Information-Retrieval Bonus

Daniel Attali, Sapir Bashan, Noam Benisho

2025-02-05

Contents

1 Part A: Human Validation on Ex4

In this part we took 35 sentence from each class that could have been classified by a human to more than one class and then tried to see how our model from

Ex4 was compared to the human classifier.

1.1 Results and Discussion

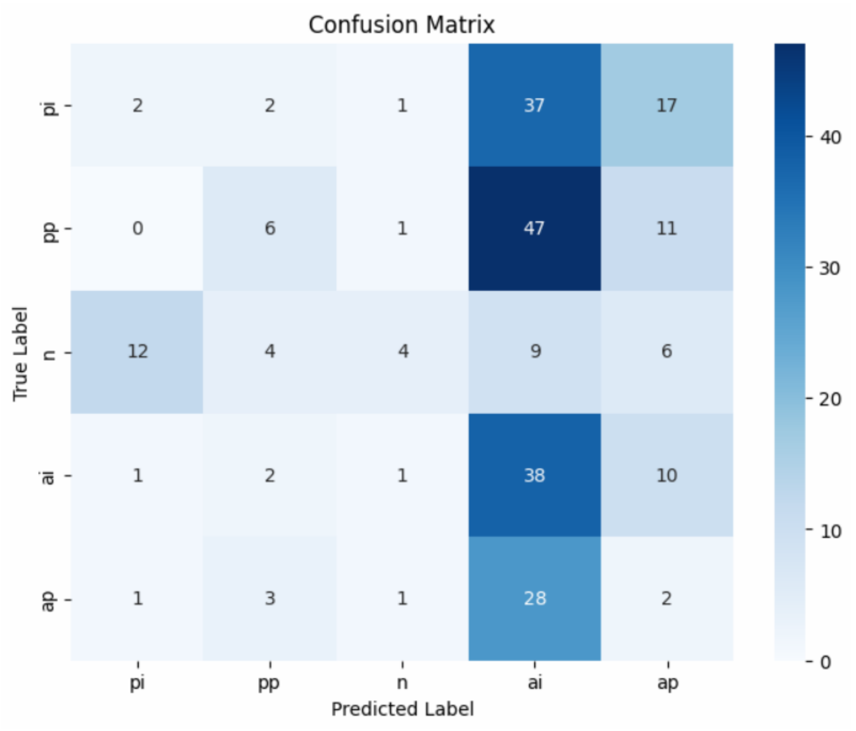
1.1.1 Results Summary

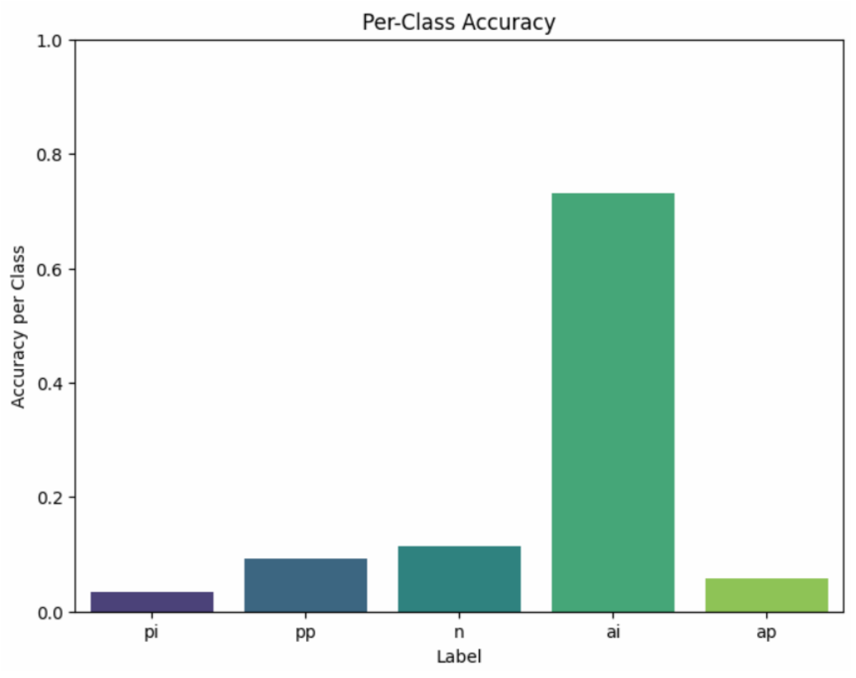
On a new test dataset (with a total support of 246 samples), one set of results for a model run was:

- **Overall Accuracy:** 21.14%
- **Classification Report:**

Class	Precision	Recall	F1-Score	Support
pi (Pro-Israel)	0.12	0.03	0.05	59
pp (Pro-Palestinian)	0.35	0.09	0.15	65
n (Neutral)	0.50	0.11	0.19	35
ai (Anti-Israel)	0.24	0.73	0.36	52
ap (Anti-Palestinian)	0.04	0.06	0.05	35

- **Macro Average:**
  - Precision: 0.25
  - Recall: 0.21
  - F1-Score: 0.16


Confusion Matrix
True Label
Predicted Label
pi
pp
n
ai
ap
2 2 1 37 17
0 6 1 47 11
12 4 4 9 6
1 2 1 38 10
1 3 1 28 2
40
30
20
10
0


Per-Class Accuracy
Accuracy per Class
Label
1.0
0.8
0.6
0.4
0.2
0.0
pi
pp
n
ai
ap

1.1.2 Explanation of the Results

The overall accuracy of **21.14%** is only marginally better than random guessing in a five-class classification problem (where chance performance is around 20%). Below are several factors that likely contributed to these results:

1. **Data Representation and Feature Quality:**
   - **Subtle Linguistic Cues:**
     Political bias is often expressed through nuanced language. While BERT and SBERT embeddings capture many aspects of language, the subtleties that distinguish one bias from another might not be sufficiently prominent in the resulting vectors.
   - **Domain Specificity:**
     The new test data might use a slightly different vocabulary or style than the training data, leading to a drop in performance as the model struggles to generalize.
2. **Labeling and Overlap Between Classes:**
   - **Noisy Labels:**
     Our initial labeling was based on a rule-based approach using keyword matching. Such labels can be noisy or ambiguous, as sentences might contain elements of more than one bias, leading to overlapping class characteristics.
   - **Ambiguous Class Boundaries:**
     For example, a sentence with subtle cues might be misclassified between "pro-" and "anti-" categories. This ambiguity negatively affects both recall and precision for several classes.
3. **Class Imbalance and Sample Size:**
   - **Imbalanced Data Distribution:**
     Although we balanced the training set by sampling an equal number of examples per class, the inherent complexity and distinctiveness of each class vary. Classes like neutral or anti-palestinian have fewer distinct features in the text, making them harder to learn.
   - **Limited Support in Test Set:**
     With some classes having a relatively small support (e.g., 35 samples for neutral and anti-palestinian), even a few misclassifications can dramatically lower the metrics.
4. **Model Complexity and Training:**
   - **Underfitting vs. Overfitting:**
     The modest overall improvement above chance (only ~1.14% above 20%) suggests that the models might not have been complex enough to capture the nuanced patterns in the data—or that they overfitted the training data and thus failed to generalize.

