IS assignment 2

Vid Kališnik and Tomo Testen

January 2024

1 Data Preprocessing

1.1 Data Description

The dataset comprises the following attributes:
link, **headline**, **short_description**, **authors**, **date**, **category**

Attribute **category** is being predicted based on the other attributes.

1.1.1 Number of Samples

Total number of samples in the dataset: **1**48122

1.1.2 Null elements

Null elements are only found in **headline** (731 samples) and **short_description** (736 samples).

1.1.3 Reappearing authors

Some authors appear more than once:

Authors	Count
Lee Moran	2058
Ron Dicker	1726
Cole Delbyck	1263
Reuters, Reuters	1141
...	...

Table 1: Authors and their respective counts.

What is more interesting, is that only about 4% off all authors write for more than one category.

1.1.4 Target Class Balance

Distribution of target classes can be seen in Figure 1, By far the most articles are from POLITICS category.

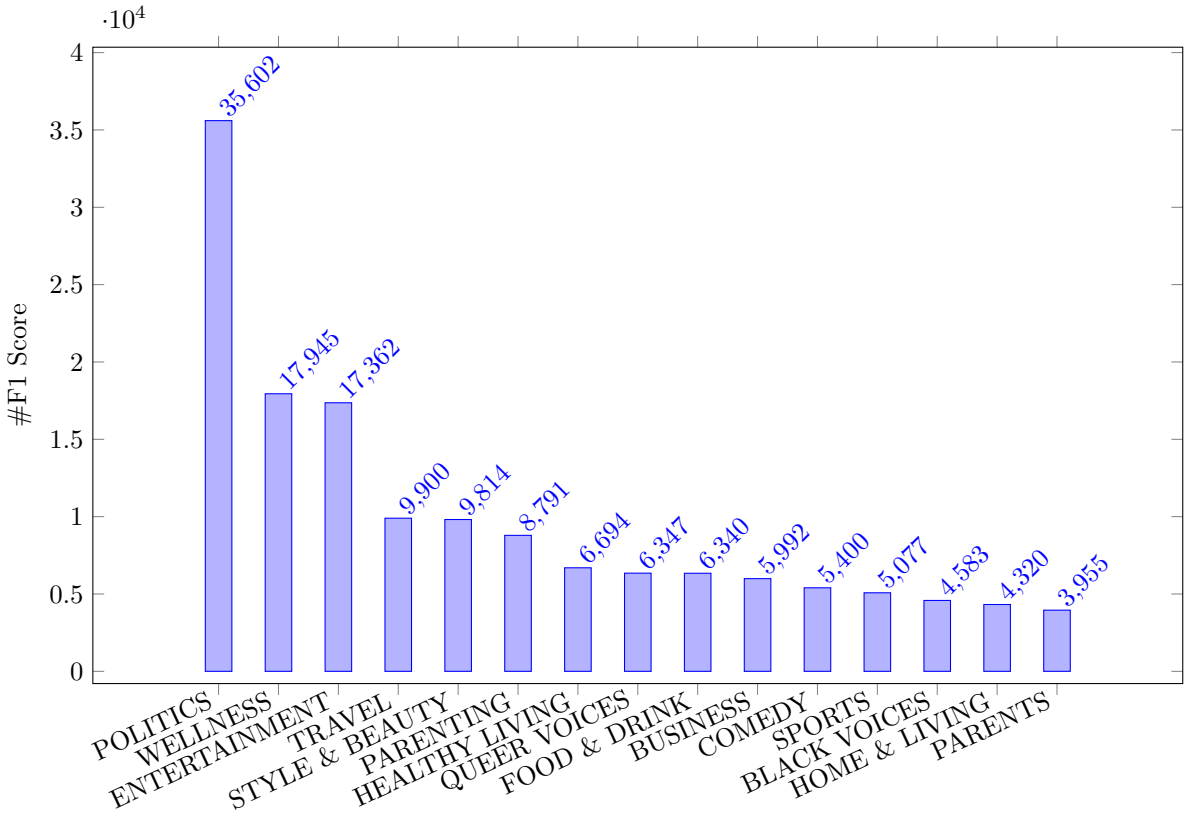

Figure 1: Articles per category

1.2 Data Cleaning

As **link** and **date** are not needed they are dropped.
Rows where either **headline** or **short_description** is null are also dropped (there is about 1% of them).

