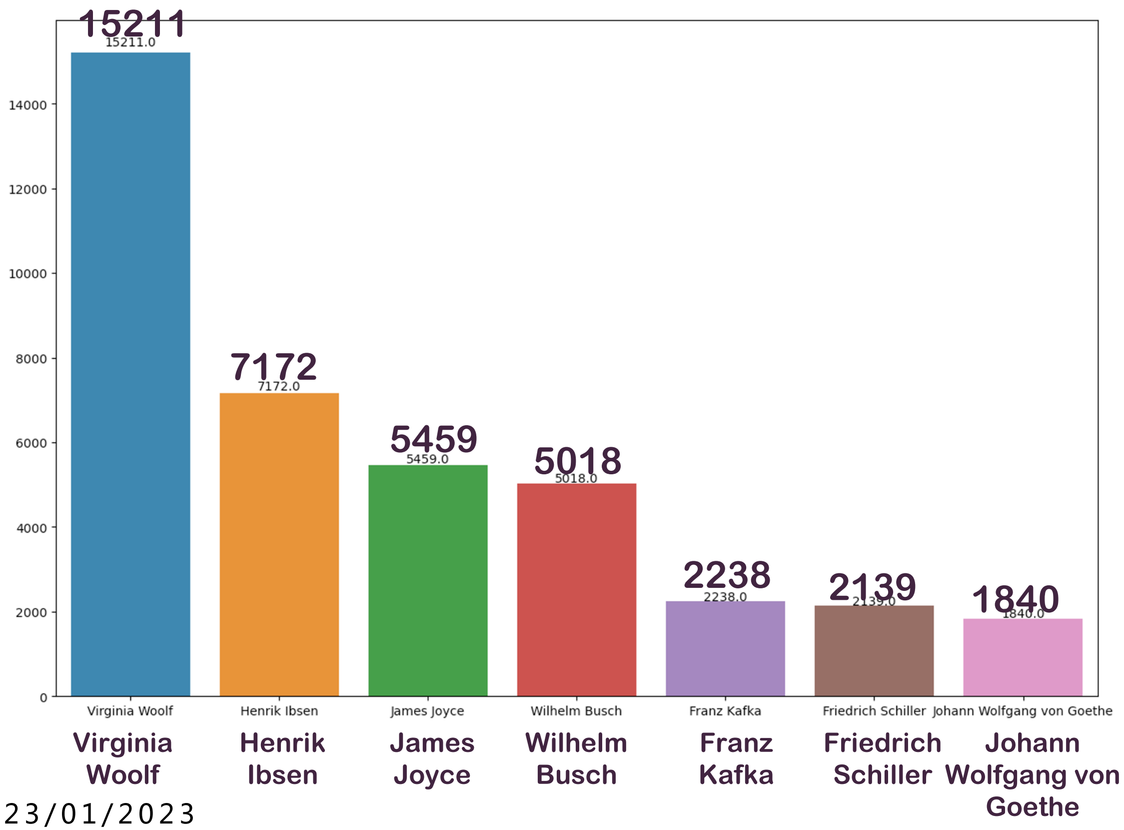


Experiment 2: Author Identification

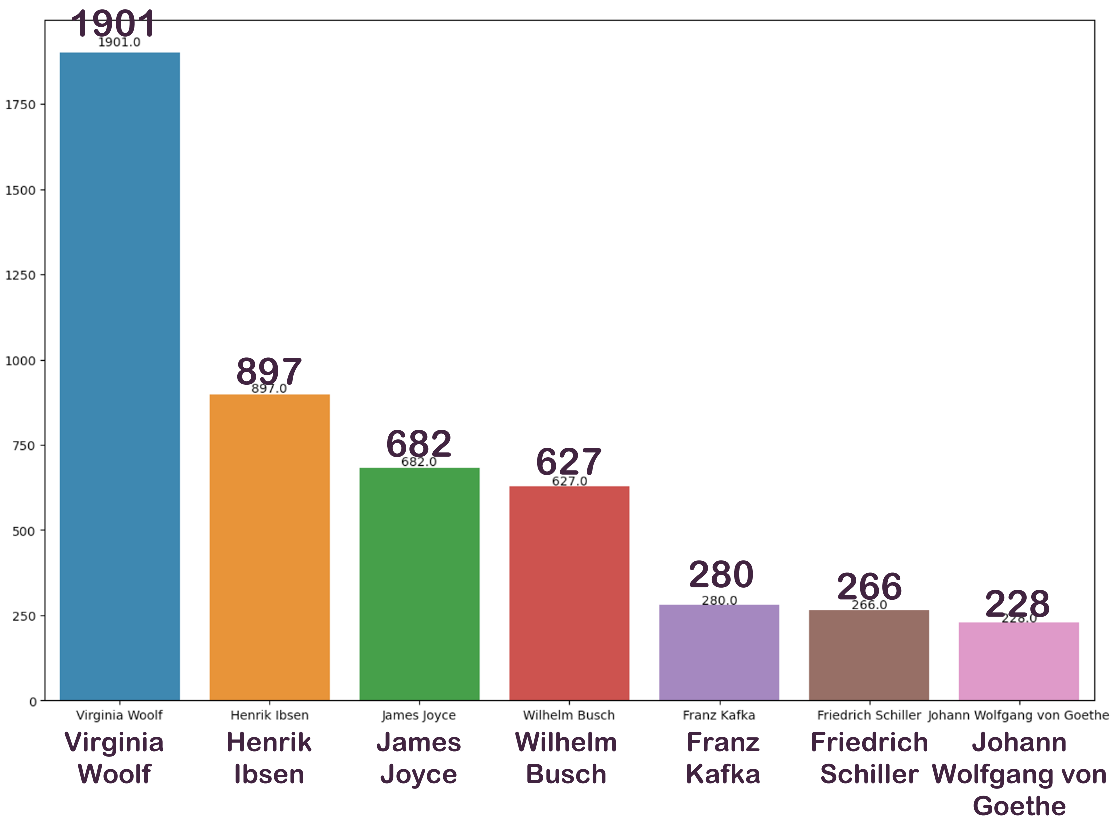
Dataset: **Letters**
Label: **Authors**

39, 077 training data
4, 881 testing data

- Training set:



