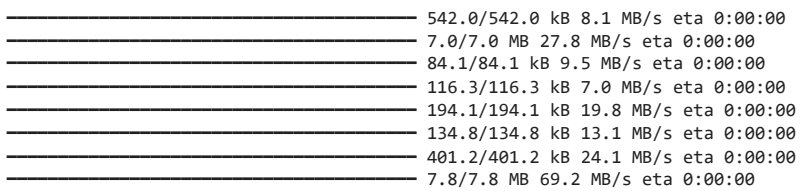


✓ Setup Python Libraries

```
%%bash
pip install numpy torch datasets transformers~=4.28.0 evaluate tqdm --quiet
pip freeze | grep -E '^numpy|^torch|^datasets|^transformers|^evaluate'
```

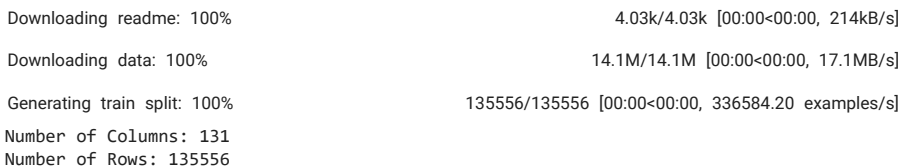


```
datasets==2.19.0
evaluate==0.4.2
numpy==1.25.2
torch @ https://download.pytorch.org/whl/cu121/torch-2.2.1%2Bcu121-cp310-cp310-linux_x86_64.whl#sha256=1adf430f01ff649c848ac021785e18007
torchaudio @ https://download.pytorch.org/whl/cu121/torchaudio-2.2.1%2Bcu121-cp310-cp310-linux_x86_64.whl#sha256=23f6236429e2bf676b820e8
torchdata==0.7.1
torchsummary==1.5.1
torchtext==0.17.1
torchvision @ https://download.pytorch.org/whl/cu121/torchvision-0.17.1%2Bcu121-cp310-cp310-linux_x86_64.whl#sha256=27af47915f6e762c1d44
transformers==4.28.1
```

✓ Clone Raw Dataset

```
from datasets import load_dataset
# source: https://huggingface.co/datasets/ucberkeley-dlab/measuring-hate-speech
raw_datasets = load_dataset("ucberkeley-dlab/measuring-hate-speech")

print(f"Number of Columns: {raw_datasets['train'].num_columns}")
print(f"Number of Rows: {raw_datasets['train'].num_rows}")
```



```
Downloading readme: 100% 4.03k/4.03k [00:00<00:00, 214kB/s]
Downloading data: 100% 14.1M/14.1M [00:00<00:00, 17.1MB/s]
Generating train split: 100% 135556/135556 [00:00<00:00, 336584.20 examples/s]
Number of Columns: 131
Number of Rows: 135556
```

✓ Clean Target Labels

```
# keep only text and specific targets
raw_columns = raw_datasets['train'].column_names
keep_columns = ['text', 'target_race', 'target_religion', 'target_origin', 'target_gender', 'target_sexuality', 'target_age', 'target_disability']
remove_columns = set(raw_columns)-set(keep_columns)

preprocessed_datasets = raw_datasets.remove_columns(remove_columns)
preprocessed_datasets

DatasetDict({
  train: Dataset({
    features: ['text', 'target_race', 'target_religion', 'target_origin', 'target_gender', 'target_sexuality', 'target_age', 'target_disability'],
    num_rows: 135556
  })
})

column_mapping = {column:column.split('_')[1] for column in keep_columns if column.startswith('target')}
print(f"COLUMN_MAPPING: {column_mapping}")

preprocessed_datasets = preprocessed_datasets.rename_columns(column_mapping)
preprocessed_datasets

COLUMN_MAPPING: {'target_race': 'race', 'target_religion': 'religion', 'target_origin': 'origin', 'target_gender': 'gender', 'target_sex': 'sex'}
DatasetDict({
  train: Dataset({
    features: ['text', 'race', 'religion', 'origin', 'gender', 'sex', 'age', 'disability'],
    num_rows: 135556
  })
})
```

```

        features: ['text', 'race', 'religion', 'origin', 'gender', 'sexuality', 'age', 'disability'],
        num_rows: 135556
    })
})

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

label_id = 0
for label in preprocessed_datasets['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}")

ID2LABEL:
{0: 'race', 1: 'religion', 2: 'origin', 3: 'gender', 4: 'sexuality', 5: 'age', 6: 'disability'}

LABEL2ID:
{'race': 0, 'religion': 1, 'origin': 2, 'gender': 3, 'sexuality': 4, 'age': 5, 'disability': 6}

