Practical Machine Learning Documentation

Mihalcea Dragos Stefan, 407

The assigned task was to train a classifier on a data set containing pairs of Romanian sentences split in the following way: 58k training, 3k validation and 3k test.

For this I have chosen to implement the following two machine learning methods: LinearSVC and Neural Networks.

Before getting into the models themselves and their hyperparameter tuning, in the next paragraphs I am going to present what feature engineering I did on the dataset, as I used the same techniques on all models.

Feature Engineering

For the extraction of features, I used 2 different methods: Term Frequency - Inverse Document Frequency (TF-IDF) and Bag of Words (BoW).

TF-IDF is a statistical metric that assesses the significance of a word within a document relative to a set of documents, hence I considered it to be useful for feature extraction in training a model that understands the correlation between 2 sentences. By having words that have a similar significance in common, the pair of sentences might be correlated.

Before extracting the features, I concatenated each pair of sentences in order to get a better understanding of both sentences together, taking in consideration their order. For my implementation I used the ***TfidfVectorizer*** from ***sklearn*** on each pair, with different parameters that will later be revealed in the model results table. After doing some hyperparameter tuning, I got the best results using: *ngram_range = (1, 2)* (i.e. using unigrams and bigrams in vectorization process), *strip_accents = 'unicode'* (for removing accents) and *max_features = 10000* (used for training the neuronal networks).

Bag of words is a technique that represents a document as an unordered set of its words, disregarding grammar and word order but keeping track of word frequency.

For this case I used the ***CountVectorizer*** from ***sklearn*** on each pair with different parameters the details will be disclosed and clarified in the table presenting the model results. This hyperparameter tuning resulted in the same way as TF-IDF: *ngram_range = (1, 2)* (i.e. using unigrams and bigrams in vectorization process), *strip_accents = 'unicode'* (for removing accents) and *max_features = 10000* (used for training the neuronal networks).

Machine learning models

This section will be divided in 2 pillars for each model I chose to implement: LinearSVC and Neural Networks. For each one I will present the model itself, my implementation and hyperparameter tuning, and a table with results and metrics.

LinearSVC

LinearSVC is a model used for binary and multiclass classification problems.It is a variant of the Support Vector Machine (SVM) algorithm that is designed to use a linear kernel function when trying to find the hyperplane that best separates different classes in the feature space. While it’s similar to the ***SVC*** model with parameter *kernel=’linear’*, this variant offers greater flexibility in selecting penalties and loss functions, and is expected to exhibit improved scalability with larger sample sizes and, also, in cases where the sample set size is smaller than the number of dimensions.

The parameters that i have used during training are:

- **C** - the regularization parameter. A higher value for C leads to a lower regularization value and a larger margin, making it more tolerant with misclassifications, while a smaller value leads to a higher regularization value and a tighter margin, making it stricter in regards to misclassified points. The default value for this parameter is 1.
- **max_iter** - the maximum number of iterations. I found this parameter useful when dealing with larger and higher dimensionality datasets that need more iterations to converge. The default value is 1000.

- **loss** - the loss function. It has 2 possible values: ‘hinge’ and ‘squared_hinge’ with later one being the default. While Squared hinge is simply the square of hinge loss function, it could be used when you want to punish the large errors as it quadratically increases with error. In the end the purpose is to minimize the loss during training.

In my experiments i tried the following:

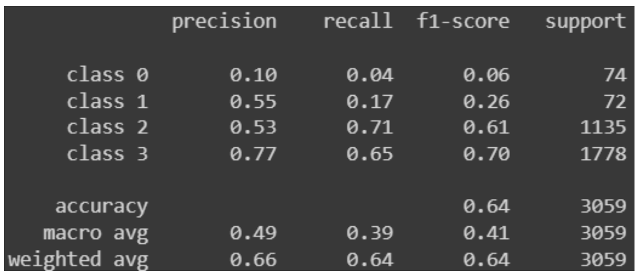
- C: 0.1, 1, 10
- loss: ‘hinge’, ‘squared_hinge’
- max_iter: 10000 (to make sure it converges)

In addition I’m gonna introduce one more parameter called *isSum*. This refers to the way in which the sentences were vectorized. For this model in particular, aside from the normal vectorization of the concatenated pair, I vectorized each sentence and then summed the 2 vectors from each pair. This makes the order of the sentences irrelevant.

