Google Developer Groups
On Campus • Jamia Millia Islamia

Multilingual Sentiment Analysis via Youtube Comments

Presented by Arib Ansari

The Problem

Developing a classification model that is capable of extracting user sentiments, particularly focusing on youtube comments on educational content.

The main point of concern is that the comments are in Hinglish and as such regular techniques don’t quite make any outstanding impact.

Preliminary Steps

1.I have chosen not to remove stop-words as during testing I came to the conclusion that their inclusion leads to better classification

2.Preprocessing has been done to remove special characters and lowercase all the text.

3.Lemmatization and Tokenization are carried out for every approach and will not be explicitly mentioned.

Methodological Approach and Procedure

01

XGDClassifier - A rarely used multi class tree based classification technique i thought could work well. Used it alongside TF-IDF vectorization of dataset.

Result - F1 score of 0.67

02

Linear SVC - A simple but effective approach for sentiment analysis. I used an ensemble of TF-IDF and Word2Vec to generate a combined feature vector for training.

Result - F1 score of 0.703

03

Linear SVC/Logistic Regression and TF-IDF but with MuRIL instead of word2vec to create better embeddings for semantic context.

Result - max F1 score of 0.71

04

Final Approach - Fine-tuning MuRIL. I stuck to LSVC as my ML model and tuned the BERT based model on my problem dataset for better performance.

Result -F1 score of 0.722

Why an ensemble model method for Feature Engineering

Using multiple techniques allows proper leverage of each of their strong suits to further enhance pipeline performance and get a much better idea of what the actual sentiment may be.

01. TF-IDF specializes in finding Importance of specific words. It can automatically reduce the impact of frequently used words on final predictions.

02. Embeddings are useful for gathering sematic context within a sentence. Using MuRIL intead of word2vec is beneficial because it allows for fine-tuning and was built with Indian Languages in mind.

03. Combining the frequency analysis of TF-IDF with the semantic analysis of MuRIL gives a more complete picture of the text at hand. F1 increases for all models with this approach.

Text Preprocessing

```python
# Cleanup + Tokenization + Lemmatization
def clean_text(text):
    text = re.sub(r'[^a-zA-Z\s]', '', text).lower().strip()
    tokens = word_tokenize(text)
    tokens = [lemmatizer.lemmatize(word) for word in tokens]
    return ' '.join(tokens)
```

MuRIL Mean Embeddings

```python
def get_mean_embedding(text, tokenizer, model):
    inputs = tokenizer(text, return_tensors="pt", truncation=True, padding=True,
    with torch.no_grad():
        outputs = model(**inputs)
    embedding = outputs.last_hidden_state.mean(dim=1).squeeze()
    return embedding.cpu().numpy()
```

Custom built GridSearchCV with a progress bar

```python
class GridSearchCVWithProgress:
    def __init__(self, estimator, param_grid,X,y, scoring=None, cv=5, n_jobs=None):
        self.estimator = estimator
        self.param_grid = param_grid
        self.cv = cv
        self.scoring = scoring
        self.n_jobs = n_jobs
        self.best_params_ = None
        self.best_score_ = -np.inf
        self.X = X
        self.y = y

    def fit(self):
        param_combinations = list(ParameterGrid(self.param_grid))

        # Initializing the progress bar
        with tqdm(total=len(param_combinations), desc="Grid Search Progress", unit="combination") as pbar:
            for params in param_combinations:
                self.estimator.set_params(**params)

                scores = cross_val_score(self.estimator, self.X, self.y, cv=self.cv, scoring=self.scoring,
                mean_score = np.mean(scores)

                # Save the vbest
                if mean_score > self.best_score_:
                    self.best_score_ = mean_score
                    self.best_params_ = params

                # INC pbar
                pbar.update(1)

        return self

    def predict(self,X_test):
        self.estimator.set_params(**self.best_params_)
        self.estimator.fit(self.X,self.y)
        y_pred = self.estimator.predict(X_test)
        return y_pred
```

Some custom functions made to assist

What’s so special about MuRIL ?

- MuRIL is trained on significantly large amounts of IN text corpora only.
- It is based on BERT and outperforms even mBERT on all benchmark datasets.
- Possibly the only homegrown Hinglish embedding generator with minimal effort for finetuning.
- Worked upon by Google and is lightweight.
- Reference: https://arxiv.org/pdf/2103.10730

Fine-Tuning MuRIL

Dataset preparation for transformer

```
hf_dataset = Dataset.from_pandas(df)

model_name = "google/muril-base-cased"
tokenizer = AutoTokenizer.from_pretrained(model_name)
model = AutoModelForSequenceClassification.from_pretrained(model_name, num_labels=len(df['label'].unique()))

def tokenize(batch):
    return tokenizer(batch['comment'], padding=True, truncation=True, max_length=128)

hf_dataset = hf_dataset.map(tokenize, batched=True)

hf_dataset.set_format(type="torch", columns=["input_ids", "attention_mask", "label"])
```