```

✎ Explore Dataset

```

# get target label counts and percentages
label_counts = {}
label_percentages = {}

for label in LABEL2ID:
    label_counts[label] = sum(preprocessed_datasets['train'][label])
    label_percentages[label] = float(f"{sum(preprocessed_datasets['train'][label]) / len(preprocessed_datasets['train'])*100:.2f}")

print(f"LABEL_COUNTS:\n{label_counts}\n")
print(f"LABEL_PERCENTAGES:\n{label_percentages}")

LABEL_COUNTS:
{'race': 48352, 'religion': 26182, 'origin': 24372, 'gender': 40439, 'sexuality': 22407, 'age': 2005, 'disability': 3716}

LABEL_PERCENTAGES:
{'race': 35.67, 'religion': 19.31, 'origin': 17.98, 'gender': 29.83, 'sexuality': 16.53, 'age': 1.48, 'disability': 2.74}

import matplotlib.pyplot as plt

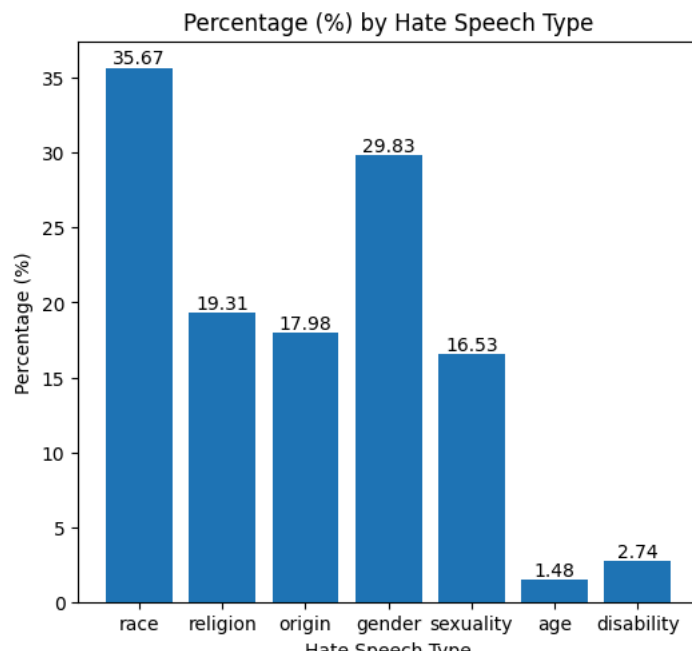
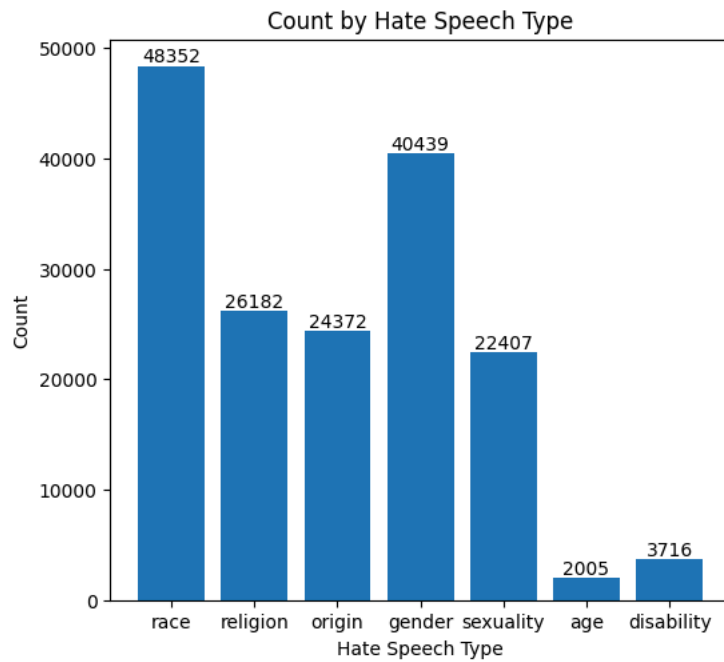
# create 2 graphs
fig, axs = plt.subplots(2, figsize=(6,12))

# create bar graphs of label counts
bar_container0 = axs[0].bar(label_counts.keys(), label_counts.values())
axs[0].bar_label(bar_container0, label_type='edge')
axs[0].set_xlabel('Hate Speech Type')
axs[0].set_ylabel('Count')
axs[0].set_title('Count by Hate Speech Type')

# create bar graphs of label percentages
bar_container1 = axs[1].bar(label_percentages.keys(), label_percentages.values())
axs[1].bar_label(bar_container1, label_type='edge')
axs[1].set_xlabel('Hate Speech Type')
axs[1].set_ylabel('Percentage (%)')
axs[1].set_title('Percentage (%) by Hate Speech Type')

plt.show()