Feature	C	loss	isSum	Recall	Accuracy	Precision	F1
TF-IDF	0.1	hinge	false	34.85	65.45	32.88	33.31
TF-IDF	1	hinge	false	38.04	66.72	54.04	38.32
TF-IDF	10	hinge	false	38.53	65.64	46.92	39.4
TF-IDF	0.1	squared	false	36.32	67.05	58.95	34.89
TF-IDF	1	squared	false	38.08	67.05	48.64	38.33
TF-IDF	10	squared	false	38.55	66.23	47.73	39.32
BoW	0.1	hinge	false	38.84	65.74	48.55	39.99
BoW	1	hinge	false	38.45	65.64	46.41	39.33
BoW	10	hinge	false	38.84	65.71	47.69	39.9
BoW	0.1	squared	false	38.6	65.81	49.12	39.63
BoW	1	squared	false	38.41	65.54	46.88	39.33
BoW	10	squared	false	38.84	65.71	47.69	39.9
TF-IDF	10	hinge	true	38.59	64.86	46.91	39.81
Bow	0.1	hinge	true	39.04	64.37	48.54	40.57

The best performance (based on F1 score) of the LinearSVC model was obtained using C=1 and loss='hinge' with a Bag of Words feature extraction on each sentence and then summing the vectors as seen in the table above. That being said, the difference in performance between the top models is <1% suggesting they perform in a very similar way.

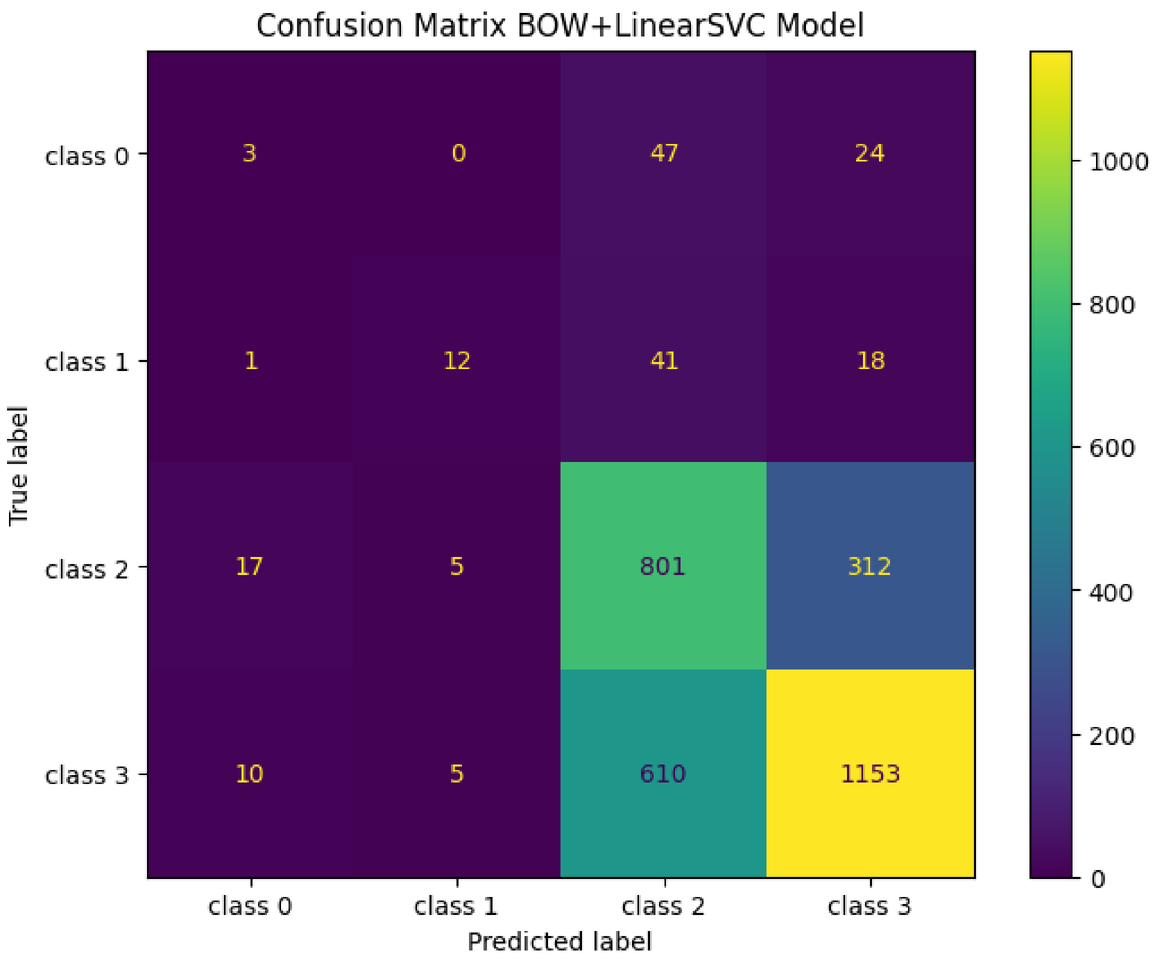
An observation we can make from the table is that although the accuracy is around 65% all the other metrics are way lower suggesting that the model is in fact not really learning well. This is further backed up by the imbalanced training dataset, the metrics on each class and the confusion matrix down below which clearly reveals that the model did not learn to classify classes 0 and 1 at all and is still confusing classes 2 and 3 one with the other.



```
              precision    recall  f1-score   support

     class 0       0.10      0.04      0.06        74
     class 1       0.55      0.17      0.26        72
     class 2       0.53      0.71      0.61      1135
     class 3       0.77      0.65      0.70      1778

    accuracy                           0.64      3059
   macro avg       0.49      0.39      0.41      3059
weighted avg       0.66      0.64      0.64      3059
```