- **Hyperparameter Tuning:**
  The hyperparameters chosen (or the network architecture in the ANN) might not be optimal for this complex task, resulting in low precision for some classes and a tendency to over-predict others (e.g., the anti-Israel class shows high recall but very low precision).

5. **Bias in Feature Extraction:**

   - **Distinct Features for "ai" Class:**
     The anti-Israel (ai) class shows a high recall (73%), indicating that the features associated with this class might be more prominent or consistent in the text. However, the low precision (24%) suggests that many sentences not belonging to this class are misclassified as anti-Israel, likely due to overlapping vocabulary or insufficient discriminative features.

2 Part B: Vector Based Model

In this part we are tasked to create a new model based on vectors (BERT and SBERT) and then use 3 different classifiers to classify the sentences into 5 categories. So the work we did was as follows:

1. **Prepare the data** - Get the original article data and prepare it for vectorization, meaning separte into sentences and clean the text etc.
2. **Create the vectors** - Create the vectors using BERT and SBERT methods.
3. **Prepare the data** - Create a base classifier using simple logic.
4. **Separte the data into test and train** - Split the data into test and train + validation.
5. **Create the classifiers** - Create the 3 classifiers as mentioned in the assignment.
6. **Train the classifiers** - Train the classifiers on the data.
7. **Test the classifiers** - Test the classifiers on the test data.
8. **Show the results** - Show the results in with graph and matrices.

2.1 Data Preparation

2.1.1 Basic Data Preparation

In order to save time we took the output from the first assignment (meaning the articles after basic processing) meaning we have for each of our 4 journals (NYT, JP, AJ, BBC) a `csv` file (stored in our GitHub repo for the course) that contains the following columns:

- `id` - the id of the article in the following format `aj_1` meaning it is the first article from AJ.
- `document` - the text body of the article.

Then we have a function called `extract_all_sentences` that will read through an article and extract all the sentences and then clean them. The cleaning process is as follows:

```python
import spacy

nlp = spacy.load("en_core_web_sm")

def extract_all_sentences(df):
    all_sentences = []
    for index, row in df.iterrows():
        doc = nlp(row["document"])
        for sent in doc.sents:
            # Optionally clean text
            sentence = clean_text(sent.text.strip())
            all_sentences.append({"id": row["id"], "document": sentence})
    return all_sentences
```

The `clean_text` function make sure that we have a clean text that contain only UTF-8 characters and no special characters.

After applying this function for each of our 4 journals we have a new dataframe that contains the following columns:

- `id` - the id of the article of the sentence.
- `sentence` - the text body of the article.

Now we concat all the dataframe into 1 big dataframe that contains all the sentences from all the journals. This new dataframe contains 46242 sentences.

2.1.2 Basic Logical Classification

Now we will use a simple logical classification of our data. We have 4 files `pro-israel.txt`, `pro-palestine.txt`, `anti-israel.txt`, `anti-palestine.txt` those file contains ≈ 20 words with clear bias towards the mentioned group. We will use those words to classify our sentences.

The idea of the classifier is simple if a sentence contains a word from a certain group we classify it as that group. But this time we can have a multi-calss classification meaning a sentence can be classified as two or more classes at the same time, We did this by adding to our dataframe for each sentence 5 columns one for each class, and we put a `1` if the sentence contains a word from that class and `0` otherwise. For the neutral class we classify a sentence as neutral if it doesn't contain a word from any of the classes.

```python
def classify_sentence(sentence):
    # Tokenize and stem the sentence
    tokens = word_tokenize(sentence.lower())
    # Initialize the one-hot encoded vector
    # [pro-israel, pro-palestine, neutral, anti-israel, anti-palestine]
    vector = [0, 0, 0, 0, 0]
    # Check for each class
    if stemmed_tokens & pro_israel_words:
        vector[0] = 1 # pro-israel
    if stemmed_tokens & pro_palestine_words:
        vector[1] = 1 # pro-palestine
    if stemmed_tokens & anti_israel_words:
        vector[3] = 1 # anti-israel
    if stemmed_tokens & anti_palestine_words:
        vector[4] = 1 # anti-palestine
    # Check if the sentence is neutral (no words from any class)
    if not any(vector):
        vector[2] = 1 # neutral
    return vector
```

Now we have a dataframe that contains the following columns:

- `id` - the id of the article of the sentence.
- `sentence` - the text body of the article.
- `pro-israel`
- `pro-palestine`
- `neutral`
- `anti-israel`
- `anti-palestine`

2.1.3 Vectorization of the Sentences

Now that we have the sentences and their basic classification we can start the vectorization process. We will use the `sentence-transformers` library to create the vectors for our sentences. We will use two methods:

1. BERT - We will use the `bert-base-uncased` model to create the vectors.
2. SBERT - We will use the `paraphrase-MiniLM-L6-v2` model to create the vectors.

```
# Load BERT model and tokenizer
bert_tokenizer = BertTokenizer.from_pretrained('bert-base-uncased')
bert_model = BertModel.from_pretrained('bert-base-uncased')

# Load SBERT model
sbert_model = SentenceTransformer('all-MiniLM-L6-v2')
```

Now we have 2 functions to convert each sentence into a vector:

```
def get_bert_embedding(sentence):
    device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
    bert_model.to(device)
    inputs = bert_tokenizer(sentence, return_tensors='pt',
         truncation=True, padding=True).to(device)
    with torch.no_grad():
        outputs = bert_model(**inputs)
    embeddings = outputs.last_hidden_state.sum(dim=1).squeeze().cpu().numpy()
    return embeddings

def get_sbert_embedding(sentence):
    device = "cuda" if torch.cuda.is_available() else "cpu"
    return sbert_model.encode(sentence, device=device)
```

Then we converted each sentence into both the BERT and SBERT vectors and added them to our dataframe.