- Test set:



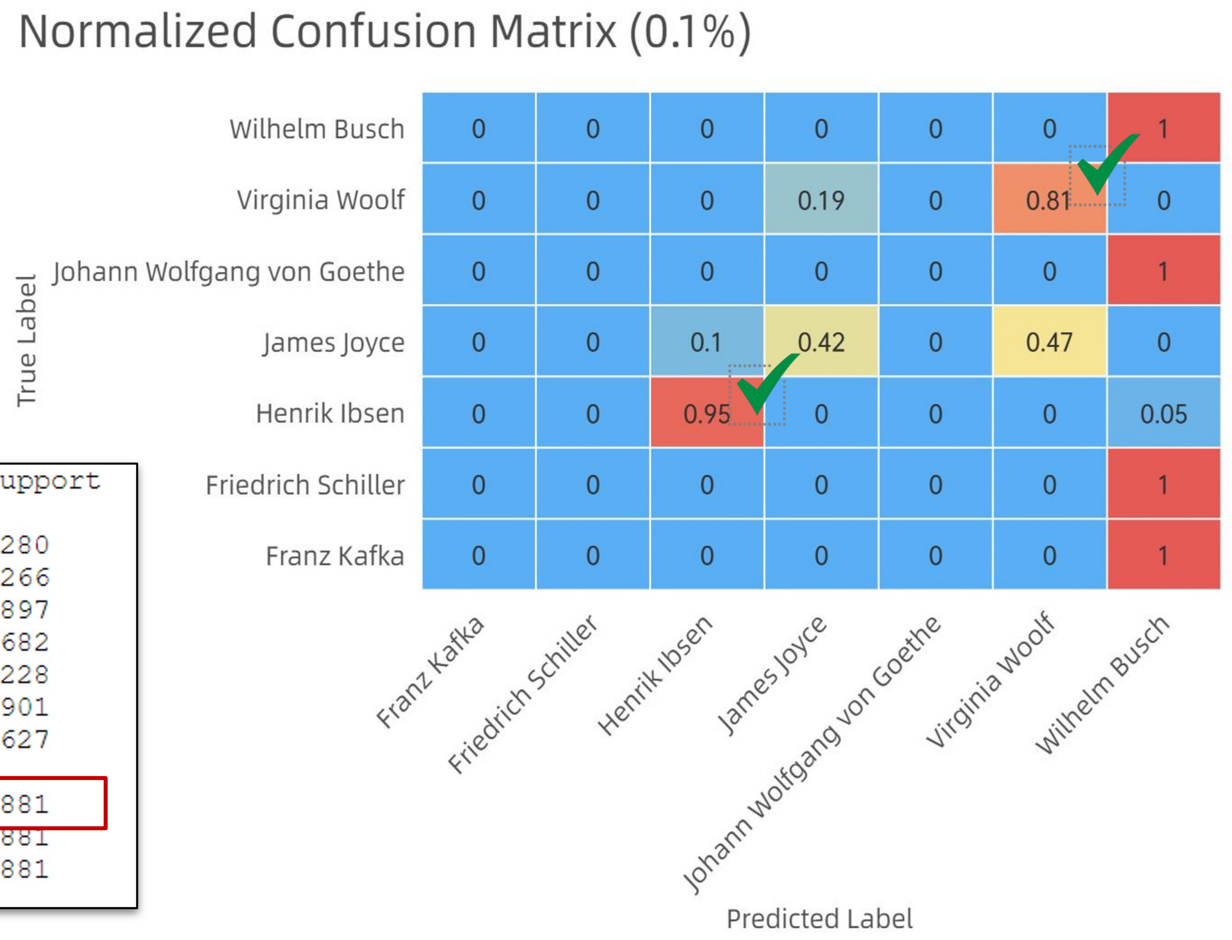
Experiment 2: Author Identification

Annotated Data: 0.1% (39 texts)

BERT: mBERT

Epochs: 3

-	precision	recall	f1-score	support
Franz Kafka	0.000	0.000	0.000	280
Friedrich Schiller	0.000	0.000	0.000	266
Henrik Ibsen	0.924	0.949	0.936	897
James Joyce	0.438	0.422	0.430	682
Johann Wolfgang von Goethe	0.000	0.000	0.000	228
Virginia Woolf	0.826	0.806	0.816	1901
Wilhelm Busch	0.433	1.000	0.604	627
accuracy			0.676	4881
macro avg	0.374	0.454	0.398	4881
weighted avg	0.608	0.676	0.628	4881

