HATE SPEECH

MULTILABEL BINARY CLASSIFICATION

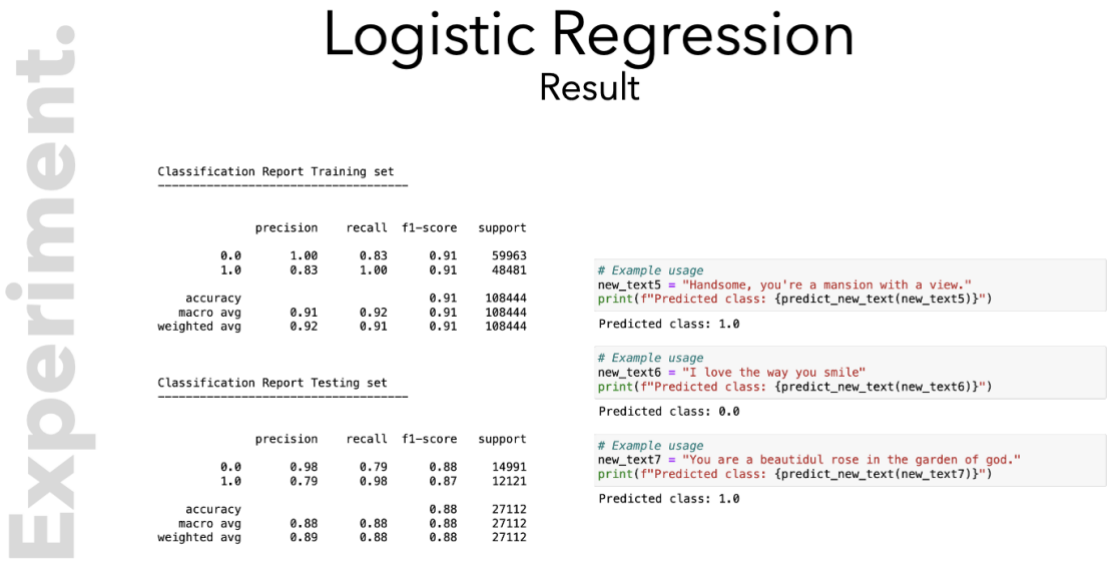
MAIB23 - Natural Language and Conversational Systems with Business Applications
Tran Khanh Ngoc Le - AJ23SYD008

Table of Contents

I. INTRODUCTION

In recent years, the proliferation of digital communication platforms has given rise to an increased prevalence of hate speech, posing significant challenges for social harmony and individual well-being. Recognizing the urgency of addressing this issue, this project embarks on the development of an advanced hate speech classification model, leveraging cutting-edge techniques in Natural Language Processing (NLP) and machine learning.

The initial phase of the project commenced with the development of a rudimentary model focused on binary classification of textual content, labeling it as either “true” hate speech or otherwise. This foundational model, Logistic Regression, despite its simplicity, played a crucial role in establishing a baseline for understanding the nuances of hate speech in digital comments. However, it soon became evident that the model was limited in its scope, particularly in dealing with sophisticated or lengthy textual content. It displayed a marked underperformance when confronted with complex language structures and subtleties inherent in human communication.



To address these limitations and advance the project’s objectives, a strategic pivot was made towards a more holistic and nuanced approach. This paper presents a comprehensive report on the development and implementation of a multicriteria hate speech classification model. The core of this model is built upon the BERT (Bidirectional Encoder Representations from Transformers) framework, renowned for its efficacy in processing natural language. By integrating BERT with NLP techniques, the model is designed to not only identify hate speech but also to understand its various dimensions and contexts. The model’s multicriteria approach allows for a more granular and accurate classification of text, going beyond the binary categorization of hate speech. It evaluates textual content on multiple parameters, encompassing not just the presence of hate speech, but also its intensity, target, and contextual nuances. This comprehensive evaluation framework enables the model to effectively tackle the complexities and varied manifestations of hate speech in digital communications.

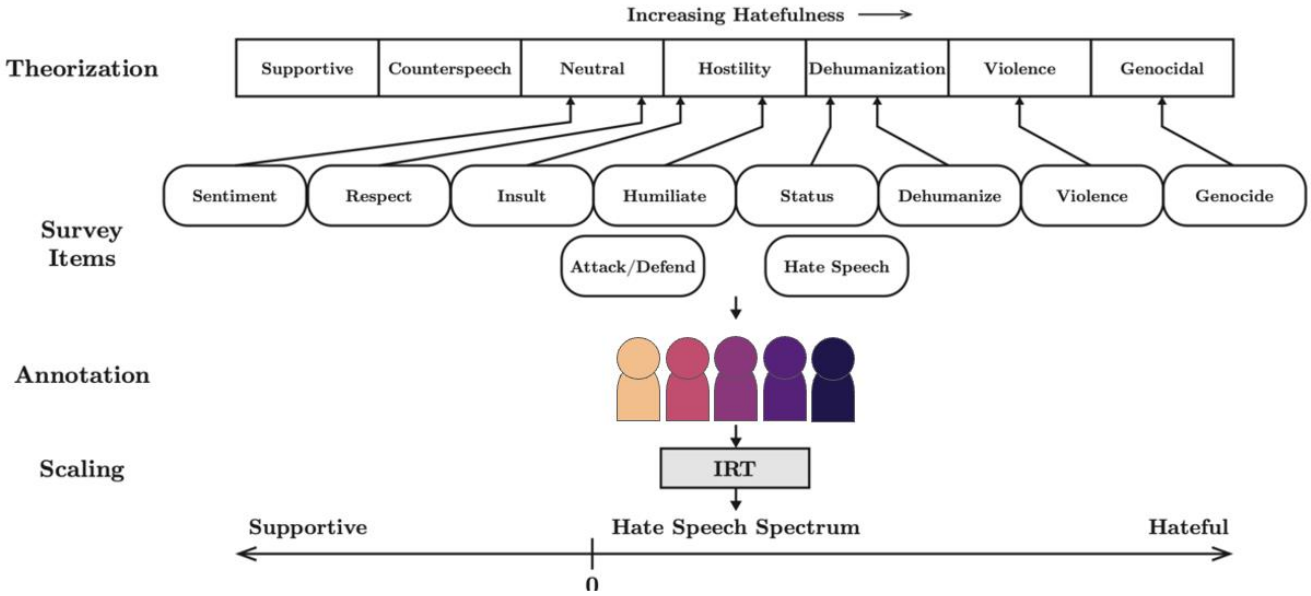
This paper details the journey of the project from its inception, through the challenges encountered in the initial model, to the development of the sophisticated multicriteria classification system. It elucidates the methodologies employed, the data analysis techniques, the model architecture, and the insights gained from the project. The findings of this research contribute significantly to the field of NLP and hate speech detection, providing a robust framework for digital platforms to identify and mitigate hate speech, thereby fostering a safer and more respectful online environment.

