Kaggle Competition - "Sentence Pair Classification"

Competitor - Sasu Alexandru-Cristian

December 29, 2023

1 General observations

Regarding the provided datasets, it can easily be noticed that both the train and validation datasets are imbalanced, given that the number of samples from each class is as follows:

Label/Dataset	Train	Validation
0	2592	74
1	1300	72
2	25722	1135
3	28500	1778

Due to the imbalance of the train dataset, methods to address this issue should be implemented from the get-go, such as using balanced weights for the models, or undersampling the train dataset.

Below will be described all the strategies used in order to obtain as high a macro F1 score on Kaggle as possible. By "strategy", it will be understood a combination of the following: data preprocessing methods, feature extraction methods, machine learning models.

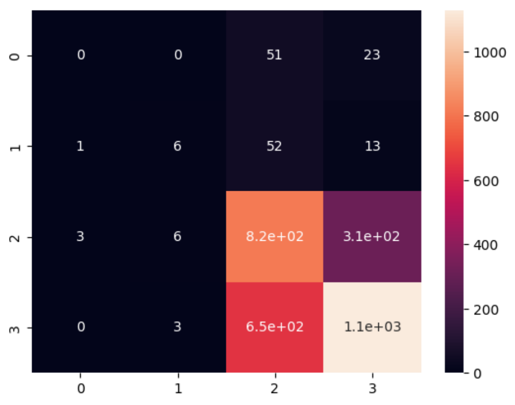
2 Strategies used

2.1 Strategy 1.

As each entry in the training dataset was made up of two sentences, the data that went into the classification model was further processed so that each sample consisted of the concatenation of the two sentences, and between them, a custom string separator. Afterwards, the samples were applied the following preprocessing methods: text to lowercase, punctuation and other special characters removal, stop words removal; and then fed to the model.

Even though the training dataset is imbalanced, a method that didn't explicitly address this issue was still tried. As such, the model *fastText* was used, which can perform moderately well on imbalanced data due to its word embedding strategy. The model's parameters had the following settings: learning rate of 0.1, 5 epochs, *Softmax* loss function.

The following results were obtained on the validation dataset:



	precision	recall	f1-score	support
0	0	0	0	74
1	0.40	0.08	0.14	72
2	0.52	0.72	0.60	1135
3	0.76	0.63	0.69	1778
accuracy	-	-	0.64	3059
macro avg	0.42	0.36	0.36	3059
weighted avg	0.6	0.64	0.63	3059

Using this strategy, the Kaggle submission achieved a macro F1 score of 45.426 on the public leaderboard.

As expected, the model has a high accuracy and a low macro f1 score, being unable to correctly classify almost all of the data samples with labels 0 and 1.

2.2 Strategy 2.

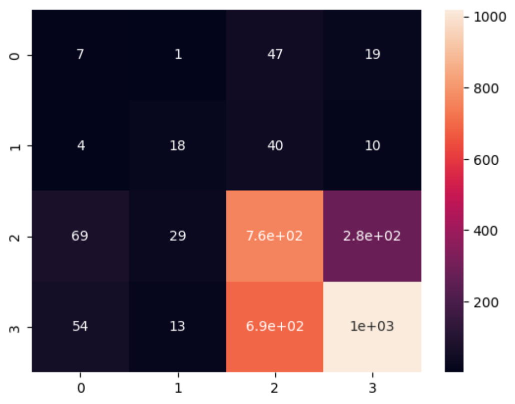
This strategy sought to address the data imbalance problem. In order to do this, more appropriate weights have been chosen for each class, through the function *compute_class_weight* from *sklearn*, namely: 0: 5.60, 1: 11.17, 2: 0.56, 3: 0.50. And to set these weights, there had to be used a model architecture that allowed the user to specify the class weights, meaning that *fastText* was no longer usable, and instead an *LSTM* model from the *TensorFlow* library took its place. The preprocessing methods used were the same as the ones from the previous strategy.

Model details:

- Architecture: an embedding layer that received sequences of length 200 and outputted embeddings of length 100, an LSTM layer with 128 units, a dense layer using the *Softmax* activation function
- Hyperparameters: *Categorical Cross Entropy* loss, *Adam* optimizer, batch size of 1024, 15 epochs with early stopping that monitors the validation loss

In the implementation of this strategy, the *Tokenizer* class from *tensorflow.keras.preprocessing.text* has been used, at word level, in order to transform each sample into an array of 200 features consisting of integer numbers, after which these arrays were fed to the model.