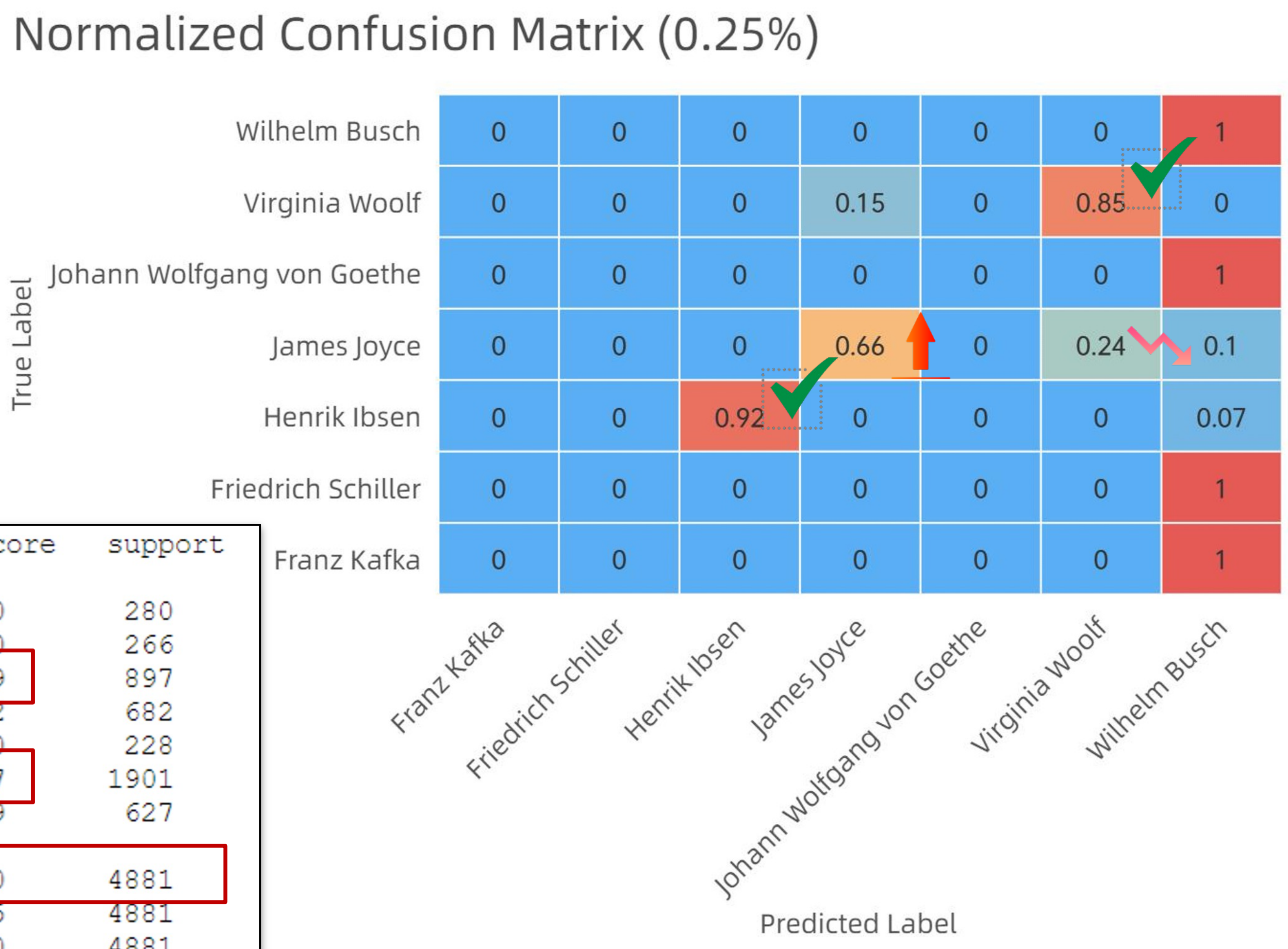


Experiment 2: Author Identification

Annotated Data: 0.25% (97 texts)