2.1.4 Filtering the Data

Now we needed to take a subset of the data for the training process. We did some analilysis on the data to find out what kind is the distribution of the data. We took only the data that only had 1 class and printed the amount for each class:

```
pro_israel: 2523,
pro_palestine: 3380,
neutral: 30222,
anti_israel: 1495,
anti_palestine: 1255
```

Meaning if we want each class to have the same amount of samples to train on we need to take at most `1255` samples from each class. So we took a random sample of `1255` samples from each class.

```
pro_israel_sample = pro_israel_df.sample(n=num_samples, random_state=42)
pro_palestine_sample = pro_palestine_df.sample(n=num_samples, random_state=42)
neutral_sample = neutral_df.sample(n=num_samples, random_state=42)
anti_israel_sample = anti_israel.sample(n=num_samples, random_state=42)
anti_palestine_sample = anti_palestine_df.sample(n=num_samples,
                                                    random_state=42)
```

2.1.5 Train and Test Split

For the train and test data we needed to take only the vectors (BERT, SBERT) and their class:

```
pro_palestine_bert_data = df_pro_palestine["bert_embedding"]
pro_palestine_sbert_data = df_pro_palestine["sbert_embedding"]

pro_palestine_bert_data = np.array([eval(instance) for instance in
                    pro_palestine_bert_data])
pro_palestine_sbert_data = np.array([eval(instance) for instance in
                    pro_palestine_sbert_data])

print(pro_palestine_bert_data.shape)
print(pro_palestine_sbert_data.shape)
```

```
(1255, 768) (1255, 384)
```

And it was the same for all the classes.

Now we needed to create our data:

```
X_bert = np.vstack([
    pro_israel_bert_data,
    pro_palestine_bert_data,
    neutral_bert_data,
    anti_palestine_bert_data,
    anti_israel_bert_data
])

y_bert = np.array(
    [0] * len(pro_israel_bert_data) + # 0s for pro-israel
    [1] * len(pro_palestine_bert_data) + # 1s for pro-palestinian
    [2] * len(neutral_bert_data) + # 2s for neutral
    [3] * len(anti_palestine_bert_data) + # 3s for anti-palestinian
    [4] * len(anti_israel_bert_data) # 4s for anti-israel
)

y_bert_onehot = to_categorical(y_bert)
```

Same for the SBERT data.

Then we split the data into train and test:

```
# Split the data (80% train, 20% test)
X_bert_train, X_bert_test, y_bert_train, y_bert_test = train_test_split(
    X_bert, y_bert_onehot, test_size=0.2, random_state=42
)

# Further split training data to create validation set (10% of original data)
X_bert_train, X_bert_val, y_bert_train, y_bert_val = train_test_split(
    X_bert_train, y_bert_train, test_size=0.125, random_state=42
    # 0.125 of 80% is 10% of total
)
```

Same for the SBERT data.

Now we have the data ready for training.

2.2 Model Preparation

Now we will show the implementation of the 3 models:

1. SVM
2. Logistic Regression
3. ANN

2.2.1 SVM

```
from sklearn.svm import SVC
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import cross_val_score, GridSearchCV
from sklearn.metrics import classification_report, confusion_matrix
import numpy as np

# SVM for BERT
svm_bert = SVC(probability=True)
svm_bert_params = {
    'C': [0.1, 1, 10],
    'kernel': ['linear', 'rbf'],
    'class_weight': [None, 'balanced']
}
grid_svm_bert = GridSearchCV(svm_bert, svm_bert_params, cv=10,
    scoring='accuracy')
grid_svm_bert.fit(X_bert_train, np.argmax(y_bert_train, axis=1))
best_svm_bert = grid_svm_bert.best_estimator_
```

What we did is we created an SVM model, and then we used `GridSearchCV` to find the best parameters for the model.

2.2.2 Logistic Regression

```
# Logistic Regression for SBERT
lor_sbert = LogisticRegression(multi_class='multinomial', max_iter=1000)
lor_sbert_params = {
    'C': [0.1, 1, 10],
    'solver': ['lbfgs', 'newton-cg'],
    'class_weight': [None, 'balanced']
}

grid_lor_sbert = GridSearchCV(lor_sbert, lor_sbert_params, cv=10, scoring='accuracy')
grid_lor_sbert.fit(X_sbert_train, np.argmax(y_sbert_train, axis=1))
best_lor_sbert = grid_lor_sbert.best_estimator_
```

Same as the SVM model we used `GridSearchCV` to find the best parameters for the model.

2.2.3 ANN

The ANN had to be created with the following requirements:

- Data split: 80% training (with 10% validation), 20% testing
- Maximum 15 epochs
- Batch size of 32
- Specific architecture requirements:
  - 4 hidden layers (3×32 nodes, 1×16 nodes)
  - ReLU activation for hidden layers
  - Softmax for output layer
  - Include callbacks:
    - Early stopping after 5 iterations without improvement
    - Model checkpoint for best accuracy

Callbacks

```
# Callbacks
early_stopping = EarlyStopping(
    monitor='val_accuracy',
    patience=5,
    restore_best_weights=True
)



# Create separate checkpoints for BERT and SBERT models
bert_checkpoint = ModelCheckpoint(
    'best_bert_model.h5',
    monitor='val_accuracy',
    save_best_only=True,
)

...
```

Model Creation + Compilation

```
# BERT Model (input dimension 768)
bert_model = Sequential([
    Dense(32, activation='relu', input_dim=768),
    Dense(32, activation='relu'),
    Dense(32, activation='relu'),
    Dense(16, activation='relu'),
    Dense(5, activation='softmax') # 5 classes
])

sbert_model = ...
```

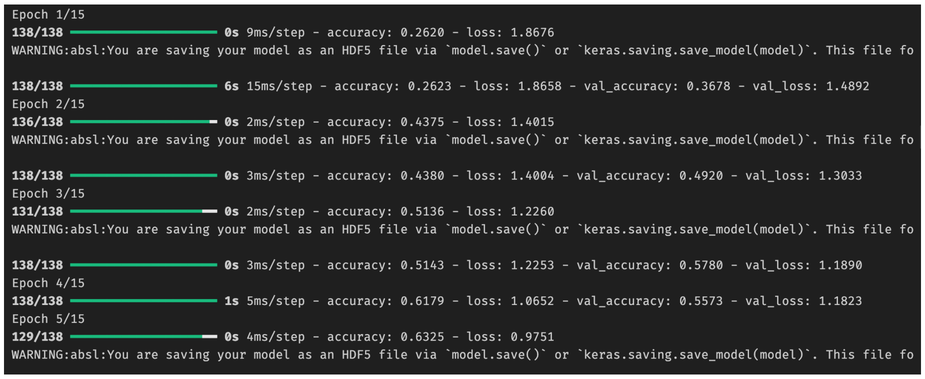
```python
# Compile models
bert_model.compile(
    optimizer='adam',
    loss='categorical_crossentropy',
    metrics=['accuracy']
)

...
```