Neural Networks

An **artificial neural network** is a model employed for tasks involving supervised learning, extending beyond regression or classification challenges. These networks are structured with layers, accommodating a wide range of applications.

In the architectures of my neuronal networks I used 2 types of layers: **Dense** and **Dropout**.

- The Dense layer calculates the result by applying the ***activation function*** element-wise to the sum of the ***dot product*** (of the ***input*** and ***kernel)*** and ***the bias***
- The Dropout layer randomly sets input units to 0 with a frequency of a given ***rate*** parameter at each step during training time. This is used in order to prevent ***overfitting***.

The activation functions I used during training are **ReLu** and **Softmax**.

- ***ReLu*** is a commonly used activation function that returns 0 if it receives any negative input, but for any positive value x it returns that value back
- ***Softmax*** is an activation function that takes as input a vector and returns a normalized vector of probabilities.

I used ***Adam*** optimizer with 2 learning rates: 0.00001 and 0.0001.

I also used ***EarlyStopping*** with a ***patience*** of: 2 and 4; in order to try and prevent overfitting.

My neural network architectures are:

- Model 1: Dense(256, activation = "relu")

  Dropout(0.4)

  Dense(128, activation = "relu")

  Dropout(0.2)

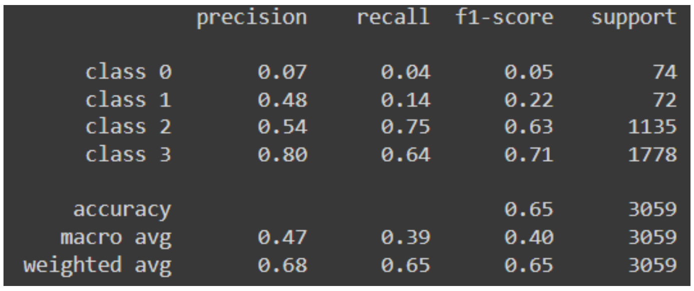
  Dense(4, activation = "softmax")
- Model 2: Dense(128, activation = "relu")

  Dropout(0.2)

  Dense(4, activation = "softmax")

Feature	Learning rate	Model	Epochs	Early Stopping Patience	Accuracy	Recall	Precision	F1
TF-IDF	0.0001	1	20	2	65.41	34.99	33.05	33.35
TF-IDF	0.0001	1	20	4	65.02	36.69	45.87	36.31
TF-IDF	0.00001	1	20	2	66.2	35.4	33.41	33.74
TF-IDF	0.00001	1	20	4	66.66	35.6	33.59	33.96
TF-IDF	0.0001	2	20	2	66.39	35.39	33.39	33.82
TF-IDF	0.0001	2	20	4	65.58	35.68	49.76	34.75
TF-IDF	0.00001	2	20	2	66.13	35.37	33.39	33.71
TF-IDF	0.00001	2	20	4	66.1	35.41	33.45	33.71
BoW	0.0001	1	20	2	65.09	37.18	45.7	37.33
BoW	0.0001	1	20	4	64.86	39.37	50.94	40.91
BoW	0.00001	1	20	2	66.0	35.4	33.44	33.67
BoW	0.00001	1	20	4	66.46	35.52	33.53	33.87
BoW	0.0001	2	20	2	65.94	37.13	52.72	37.33
BoW	0.0001	2	20	4	65.97	39.34	52.46	40.78
BoW	0.00001	2	20	2	66.49	36.57	52.31	35.86
BoW	0.00001	2	20	4	66.23	36.69	53.41	36.34