BERT: mBERT

Epochs: 3

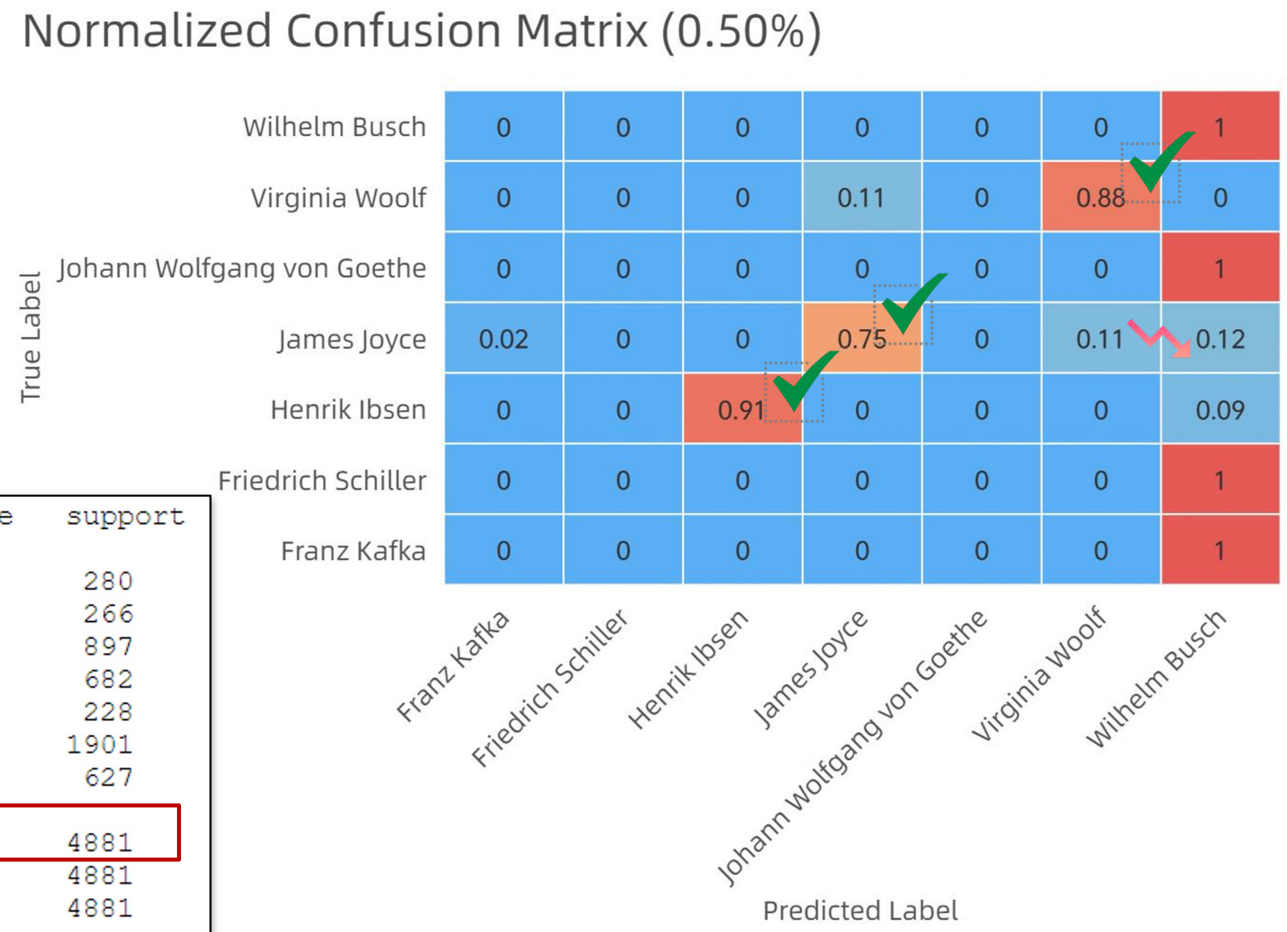


-	precision	recall	f1-score	support
Franz Kafka	0.000	0.000	0.000	280
Friedrich Schiller	0.000	0.000	0.000	266
Henrik Ibsen	0.998	0.924	0.959	897
James Joyce	0.607	0.658	0.632	682
Johann Wolfgang von Goethe	0.000	0.000	0.000	228
Virginia Woolf	0.909	0.847	0.877	1901
Wilhelm Busch	0.408	1.000	0.579	627
accuracy			0.720	4881
macro avg	0.417	0.490	0.435	4881
weighted avg	0.674	0.720	0.680	4881

Experiment 2: Author Identification

Annotated Data: 0.5% (195 texts)
BERT: mBERT
Epochs: 3

-	precision	recall	f1-score	support
Franz Kafka	0.000	0.000	0.000	280
Friedrich Schiller	0.000	0.000	0.000	266
Henrik Ibsen	0.995	0.905	0.948	897
James Joyce	0.704	0.746	0.725	682
Johann Wolfgang von Goethe	0.000	0.000	0.000	228
Virginia Woolf	0.956	0.884	0.919	1901
Wilhelm Busch	0.400	1.000	0.571	627
accuracy			0.743	4881
macro avg	0.436	0.505	0.452	4881
weighted avg	0.705	0.743	0.707	4881