II. DATASET

The dataset, developed by the D-Lab at the University of California, Berkeley, discussed in the papers is a specialized corpus designed for the nuanced measurement and analysis of hate speech, comprising 50,070 social media comments sourced from platforms like YouTube, Reddit, and Twitter. It was annotated by a large cohort of 11,143 annotators recruited from Amazon Mechanical Turk (Sachdeva et al., 2022). Each comment in this dataset was subjected to a multi-faceted annotation process,

where annotators provided labels on ten different aspects, including sentiment, disrespect, insult, attacking/defending, humiliation, inferior/superior status, dehumanization, violence, genocide, and a three-valued hate speech benchmark label.

A key feature of this dataset is the incorporation of annotators' perspectives into the labeling process, acknowledging the subjectivity inherent in identifying and classifying hate speech. The annotations also included the identification of the target identity groups of each comment, covering a range of categories such as race/ethnicity, religion, national origin, gender, sexual orientation, age, disability status, and political identity.



The aggregation of these labels utilized faceted Rasch measurement theory (RMT), culminating in a continuous score that reflects each comment's position on a spectrum of hate speech (Kennedy, 2020). This methodological approach acknowledges and statistically analyzes the variations in annotator perspectives, making it a significant departure from traditional methods that seek uniformity in annotator agreement (Sachdeva et al., 2022).

The dataset, thus, serves as a rich resource for understanding the complexities of hate speech, offering insights into how different annotators perceive and rate various components associated with hate speech. Furthermore, the dataset, along with the associated analytical code, has been made publicly available, providing a valuable tool for the wider research community to conduct more in-depth and diverse analyses in the realm of hate speech detection and measurement.

III. LITERATURE REVIEW

In the innovative study titled "Emotion Based Hate Speech Detection using Multimodal Learning", researchers developed a multimodal deep learning framework specifically designed to detect hate speech in multimedia content. This approach is unique in its incorporation of both semantic and emotional features extracted from speech. Utilizing three machine learning models, the study successfully demonstrated the value of analyzing a speaker's emotional state alongside verbal content. A custom dataset, the Hate Speech Detection Video Dataset (HSDVD), was created using multimedia data from Twitter and YouTube, focusing on sexist and ethnic defamation . The results highlighted the improved precision and recall of the multimodal learning framework over traditional text-based models, confirming the hypothesis that multiple modalities enhance hate speech detection. However, the study faced limitations due to the dataset's restricted scope, particularly in analyzing visual features. Future work was recommended to include a more extensive analysis with an expanded dataset (Rana & Jha, 2022).

On the other side, in "Deep Learning for Hate Speech Detection: A Comparative Study" the researchers conducted a large-scale empirical comparison of deep and shallow hate speech detection methods. The study employed 14 different classification-based hate speech detectors, utilizing various word representation methods, and evaluated these on three widely-used datasets: Davidson, Founta, and Twitter Sentiment Analysis (TSA). Each dataset presented its own unique challenges, with varied classes of hatred texts and labeling methods. The study found that while transformer-based models, particularly BERT-based CNN and Electra-based MLP, showed superior performance on certain datasets, there was no consistent superiority of one method over others across all datasets. This indicates the complexity and variability of hate speech detection across different contexts. The study serves as a guide for future research directions, highlighting the need for exploring diverse datasets and methodologies (Malit et al., 2022).

