IN5550 - Spring 2023

Mandatory assignment 1
bag of word document classification

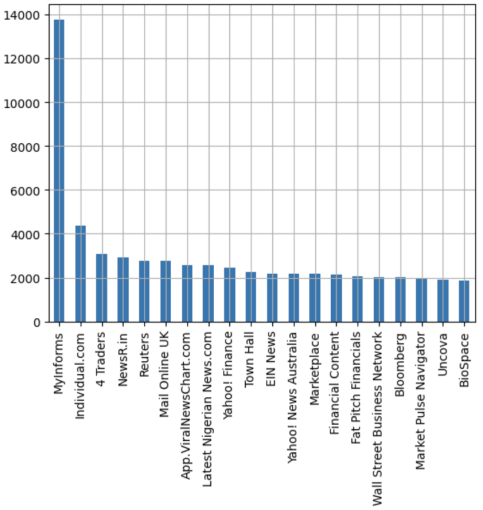
Cornelius Bencsik, Amir Basic & Torstein Forseth
corneb, amirbas & torfor

Git repository: https://github.uio.no/torfor/IN5550-Oblig1
Fox directory: ../../../projects01/ec30/CocoAmirTorfor

1.1 - data processing

The data given is the "Signal Media One-Million News Articles Dataset", which consists of texts from news sites and blogs from September 2015. The dataset contains a total of 75 000 articles and around 16 million word tokens from 20 different news sites. Stop words and punctuation signs are removed, and the text is lemmatized and POS-tagged. Additionally, the documents are shuffled, which implies no order in the dataset.

The problem is to predict the source of a text based on the occurring words using a feed-forward neural network. To accomplish this, we want to represent the texts as bags-of-words representing the frequency of each word from a predetermined vocabulary in the text. Individual texts are then represented as vectors of the same size as the vocabulary, with values equal to the frequency of each vocabulary word in that document.



Our selection regarding data splitting is to perform a stratified test train split, where we stratify on the labels (articles), which means the ratio of each label is equal in the train and test set. The ratio is 80% training data and 20% test data. A quick overview of the data, more specifically the labels, show that the total quantity of each label ranges from 2326 to 17210. Articles from MyInforms are overrepresented with 17210 articles, followed by articles from Individual.com with 5416 articles. The 18 other websites are represented with 2326 to 4000 articles each, with no clear outliers.

The most important data processing decision regarding this problem is to indicate a vocabulary of words that will be the baseline of text representation for the classification. Throughout the assignment, we are creating a total of three different vocabularies for the BoW vectorization. The process consists of mainly two steps; pure text preprocessing and deciding rules for the vocabulary. The pure text processing is done through regex preprocessing which is passed as a preprocessing step to the "TfidfVectorizer" function from sklearn. The vocabulary rules are tuned in the "TfidfVectorizer". The development of all vocabularies uses only the training data (documents/text) and are then used to create a new bag of words.

we use three different preprocessing methods across our three separate vocabularies:

1. Basic text preprocessing:
    a. remove all "_NUM" tagged words
    b. remove all pure digit words
    c. convert all words to lower letters
2. Setting vocabulary max size/features:
    a. We use either 10 000 features or 1 000 features in our vocabulary. The features are chosen on their frequency, where the 10 000 or 1 000 most frequent words across all documents are chosen to be in the BoW vocabulary. We also allow both single words and bi-grams in all our BoW representations

3. Ignoring “irrelevant” words:
    a. Remove all words defined in a stopword list that consists of words previously found in the vocabulary that are manually eliminated because they are regarded as either:
        i. Noise: “k_noun”, “rt_propn”, “m2”
        ii. Frequent words/stopwords: “be”, “in”, “to”
        iii. Irrelevant words: “10th”, Names, States/cities
        iv. misspelled: “issu”, “describ”, “promot”

        The stopword list consists of 312 of these words, which are replaced by the next most frequent words across all documents in the training data.

After settling a vocabulary, we process both the training and test text data through the vocabulary, which results in each text being represented as a tensor denoting the count of occurrence of each word in our vocabulary in the text, a bag of words representation. Our gold label (source/website) data goes through a simple label encoder that gives each unique label a numeric value, which results in the label data being represented as a tensor of numerical values 0 - 19, where each numerical value represents one of the 20 websites.

1.2 - Training classifier

Our neural network is a feed-forward neural network that uses linear input and output layers and the gelu activation function. The number of hidden layers and neurons in each hidden layer are parameters that can be tuned, but in general, we used only 1 hidden layer with 100 neurons.