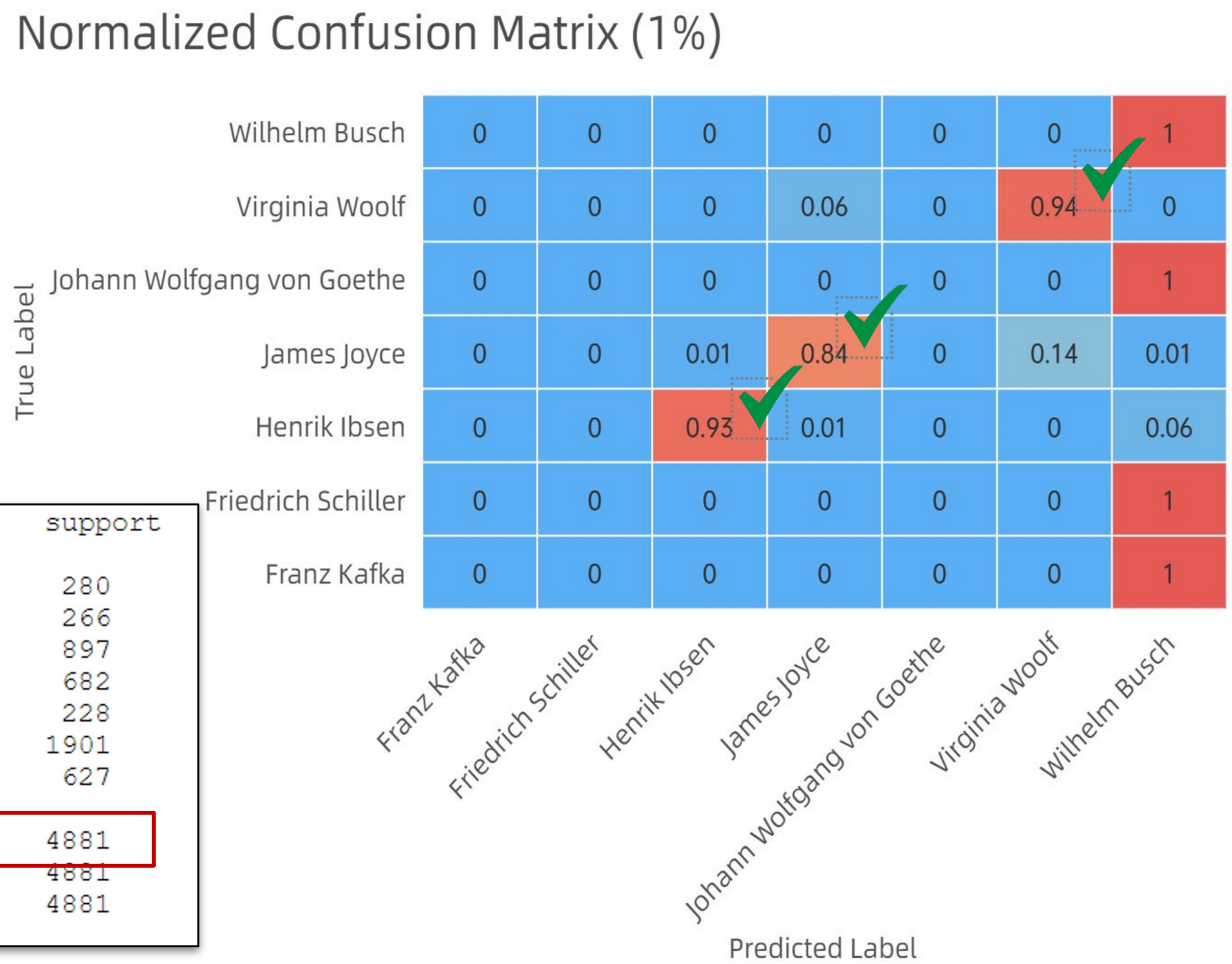


Experiment 2: Author Identification

Annotated Data: 1% (390 texts)

BERT: mBERT

Epochs: 3



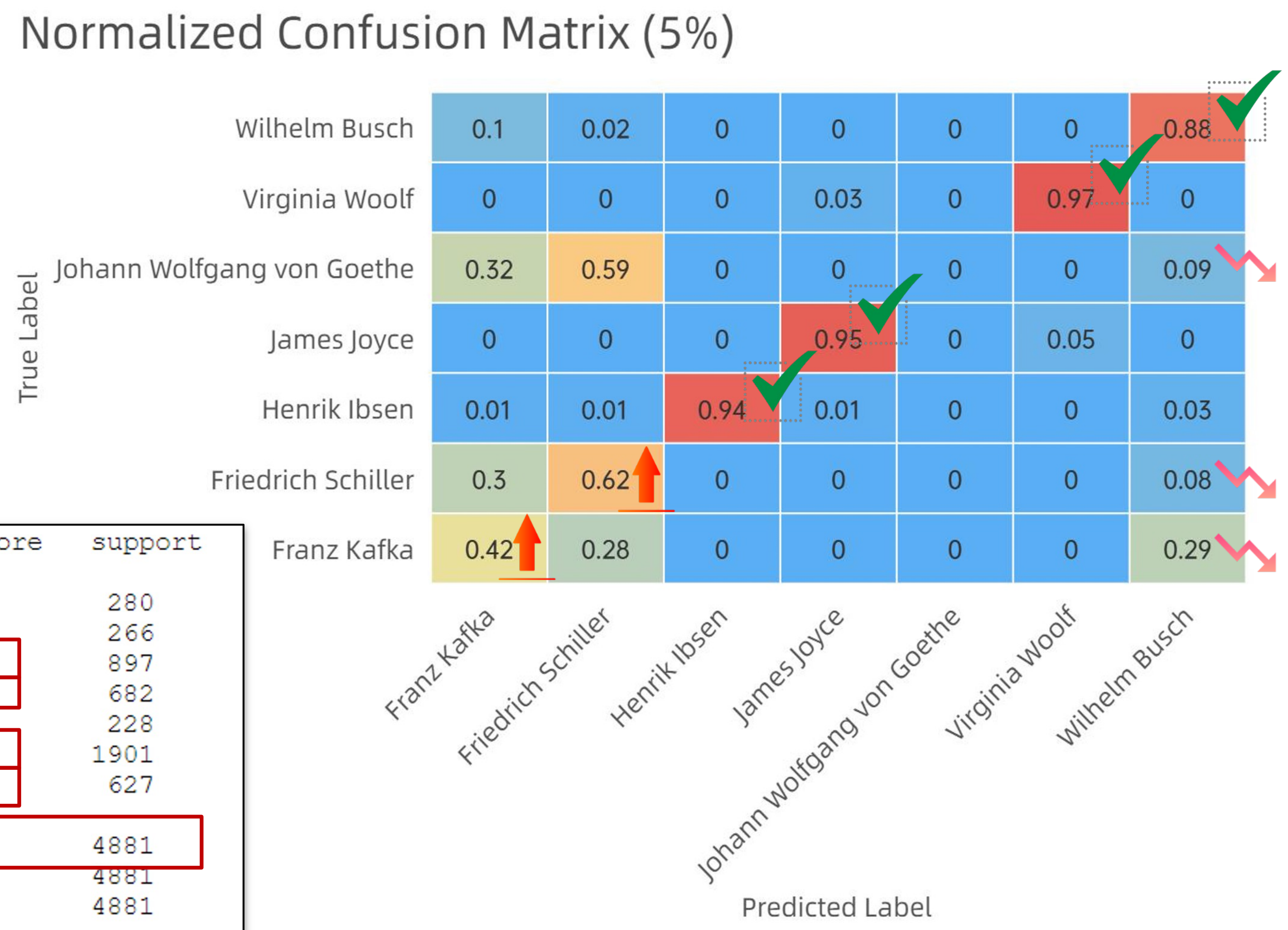
-	precision	recall	f1-score	support
Franz Kafka	0.000	0.000	0.000	280
Friedrich Schiller	0.000	0.000	0.000	266
Henrik Ibsen	0.994	0.933	0.963	897
James Joyce	0.820	0.840	0.830	682
Johann Wolfgang von Goethe	0.000	0.000	0.000	228
Virginia Woolf	0.948	0.937	0.942	1901
Wilhelm Busch	0.429	1.000	0.601	627
accuracy			0.782	4881
macro avg	0.456	0.530	0.476	4881
weighted avg	0.722	0.782	0.737	4881

