Support Vector Machine from scratch with Python

Anna Granberg
Filip Hamrelius

Sentiment Analysis on Political Biased News

Introduction - Sentiment Analysis

Sentiment analysis involves examining text to decide whether the expressed emotional tone within the message is positive, negative, or neutral. The method can also be used to examine other sentiments, in our case; Political bias.

Sentiment analysis in the context of political biased news aims to assessing the tone of articles to determine the attitude and opinions presented in the text. Political bias means that the source expresses sentiments that align with a particular political stance, influencing the way they present and frame specific information.

Using sentiment analysis we can shed light on the otherwise hidden meaning and information in the news and to an extent classify entire news outlets depending on their political agenda.

This project aims to try out different methods for political bias, especially in the realm of support vector machines and feature extraction.

Ch. 1 Research Question
Aim & Background

The aim for this projects research questions lies within evaluating the implementation and understanding support vector machines. The particular area within political bias was not chosen to dictate the research questions.

1. Can a support vector machine be implemented in a sufficient way without predefined libraries?
2. If so, how do the implemented SVM compare to the predefined one provided by the sklearn library?
3. How does one choose the right document embedding method for feature extraction?

Ch. 2 Data
The Dataset

The dataset used is constructed of 6000 news articles with predefined political bias. The articles are labelled as either left, neutral or right bias. For the implementation we split the dataset into two, 75% training and 25% testing. The dataset was relatively well balanced and by shuffling before splitting we get a reasonable good balance. [1]

6000 news articles

Left (0), neutral (1) or right (2)

25% Testing & 75% Training

Ch. 3 Processing
Cleaning the dataset

Before utilizing the dataset, we preprocessed it by removing stop words, punctuations, URL's and HTML syntax. Since all news articles are written in Macedonian, the stop words needed to be imported as a distinct dataset.[1]

1. **Remove HTML syntax**
2. **Drop URL's**
3. **Remove punctuations & other symbols**
4. **Remove stop words**

Ch. 3 Processing
Learning document embeddings

After the cleaning process, the dataset needed to be vectorized and tokenized to map the document embeddings. This resulted in a dataset containing 1000+ vectors, each with a length of 300, accompanied by a complementary dataset with labels: 0, 1, and 2. "*Learning document embeddings*" refers to the process of representing documents as vectors in a continuous vector space. The idea is to capture the semantic meaning, relationships and context of the entire documents in a dense vector, making it suitable for various natural language processing tasks, and in this case classifying the data with among others the SVM. [1]

PV-DBoW (Distributed Bag of Words)

The model ignores the context words in a document and tries to predict the target words (or a set of words) based solely on the document's representation. It treats each document as a bag of words and learns a fixed-size vector representation for each document. [4]

PV-DM (Distributed Memory)

The model takes into account the context words as well as the document itself. It tries to predict the target words by considering both the document's representation and the context words. This allows the model to capture the overall meaning of the document in addition to the individual word meanings. [4]

During this project both methods were implemented in a attempt to evaluate which one is better suited for sentiment analysis and its correlation with the use of SVM.

Both models were imported with the Gensim library using their Doc2Vec class.

Ch. 3 Processing
Finished Dataset

Orignal dataset:

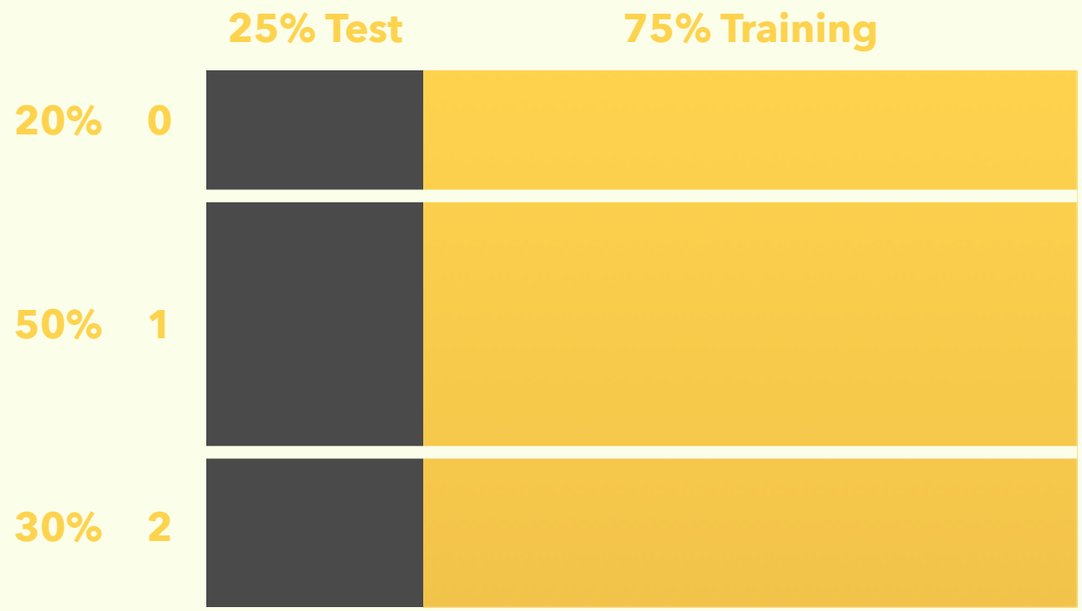
6000 articles with a respective label; left, neutral, right