```



```
def create_labels(batch):
    # one-hot encode targets for training
    batch['labels'] = [[float(batch[label][i]) for label in LABEL2ID] for i in range(len(batch['text']))]
    return batch

preprocessed_datasets = preprocessed_datasets.map(create_labels, batched=True, remove_columns=LABEL2ID.keys())
preprocessed_datasets
```

```
Map: 100%                               135556/135556 [00:01<00:00, 121341.15 examples/s]
DatasetDict({
  train: Dataset({
    features: ['text', 'labels'],
    num_rows: 135556
  })
})
```

✧ Setup Model Training

```

import numpy as np
import torch

# set seed for reproducibility
SEED = 42
torch.manual_seed(SEED)
np.random.seed(SEED)

from datasets import DatasetDict

# train (80%), validation (10%), test (10%) split
train_test_datasets = preprocessed_datasets['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=True)

preprocessed_datasets = DatasetDict({
    'train': train_test_datasets['train'],
    'validation': validation_test_datasets['train'],
    'test': validation_test_datasets['test']
})
preprocessed_datasets

DatasetDict({
  train: Dataset({
    features: ['text', 'labels'],
    num_rows: 108444
  })
  validation: Dataset({
    features: ['text', 'labels'],
    num_rows: 13556
  })
  test: Dataset({
    features: ['text', 'labels'],
    num_rows: 13556
  })
})

```

✓ Tokenize Dataset

```

from transformers import AutoTokenizer

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

/usr/local/lib/python3.10/dist-packages/huggingface_hub/file_download.py:1132: FutureWarning:
  warnings.warn(
tokenizer_config.json: 100% 28.0/28.0 [00:00<00:00, 1.59kB/s]
config.json: 100% 483/483 [00:00<00:00, 22.7kB/s]
vocab.txt: 100% 232k/232k [00:00<00:00, 6.61MB/s]
tokenizer.json: 100% 466k/466k [00:00<00:00, 24.5MB/s]
Map: 100% 108444/108444 [00:37<00:00, 2689.77 examples/s]
Map: 100% 13556/13556 [00:03<00:00, 3611.34 examples/s]
Map: 100% 13556/13556 [00:02<00:00, 4709.32 examples/s]
DatasetDict({
  train: Dataset({
    features: ['labels', 'input_ids', 'attention_mask'],
    num_rows: 108444
  })
  validation: Dataset({
    features: ['labels', 'input_ids', 'attention_mask'],
    num_rows: 13556
  })
  test: Dataset({
    features: ['labels', 'input_ids', 'attention_mask'],
    num_rows: 13556
  })
})