Column **text** is then concatenated from **headline**, **short_description** and **authors**.
clean_text(text) function transforms the text to lower case, separates the words as tokens, removes punctuations and stopwords and applies lemmatization.

```
    def clean_text(text):

        # Transform to lower case
        text = text.lower()

        # Tokenization
        words = word_tokenize(text)

        # Remove punctuation
        table = str.maketrans('', '', string.punctuation)
        words = [word.translate(table) for word in words if word.isalpha()]

        # Remove stopwords
        stop_words = set(stopwords.words('english'))
        words = [word for word in words if word not in stop_words]

        # Lemmatization
        lemmatized_words = [lemmatizer.lemmatize(word) for word in words]

        # Join the words back into a string
        preprocessed_text = ' '.join(lemmatized_words)

        return preprocessed_text
```

This is applied to **text** and saved as **clean_text**.

To enable compatibility with algorithms that require numerical input, **category** attribute is encoded to unique integer values with **sklearn.preprocessing.LabelEncoder()**

1.3 Data Splits

1.3.1 Test,train split

Division of dataset into training, validation, and test sets:

- Training set size: 80%
- Test set size: 20%

```
    from sklearn.model_selection import train_test_split

    X_train, X_test, y_train, y_test = train_test_split(data['clean_text'], data['
    category'], test_size=0.2, random_state=42)
```

1.3.2 Resampling

From Figure 1, it is observed that the category POLITICS has a considerably greater number of cases than other categories. An attempt was made to address this by employing resampling, ensuring that all categories are represented by an equal number.

Multiple types of resampling exist, with the simplest ones being down and up sampling. In the former, random cases from categories with a surplus are deleted, equalizing the quantities across all categories. In the latter, random cases from categories with fewer instances are duplicated.

Due to the already substantial size of our dataset, downsampling was chosen and implemented using TF-IDF vectorization.

2 Details of Classification Models

2.1 Vectorization

The classification models cannot make predictions on sentences/words, so they need to be transformed into numeric values. Three methods were utilized: **TF-IDF**, **Word2Vec**, and a **pretrained BERT** model.

2.1.1 TF-IDF

TF-IDF is a numerical statistic that reflects the importance of a term in a document relative to a collection of documents. The term frequency (**TF**) is the number of times a term appears in a document, and it is calculated as:

$$\text{TF}(t, d) = \frac{\text{Number of times term } t \text{ appears in document } d}{\text{Total number of terms in document } d}$$

The inverse document frequency (**IDF**) is the logarithmically scaled inverse fraction of the documents that contain the term, and it is calculated as:

$$\text{IDF}(t, D) = \log\left(\frac{\text{Total number of documents in the collection } N}{\text{Number of documents containing term } t}\right)$$

Finally, the **TF-IDF** score for a term in a document is given by the product of **TF** and **IDF**:

$$\text{TF-IDF}(t, d, D) = \text{TF}(t, d) \times \text{IDF}(t, D)$$

```
from sklearn.feature_extraction.text import TfidfVectorizer

vectorizer = TfidfVectorizer()
X_train_tfidf = vectorizer.fit_transform(X_train)
X_test_tfidf = vectorizer.transform(X_test)
```

2.1.2 Word2Vec

Word2Vec represents words as dense vectors, allowing for semantic similarities between words to be captured in the vector space. For the whole text vector, we took the average of word vectors.

```
from gensim.models import Word2Vec

w2v_model = Word2Vec(tokenized_train_text,
        vector_size=100,
        window=5,
        min_count=1,
        workers=4,
        epochs=10)
```

2.1.3 Sentence Embeddings with BERT

BERT (Bidirectional Encoder Representations from Transformers) is a natural language processing model that captures contextual information bidirectionally within sentences. Sentence embeddings with **BERT** involve obtaining fixed-size vector representations for entire sentences or phrases.
all-mpnet-base-v2 is a sentence-transformers model from **sentence_transformers** library: It maps sentences and paragraphs to a 768 dimensional dense vector space and can be used for tasks like clustering or semantic search.

```
def embedding(texts):
    device = torch.device("cuda")

    # Load pre-trained sentence embedding model(all-mpnet-base-v2)
    model = SentenceTransformer('all-mpnet-base-v2').to(device)

    # Calculate embeddings
    embeddings = model.encode(texts)

    return embeddings
```