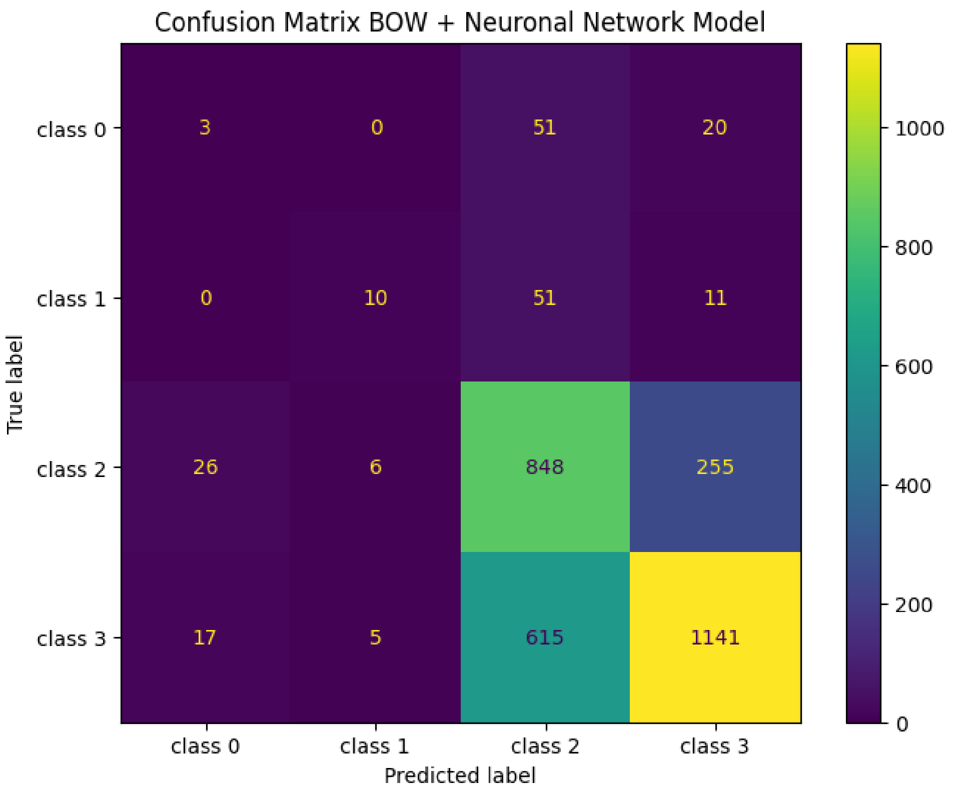
The TF-IDF and BoW used for feature engineering had the ***max_features*** parameter set to **10000** in order to reduce dimensionality and be able to train the neural networks.
The best performance I got was on Model 1 with Bag of Words features, learn_rate =0.0001, 20 epochs and patience=4.
The results in the above are similar to the LinearSVC result and point out the same problems of the model: accuracy is higher than other metrics because of the imbalance of samples from different classes in the dataset as shown in the pictures below.



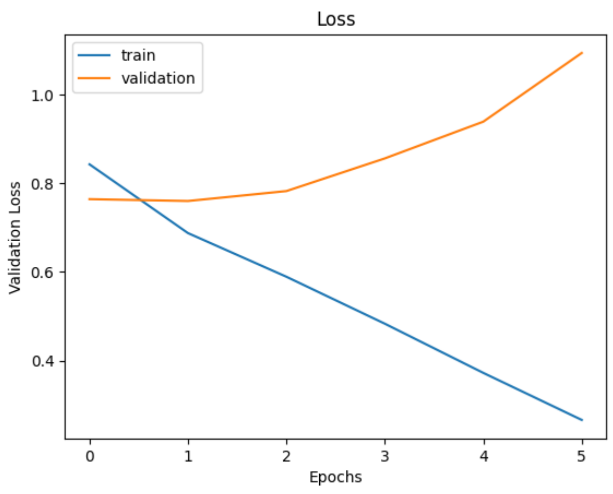
```
              precision    recall  f1-score   support

     class 0       0.07      0.04      0.05        74
     class 1       0.48      0.14      0.22        72
     class 2       0.54      0.75      0.63      1135
     class 3       0.80      0.64      0.71      1778

    accuracy                           0.65      3059
   macro avg       0.47      0.39      0.40      3059
weighted avg       0.68      0.65      0.65      3059
```



One last observation that I want to point out from the last picture is that during the training the validation loss doesn't seem to minimize while the training validation is getting closer to 0. That suggests that the model is overfitting and maybe a bigger dataset is required to get a better performance.


Loss
train
validation
Validation Loss
Epochs