2.3 Model Implementation

Model Training

```python
# Train BERT model
bert_history = bert_model.fit(
    X_bert_train,
    y_bert_train,
    batch_size=32,
    epochs=15,
    validation_data=(X_bert_val, y_bert_val),
    callbacks=[early_stopping, bert_checkpoint],
    verbose=1
)
```

We have this for the training process:



```
Epoch 1/15
138/138 ━━━━━━━━━━━━━━━━━━━━ 0s 9ms/step - accuracy: 0.2620 - loss: 1.8676
WARNING:absl:You are saving your model as an HDF5 file via `model.save()` or `keras.saving.save_model(model)`. This file fo

138/138 ━━━━━━━━━━━━━━━━━━━━ 6s 15ms/step - accuracy: 0.2623 - loss: 1.8658 - val_accuracy: 0.3678 - val_loss: 1.4892
Epoch 2/15
136/138 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.4375 - loss: 1.4015
WARNING:absl:You are saving your model as an HDF5 file via `model.save()` or `keras.saving.save_model(model)`. This file fo

138/138 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.4380 - loss: 1.4004 - val_accuracy: 0.4920 - val_loss: 1.3033
Epoch 3/15
131/138 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.5136 - loss: 1.2260
WARNING:absl:You are saving your model as an HDF5 file via `model.save()` or `keras.saving.save_model(model)`. This file fo

138/138 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.5143 - loss: 1.2253 - val_accuracy: 0.5780 - val_loss: 1.1890
Epoch 4/15
138/138 ━━━━━━━━━━━━━━━━━━━━ 1s 5ms/step - accuracy: 0.6179 - loss: 1.0652 - val_accuracy: 0.5573 - val_loss: 1.1823
Epoch 5/15
129/138 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - accuracy: 0.6325 - loss: 0.9751
WARNING:absl:You are saving your model as an HDF5 file via `model.save()` or `keras.saving.save_model(model)`. This file fo
```

2.4 Testing and Evaluation

2.4.1 SVM

```python
def evaluate_model(model, X_test, y_test, class_names):
    y_pred = model.predict(X_test)
    print("Classification Report:")
    print(classification_report(np.argmax(y_test, axis=1),
            y_pred, target_names=class_names))
    # Confusion Matrix
    cm = confusion_matrix(np.argmax(y_test, axis=1), y_pred)
    print("\nConfusion Matrix:")
    print(cm)
    return model.predict_proba(X_test)

# Evaluate models (assuming class_names is already defined)
print("\nSVM BERT:")
svm_bert_proba = evaluate_model(best_svm_bert, X_bert_test,
                y_bert_test, class_names)

print("\nSVM SBERT:")
svm_sbert_proba = evaluate_model(best_svm_sbert, X_sbert_test,
                y_sbert_test, class_names)
```

```
SVM BERT:

Classification Report:

    precision recall f1-score support
pro_israel      0.80 0.43 0.56 28
pro_palestine   0.53 0.71 0.61 14
neutral         0.26 0.50 0.34 10
anti_israel     0.60 0.50 0.55 24
anti_palestine  0.44 0.50 0.47 24

accuracy     0.51 100
macro avg    0.53 0.53 0.51 100
weighted avg 0.57 0.51 0.52 100

Confusion Matrix:
    [[12 3 2 5 6]
    [ 1 10 3 0 0]
    [ 1 0 5 1 3]
    [ 1 3 2 12 6]
    [ 0 3 7 2 12]]

SVM SBERT:

Classification Report:

    precision recall f1-score support
pro_israel       0.78 0.50 0.61 28
pro_palestine    0.56 0.71 0.62 14
neutral          0.33 0.70 0.45 10
anti_israel      0.73 0.67 0.70 24
anti_palestine   0.71 0.62 0.67 24
```

```
accuracy      0.62 100
macro avg     0.62 0.64 0.61 100
weighted avg  0.67 0.62 0.63 100

Confusion Matrix:

[[14 3 7 3 1]
[ 1 10 2 1 0]
[ 1 1 7 0 1]
[ 0 1 3 16 4]
[ 2 3 2 2 15]]
```

2.4.2 Logistic Regression

The testing method is the same as for the SVM model:

```
print("\nLogistic Regression BERT:")
lor_bert_proba = evaluate_model(best_lor_bert, X_bert_test,
                        y_bert_test, class_names)
```

```
print("\nLogistic Regression SBERT:")
lor_sbert_proba = evaluate_model(best_lor_sbert, X_sbert_test,
                        y_sbert_test, class_names)
```

```
Logistic Regression BERT:

Classification Report:
    precision recall f1-score support
pro_israel      0.68 0.46 0.55 28
pro_palestine   0.47 0.57 0.52 14
neutral         0.25 0.50 0.33 10
anti_israel     0.61 0.58 0.60 24
anti_palestine  0.43 0.38 0.40 24

accuracy      0.49 100
macro avg     0.49 0.50 0.48 100
weighted avg  0.53 0.49 0.50 100

Confusion Matrix:

    [[13 4 3 4 4]
    [ 1 8 3 0 2]
    [ 1 0 5 1 3]
    [ 2 1 4 14 3]
    [ 2 4 5 4 9]]

Logistic Regression SBERT:

Classification Report:
    precision recall f1-score support
pro_israel       0.79 0.54 0.64 28
pro_palestine    0.56 0.71 0.62 14
neutral          0.28 0.50 0.36 10
anti_israel      0.70 0.58 0.64 24
anti_palestine   0.64 0.67 0.65 24

accuracy      0.60 100
macro avg     0.59 0.60 0.58 100
weighted avg  0.65 0.60 0.61 100

Confusion Matrix:

    [[15 3 6 2 2]
    [ 1 10 2 1 0]
    [ 1 1 5 1 2]
    [ 1 1 3 14 5]
    [ 1 3 2 2 16]]
```