Processed Training dataset:

4500 vectors with a size of 300 tokens each representing one article. A complementary set with the labels; 0,1,2

Processed Test Dataset:

1500 vectors with a size of 300 tokens each representing one article. With a key set containing the labels; 0, 1, 2



Ch. 4 Modeling
SVM Implementation without Libraries

SVM using a basic gradient descent approach with hinge loss and L2 regularization

The gradient descent optimizes the SVM iteratively, the method to adjusts the parameters to find the optimal values that result in the best classification.

The hinge loss is a function that penalized misclassifications and encourages correct classifications. This can be done since SVM is a supervised learning method, meaning we use the labels during the learning process. The goal is to maximize the margin between the hyperplane and the nearest data point of each class.

The L2 regularization adds a penalty based on the sum of all parameters squared and adds them to the loss function. This helps prevent overfitting by discouraging complex models with large parameter values. [3].

```
class SVM:

    def __init__(self, learning_rate=0.01, lambda_param=0.01, n_iters=10):

        self.lr = learning_rate  # Learning rate for gradient descent
        self.lambda_param = lambda_param  # Regularization parameter
        self.n_iters = n_iters  # Number of iterations for training
        self.w = None  # Weight vector
        self.b = None  # Bias term

    def fit(self, X, y):

        n_samples, n_features = X.shape  # Number of samples and features
        self.w = np.zeros((len(np.unique(y)), n_features))   Weight matrix
        self.b = np.zeros(len(np.unique(y)))  # Bias vector

        for class_label in np.unique(y):

            # Set class labels,-1 for samples not belonging to the current class
            binary_labels = np.where(y == class_label, 1, -1)

            for _ in range(self.n_iters):

                for idx, x_i in enumerate(X):

                    # Compute the condition based on the hinge loss
                    condition = binary_labels[idx] * (np.dot(x_i,
self.w[class_label].T) - self.b[class_label]) >= 1

                    if condition.all():
                # If the condition holds, update weights with regularization term
                        self.w[class_label] = self.w[class_label] - self.lr * (2
* self.lambda_param * self.w[class_label])

                    else:
                # Else, update weights and bias to account for misclassifications
                        self.w[class_label] = self.w[class_label] - self.lr * (2
* self.lambda_param * self.w[class_label] - np.outer(binary_labels[idx], x_i))
                        self.b[class_label] = self.b[class_label] - self.lr *
binary_labels[idx]


    def predict(self, X):
        return np.argmax(np.dot(X, self.w.T) - self.b, axis=1)
```

Ch. 4 Modeling
Sklearn - SVM, Random Forrest, Bayes

Generally

Doc2Vec embeddings capture the semantic relationships within a document. In contrast, Support Vector Machines (SVM) and other methods like Naive Bayes or Random Forests use these features to classify the given data.

Naive Bayes Classifier (sklearn.naive_bayes)

The Naive Bayes classifier is a probabilistic machine learning model based on Bayes' theorem. In scikit-learn, the `sklearn.naive_bayes` module provides various Naive Bayes implementations, including Gaussian Naive Bayes for continuous data and Multinomial Naive Bayes for discrete data. Despite its "naive" assumption of independence between features, Naive Bayes classifiers are efficient and perform well in many real-world applications, particularly when dealing with high-dimensional data and large feature spaces. [2]

Support Vector Machine (SVM) (sklearn.svm)

Scikit-learn's SVM implementation, found in the `sklearn.svm` module, is a supervised learning algorithm for classification and regression tasks. SVMs aim to find an optimal hyperplane that separates data points of different classes in a high-dimensional space. The algorithm is effective in handling non-linear relationships through the use of kernel functions. [2]