The following results were obtained on the validation dataset:



	precision	recall	f1-score	support
0	0.05	0.09	0.07	74
1	0.30	0.25	0.27	72
2	0.49	0.67	0.57	1135
3	0.77	0.57	0.66	1778
accuracy	-	-	0.59	3059
macro avg	0.40	0.40	0.39	3059
weighted avg	0.64	0.59	0.60	3059

Using this strategy, the Kaggle submission achieved a macro F1 score of 56.674 on the public leaderboard.

In comparison with Strategy 1., this time the model classified more validation samples as belonging to classes 0 and 1, although not by a considerable amount. As such, the macro F1 score is a small amount better, but the accuracy dropped by 5%.

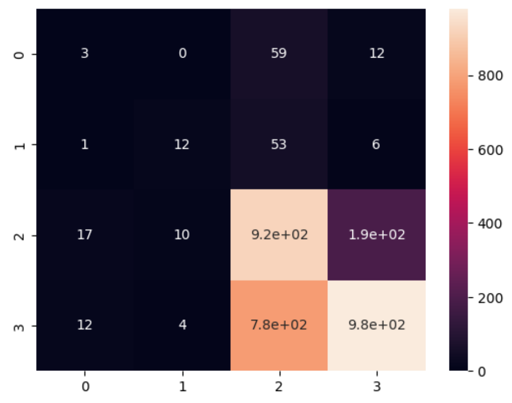
The use of custom class weights did not bring a considerable advantage over the predefined class weights from the previous strategy, which may indicate that further methods to approach the training dataset imbalance have to be used, such as under-sampling. Moreover, a large number of samples from classes 2 and 3 are still being misclassified, which could mean that the feature extraction method needs improvement.

2.3 Strategy 3.

In order to address the problem of having an underperforming feature extraction method, this strategy had the purpose of finding a better method. As such, the *TF-IDF* vectorizer method was used, on unigrams, with a maximum of 5000 features. Moreover, the preprocessing methods were the same as the ones used before.

The model used in this strategy was again changed, as the *LSTM* required integer numbers, but this time the input consisted of floating point numbers between 0 and 1. The model used this time was an *SVM*, that maintained the properties of the weights from the previous strategy. Its parameters were the following: regularization factor of 1, "auto" gamma, Radial Basis Function kernel, balanced weights.

The following results were obtained on the validation dataset:



	precision	recall	f1-score	support
0	0.09	0.04	0.06	74
1	0.46	0.17	0.24	72
2	0.51	0.81	0.62	1135
3	0.83	0.55	0.66	1778
accuracy	-	-	0.63	3059
macro avg	0.47	0.39	0.40	3059
weighted avg	0.68	0.63	0.62	3059

Using this strategy, the Kaggle submission achieved a macro F1 score of 60.496 on the public leaderboard.

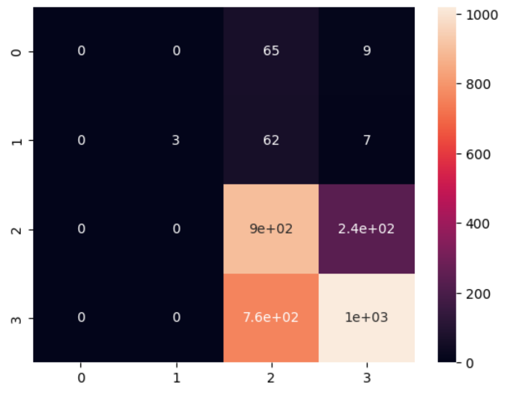
Compared to the previous strategy, overall all of the metrics have improved by a small amount, although the model misclassified more samples with labels 0 and 1. The *Tokenizer* from before was tested with this SVM, although it performed considerably worse, having an accuracy and macro F1-score with more than 20% less than the newly used TF-IDF method. This may indicate that the use of the *TF-IDF* vectorizer method is able to produce more meaningful features compared to the *Tokenizer* method.

The fact that the macro F1 score from Kaggle is noticeably higher than the macro F1 score obtained on the validation set indicates that the model has an easier time classifying the samples from the test dataset, which may mean that training samples have leaked in the test dataset. Upon further inspection, all samples from the test dataset can be found in both the training and validation datasets.