2.4.3 ANN

The testing process for the ANN is a bit different:

```python
def get_metrics(y_true, y_pred, class_names):
    # Convert one-hot encoded labels back to class indices
    if len(y_true.shape) > 1: # if one-hot encoded
        y_true = np.argmax(y_true, axis=1)
    if len(y_pred.shape) > 1: # if one-hot encoded
        y_pred = np.argmax(y_pred, axis=1)

    # Calculate metrics
    accuracy = accuracy_score(y_true, y_pred)
    precision, recall, f1, _ = precision_recall_fscore_support(y_true,
                                 y_pred, average=None)

    # Print detailed results
    print("Overall Accuracy:", accuracy)
    print("\nPer-class metrics:")
    for i, class_name in enumerate(class_names):
        print(f"\n{class_name}:")
        print(f"Precision: {precision[i]:.4f}")
        print(f"Recall: {recall[i]:.4f}")
        print(f"F1-score: {f1[i]:.4f}")

    # Return metrics for further use if needed
    return {
        'accuracy': accuracy,
        'precision': precision,
        'recall': recall,
        'f1': f1
    }
```

```
BERT Model Metrics:
==================
Overall Accuracy: 0.6597609561752988
Per-class metrics:
    Pro-Israel:
        Precision: 0.6992
        Recall: 0.6300
        F1-score: 0.6628
    Pro-Palestinian:
        Precision: 0.5831
        Recall: 0.7511
        F1-score: 0.6565
    Neutral:
        Precision: 0.6545
        Recall: 0.5000
        F1-score: 0.5669
    Anti-Palestinian:
        Precision: 0.6850
        Recall: 0.7016
        F1-score: 0.6932
    Anti-Israel:
        Precision: 0.6877
        Recall: 0.7255
        F1-score: 0.7061

BERT Confusion Matrix:

    Confusion Matrix:
    ----------------
    True\Pred Pro- Pro- Neut Anti Anti
    Pro- 172 32 30 22 17
    Pro- 15 172 19 8 15
    Neut 29 45 125 23 28
    Anti 16 23 11 174 24
    Anti 14 23 6 27 185

SBERT Model Metrics:
```

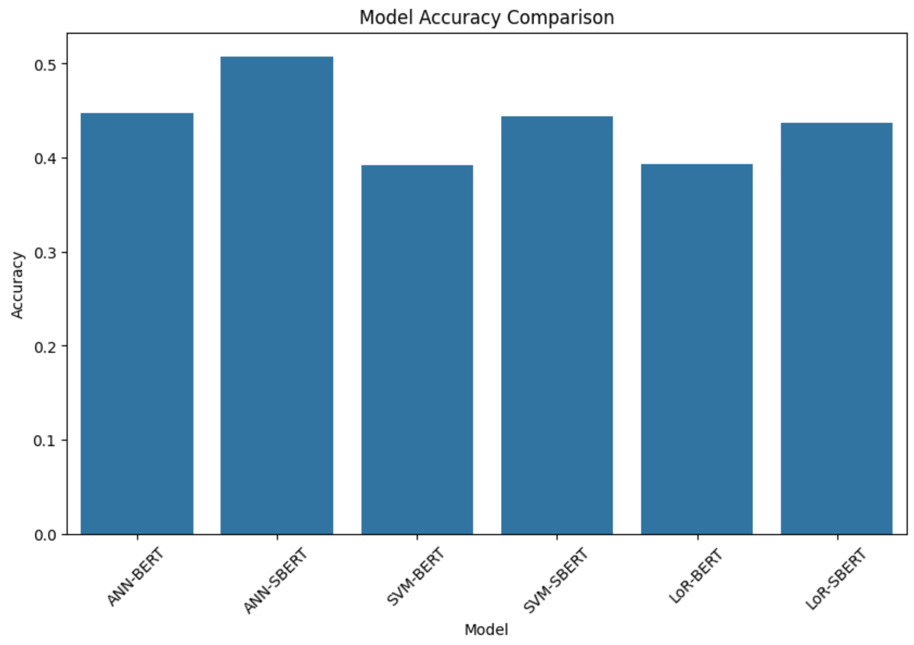
```
===================
Overall Accuracy: 0.7450199203187251

Per-class metrics:
    Pro-Israel:
        Precision: 0.8376
        Recall: 0.7179
        F1-score: 0.7732
    Pro-Palestinian:
        Precision: 0.7040
        Recall: 0.8515
        F1-score: 0.7708
    Neutral:
        Precision: 0.7015
        Recall: 0.5640
        F1-score: 0.6253
    Anti-Palestinian:
        Precision: 0.7385
        Recall: 0.7742
        F1-score: 0.7559
    Anti-Israel:
        Precision: 0.7456
        Recall: 0.8275
        F1-score: 0.7844

SBERT Confusion Matrix:

    Confusion Matrix:
    ----------------
    True\Pred Pro- Pro- Neut Anti Anti
    Pro- 196 14 24 13 26
    Pro- 8 195 10 9 7
    Neut 21 43 141 25 20
    Anti 4 17 16 192 19
    Anti 5 8 10 21 211
```

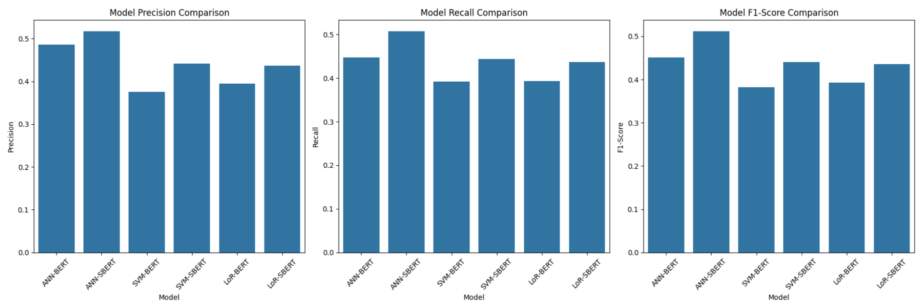
2.5 Results Documentation

This is the table comparing the results of the 3 models:

Model	Accuracy	Precision	Recall	F1-Score
ANN-BERT	0.447490	0.486108	0.447490	0.451738
ANN-SBERT	0.507729	0.517275	0.507729	0.511968
SVM-BERT	0.391873	0.375541	0.391873	0.382335
SVM-SBERT	0.444303	0.441490	0.444303	0.440945

Model	Accuracy	Precision	Recall	F1-Score
LoR-BERT	0.392829	0.394762	0.392829	0.392715
LoR-SBERT	0.436653	0.436328	0.436653	0.436001