The parameters we used to train our models:

- 1 hidden layer
- 100 neurons
- batch size 64
- learning rate 0.001
- trained over 4 runs

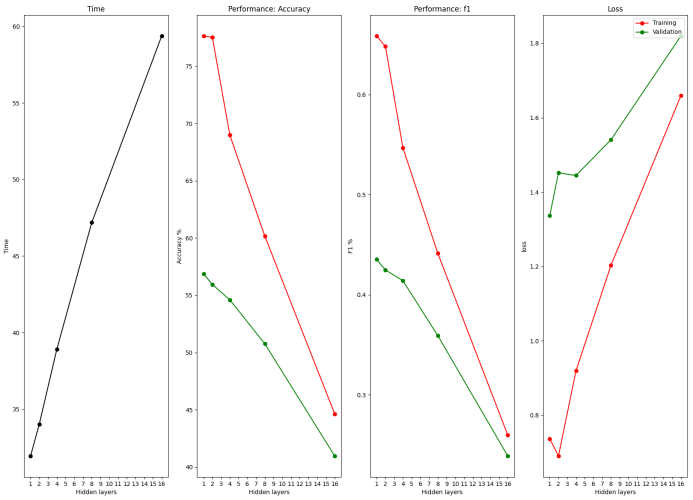
1.3 / 1.4 - Feature tuning - 3 experiments with evaluation, timing and performance

Experiment 1 (best model):

- creating vocabulary:
    - 1. Basic text preprocessing
    - 2. Setting vocabulary max size = 10 000

- Evaluation/performance with 1 hidden layer:
    - accuracy: 57.03 %
    - recall: 45.72 %
    - precision: 46.07 %
    - f1-macro: 43.46 %

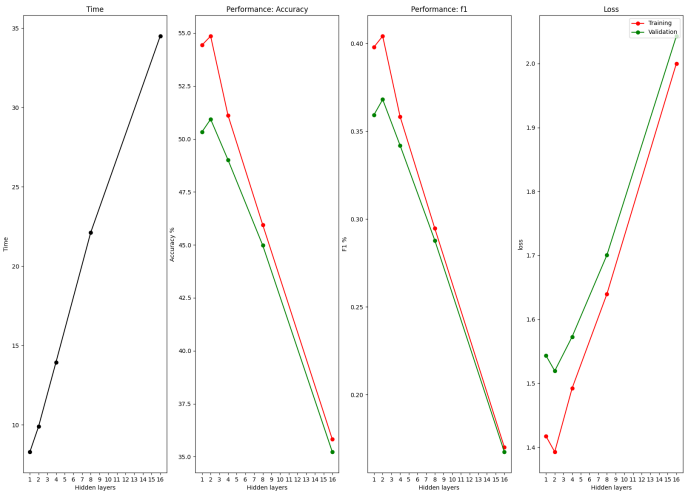
Experiment 1 performed very well, and is the “baseline” for the rest of the experiments. Experiment 1 has fairly basic text preprocessing and a decent vocabulary size that is not too large and not too small.



Experiment 1 (best model): Training time, accuracy, f1-macro & final loss on training and validation data with 1, 2, 4, 8 & 16 hidden layers

Experiment 2 (small vocabulary):
creating vocabulary:

- 1. Basic text preprocessing
    - 2. Setting vocabulary max size = 1 000
- Evaluation/performance:
    - accuracy: 50.46%
    - recall: 38.04%
    - precision: 38.7%
    - f1-macro: 35.82%

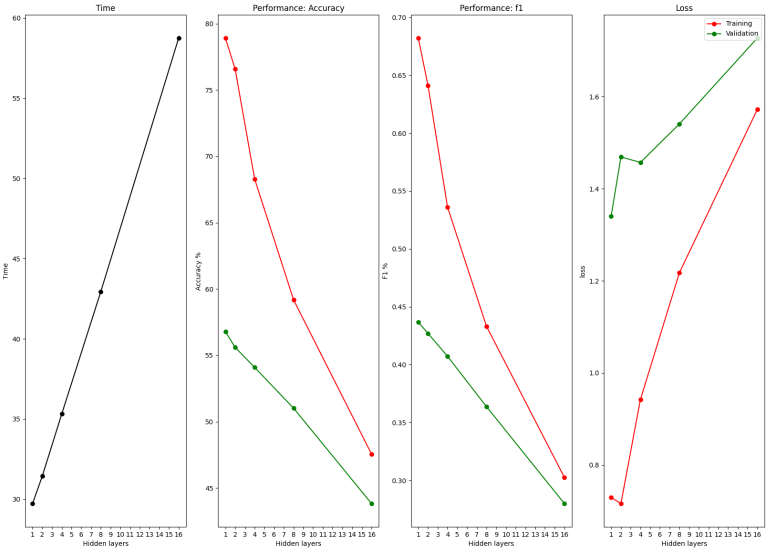


Experiment 2 (small vocabulary): Training time, accuracy, f1-macro & final loss on training and validation data with 1, 2, 4, 8 & 16 hidden layers

Experiment 2 performed, as expected, worse than Experiment 1. The only difference between the two is that the vocabulary size is scaled from 10 000 features to 1 000 features. It is reasonable to believe that this change causes the network to underfit and maybe can't separate some websites from each other, or don't predict some websites at all due to lack of information from these websites. This shows that the vocabulary size hyperparameter is important for the performance and can create underfit if the vocabulary is too small, and assumingly and similarly an overfit if the vocabulary is too large.