2.4 Strategy 4.

This strategy consisted only in trying another machine learning model, while preserving the same preprocessing and feature extraction methods as before. The model used was a *Random Forest* classifier, with the following parameters: 100 trees, no max depth, a minimum of 2 samples required to split an internal node, Gini criterion, balanced weights.

The following results were obtained on the validation dataset:



	precision	recall	f1-score	support
0	0.00	0.00	0.00	74
1	1.00	0.04	0.08	72
2	0.50	0.79	0.62	1135
3	0.80	0.57	0.67	1778
accuracy	-	-	0.63	3059
macro avg	0.58	0.35	0.34	3059
weighted avg	0.68	0.63	0.62	3059

Using this strategy, the Kaggle submission achieved a macro F1 score of 66.663 on the public leaderboard.

During this strategy, there was an attempt to instead use the features generated by a *Tokenizer* with the same settings as the one from Strategy 2., although this method produced worse overall results than the ones obtained with the *TF-IDF* vectorizer.

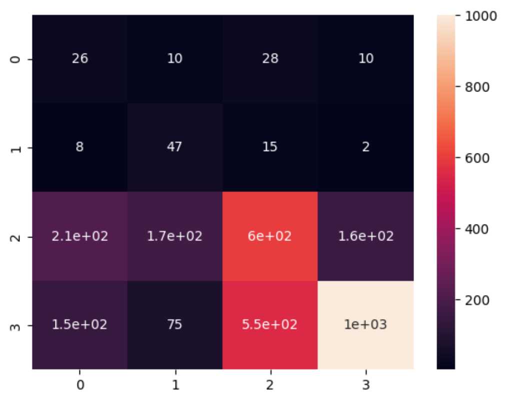
This strategy's Kaggle submission obtained the highest macro F1 score up to this point, although the results obtained on the validation data are similar to those obtained for Strategy 1. Moreover, the difference between the Kaggle macro F1 score for this strategy and the Kaggle macro F1 score for Strategy 1. is $\approx 20\%$. As such, one possibility as to why the current strategy obtained a higher competition score could be due to overfitting, given the discovery made at the previous strategy.

2.5 Strategy 5.

For this approach, the *LSTM* model from Strategy 2. was revisited. The goal was to use custom embeddings for the embeddings layer of the model. In order to do this, the *Word2Vec* method from the *Gensim* library was used. The model chosen for embeddings creation was the *skip-gram* model. Moreover, for the model to be able to efficiently process the new embeddings, its architecture had to change, now consisting of: an embedding layer that received sequences of length 200 and outputted embeddings of length 100, two bidirectional LSTM layers each with 128 units, two dropout operations each with a rate of 0.8, a dense layer using the *Softmax* activation function; while the hyperparameters stayed the same.

The preprocessing and feature extraction methods used were the same as the ones in Strategy 2.

The following results were obtained on the validation dataset:



	precision	recall	f1-score	support
0	0.07	0.35	0.11	74
1	0.15	0.65	0.25	72
2	0.50	0.53	0.51	1135
3	0.86	0.56	0.68	1778
accuracy	-	-	0.55	3059
macro avg	0.39	0.52	0.39	3059
weighted avg	0.69	0.55	0.59	3059

No Kaggle submission was uploaded using this strategy.

Using the custom embeddings did not bring noticeable improvements over the previous strategies. As such, the use of them was discontinued.

2.6 Strategy 6.

Given that all of the test data can be found inside the train and validation data, this strategy consisted of training a model on both the train and validation data. The Random Forest model, preprocessing methods – minus the removal of stop words, as a bit better results were obtained by leaving them in the data – and feature extraction methods from Strategy 3. were chosen, through which all of the metrics on train and validation data reached a score of 100%.

Using this strategy, the Kaggle submission achieved a macro F1 score of 67.688 on the public leaderboard.

3 Conclusions

The features obtained using *TF-IDF* brought better results than the ones obtained using the *keras Tokenizer*.

Traditional machine learning methods may be powerful enough for the given task, as they obtained similar results with the LSTM deep learning method.

There is a possibility that the training dataset does not have enough features, or its features don't bring enough information, such that there can be obtained desirable results on data that the model hasn't been trained on, and so the validation dataset contains data that is too difficult to classify.

As discovered during the implementation of Strategy 3., the test dataset is faulty, as its entirety can be found in the training and validation datasets. This means that it cannot be used for an accurate measure of the model on real-world data.