Random Forest (sklearn.ensemble.RandomForestClassifier)

The Random Forest algorithm, available in scikit-learn's `sklearn.ensemble` module, is a learning method that builds a multitude of decision trees during training and outputs the mode of the classes for classification tasks of the individual trees. Random Forests are robust, versatile, and less prone to overfitting compared to individual decision trees. By combining the predictions from multiple trees, they provide a more accurate and stable model. Random Forests are commonly used for tasks such as classification, regression, and feature importance analysis in various domains. [2]

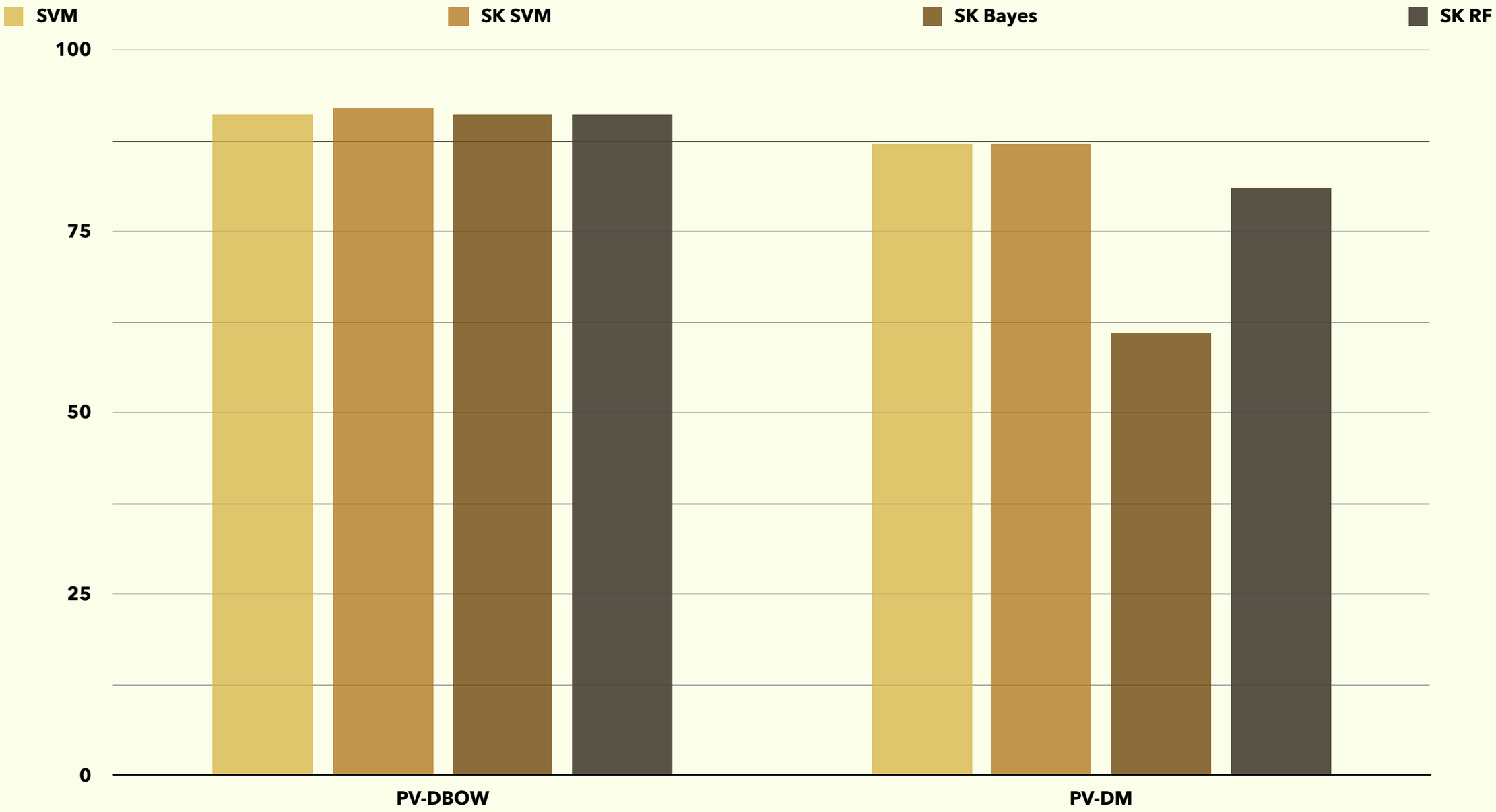
Ch. 4 Modeling
Sklearn - SVM, Random Forrest, Bayes

```
# ----- Naive Bayes Classifier from sklearn -----
bayes_0 = GaussianNB()
bayes_1 = GaussianNB()
bayes_0.fit(train_x_0,train_y_0)
bayes_1.fit(train_x_1,train_y_1)

print("Naive Bayes Classifier")
print(acc(test_y_0,bayes_0.predict(test_x_0)))
print(acc(test_y_1,bayes_1.predict(test_x_1)))

# -------- Random Forest from sklearn --------
forest_0 = RandomForestClassifier(n_estimators=100)
forest_1 = RandomForestClassifier(n_estimators=100)
forest_0.fit(train_x_0,train_y_0)
forest_1.fit(train_x_1,train_y_1)

print("Random Forest Classifier")
print(acc(test_y_0,forest_0.predict(test_x_0)))
print(acc(test_y_1,forest_1.predict(test_x_1)))

# -- Support Vector Machine from sklearn ----
svc_1 = SVC()
svc_1.fit(train_x_1,train_y_1)
svc_0 = SVC()
svc_0.fit(train_x_0,train_y_0)

print("Support Vector Machine")
print(acc(test_y_1,svc_1.predict(test_x_1)))
print(acc(test_y_0,svc_0.predict(test_x_0)))
```

```
from sklearn import utils
from sklearn.naive_bayes import
GaussianNB
from sklearn.ensemble import
RandomForestClassifier
from sklearn.svm import SVC
```