Experiment 2: Author Identification

Annotated Data: 5% (1953 texts)

BERT: mBERT

Epochs: 3

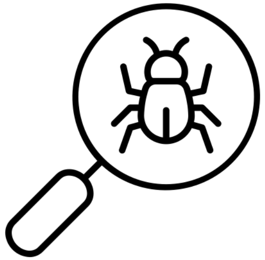


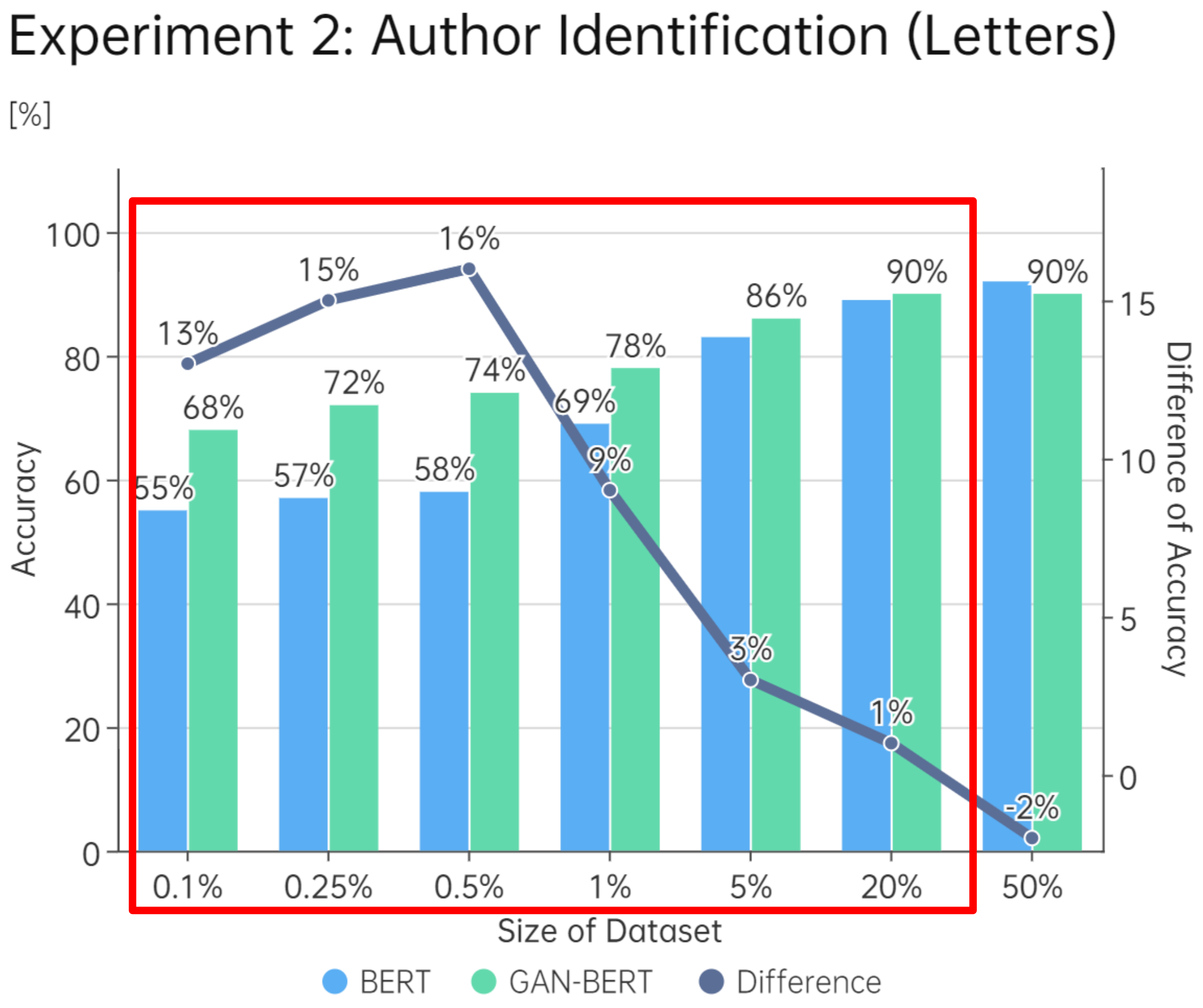
-	precision	recall	f1-score	support
Franz Kafka	0.352	0.425	0.385	280
Friedrich Schiller	0.411	0.620	0.495	266
Henrik Ibsen	0.996	0.944	0.970	897
James Joyce	0.917	0.946	0.931	682
Johann Wolfgang von Goethe	0.000	0.000	0.000	228
Virginia Woolf	0.982	0.974	0.978	1901
Wilhelm Busch	0.783	0.879	0.828	627
accuracy			0.856	4881
macro avg	0.635	0.684	0.655	4881
weighted avg	0.837	0.856	0.845	4881