We will show graphs and metrics for each of the model

2.5.1 SVM & LoR

2.5.2 ANN

2.6 Final File

Provided in the `.xlx` file

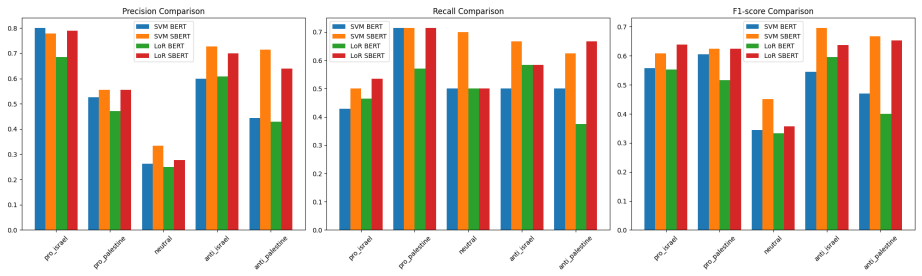


Figure 1: LoR and SVM

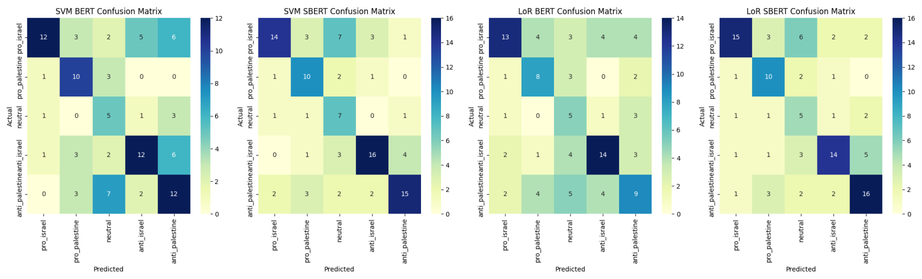


Figure 2: LoR and SVM CM

2.7 Conclusion

We have shown that the ANN model is the best model for this task, and we have shown the results for the other models. We have also shown the training process for the ANN model.

The best model was the ANN model with a test accuracy of 0.64. For the other models the accuracy was 0.59 for the SVM model and 0.57 for the LoR model.

The low accuracy of those model compared to our 95% in the last assignment is due to the complexity of the data natural-language and understanding the sentiment of the text is not an easy task.