To use the models by sklearn you simply import the wanted model trough the sklearn library by specifying which one. Cleaning and vectorizing the data has to be done beforehand. In the case of a supervised model such as an SVM you do have to have a label set as well with the real and correct classifications.

Ch. 5 Evaluation
SVM vs Sklearns SVM vs other methods



Ch. 6 Summary
SVM vs SKlearns SVM

Regarding the performance, our own SVM and sklearns SVM preforms almost identical. Note that the other methods presented are included solely for the purpose of comparison and as a starting point. This is intended to provide a better understanding of the baseline performance of PV-DBOW and PV-DM.

By definition Sklearns implementation should be more advanced, but that may not matter in this case. The performance of linear SVMs can depend on the characteristics of the data. If the data has distinct classes that are separable by a hyperplane, both implementations may produce comparable results. In summary the results depend on the characteristics of the data, since both SVM's preform really well, the conclusion that the data is linearly separable can be drawn.

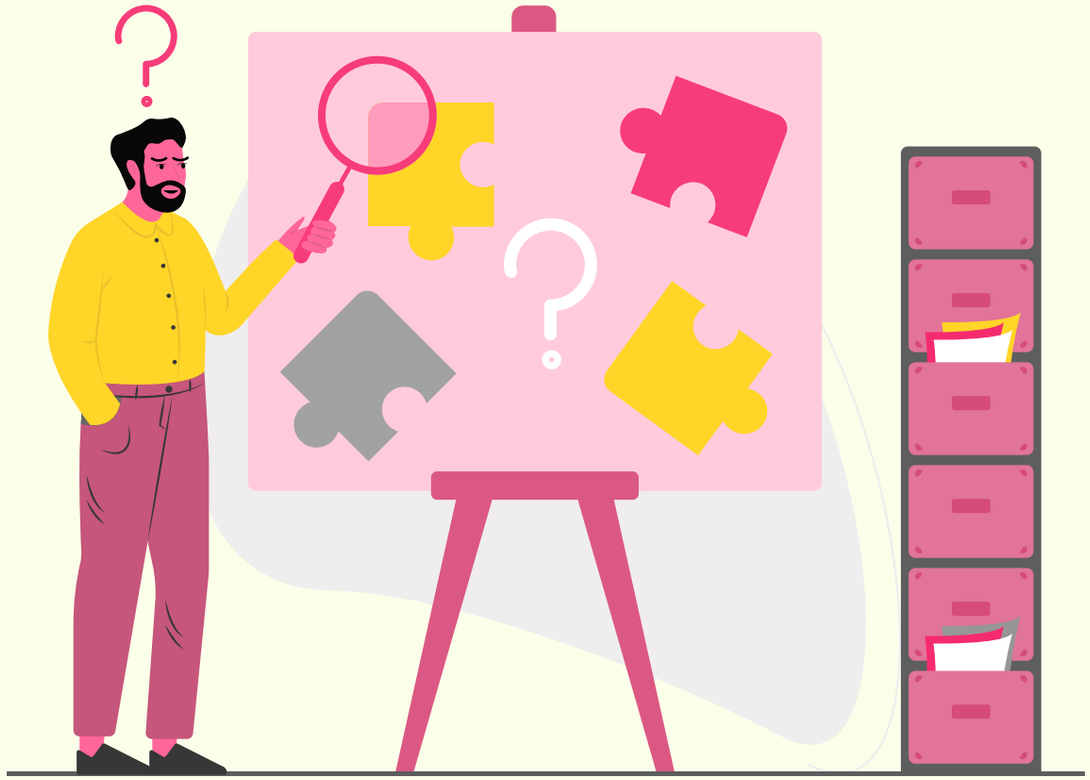
Even though the produced results are very close, it is important to note that sklearns implementation of the SVM includes other beneficial aspects. It is more optimized, robust, and flexible. For example, one can train the model independently on a number of features, while our own implementation has to be changed if the number of features changes. I.e if one wishes to perform binary classification instead of three labels with the implemented SVM, the model needs to be modified. It is also worth noting that our model does not perform any kind of cross-validation or grid search.