Saving the model

```
results = trainer.evaluate()
print(results)

model.save_pretrained("fine_tuned_muril")
tokenizer.save_pretrained("fine_tuned_muril")
```

✓ 1m 33.7s

100%

```
{'eval_loss': 0.6469289064407349, 'eval_runtime': 92.6172, 'eval_samples_per_second': 692.366, 'eval_steps_per_second': 21.637, 'epoch': 1

('fine_tuned_muril\\tokenizer_config.json',
 'fine_tuned_muril\\special_tokens_map.json',
 'fine_tuned_muril\\vocab.txt',
 'fine_tuned_muril\\added_tokens.json',
 'fine_tuned_muril\\tokenizer.json')
```

Training Parameters

```
training_args = TrainingArguments(
    output_dir="./results",
    num_train_epochs=1,                    #Make this 5 for optimal training
    per_device_train_batch_size=8,
    per_device_eval_batch_size=32,
    weight_decay=0.01,
    logging_dir="./logs",
    logging_steps=500,
    learning_rate=2e-5,
    warmup_ratio=0.1,
    dataloader_num_workers=4,
    fp16 = True,
    save_steps=1000,
    save_total_limit=1,
)


train_test_split = hf_dataset.train_test_split(test_size=0.3, seed=42)
train_dataset = train_test_split["train"]
test_dataset = train_test_split["test"]

trainer = Trainer(
    model=model,
    args=training_args,
    train_dataset=train_dataset,
    eval_dataset=test_dataset,
    tokenizer=tokenizer,
)

# Fine-tune the model
trainer.train()
```

Training Pipeline

Assuming fine-tuning is done separately

Data preprocessing using regex and nltk. The data is tokenized and lemmatized i.e. prepared for feature engineering.

Generating TF-IDF document vectors and mean of fine-tuned MuRIL word embedding

Normalizing (L2) both the feature vectors and then combining them using scipy.

Using my custom GridSearchwithProgress class to find optimal hyperparameters for the chosen model

Generating final classification report heatmap as well as matrix for the best parameters found.

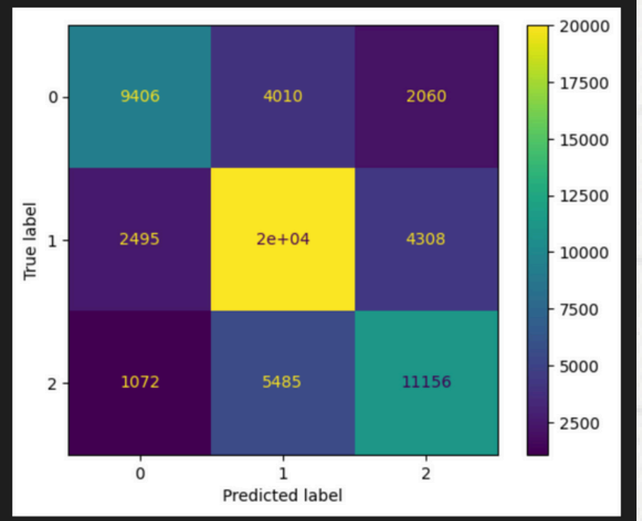
Exporting final model for use in creating the solution.csv using the test dataset.

-- Results --

Approach 1

Best parameters found: {'colsample_bytree': 1.0, 'gamma': 0, 'learning

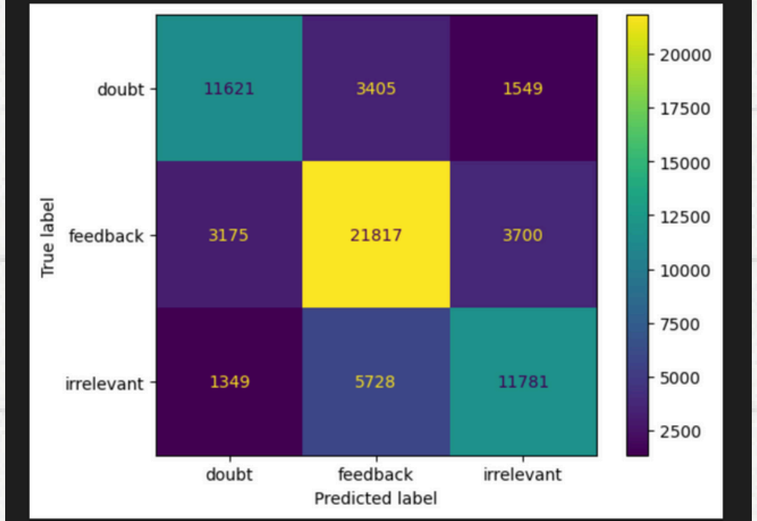
	precision	recall	f1-score	support
doubt	0.73	0.61	0.66	15476
feedback	0.68	0.75	0.71	26811
irrelevant	0.64	0.63	0.63	17713
accuracy			0.68	60000
macro avg	0.68	0.66	0.67	60000
weighted avg	0.68	0.68	0.68	60000