IV. METHODOLOGY

1. Exploratory Data Analysis

Rename features

To make it easier for later interpretation, some attributes are renamed.

```
# Rename features
dataset1['train'] = dataset1['train'].rename_column("sentiment", "pos/neg sentiment")
dataset1['train'] = dataset1['train'].rename_column("respect", "(dis)respect")
dataset1['train'] = dataset1['train'].rename_column("status", "inf/sup status")
dataset1['train'] = dataset1['train'].rename_column("hatespeech", "bias motivated language")
dataset1['train'] = dataset1['train'].rename_column("hate_speech_score", "hate speech")
```

Create a working dataset

The original dataset contains more than a hundred features which most of them are not needed for this project. Therefore, I create a working dataset by keeping some required features including:

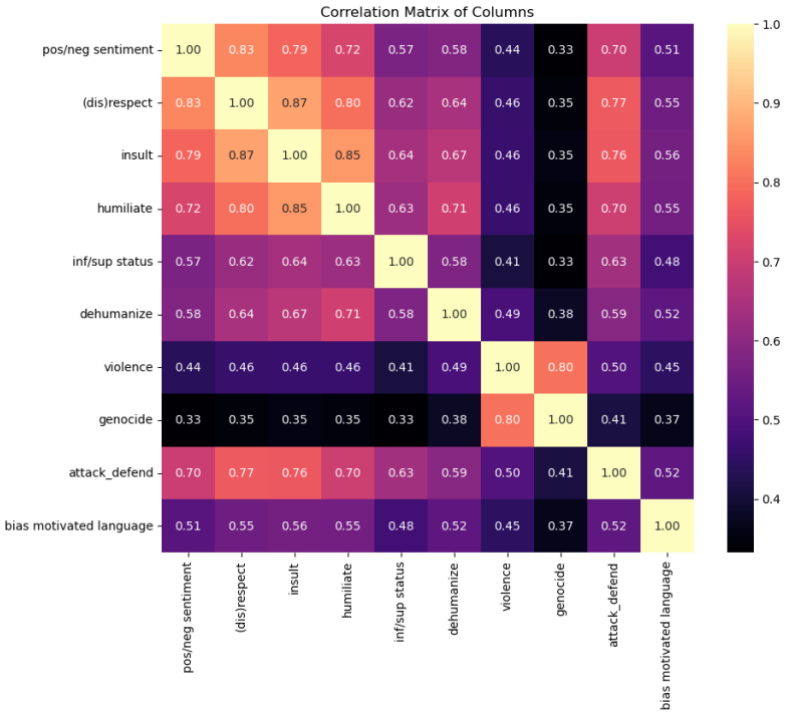
- Positive or Negative Sentiment
- Respectful or disrespectful
- Insult
- Humiliate
- Superior or Inferior Staus
- Dehumanize
- Violence
- Genocide
- Attack_deffend
- Bias motivated langauge
- Hate speech
- Text

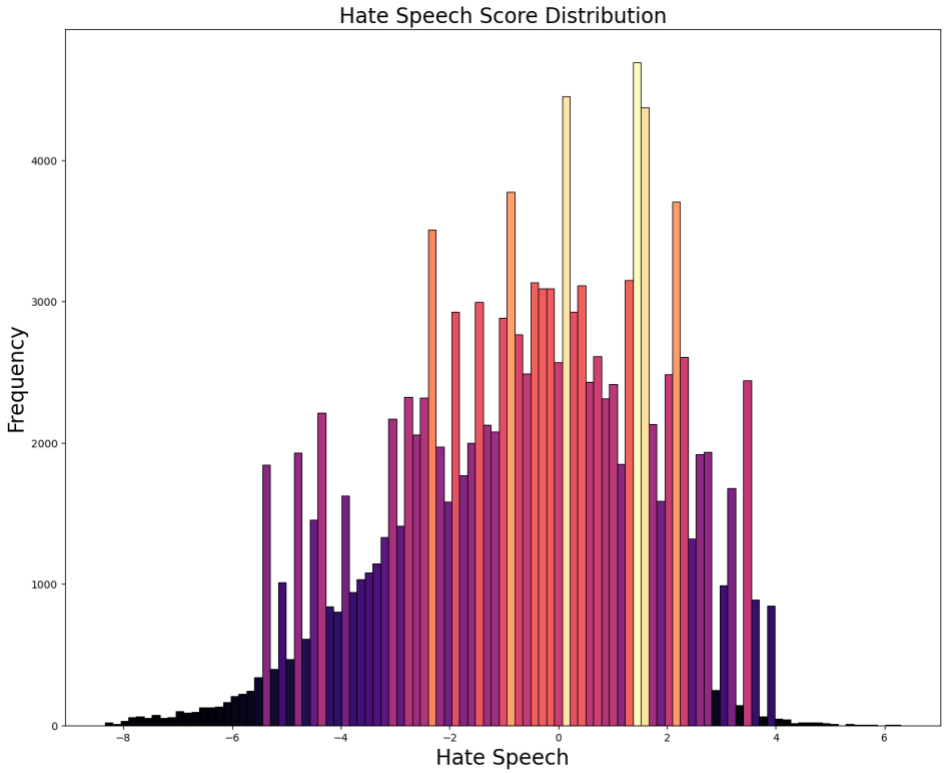
```
# Clean Target Labels
# keep only text and specific targets
column1 = dataset1['train'].column_names
column2 = ['text', 'pos/neg sentiment', '(dis)respect', 'insult', 'humiliate',
           'inf/sup status', 'dehumanize', 'violence', 'genocide',
           'attack_defend',"bias motivated language", 'hate speech']
remove_columns = set(column1)-set(column2)

dataset2 = dataset1.remove_columns(remove_columns)
dataset2
```

	pos/neg sentiment	(dis)respect	insult	humiliate	inf/sup status	dehumanize	violence	genocide	attack_defend	bias motivated language	hate speech	text
0	0.0	0.0	0.0	0.0	2.0	0.0	0.0	0.0	0.0	0.0	-3.90	Yes indeed. She sort of reminds me of the elde...
1	0.0	0.0	0.0	0.0	2.0	0.0	0.0	0.0	2.0	0.0	-6.52	The trans women reading this tweet right now i...
2	4.0	4.0	4.0	4.0	4.0	4.0	0.0	0.0	4.0	2.0	0.36	Question: These 4 broads who criticize America...
3	2.0	3.0	2.0	1.0	2.0	0.0	0.0	0.0	3.0	0.0	0.26	It is about time for all illegals to go back t...
4	4.0	4.0	4.0	4.0	4.0	4.0	4.0	1.0	3.0	2.0	1.54	For starters bend over the one in pink and kic...