Experiment 3 (stop words):

- creating vocabulary:
    - 1. Basic text preprocessing
    - 2. Ignoring "irrelevant" words
    - 3. Setting vocabulary max size = 10 000
- Evaluation/performance:
    - accuracy: 56.67%
    - recall: 45.33%
    - precision: 45.77%
    - f1-macro: 43.16%



Experiment 3 (ignoring stop words): Training time, accuracy, f1-macro & final loss on training and validation data with 1, 2, 4, 8 & 16 hidden layers

Experiment 3 performs almost as well as experiment 1, which is expected, but surprisingly it doesn't perform better than experiment 1. The only difference between the two experiments is that in experiment 3, we remove/ignore 312 words that were included in the 10 000 features in experiment 1, that we regarded as noise, stopwords, irrelevant, etc. We removed these words with the impression that it would make a significant difference in performance, which it obviously didn't. One explanation for this small drop in performance is that these "noisy" or "irrelevant" words, in fact, are the words that help the model separate certain websites from one another, which means we have removed some relevant words.5

As mentioned earlier, we use the same hyperparameters in the neural network when training all the different experiments. The effect of tuning hyperparameters is an increase/decrease towards overfitting/underfitting. For the values we have chosen, our opinion regarding different hyperparameter values is the following:

- Hidden layers: 1
    - Increasing the number of hidden layers can lead to overfitting. The model can start to see relationships that are not there and over/under-value some relationships in the data

- Neurons: 100
    - Increasing the number of neurons in each layer can lead to overfitting. Too many neurons can over-complicate the model, especially if the model also has a lot of hidden layers with large sizes.
    - Decreasing the number of neurons in each layer can lead to underfitting. Having too few neurons can lead to a too general and sparse model.
- Learning rate: 0.001 & epochs: 4
    - increasing/decreasing the learning rate combined with the number of epochs we train the model determines if we overfit or underfit the model.
    - Having a low learning rate can lead to an underfit model that doesn't reach its potential/maximum. Having a too large learning rate can lead to an overfit model
    - Training over too many epochs can lead to overfitting (especially with a high learning rate) and training over too few epochs can lead to underfitting (especially with a small learning rate)
    - In terms of these hyperparameters, the strategy is to find a balance between the two in order to create a “perfect compromise” so that the model learns the relationships well enough to predict good, but also generalize and not overfit
- Vocabulary size: 10 000
    - As mentioned, it is important to have enough features to train the model on, but not too many. Having a huge set of features can lead to overfitting, since irrelevant features can become overly important. A too small vocabulary, as we saw in experiment 2, can lead to underfitting by not having enough features, relationship, or information to create a good performing and generalized model
- Batch size 64:
    - Batch size is a balance between speed, generalization and performance. A large batch size will lead to a faster training, but a too large batch size can lead to poor generalization since, and the model will not train until it has seen a lot of the data. However, with a smaller batch size the model will train and fit to smaller portions of the data, leading to a good and general solution

1.5 - Evaluation - best model trained and evaluated three separate times

validation score	scores	mean	sd
Accuracy	[56.93, 56.72, 56.82]	56.823%	0.086
Recall	[0.45, 0.45, 0.45]	0.45	0.0
Precision	[0.46, 0.46, 0.46]	0.46	0.0
F1-macro	[0.43, 0.43, 0.43]	0.43	0.0

The results are very consistent over the three separate training sessions. The results on the validation data are worse than on the training data which had over 72% accuracy. This is expected since the model is in fact trained on the training data, hence learning the relationships in the training data and correcting/improving itself with respect to the loss of the training data. When the model sees new data (the validation data), it uses the relationships found in the training data to predict the new data, which at best should predict the same as on the training data, but generally predict a little worse.
An important aspect of this, especially with our model, is that we “only” have a final accuracy on the training data of around 72%, which is fairly low. This is a result of the hyper parameters chosen, especially epochs and learning rate. By training the model over more epochs, we could have pushed the training accuracy to the 90’s in accuracy, but that would lead to overfitting, and a less general model that would predict new data worse than it does now. Therefore, we decided to create a more general model by training over “only” 4 epochs.