2.2 Classification Models

2.2.1 Logistic Regression

Logistic Regression models the probability of a binary outcome by fitting a logistic function to a set of independent variables.

```
from sklearn.linear_model import LogisticRegression
logistic_model = LogisticRegression(max_iter=1000)
logistic_model.fit(X_train, y_train)
```

2.2.2 Random Forest

Random Forest builds multiple decision trees during training and merges them to get a more accurate and stable prediction.

```
from sklearn.ensemble import RandomForestClassifier
rf_model = RandomForestClassifier()
rf_model.fit(X_train, y_train)
```

2.2.3 K-Nearest Neighbors

K-Nearest Neighbors is a simple algorithm where an object is classified by its neighbors based on majority voting.

```
from sklearn.neighbors import KNeighborsClassifier
knn_model = KNeighborsClassifier(n_neighbors=1000)
knn_model.fit(X_train, y_train)
```

2.2.4 XGBoost

XGBoost (Extreme Gradient Boosting) is a gradient boosting framework that uses decision trees as base learners and builds a predictive model.

```
from xgboost import XGBClassifier
xgb_model = XGBClassifier()
xgb_model.fit(X_train, y_train)
```

2.2.5 LinearSVC

Linear Support Vector Classification is a linear model that tries to find a hyperplane that best separates different classes. LinearSVC was used instead of normal SVC, because the normal one is too computationally expensive.

```
from sklearn.svm import LinearSVC
svc_model = LinearSVC()
svc_model.fit(X_train, y_train)
```

2.2.6 DistilBERT

DistilBERT is a distilled version of BERT, which is a pre-trained transformer-based model for natural language understanding and processing. As DistilBERT is pre-trained and it comes with its own tokenizer.

```
tokenizer = DistilBertTokenizerFast.from_pretrained('distilbert-base-uncased')
model = DistilBertForSequenceClassification.from_pretrained('distilbert-base-
uncased', num_labels=15)
```

2.3 DistilBERT Fine-Tunning

DistilBERT initially demonstrates strong performance, leveraging its pre-training on diverse data. However, fine-tuning becomes essential to tailor the model to specific task nuances. In this case, the fine-tuning process is limited to just three epochs, as an increase in validation loss indicates a potential risk of overfitting or diminished generalization beyond this point. This strategic approach ensures that DistilBERT is optimized for the task while avoiding performance degradation.

```
def train_model(model, train_loader, valid_loader, num_epochs, device, optim):
    for epoch in range(NUM_EPOCHS):

        model.train()

        for batch_idx, batch in enumerate(train_loader):

            # Prepare data
            input_ids = batch['input_ids'].to(DEVICE)
            attention_mask = batch['attention_mask'].to(DEVICE)
            labels = batch['labels'].to(DEVICE)

            # Forward
            outputs = model(input_ids, attention_mask=attention_mask, labels=labels)
            loss, logits = outputs['loss'], outputs['logits']

            # Backward
            optim.zero_grad()
            loss.backward()
            optim.step()

            ### Logging
            if not batch_idx % 250:
                print (f'Epoch: {epoch+1:04d}/{NUM_EPOCHS:04d} | '
                    f'Batch {batch_idx:04d}/{len(train_loader):04d} | '
                    f'Loss: {loss:.4f}')


        model.eval()

        with torch.set_grad_enabled(False):

            valid_accuracy = compute_accuracy(model, valid_loader, DEVICE)

            print(f'\nvalid accuracy: '
                        f'{valid_accuracy:.2f}%')
```

3 Results

3.1 Performance Matrices

3.1.1 TF-IDF

Model	Accuracy	Precision	Recall	F1-Score
Logistic Regression	0.832	0.833	0.832	0.829
Random Forest	0.79	0.793	0.79	0.781
KNN (1000)	0.681	0.735	0.681	0.646
XGBoost	0.805	0.808	0.805	0.802
Linear SVC	**0.847**	**0.845**	**0.847**	**0.845**

Table 2: Performance Metrics of TF-IDF

Linear SVC outperforms all models.