Correlation

It is demonstrated the aspect of hatefulness located close to each other show strong correlations. Notably, the correlation diminishes for pairs of survey items that are more distant on the spectrum of hatefulness. For instance, "violence" and "genocide" display a weak correlation with other items, yet they are strongly correlated with each other. Additionally, the item for hate speech displays moderate correlations with all other items, suggesting that each item captures an aspect of hate speech. These findings confirm that the selected features effectively explore the spectrum of hatefulness theorized in the study (Sachdeva et al., 2022).

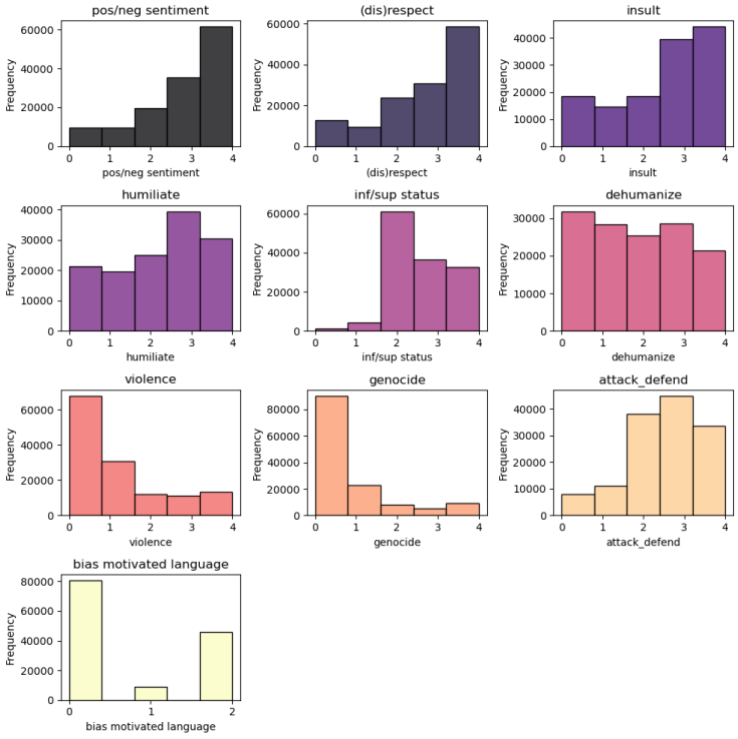




Data distribution

*Continous variable [‘**hate speech’**]* - With 0 being a neutral mark, the scores on the negative side signify a tendency towards supportive language, while positive scores indicate hateful language. The distribution is bimodal, showing two peaks, which suggests there are two prevalent score ranges where the data clusters — one on the supportive side and one on the hateful side. This pattern could suggest that the dataset contains a significant number of instances that are clearly identified as either supportive or hateful, with fewer instances near the neutral center. The tails of the distribution taper off, indicating fewer occurrences of extreme supportive or extreme hateful scores.

*Discrete variable [**Other features except ‘text’**]* - Each histogram represents a different aspect of hate speech, scored on a scale from 0 to 4, where 0 indicates the absence (e.g., respectful, non-violent) and 4 indicates a strong presence of the characteristic (e.g., extremely insulting, highly violent) except for “bias motivated language” in which the indication is around “Yes”, “No”, “Unclear”.



Stratify the aspects into binary variable

Understanding the distribution of features is key to applying the appropriate bins for categorization. For continuous variables, the scores are dichotomized: a score of 0 or less signifies supportive content with less spectrum of hatefulness, while any score above 0 is indicative of hateful content. In the case of discrete variables, the classification is tripartite: scores of 0, 1, or 2 denote supportive or neutral content, and scores of 3 or 4 are categorized as hateful.

By then, the features can be interpreted as, for example:

- Prompt: Is the content likely insulted?
- Response:
  - 0 is False – the content likely weakly insulted.
  - 1 is True – the content likely strongly insulted.

```python
from datasets import Dataset

def bin_values(column, value):
    if column == 'bias motivated language':
        if value == 0:
            return False
        elif value == 1 or value ==2:
            return True
        else:
            return None
    elif column == 'hate speech':
        if value <= 0:
            return False
        else:
            return True
    else:
        if 0 <= value <= 2:
            return False
        elif 3 <= value <= 4:
            return True
        else:
            return None

def apply_binning(example):
    for key, value in example.items():
        if key != 'text':
            example[key] = bin_values(key, value)
    return example

# Apply the function to the 'train' split of the dataset
dataset2['train'] = dataset2['train'].map(apply_binning)
```

Mapping Labels to IDs and IDs to Labels

```python
# get two-way label and label id
ID2LABEL = {}
LABEL2ID = {}

label_id = 0
for label in dataset2['train'].features.keys():
    if label in ['text']:
        continue

    ID2LABEL[label_id] = label
    LABEL2ID[label] = label_id

    label_id += 1

print(f"ID2LABEL:\n{ID2LABEL}\n")
print(f"LABEL2ID:\n{LABEL2ID}")
```

The code shows that it creates two dictionaries, ID2LABEL and LABEL2ID, which serve as mappings between numeric IDs and label names. This is done by iterating over each feature name in the train subset of dataset2, skipping the 'text' feature, and assigning an incremental label_id to each. This is typically done to facilitate referencing labels in a numerical format, which is a common requirement for the training later.