Results

GAN-BERT performs **better** than **BERT**:

with 0.1% (39 cases),
0.25% (97 cases),
0.5% (195 cases),
1% (390 cases),
5% (1953 cases),
20% of Dataset

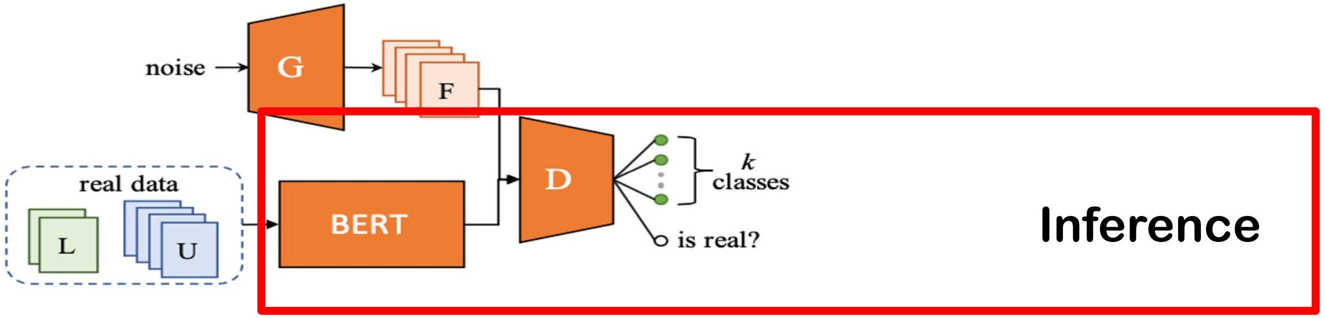




Conclusion

- GAN-BERT improved the robustness of fine-tuning
  - even in poor training condition (low training data)
  - (intuitively) GAN provide massive negative samples during training

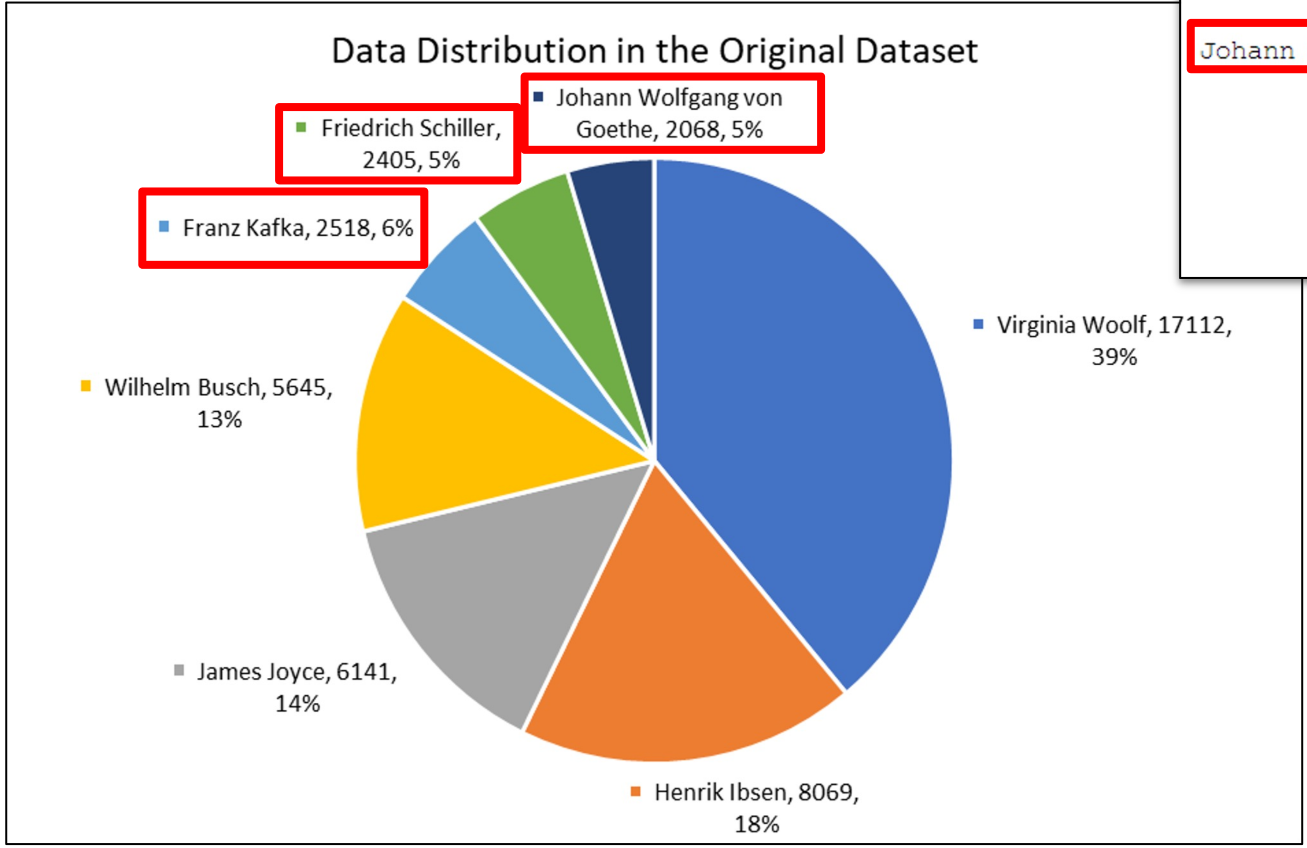
- GAN-BERT does not introduce **additional cost** during inference
  - Only additional discriminator (simple)