```

✓ Setup DataLoaders

```
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,
    )
```

✓ Get Current Device

```
# get current device
device = 'cuda' if torch.cuda.is_available() else 'cpu'
device
```

```
'cuda'
```

```
!pip install emoji
```

```
Collecting emoji
  Downloading emoji-2.11.1-py2.py3-none-any.whl (433 kB)
  ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 433.8/433.8 kB 6.7 MB/s eta 0:00:00
Installing collected packages: emoji
Successfully installed emoji-2.11.1
```

```

from transformers import AutoModelForSequenceClassification
import gc
#import os
import emoji as emoji
import re
import string
import numpy as np
import pandas as pd
import torch
import torch.nn as nn
from sklearn.model_selection import train_test_split
from sklearn.metrics import classification_report, accuracy_score
from transformers import AutoModel
from transformers import BertModel, BertTokenizer
class BERT_Arch(nn.Module):

    def __init__(self, bert):
        super(BERT_Arch, self).__init__()
        self.bert = BertModel.from_pretrained('bert-base-uncased')
        self.conv = nn.Conv2d(in_channels=13, out_channels=13, kernel_size=(3, 768), padding=True)
        self.relu = nn.ReLU()
        self.pool = nn.MaxPool2d(kernel_size=3, stride=1)
        self.dropout = nn.Dropout(0.1)
        self.fc = nn.Linear(442, 3) # before : 442 with max_length 36 # 806 with max_length 64
        self.flat = nn.Flatten()
        self.softmax = nn.LogSoftmax(dim=1)

    def forward(self, sent_id, mask):
        _, _, all_layers = self.bert(sent_id, attention_mask=mask, output_hidden_states=True)
        # all_layers = [13, 32, 64, 768]
        x = torch.transpose(torch.cat(tuple([t.unsqueeze(0) for t in all_layers])), 0), 0, 1)
        del all_layers
        gc.collect()
        torch.cuda.empty_cache()
        x = self.pool(self.dropout(self.relu(self.conv(self.dropout(x)))))
        x = self.fc(self.dropout(self.flat(self.dropout(x))))
        return self.softmax(x)

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)

```

```

/usr/local/lib/python3.10/dist-packages/huggingface_hub/file_download.py:1132: FutureWarning:
warnings.warn(
model.safetensors: 100% 268M/268M [00:01<00:00, 196MB/s]
Some weights of the model checkpoint at distilbert-base-uncased were not used when initializing
- This IS expected if you are initializing DistilBertForSequenceClassification from the
- This IS NOT expected if you are initializing DistilBertForSequenceClassification from
Some weights of DistilBertForSequenceClassification were not initialized from the model
You should probably TRAIN this model on a down-stream task to be able to use it for predictions
DistilBertForSequenceClassification(
  (distilbert): DistilBertModel(
    (embeddings): Embeddings(
      (word_embeddings): Embedding(30522, 768, padding_idx=0)
      (position_embeddings): Embedding(512, 768)
      (LayerNorm): LayerNorm((768,), eps=1e-12, elementwise_affine=True)
      (dropout): Dropout(p=0.1, inplace=False)
    )
    (transformer): Transformer(
      (layer): ModuleList(
        (0-5): 6 x TransformerBlock(
          (attention): MultiHeadSelfAttention(
            (dropout): Dropout(p=0.1, inplace=False)
            (q_lin): Linear(in_features=768, out_features=768, bias=True)
            (k_lin): Linear(in_features=768, out_features=768, bias=True)
            (v_lin): Linear(in_features=768, out_features=768, bias=True)
            (out_lin): Linear(in_features=768, out_features=768, bias=True)
          )
          (sa_layer_norm): LayerNorm((768,), eps=1e-12, elementwise_affine=True)
          (ffn): FFN(
            (dropout): Dropout(p=0.1, inplace=False)
            (lin1): Linear(in_features=768, out_features=3072, bias=True)
            (lin2): Linear(in_features=3072, out_features=768, bias=True)
            (activation): GELUActivation()
          )
          (output_layer_norm): LayerNorm((768,), eps=1e-12, elementwise_affine=True)
        )
      )
    )
  )
  (pre_classifier): Linear(in_features=768, out_features=768, bias=True)
  (classifier): Linear(in_features=768, out_features=7, bias=True)
  (dropout): Dropout(p=0.2, inplace=False)
)