One-hot encoder

The create_labels function one-hot encodes target variables in a data batch. It creates a 'labels' entry in each example with a list of floats, corresponding to the labels defined by LABEL2ID. This function is applied to dataset2 using .map with batched=True, allowing efficient batch processing. The remove_columns=LABEL2ID.keys() argument removes original label columns from dataset2, replacing them with the new one-hot encoded 'labels'. This updates dataset2 for efficient training in machine learning models.

	text	labels
0	Yes indeed. She sort of reminds me of the elde...	[0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, ...
1	The trans women reading this tweet right now i...	[0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, ...
2	Question: These 4 broads who criticize America...	[1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 0.0, 0.0, 1.0, ...
3	It is about time for all illegals to go back t...	[0.0, 1.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 1.0, ...
4	For starters bend over the one in pink and kic...	[1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 0.0, 1.0, ...

2. Preprocessing

Spliting dataset into Train – Validation – Test

The original dataset is divided into three distinct subsets: a training set comprising 80% of the data, a validation set, and a test set, each accounting for 10% of the data.

```
# train (80%), validation (10%), test (10%) split
train_test_datasets = dataset2['train'].train_test_split(test_size=0.2, seed=SEED,
                                                         shuffle=True)
validation_test_datasets = train_test_datasets['test'].train_test_split(test_size=0.5,
                                                                        seed=SEED, shuffle=Tr
```

Tokenizing

This project employs the transformers library to tokenize a dataset using DistilBERT, a streamlined version of BERT trained on lowercase English. The AutoTokenizer class is utilized to initialize a tokenizer with the 'distilbert-base-uncased' model. This tokenizer is then applied to dataset using the map function, which tokenizes the 'text' field of each batch, truncates texts longer than the model's maximum length, and processes the data in batches for efficiency. The original 'text' columns are removed, resulting in tokenized_datasets containing the tokenized data, ready for training with transformer models.

```
from transformers import AutoTokenizer

CHECKPOINT = 'distilbert-base-uncased'
tokenizer = AutoTokenizer.from_pretrained(CHECKPOINT)
tokenized_datasets = dataset2.map(lambda batch: tokenizer(batch['text'], truncation=True),
                                  batched=True, remove_columns=['text'])
tokenized_datasets
```

Create Dataloader

To efficiently process and feed data into machine learning models, I create data loaders for a machine learning model using transformers. It uses DataCollatorWithPadding to dynamically pad batches of tokenized data, ensuring uniform length in each batch. The data loaders are set up for different subsets of the data (like training, validation, and test) using the DataLoader class from PyTorch, with a batch size of 64. The training data is shuffled each epoch to prevent the model from learning data order, a feature not applied to other data subsets. These data loaders streamline the process of batching, padding, and shuffling, enhancing the efficiency of training and evaluating transformer models.

```
from transformers import DataCollatorWithPadding
from torch.utils.data import DataLoader

# get data collator for data loader
data_collator = DataCollatorWithPadding(tokenizer=tokenizer)

# setup dataloaders with tokenized dataset
# to shuffle only be train for each epoch
# in 64 batch sizes with dynamic padding

dataloaders = {}
for dataset_type in tokenized_datasets.keys():
    dataloaders[dataset_type] = DataLoader(
        dataset=tokenized_datasets[dataset_type],
        batch_size=64,
        shuffle=(dataset_type == 'train'),
        collate_fn=data_collator,
    )
```

3. Compilation

Pre-trained Model

A pre-trained model from the transformers library is initialized and configured for a multi-label classification task. It uses AutoModelForSequenceClassification.from_pretrained with a specified checkpoint to load a model, setting it up for multi-label classification with mappings for label names and IDs based on LABEL2ID and ID2LABEL. The model is then moved to a specified

computing device (CPU) for efficient processing, using the model.to(device) command. This setup is essential for tasks where each input can be associated with multiple labels.

```
from transformers import AutoModelForSequenceClassification

model = AutoModelForSequenceClassification.from_pretrained(
    CHECKPOINT,
    problem_type='multi_label_classification',
    num_labels=len(LABEL2ID),
    label2id=LABEL2ID,
    id2label=ID2LABEL,
)

# move model to device
model.to(device)
```

The DistilBertForSequenceClassification model encapsulates the distilled version of the BERT architecture, optimized for sequence classification tasks. It consists of the core DistilBertModel, which includes word and position embedding layers, augmented with dropout and layer normalization for stability and efficiency. The model's backbone is formed by six transformer blocks, each comprising a multi-head self-attention mechanism and a feed-forward network, with additional layer normalizations and dropout for effective learning. The architecture concludes with a pre-classifier linear layer followed by the final classifier layer, designed to output logits for eleven distinct classes. This structure demonstrates a balance between the model's complexity and its capacity for handling multi-label classification scenarios.