Approach 2

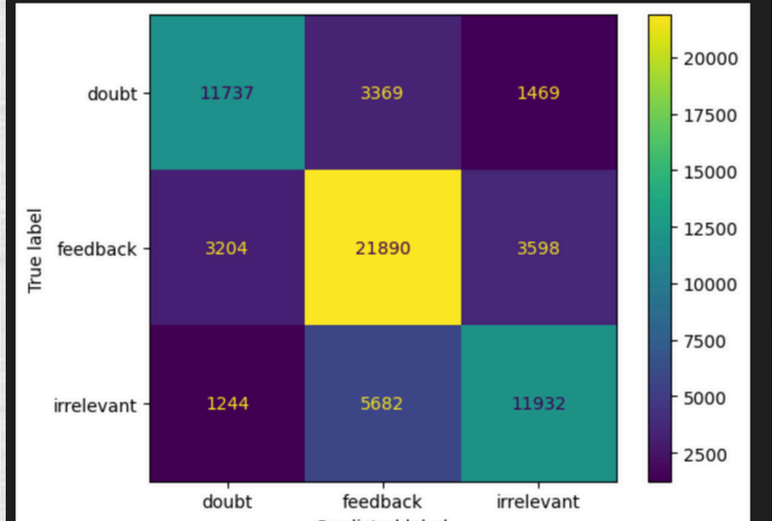
Best parameters found: {'C': 0.1, 'class_weight': None, 'max_iter': 1000, 'tol'

	precision	recall	f1-score	support
doubt	0.72	0.70	0.71	16575
feedback	0.70	0.76	0.73	28692
irrelevant	0.69	0.62	0.66	18858
accuracy			0.71	64125
macro avg	0.71	0.70	0.70	64125
weighted avg	0.70	0.71	0.70	64125



Approach 3

	precision	recall	f1-score	support
doubt	0.73	0.71	0.72	16575
feedback	0.71	0.76	0.73	28692
irrelevant	0.70	0.63	0.67	18858
accuracy			0.71	64125
macro avg	0.71	0.70	0.71	64125
weighted avg	0.71	0.71	0.71	64125



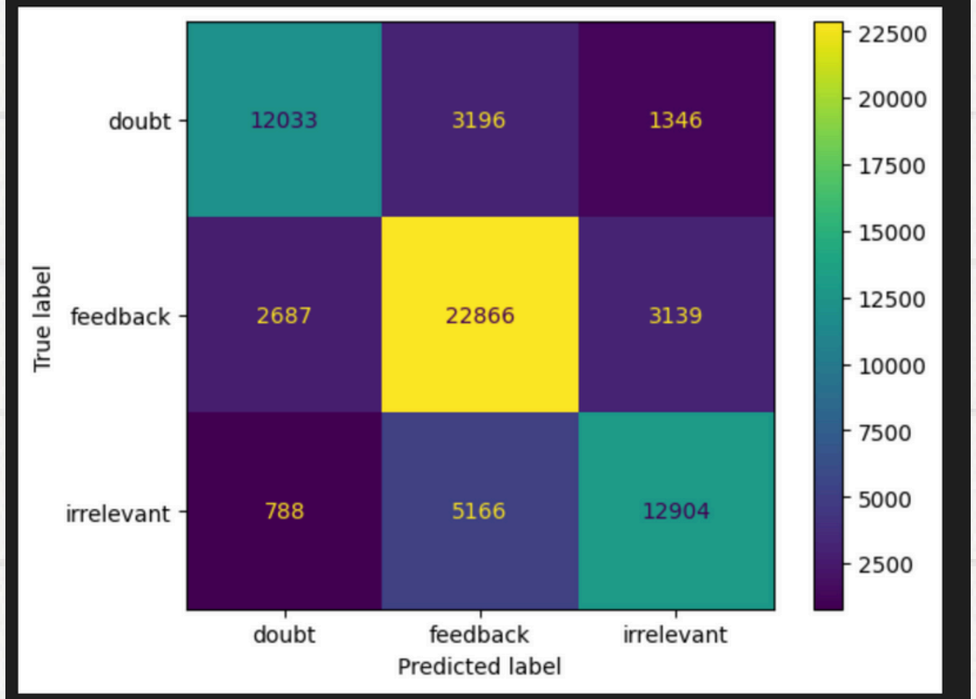
After all that work...

74%

Success

My final approach boasts a very respectable 0.722 F1 score when evaluated on test dataset on kaggle

	precision	recall	f1-score	support
doubt	0.78	0.73	0.75	16575
feedback	0.73	0.80	0.76	28692
irrelevant	0.74	0.68	0.71	18858
accuracy			0.75	64125
macro avg	0.75	0.74	0.74	64125
weighted avg	0.75	0.75	0.74	64125



Thank you very much!

All the code is available on my github:https://github.com/Arperior