```

✎ Setup Optimizer and Scheduler

```

from transformers import AdamW, get_scheduler

# setup optimizer and scheduler
scheduler_name = 'linear'
optimizer = AdamW(model.parameters(), lr=5e-5, weight_decay=0, no_deprecation_warning=True)
num_training_epochs = 1
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,
)

print(f"          SCHEDULER NAME: {scheduler_name}")
print(f"          OPTIMIZER: {optimizer.__class__.__name__}")
print(f"NUMBER OF TRAINING EPOCHS: {num_training_epochs}")
print(f"NUMBER OF TRAINING STEPS: {num_training_steps}")

          SCHEDULER NAME: linear
          OPTIMIZER: AdamW
NUMBER OF TRAINING EPOCHS: 1
NUMBER OF TRAINING STEPS: 1695

```

✎ Setup Training Metrics

```

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,
    }

```

Setup Training and Evaluation Scripts

```

from tqdm.notebook import tqdm

def train(model, dataloader):
    # setup train metrics
    loss = 0
    train_predictions = []
    train_labels = []

    # set to train mode
    model.train()
    # iterate through dataloader
    for batch in tqdm(dataloader):
        # zero the gradients
        optimizer.zero_grad()

        # predict batch in current device
        batch.to(device)
        outputs = model(**batch)

        # compute multilabel outputs
        predictions = torch.nn.functional.sigmoid(outputs.logits).cpu()
        predictions = (predictions >= 0.50).int().numpy()
        labels = batch['labels']

        # backprop and update learning rate
        outputs.loss.backward()
        optimizer.step()
        lr_scheduler.step()

        # accumulate train metrics
        loss += outputs.loss.item()
        train_predictions += predictions.tolist()
        train_labels += labels.tolist()

    # compute train metrics
    loss /= len(dataloader)
    samples_accuracy = samples_accuracy_score(np.array(train_labels), np.array(train_predictions))
    samples_f1 = f1_score(np.array(train_labels), np.array(train_predictions), average='samples', zero_division=0)
    return {
        'loss': loss,
        'accuracy': samples_accuracy,
        'f1': samples_f1,
    }

```



```
def evaluate(model, dataloader):
    # setup evaluation metrics variables
    loss = 0
    evaluate_predictions = []
    evaluate_labels = []

    # set to evaluation mode
    model.eval()
    with torch.no_grad():
        # iterate through dataloader
        for batch in tqdm(dataloader):
            # predict batch in current device
            batch.to(device)
            outputs = model(**batch)

            # compute multilabel outputs
            predictions = torch.nn.functional.sigmoid(outputs.logits).cpu()
            predictions = (predictions >= 0.50).cpu().numpy()
            labels = batch['labels']

            # accumulate evaluation metrics
            loss += outputs.loss.item()
            evaluate_predictions += predictions.tolist()
            evaluate_labels += labels.tolist()

    # compute evaluation metrics
    loss /= len(dataloader)
    samples_accuracy = samples_accuracy_score(np.array(evaluate_labels), np.array(evaluate_predictions))
    samples_f1 = f1_score(np.array(evaluate_labels), np.array(evaluate_predictions), average='samples', zero_division=0)
    return {
        'loss': loss,
        'accuracy': samples_accuracy,
        'f1': samples_f1,
    }
```

✓ Train Model

```
test_metrics = evaluate(model, dataloaders['test'])
print(f"TEST ACCURACY: {test_metrics['accuracy']:.5f}", end=" | ")
print(f"TEST F1: {test_metrics['f1']:.5f}")
```

100% 212/212 [00:45<00:00, 4.75it/s]

You're using a DistilBertTokenizerFast tokenizer. Please note that with a fast tokenizer, using the `__call__` method is faster than using the `encode` and `decode` methods.