Optimizer and Scheduler

The code is to set up an AdamW optimizer and a cosine learning rate scheduler with restarts for training a machine learning model. AdamW, tailored for transformer models, is configured with specific learning rate and weight decay parameters. The scheduler, calculated based on the number of training epochs and batches, adjusts the learning rate following a cosine curve, with periodic resets and a specified warmup period. This setup aims to balance efficient optimization and adaptive learning rate control, enhancing model training effectiveness.

```
from transformers import AdamW, get_scheduler

# setup optimizer and scheduler
scheduler_name = 'cosine'
optimizer = AdamW(model.parameters(), lr=8e-5, weight_decay=0.1,
                  no_deprecation_warning=True)
num_training_epochs = 3
num_training_steps = num_training_epochs * len(dataloaders['train'])
num_warmup_steps = 0

lr_scheduler = get_scheduler(
    name=scheduler_name,
    optimizer=optimizer,
    num_training_steps=num_training_steps,
    num_warmup_steps=num_warmup_steps,
)
```

Training metric

The samples_accuracy_score function calculates sample-level accuracy by comparing individual predictions with true labels and determining the proportion of correct predictions. The compute_metrics function computes both the sample-level accuracy and the F1 score. It applies a sigmoid function to model logits to obtain probabilities, thresholds these at 0.50 for binary predictions, and then calculates the accuracy using samples_accuracy_score and the F1 score with average='samples'. These metrics are tailored to multi-label classification, considering the accuracy and balance of precision and recall for each label in each sample.

```python
from sklearn.metrics import accuracy_score, f1_score

def samples_accuracy_score(y_true, y_pred):
    return np.sum(y_true==y_pred) / y_true.size

def compute_metrics(eval_preds):
    logits, labels = eval_preds
    predictions = torch.nn.functional.sigmoid(torch.Tensor(logits))
    predictions = (predictions >= 0.50).int().numpy()
    samples_accuracy = samples_accuracy_score(labels, predictions)
    samples_f1 = f1_score(labels, predictions, average='samples', zero_division=0)
    return {
        'accuracy': samples_accuracy,
        'f1': samples_f1,
    }
```

Compose training function

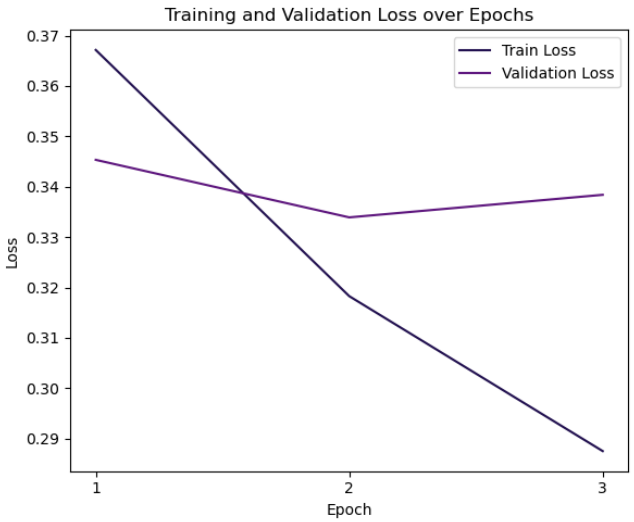
The train function is essential for training a multi-label classification model. It begins by setting the model to training mode, ensuring that layers specific to training, such as dropout, are active. During the training process, the function iterates through the training dataset, executing forward and backward passes for each batch. It processes the model's logits through a sigmoid function and thresholds these to generate binary predictions, followed by loss computation. The function then performs backpropagation and updates the model's weights. Across all batches, it accumulates the loss, predictions, and labels, leading to the computation of average loss, sample-level accuracy, and the F1 score for the entire training dataset. These actions are key for iteratively improving the model based on the training data

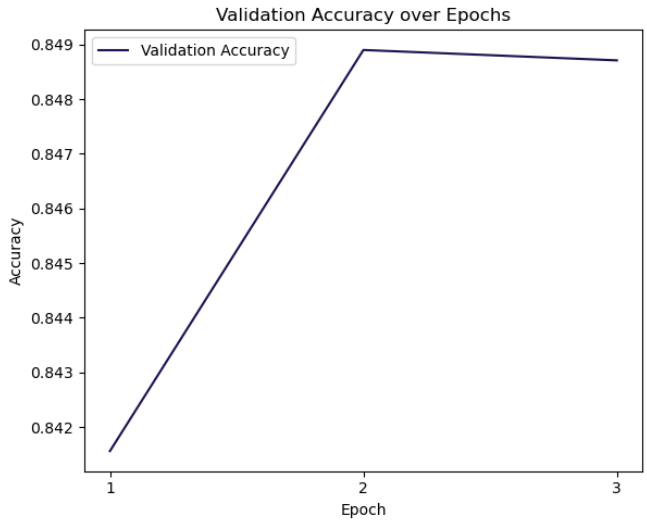
Compose evaluating function

The evaluate function plays a crucial role in assessing the performance of the model on a separate dataset, typically validation or test data. It sets the model to evaluation mode, which deactivates training-specific behaviors like dropout, optimizing the model for evaluation. The function iterates through the evaluation dataset without calculating gradients, enhancing memory efficiency and performance. Similar to the training process, it applies a sigmoid function to the logits and thresholds them to obtain binary predictions. However, unlike training, it does not perform backpropagation or update the model's weights. Instead, it focuses on accumulating the loss, predictions, and labels from each batch. From this data, it calculates the average loss, sample-level accuracy, and F1 score, providing a comprehensive assessment of the model's ability to generalize to new, unseen data.