- More improvement of GAN-BERT is more significant on **Multiclass** Classification than **Binary** Classification

Conclusion

- **Why Kafka, Schiller, Goethe performs not well?**
  - **size of training data is limited**



result of GAN-BERT on 5% of dataset

-	precision	recall	f1-score	support
Franz Kafka	0.352	0.425	0.385	280
Friedrich Schiller	0.411	0.620	0.495	266
Henrik Ibsen	0.996	0.944	0.970	897
James Joyce	0.917	0.946	0.931	682
Johann Wolfgang von Goethe	0.000	0.000	0.000	228
Virginia Woolf	0.982	0.974	0.978	1901
Wilhelm Busch	0.783	0.879	0.828	627
accuracy			0.856	4881
macro avg	0.635	0.684	0.655	4881
weighted avg	0.837	0.856	0.845	4881

Limitation:
- **Carefully separate dataset**

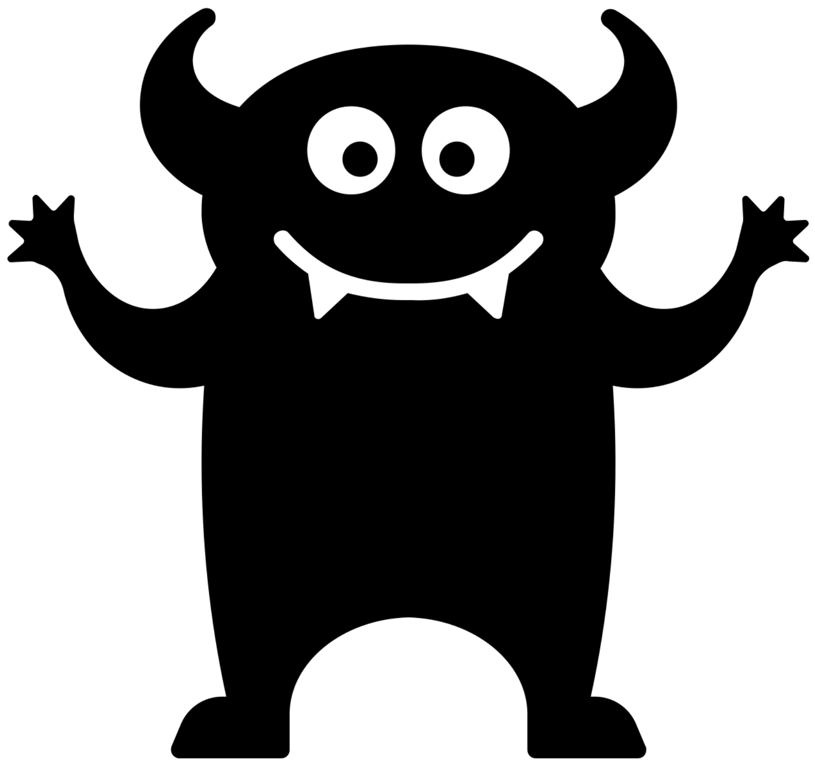
Outlook:
- **Long-Tail Distribution Problem**

Conclusion

- BERT
- GAN (Generative Adversarial Networks)
- GAN-BERT: Fine-Tune of BERT with GAN
- Experiments
  - Sentiment Analysis
  - Author Identification

- Advantage of GAN-BERT
  - GAN provides a large number of negative samples
  - Robustness with low training data
  - Not introducing additional costs during inference

- Limitation
  - Carefully separate dataset



Reference

- Danilo Croce, Giuseppe Castellucci, and Roberto Basili. 2020. GAN-BERT: Generative Adversarial Learning for Robust Text Classification with a Bunch of Labeled Examples. In Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics, pages 2114–2119, Online. Association for Computational Linguistics.
- Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. 2014. Generative adversarial nets. In Z. Ghahramani, M. Welling, C. Cortes, N. D. Lawrence, and K. Q. Weinberger, editors, Advances in Neural Information Processing Systems 27, pages 2672–2680. Curran Associates, Inc.
- Tim Salimans, Ian Goodfellow, Wojciech Zaremba, Vicki Cheung, Alec Radford, Xi Chen, and Xi Chen. 2016. Improved techniques for training gans. In D. D. Lee, M. Sugiyama, U. V. Luxburg, I. Guyon, and R. Garnett, editors, Advances in Neural Information Processing Systems 29, pages 2234–2242. Curran Associates, Inc.
- Anna Rogers, Olga Kovaleva and Anna Rumshisky. A Primer in BERTology: What We Know About How BERT Works.Transactions of the Association for Computational Linguistics (2020) 8: 842–866.January 01.2021.
- Jason Browniee. A Gentle Introduction to Generative Adversarial Networks(GANs).Generative Adversarial Networks. June 17, 2019.