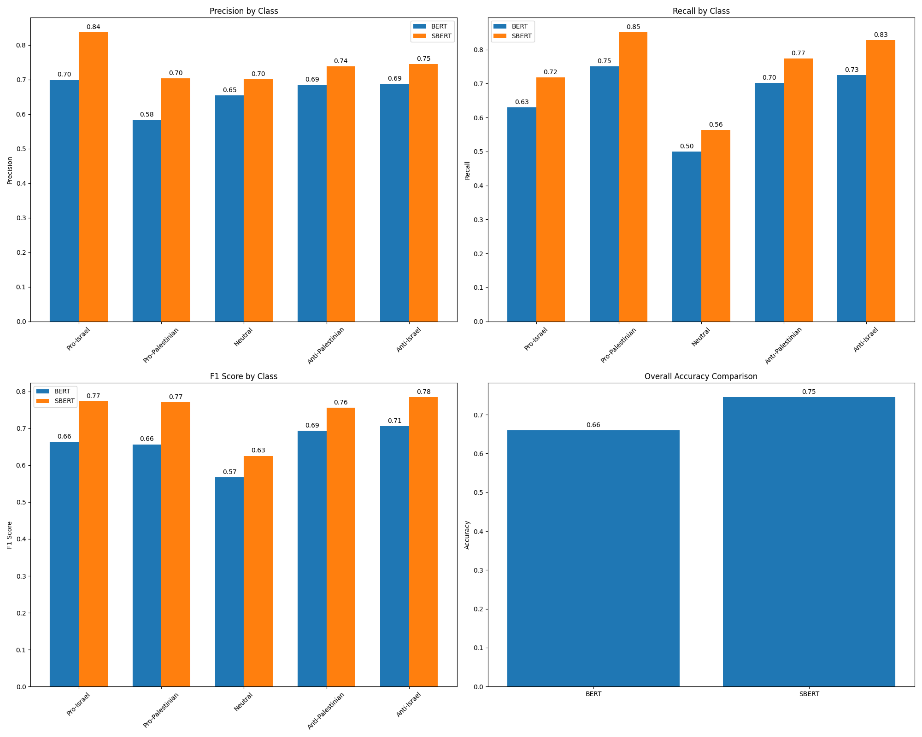


Figure 3: metric_ann

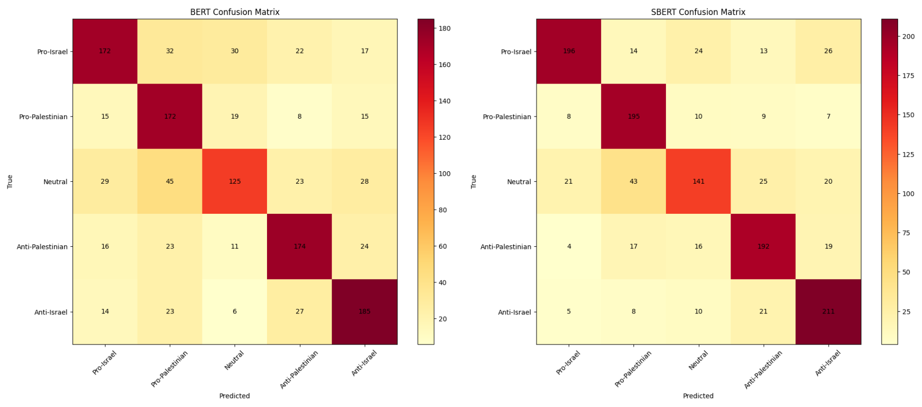


Figure 4: cm_ann

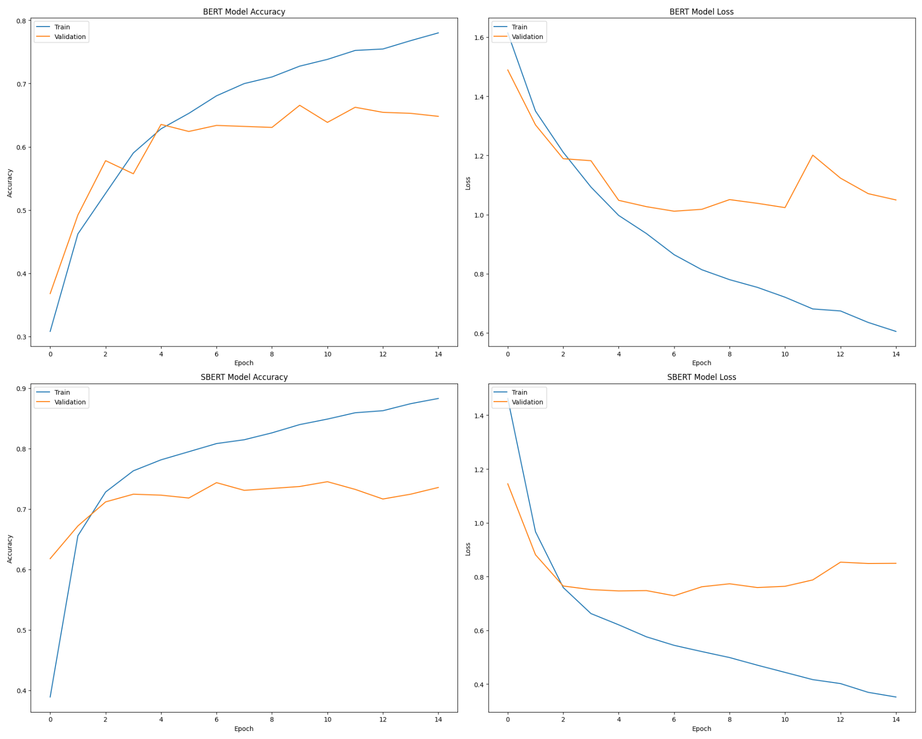
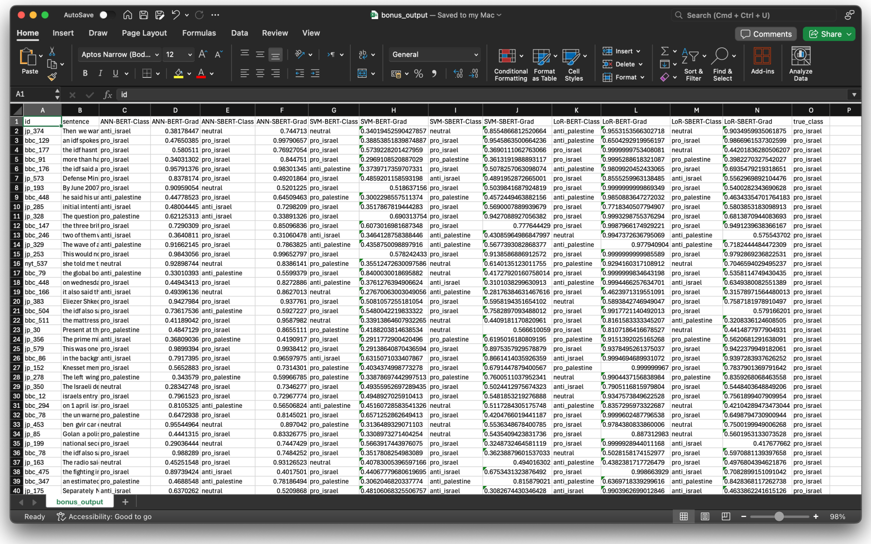