V. RESULT

1. Training & Validation





Epoch 1

- Duration: The model completed its first epoch with a training phase of around 9 minutes and a validation phase of about 6 minutes.
- Performance: The initial training loss was moderately high, indicating room for improvement in the model's learning. However, a slightly lower validation loss suggested good generalization capabilities. The model achieved a high validation accuracy of approximately 84.156%, demonstrating its effectiveness in classifying the validation data. The F1 score, while moderate, pointed to the need for further refinement in balancing precision and recall.

Epoch 2

- Duration: The second epoch saw a significant increase in training duration to over 6 hours, with the validation phase taking about 34 minutes.
- Performance: There was a notable improvement in training loss, indicating enhanced learning from the data. The validation loss continued to decrease, affirming the model's generalization. The validation accuracy improved to around 84.890%, showing better classification capabilities. However, the F1 score slightly decreased, suggesting challenges in maintaining a consistent balance between precision and recall.

Epoch 3

- Duration: The third epoch extended further, taking around 10 hours and 42 minutes for training and about 34 minutes for validation.
- Performance: The training loss continued its downward trend, showcasing ongoing learning improvements. Conversely, the validation loss saw a slight increase, though it remained in a favorable range. The validation accuracy remained high and consistent with the previous epoch. The F1 score experienced a marginal decrease, highlighting the ongoing need for model tuning to optimize performance metrics.

2. Testing

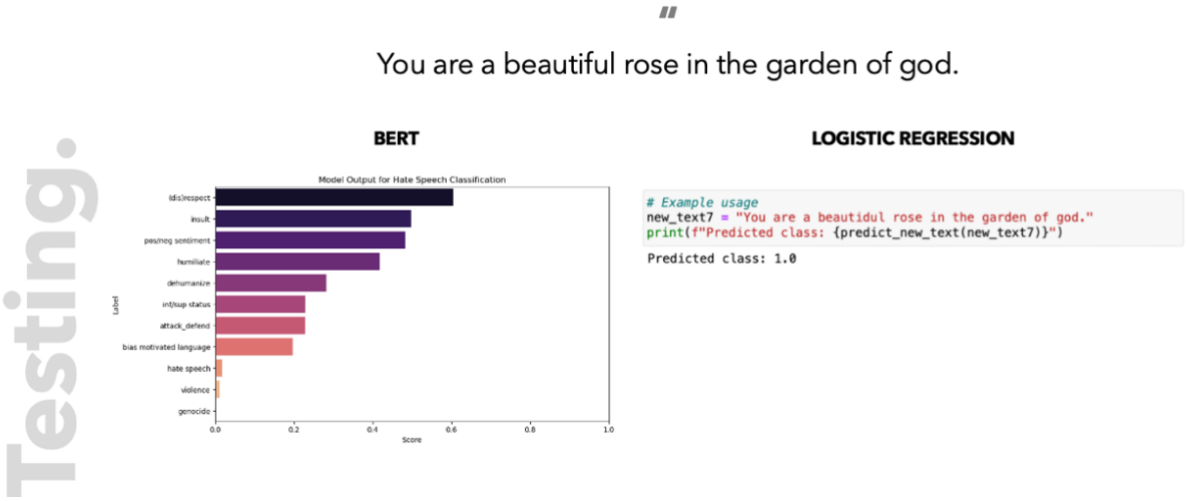
```
TEST ACCURACY: 0.85057 | TEST F1: 0.57239
```

- Accuracy (0.85057): This indicates that the model correctly classified approximately 85.057% of the test dataset. High accuracy suggests that the model is effective at identifying the correct class labels (whether hate speech or not) for a significant portion of the test data.

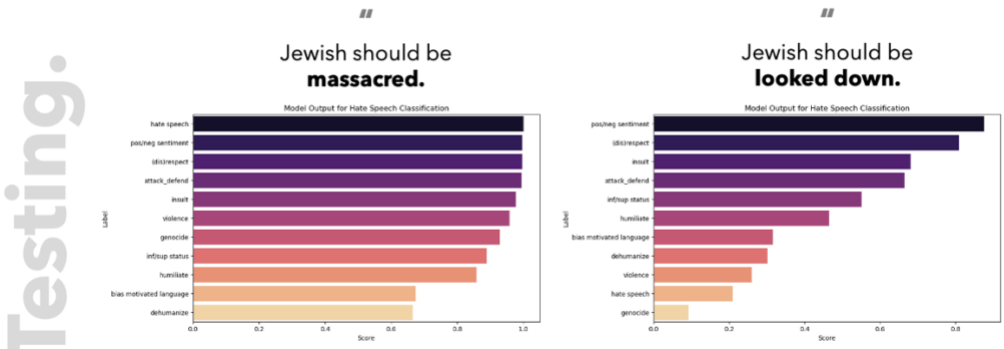
- F1 Score (0.57239): The F1 score is the harmonic mean of precision and recall, two critical metrics in classification tasks. An F1 score of 0.57239, while moderate, indicates a balance between precision (the model's ability to correctly identify positive instances) and recall (the model's ability to find all positive instances).