3.1.2 Word2Vec

Model	Accuracy	Precision	Recall	F1-Score
Logistic Regression	0.752	0.745	0.752	0.745
Random Forest	0.723	0.735	0.723	0.7
KNN (1000)	0.681	0.683	0.681	0.636
XGBoost	**0.763**	**0.757**	**0.763**	**0.757**
Linear SVC	0.739	0.731	0.739	0.72

Table 3: Performance Metrics of Word2Vec

XGBoost outperforms all models.

3.1.3 TF-IDF Resampled

Model	Accuracy	Precision	Recall	F1-Score
Logistic Regression	0.796	0.818	0.796	0.802
Random Forest	0.747	0.785	0.747	0.755
KNN (1000)	0.667	0.712	0.667	0.673
XGBoost	0.77	0.8	0.77	0.778
Linear SVC	**0.798**	**0.82**	**0.798**	**0.803**

Table 4: Performance Metrics of TF-IDF Resampled

Linear SVC outperforms all models.

3.1.4 BERT

Model	Accuracy	Precision	Recall	F1-Score
Logistic Regression	0.855	0.853	0.855	0.853
Random Forest	0.783	0.811	0.783	0.769
KNN (1000)	0.742	0.763	0.742	0.714
XGBoost	0.86	0.86	0.86	0.854
Linear SVC	**0.862**	**0.86**	**0.862**	**0.859**

Table 5: Performance Metrics of BERT

Linear SVC outperforms all models.

3.1.5 DistilBERT

Model	Accuracy	Precision	Recall	F1-Score
DistilBERT	0.876	0.877	0.876	**0.876**

Table 6: Performance Metrics of Fine-Tunned DistilBERT

As seen, DistilBERT is the best of them all.

3.2 Comparison

Let's focus only on F1 Score. Overall DistilBERT gave the best performance. Focused only on our vectorization, another pretrained model can be found at the top in **all-mpnet-base-v2**. The worst was Word2Vec. Linear SVC performed the best most time, except with Word2Vec where XGBoost was better. KNN gave the worst performance.