TEST ACCURACY: 0.49832 | TEST F1: 0.16076

We got **Test Accuracy: 49%** and **Test F1: 16%**

Updating to transfer learning on **BERT**!

✓ Train the Model

```
for epoch in range(num_training_epochs):
    train_metrics = train(model, dataloaders['train'])
    validation_metrics = evaluate(model, dataloaders['validation'])

    print(f"EPOCH {epoch+1}", end=" | ")
    print(f"TRAIN LOSS: {train_metrics['loss']:.5f}", end=" | ")
    print(f"VALIDATION LOSS: {validation_metrics['loss']:.5f}", end=" | ")
    print(f"VALIDATION ACCURACY: {validation_metrics['accuracy']:.5f}", end=" | ")
    print(f"VALIDATION F1: {validation_metrics['f1']:.5f}")
```

100% 1695/1695 [16:38<00:00, 2.05it/s]

100% 212/212 [00:45<00:00, 4.62it/s]

EPOCH 1 | TRAIN LOSS: 0.17746 | VALIDATION LOSS: 0.15467 | VALIDATION ACCURACY: 0.93854 | VALIDATION F1: 0.83531

```
test_metrics = evaluate(model, dataloaders['test'])
print(f"TEST ACCURACY: {test_metrics['accuracy']:.5f}", end=" | ")
print(f"TEST F1: {test_metrics['f1']:.5f}")
```

```
100% 212/212 [00:45<00:00, 5.36it/s]
TEST ACCURACY: 0.93738 | TEST F1: 0.82960
```

We got **Test Accuracy: 93%** and **Test F1: 82%**

```
from transformers import pipeline

# setup pipeline as a text classification with multilabel outputs
hate_speech_multilabel_classifier = pipeline(
    task='text-classification',
    model=model,
    tokenizer=tokenizer,
    device=torch.cuda.current_device(),
    top_k=None
)
```

```
race_hate_text = """
Yellow peril.
"""
```

```
hate_speech_multilabel_classifier(race_hate_text)

[[{'label': 'race', 'score': 0.9407707452774048},
 {'label': 'origin', 'score': 0.05217541381716728},
 {'label': 'gender', 'score': 0.04240966960787773},
 {'label': 'sexuality', 'score': 0.011713751591742039},
 {'label': 'religion', 'score': 0.0055633787997066975},
 {'label': 'age', 'score': 0.004907922353595495},
 {'label': 'disability', 'score': 0.0034424590412527323}]]
```

```
origin_hate_text = """
Go back to where you came from.
"""
```

```
hate_speech_multilabel_classifier(origin_hate_text)

[[{'label': 'origin', 'score': 0.9191192984580994},
 {'label': 'race', 'score': 0.22591283917427063},
 {'label': 'sexuality', 'score': 0.01085717137902975},
 {'label': 'religion', 'score': 0.010725829750299454},
 {'label': 'gender', 'score': 0.005518690217286348},
 {'label': 'age', 'score': 0.0035892582964152098},
 {'label': 'disability', 'score': 0.0032354879658669233}]]
```

```
gender_hate_text = """
You're being emotional.
"""
```

```
hate_speech_multilabel_classifier(gender_hate_text)

[[{'label': 'disability', 'score': 0.33753475546836853},
 {'label': 'gender', 'score': 0.3066709041595459},
 {'label': 'sexuality', 'score': 0.10137270390987396},
```