Figure 5: training_ann



id	sentence	ANN-BERT-Class	ANN-BERT-Grad	ANN-SBERT-Class	ANN-SBERT-Grad	SVM-BERT-Class	SVM-BERT-Grad	SVM-SBERT-Class	SVM-SBERT-Grad	LoR-BERT-Class	LoR-BERT-Grad	LoR-SBERT-Class	LoR-SBERT-Grad	true_class
jp_374	Then we war	anti_israel	0.38178447	neutral	0.744713	neutral	0.34019452590427857	neutral	0.8554866812520664	anti_palestine	0.9553153566302718	neutral	0.9034959935061875	pro_israel
bbc_129	an idf spokes	pro_israel	0.47650385	pro_israel	0.99790657	pro_israel	0.38853851839874887	pro_israel	0.9545863500664236	anti_palestine	0.6504292919956197	pro_israel	0.9866961537302599	pro_israel
bbc_177	the idf hasnt	pro_israel	0.580511	pro_israel	0.76927054	pro_israel	0.5739228201427959	pro_israel	0.3690111062763066	pro_israel	0.9999999753408081	neutral	0.44201836280506207	pro_israel
bbc_91	more than ha	pro_israel	0.34031302	pro_israel	0.844751	pro_israel	0.2969108520887029	pro_palestine	0.3613191988893117	pro_israel	0.9995288618321087	pro_palestine	0.3982270327542027	pro_israel
bbc_176	the idf said a	pro_israel	0.95791376	pro_israel	0.98301345	anti_palestine	0.3739717359707331	pro_israel	0.5078257063098074	anti_palestine	0.9809920452433065	pro_israel	0.6935479219318651	pro_israel
jp_573	Defense Mini	pro_israel	0.8378174	pro_israel	0.49201864	pro_israel	0.4859201158593198	anti_israel	0.4891952872665001	pro_israel	0.8555259963138485	anti_israel	0.5562969892104476	pro_israel
jp_193	By June 2007	pro_israel	0.90959054	neutral	0.5201225	pro_israel	0.518637156	pro_israel	0.5039841687924819	pro_israel	0.9999999999869349	pro_israel	0.5400282343690628	pro_israel
bbc_448	he said his ur	anti_palestine	0.44778523	pro_israel	0.64509463	pro_palestine	0.3002298557511374	pro_palestine	0.4572449463882156	anti_palestine	0.9850883647272032	pro_palestine	0.46343354701764183	pro_israel
jp_285	initial intenti	anti_israel	0.48004445	anti_israel	0.7298209	pro_israel	0.3517867819444283	pro_israel	0.5690007889939679	pro_israel	0.7718340507794907	pro_israel	0.5803853183098913	pro_israel
jp_328	The question	pro_palestine	0.62125313	anti_israel	0.33891326	pro_israel	0.690313754	pro_israel	0.9427088927056382	pro_israel	0.9993298755376294	pro_israel	0.6813870944083693	pro_israel
bbc_147	the three brit	pro_israel	0.7290309	pro_israel	0.85096836	pro_israel	0.6073016981687348	pro_israel	0.777644429	pro_israel	0.9987966174929221	pro_israel	0.9491239638366167	pro_israel
bbc_246	two of them v	anti_israel	0.3640811	pro_israel	0.31060478	anti_israel	0.3464128758388446	anti_palestine	0.43085964986847997	neutral	0.9947372636795069	anti_palestine	0.575543702	pro_israel
jp_329	The wave of a	anti_palestine	0.91662145	pro_israel	0.7863825	anti_palestine	0.4358750098897916	anti_palestine	0.5677393082868377	anti_palestine	0.977940904	anti_palestine	0.7182444484472309	pro_israel
jp_253	This would nc	pro_israel	0.9843056	pro_israel	0.99652797	pro_israel	0.578242433	pro_israel	0.9138586886912572	pro_israel	0.9999999999985589	pro_israel	0.9792869236822531	pro_israel
nyt_537	she told me t	neutral	0.92898744	neutral	0.8386141	pro_palestine	0.35512472630097586	neutral	0.6140135123011755	pro_palestine	0.9294160317108912	neutral	0.7046594029495237	pro_israel
bbc_79	the global bo	anti_palestine	0.33010393	anti_palestine	0.5599379	pro_israel	0.8400030018695882	neutral	0.41727920160758014	pro_israel	0.9999999834643198	pro_israel	0.5358114749430435	pro_israel
bbc_448	on wednesda	pro_israel	0.44943413	pro_israel	0.8272886	anti_palestine	0.3761276394906624	anti_israel	0.3101038299630913	anti_palestine	0.9994466257634701	anti_israel	0.6349380082551389	pro_israel
bbc_166	it also said th	anti_israel	0.49396136	neutral	0.8627013	neutral	0.27670063003049056	anti_palestine	0.28176384631467616	pro_israel	0.4623971319551091	pro_israel	0.31578971564480013	pro_israel
jp_383	Eliezer Shkec	pro_israel	0.9427984	pro_israel	0.937761	pro_israel	0.5081057255181054	pro_israel	0.5958194351654102	neutral	0.5893842746949047	pro_israel	0.7587181978910497	pro_israel
bbc_504	the idf also sa	pro_israel	0.73617536	anti_palestine	0.5927227	pro_israel	0.5480042219833322	pro_israel	0.7582897093488012	pro_israel	0.9917721140492013	pro_israel	0.579166201	pro_israel
bbc_511	the mattress	pro_israel	0.41189042	pro_israel	0.9587982	neutral	0.33913864607932265	neutral	0.4409181170820961	pro_israel	0.8161583333345207	anti_palestine	0.3208336124608505	pro_israel
jp_30	Present at th	pro_palestine	0.4847129	pro_israel	0.8655111	pro_palestine	0.4188203814638534	neutral	0.566610059	pro_israel	0.8107186416678527	neutral	0.4414877977904931	pro_israel
jp_356	The prime mi	anti_israel	0.36809036	pro_palestine	0.4190917	pro_israel	0.2911772900420496	pro_palestine	0.6195016180809195	pro_palestine	0.9151392025165268	pro_palestine	0.5620681291638091	pro_israel
jp_579	This was one	pro_israel	0.9899394	pro_israel	0.9938412	pro_israel	0.29138640870436594	pro_israel	0.8975357929578879	pro_israel	0.9378495261375037	pro_israel	0.9422379949182061	pro_israel
bbc_86	in the backgr	anti_israel	0.7917395	pro_israel	0.96597975	anti_israel	0.6315071033407867	pro_israel	0.8661414035926359	anti_israel	0.9994694689931072	pro_israel	0.9397283937626252	pro_israel
jp_152	Knesset mem	pro_israel	0.5652883	pro_israel	0.7314301	pro_palestine	0.4034374998773278	pro_israel	0.6791447879400567	pro_palestine	0.999999967	pro_israel	0.7837901369791642	pro_israel
jp_278	The left wing	pro_palestine	0.343579	pro_palestine	0.59966785	pro_palestine	0.33878697442997513	pro_palestine	0.7600511037952341	neutral	0.9904437156838984	pro_palestine	0.8359268068463558	pro_israel
jp_350	The Israeli de	neutral	0.28342748	pro_israel	0.7346277	pro_israel	0.49355952697289435	pro_israel	0.5024412975674323	anti_israel	0.7905116815979804	pro_israel	0.5448403648849206	pro_israel
bbc_12	israels entry	pro_israel	0.7961523	pro_israel	0.72967774	pro_israel	0.4948927025910413	pro_israel	0.5481853219276888	neutral	0.9347573849622528	pro_israel	0.7561899407909954	pro_israel
bbc_294	on 1 april isr	pro_israel	0.8105325	anti_palestine	0.56506824	anti_palestine	0.45160728583541326	neutral	0.5117284305175748	anti_palestine	0.8357295973322687	neutral	0.42104289473473044	pro_israel
bbc_78	the un warne	pro_palestine	0.6472938	pro_israel	0.8145021	pro_israel	0.6571252862649413	pro_israel	0.4204766019441187	pro_israel	0.9999602487796538	pro_israel	0.6498794730900944	pro_israel
jp_453	ben gvir car (	neutral	0.95544964	neutral	0.897042	pro_palestine	0.3136489329071103	neutral	0.5536348678400785	pro_israel	0.9784380833860006	neutral	0.7500199949006268	pro_israel
jp_85	Golan a polis	pro_palestine	0.4441315	pro_israel	0.83326775	pro_israel	0.3308973271404254	neutral	0.5435409423831736	pro_israel	0.887312983	neutral	0.5601953133073528	pro_israel
jp_199	national secu	pro_israel	0.29036444	neutral	0.7447429	pro_israel	0.5663917443976075	pro_israel	0.3248732464581119	pro_israel	0.9999928944011168	anti_israel	0.417677662	pro_israel
bbc_78	the idf also sa	pro_israel	0.988289	pro_israel	0.7484252	pro_israel	0.3517808254983089	pro_israel	0.36238879601537033	neutral	0.5028158174152977	pro_israel	0.5970881139397658	pro_israel
jp_163	The radio sai	neutral	0.45251548	pro_israel	0.93126523	neutral	0.40783005396597166	pro_israel	0.494016302	anti_palestine	0.4382381717726479	pro_israel	0.4976804394621876	pro_israel
bbc_475	the fighting ir	pro_israel	0.89739424	anti_israel	0.4017501	pro_israel	0.44067779680619695	anti_israel	0.6753431323876492	pro_israel	0.998663929	anti_israel	0.7082899151091042	pro_israel
bbc_347	an estimatec	pro_palestine	0.4688548	anti_palestine	0.78186494	pro_palestine	0.3062046820337774	anti_palestine	0.815879021	anti_palestine	0.6369718339299616	anti_palestine	0.8428368117262738	pro_israel
jp_175	Separately N	anti_israel	0.6370262	neutral	0.5209868	pro_israel	0.48106068325506757	anti_israel	0.3082674430346428	anti_israel	0.9903962699012846	anti_israel	0.4633862241615126	pro_israel

Figure 6: final_file