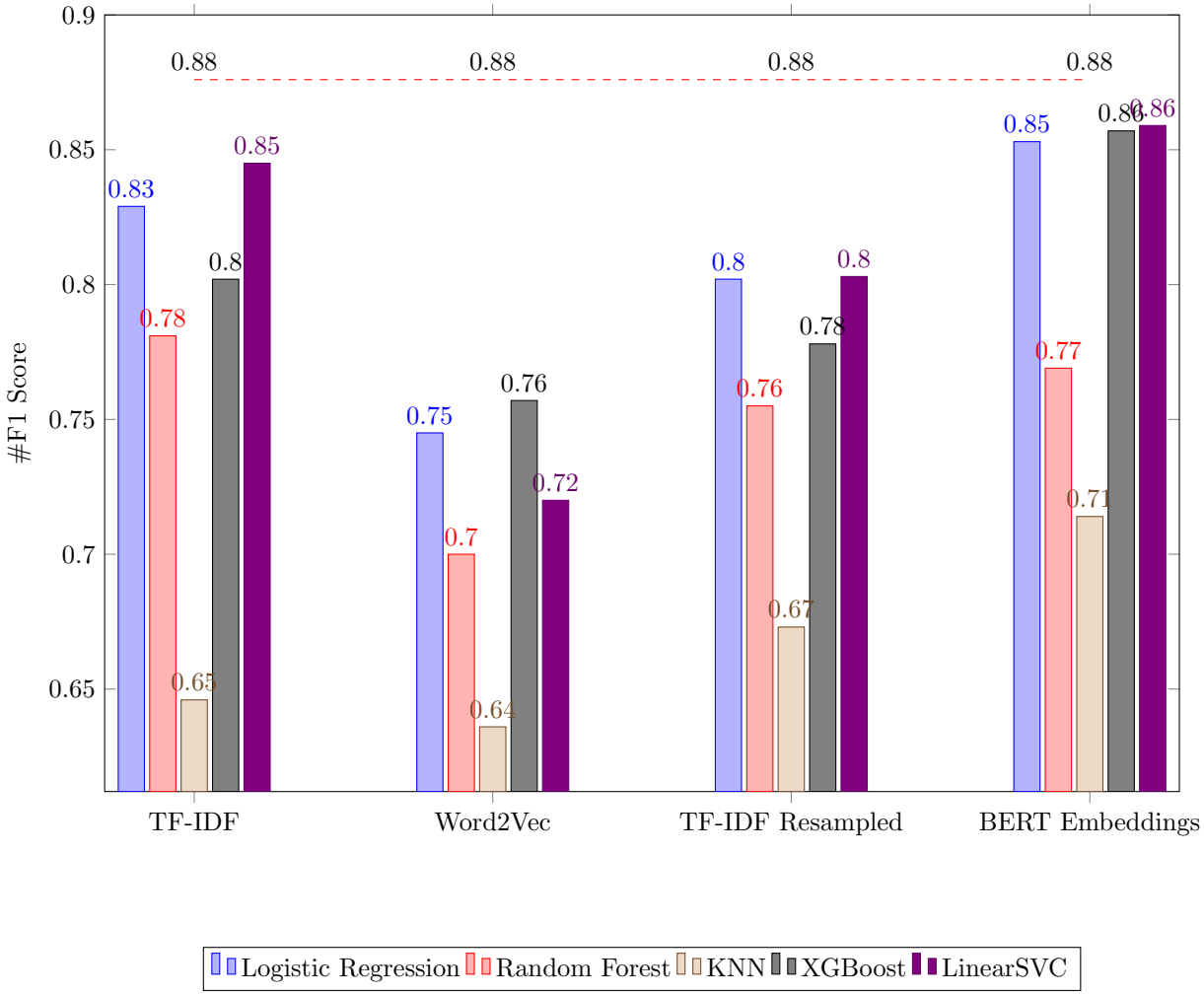

Figure 2: Model comparison, red dashed line represents DistilBERT

3.3 Best Models

3.3.1 Bert Embedding with LinearSVC

In the examination of the normalized confusion matrix, it becomes apparent that POLITICS is predicted quite frequently, leading to a considerable number of texts being wrongly classified as POLITICS. This occurrence is a direct consequence of the training dataset containing a significantly higher abundance of POLITICS compared to other categories.

Some interesting findings include the model's predictions of ENTERTAINMENT when the actual category is COMEDY, as well as its predictions of PARENTS and PARENTING etc.

