# Lab 1 – Sentence Classification: BoW vs CNN

November 19, 2025

## 1 Introduction

### 1.1 Data description

The dataset first consists of two sets of movie text reviews of the same size, one containing positive reviews and one containing negative reviews. It is first cleaned: punctuation is removed and the text is standardized by the provided helper functions. The dataset is then split into a train set (70% of all data) and a test set with the remaining data. Both test and train sets keep the same proportion of 50/50 positive and negative reviews. The labels are then constructed as 1 for positive reviews and 0 for negative reviews.

### 1.2 Data representation

In order to make the model able to understand words, we use an embedding table mapping indices of words to embeddings. For that, we use the scikit-learn embedding table of dimension 50 for computing speed and given the small size of sentences. The unique index is constructed from the vocabulary of all the sentences in the train set, of size 6556, including indices for end of sentence, start of sentence, and padding tokens, which will be used later.

As a result, some words in the test set might not appear in the training set and therefore have no index. We decide to skip them, following the guidelines, but we could also encode them with a special index.

## 2 Model 1 – Bag-of-Words (BoW) Classifier

### 2.1 Model architecture

The BoW model is very simple once the words are embedded: it operates on all the embeddings in the sentence, either summing them or averaging them, to obtain a fixed-size representation of the sentence.

Then an MLP is implemented to classify the sentence into positive or negative sentiment. This MLP can be of any size we want; we first tested a simple linear perceptron, then a multilayer perceptron with hidden layer size 64. The loss function is a standard binary cross-entropy loss, and the optimizer is Adam with a learning rate of 0.001, number of epochs is 25, as with more we observe some overfitting.

As not to overfit we fix the embedding dimension for the BoW model to $dim = 50$.

## 2.2 Results

The bagging operation of averaging is clearly better than summing after different tests, which seems coherent as it is basically the same operation but normalized. We only consider this operation from this point onward as we do not want the size of the sentence to matter in classification.

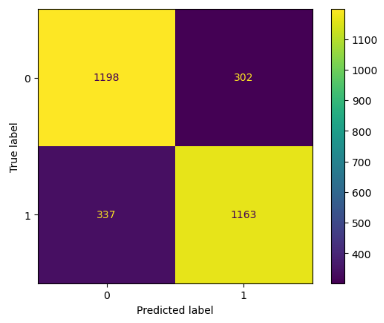

Figure 1: Confusion Matrix: 1 perceptron

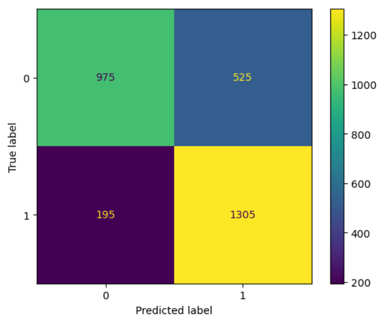

Figure 2: Confusion Matrix: hidden 64 layer

With a single linear perceptron, performance is already quite good, accuracy is 76%. The confusion matrix shows some misclassifications.

Adding a hidden layer makes the model only a little better (accuracy = 79%) probably because of overfitting, seeing that the previous model is quite good; there is a bias towards positive (high recall).

With many different settings we can see that this accuracy is the maximum possible; this is because BoW discards much information by bagging.

## 2.3 Discussion

The Bag-of-Words model ignores word order and aggregates all word embeddings in the sentence, so negation or sarcasm can induce a very different meaning that is not correctly captured in the representation. This may explain why the BoW model struggles with positive/negative classification. It might perform better on other classification tasks where word order is less important. The need for local information motivates the test of a CNN to determine more local structures in the sentences.

## 3 Model 2 – Convolutional Neural Network (CNN) Classifier

### 3.1 Model architecture

The CNN model is more complex than the BoW model. It still uses the same word embeddings, but it requires padding before and after each sentence so that convolution filters can operate properly near the sentence boundaries. This padding could be applied directly using scikit-learn functions, but I chose to implement it manually by adding padding tokens around each sentence.

Once the sentences are embedded, the CNN applies convolution filters across the entire sequence using a fixed window size to extract local features. All parameters are learned by the model, allowing it to recognize short local patterns. For this reason, the window size must not be too small (otherwise it only captures very short patterns) and not too large (otherwise it becomes too global and loses the benefit of local structure). This is a key hyper-parameter that will be studied later.

The features obtained from the convolution filters are passed through a non-linear activation function (ReLU), then pooled using a max-pooling operation. The resulting vector is fed into an MLP for binary classification.