3. Performance evaluation

- The model shows a good level of accuracy on both the validation (84.871%) and test datasets (85.057%). This indicates a strong ability to correctly classify hate speech and non-hate speech instances.
- The F1 scores for both validation (0.57390) and test (0.57239) datasets are moderate. While these scores show that the model has a reasonable balance between precision and recall, they also suggest there is significant room for improvement, especially considering the critical nature of hate speech detection.
- The indexes reflects the capability of the model that outperform the Logistic Regression when it comes to the sotisficated input such as showing below. BERT seems perform more rationale when it appears to be providing a nuanced output that takes into account multiple categories. It's assigning a higher score to what seems to be a positive category, which aligns well with the input text that doesn't contain hate speech. On the other hand, Logistic Regression seems to be underperforming in this instance, as it has classified the input text into a single class, which we might assume to be incorrect if the predicted class of 1.0 corresponds to 'hate speech'.



- The model also have a high capability of sentivity to the varying degrees of negative sentiment and potential hate speech in the input text. It assigns higher scores to more severe categories like "hate speech" and "genocide" for the first input which is associated with the term “masscred”, a direct and explicit call for violence.



VI. DISCUSSION

1. Challenges

One notable limitation in the application of BERT-based models, as encountered in this study, is the significant demand for computational resources. BERT and its variants, being large and complex models, require substantial memory and processing power, which can pose challenges, especially in resource-constrained environments. Additionally, the training and inference times for these models are considerably longer compared to more traditional, less resource-intensive models. This aspect limits the scalability and practical applicability of BERT in scenarios where rapid processing is crucial.

On the other hand, Handling data imbalance in multicriteria survey items is challenging, as efforts to balance one aspect often lead to imbalances in others. The multicriteria nature of survey items inherently complicates the balancing process. Techniques like SMOTE, while useful for generating synthetic samples for underrepresented classes, can result in an oversized dataset, increasing the risk of overfitting and straining computational resources. Conversely, under-sampling methods like SLIME might reduce the dataset's size excessively, causing a loss of crucial information and diversity.

Another critical limitation observed is related to semantic understanding, particularly in the context of detecting nuanced language aspects like sarcasm. Despite its advanced capabilities in language processing, BERT's performance in accurately identifying sarcastic content remains suboptimal. This is a significant drawback considering that sarcasm is a common element in human communication, and its misinterpretation can lead to erroneous conclusions in sentiment analysis and related tasks. The model's current architecture and training data may not effectively capture the subtleties and contextual cues essential for recognizing sarcasm, leading to this gap in performance.

2. Future considerations

Addressing these limitations, future work can take several directions. First, exploring the model's applicability and performance across different languages would be a valuable expansion. As linguistic structures and nuances vary greatly among languages, assessing and refining BERT's capabilities in a multilingual context can offer broader applicability and more inclusive linguistic analysis. This would involve not only training the model on diverse language datasets but also adapting its architecture to better capture language-specific characteristics.

In addition to linguistic diversity, future efforts could focus on enhancing the model's proficiency in understanding and processing texts with toxic and sarcastic content. Developing specialized training modules or incorporating datasets rich in such linguistic nuances could be a way forward. By training the model on datasets that extensively feature sarcasm, irony, and toxicity, the algorithm could potentially learn to recognize and interpret these complex language forms more accurately. Such advancements would significantly improve the model's utility in real-world applications where understanding the intent and tone behind text is as important as the content itself.

VII. CONCLUSION

This research represents a significant endeavor in addressing the prevalence of hate speech on digital platforms through advanced Natural Language Processing (NLP) and machine learning techniques. Beginning with a fundamental Logistic Regression model for binary classification, the project laid the groundwork for understanding the complexities of hate speech. This foundation was further built upon with the development of a sophisticated, multicriteria classification system utilizing the BERT framework. This system marked a transition from simple binary classification to a nuanced analysis of hate speech, considering various aspects like intensity, target, and context. The journey, as detailed in the paper, highlights the progression from an elementary model to an intricate multicriteria approach. The integration of the BERT framework with NLP techniques

enabled not just the identification but also the detailed analysis of hate speech's multifaceted nature. The use of a diverse dataset from the University of California, Berkeley, enriched the research with its comprehensive annotation process, incorporating a wide range of perspectives. Despite achieving considerable accuracy and a balanced F1 score, the study faced challenges, especially in managing the complexity of the multicriteria data and the significant computational resources required by the model. The model's limitations in interpreting nuanced language forms like sarcasm underscored areas needing further enhancement. Future directions include broadening the model's linguistic capabilities across different languages and improving its ability to process complex forms of language such as sarcasm and toxicity. These improvements aim to refine the model's accuracy and adaptability in various real-world contexts.

VIII. REFERENCE

Kennedy, C. J., Bacon, G., Sahn, A., & von Vacano, C. (2020). Constructing interval variables via faceted rasch measurement and multitask deep learning: a hate speech application. arXiv preprint arXiv:2009.10277.

Malik, J. S., Pang, G., & Hengel, A. V. D. (2022). Deep learning for hate speech detection: a comparative study. arXiv preprint arXiv:2202.09517.

Rana, A., & Jha, S. (2022). Emotion based hate speech detection using multimodal learning. arXiv preprint arXiv:2202.06218.

Sachdeva, P. S., Barreto, R. R., Vacano, C. v., & Kennedy, C. (2022). Assessing annotator identity sensitivity via item response theory: a case study in a hate speech corpus. *2022 ACM Conference on Fairness, Accountability, and Transparency*. https://doi.org/10.1145/3531146.3533216