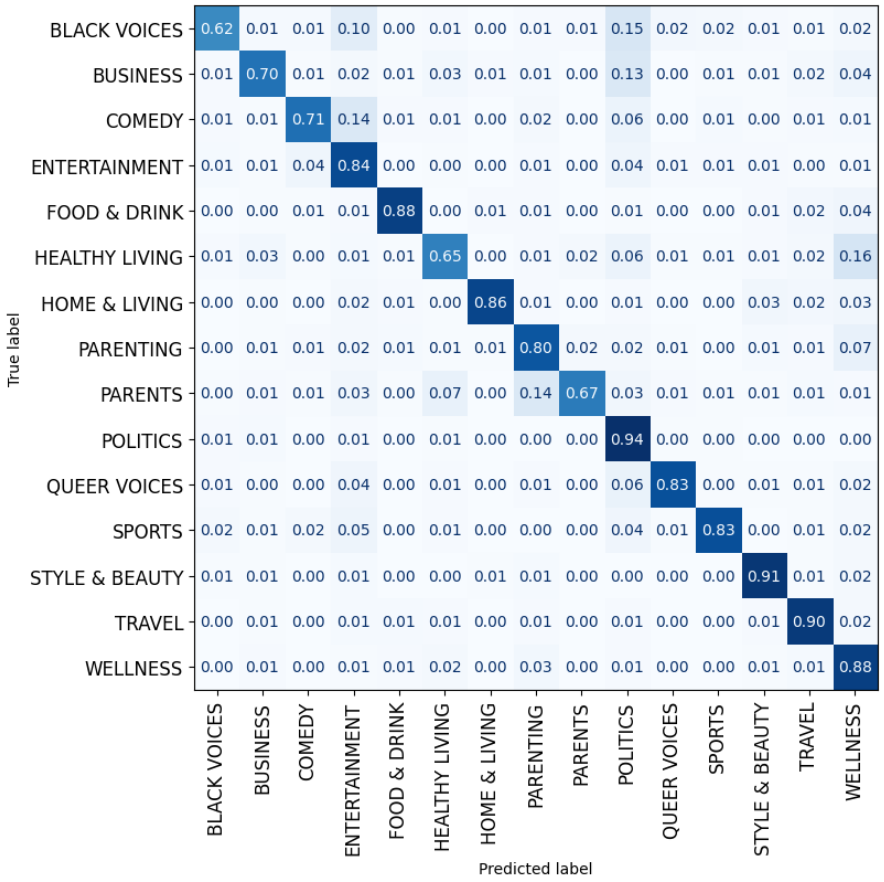

Figure 3: Bert Embedding with LinearSVC

3.3.2 DistilBERT

"The POLITICS problem" is not as apparent as in the last normalized confusion matrix, which may be the result of fine-tunning the model.

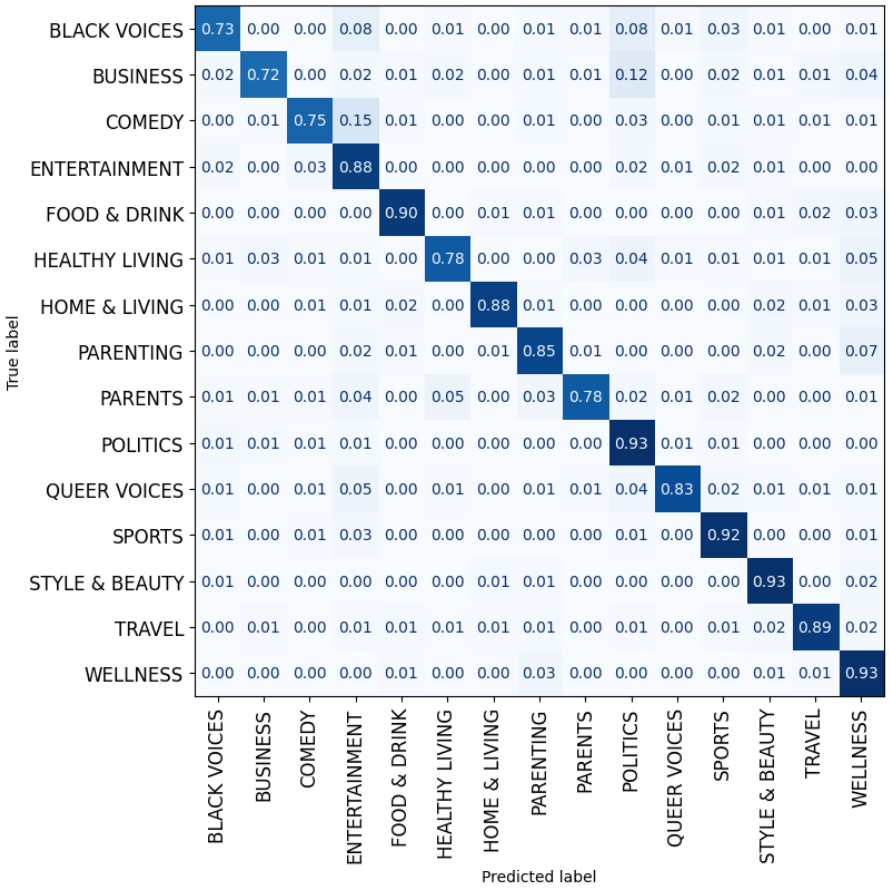

Figure 4: DistilBERT

4 Discussion

The pretrained models performed better, as expected, thanks to their ability to learn linguistic patterns and relationships from extensive text data. As anticipated, there was a notable challenge with the 'POLITICS problem,' because of the ratio of political articles to other.

However, a noteworthy deviation from expectations was observed in the subpar performance of Word2Vec. Additionally, it was intriguing to observe that TF-IDF performed worse with downsampling compared to without it, which is weird considering that vast majority of data are of political category.

Looking ahead, several potential improvements can be considered:

- Working with more computational power
- Experimenting with different kernels for SVC (**'poly'**, **'rbf'**, **'sigmoid'** or **'precomputed'**)
- Enhancing resampling techniques to better address the ratio of politics in the training dataset.
- Exploring alternative models, such as **multi-qa-mpnet-base-dot-v1,** known for its proficiency in Semantic Search.