As before, the loss is a binary cross-entropy loss, optimized with Adam using a learning rate of 0.001.

### 3.2 Hyper-parameter experimentation

In this model, there are a lot of hyperparameters to optimize by testing grids of parameters. The learning rate is only a matter of precision and speed of convergence, not really in the fundamental parameters of the models, so I just picked the best one.

#### 3.2.1 Filter size

The filter size is a major parameter, it corresponds to the size of local structure you can detect. Too small and it's just a bag of words and too much and it's just a fully connected layer. Here we can see that as the dataset is mainly composed of small sentences, groups of words that are 2-3-4 words long are sufficient to carry most of the information as in bigram and trigram models. A window too big hurts performance because it misses local structure.

#### 3.2.2 Number of filters

The number of filters corresponds to how many different convolutional feature detectors are applied to the sentence. Each filter learns to detect a

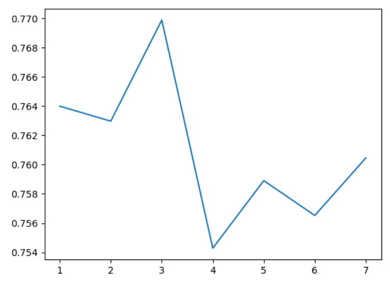

Figure 3: F1-score vs. Filter size

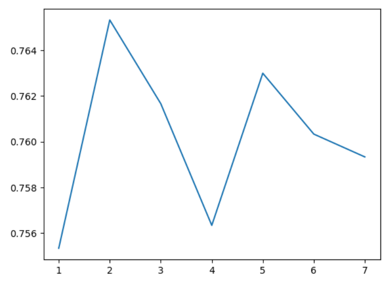

Figure 4: Accuracy vs. Filter size

particular local pattern, so increasing the number of filters increases the number of distinct features the model can extract. There is also a trade-off here between under- and overfitting.

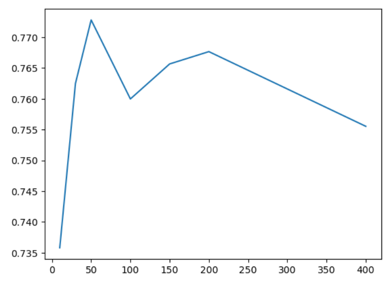

Figure 5: F1-score vs. number of filters

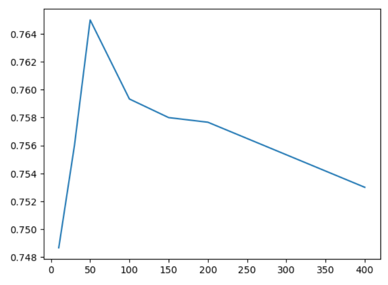

Figure 6: Accuracy vs. number of filters

### 3.2.3 Best results

To compare to the previous model, we decide to take the best parameters: number of filters = 50, window size = 3 and compare the performance of the CNN in the same conditions as the previous BoW model. Those low parameters are coherent because the dataset is simple so models are prone to overfit.

The CNN theoretically outperforms the BoW model by capturing local word-order and n-gram patterns but here, as the dataset is relatively small, learning patterns is difficult.

## 4 Conclusion

In summary, our results show a clear trade-off between simplicity and expressiveness. The BoW model is fast, easy to train, and already sufficient when the dataset is small and word order is not crucial, but it struggles with

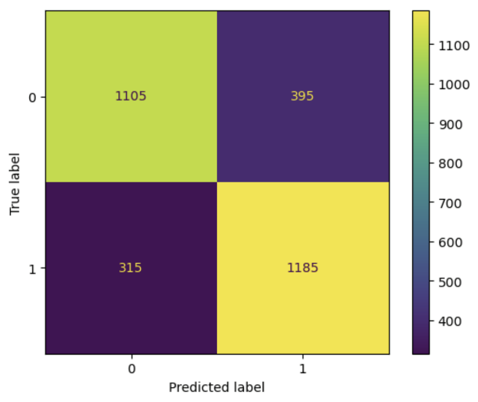

Figure 7: Confusion Matrix for Best CNN

phenomena like negation or more subtle sentiment. The CNN brings gains by exploiting local word-order patterns, suggesting that the dataset does contain informative n-gram structures, though its advantage is limited by data size and risk of overfitting. Future work could explore larger datasets, pretrained word embeddings, but also a multilayer CNN, with $p$ different size windows calculating features at different scales in parallel in the sentence with an MLP aggregating the $p$ pooled features into a prediction.