In summary it is possible implementing your own classification model, such as a basic SVM. Although it might not be worth it for simple use cases. Although it is worth mentioning that you do have more control over the model and how it operates which could prove beneficial for more complex use cases. But for this case the available libraries are sufficient.

Ch. 6 Summary
PV-DBOW vs PV-DM

Generally speaking, PV-DM should perform better for sentiment analysis since it not only considers relationships but also the context present. However, looking at the results, PV-DBOW achieves better overall performance for political bias sentiment analysis, in this case. This may depend on the news articles; if the political bias is rudimentary and doesn't contain any complex connections, PV-DBOW would perform better. That is, the bias is visible to the naked eye. With more complex and underlying opinions, PV-DM should, in theory, perform better. Thus in this case, it seems that we benefited from having context independence.

In summary, when choosing a document embedding method, it will always be beneficial choosing said method heuristically. Representing text can also be done in other manners, for example with a TF-IDF model. Which may prove beneficial for different use cases.



Ch. 6 Summary
Answer to research questions

1. Can a support vector machine be implemented in a sufficient way without predefined libraries?

Yes, a simple and sufficient SVM can be implemented without predefined libraries and tools.

2. If so, how do the implemented SVM compare to the predefined one provided by the sklearn library?

A simple SVM preforms quite similar to the predefined SVM provided by sklearn library for isolated use cases. If one is working with different datasets or multiple use cases you should use the predefined models since these are more robust, optimized and allows for a more efficient implementation. Although it is worth mentioning, with a SVM from scratch you have control over every single parameter and how the SVM handels the features and data. This could prove to be useful for specific use cases, it might also be easier to identify if anything goes wrong since you have a peek directly into the source code.

3. How does one choose the right document embedding method for feature extraction?

In this case, the PV-DBOW method yielded better results, although PV-DM, at first glance, should perform better in sentiment analysis. This discrepancy is likely due to the characteristics of the dataset, particularly how the political bias is presented. The conclusion drawn is that the choice of method needs to be made heuristically and is highly dependent on the dataset.

Ch. 7 Sources
Bibliography

[1] Najkov, Danilo, Detecting political bias in online articles using NLP and classification models, Medium, Jul 19 2022, https://medium.com/@danilo.najkov/detecting-political-bias-in-online-articles-using-nlp-and-classification-models-c1a40ec3989b, Collected: 2023-11-12

[2] Umair, Muhammed, Implementing Naive Bayes, Random Forest, and SVM for Classification: A Tutorial with Code and Dataset, Medium, Mar 17 2023
https://ai.plainenglish.io/implementing-naive-bayes-random-forest-and-svm-for-classification-a-tutorial-with-code-and-47f76d7361dc
Collected: 2023–11-20

[3] Luo, Shuyu, Loss Function: Support Vector Machine, Medium: Towards Data Science, Oct 15, 2018,
https://towardsdatascience.com/optimization-loss-function-under-the-hood-part-iii-5dff33fa015d
Collected: 2023-11-25

[4] Tsang, Sik-Ho, Review: Distributed Representations of Sentences and Documents (Doc2Vec), Medium, Nov 12 2021,
https://sh-tsang.medium.com/review-distributed-representations-of-sentences-and-documents-doc2vec-86ef911d4515
Collected: 2023-12-05