Proceedings of the 12th Conference on Language Resources and Evaluation (LREC 2020), pages 5000–5009
Marseille, 11–16 May 2020

Cooking Up a Neural-based Model for Recipe Classification

Elham Mohammadi[1,2,3], Nada Naji[1], Louis Marceau[1], Marc Queudot[1,3]
Eric Charton[1], Leila Kosseim[2], Marie-Jean Meurs[3]
[1]Banque Nationale du Canada, [2]Concordia University, [3]Université du Québec à Montréal UQAM,
Montreal, QC, Canada
{nada.naji, louis.marceau, marc.queudot, eric.charton}@bnc.ca
{elham.mohammadi, leila.kosseim}@concordia.ca
meurs.marie-jean@uqam.ca

Abstract

In this paper, we propose a neural-based model to address the first task of the DEFT 2013 shared task, with the main challenge of a highly imbalanced dataset, using state-of-the-art embedding approaches and deep architectures. We report on our experiments on the use of linguistic features, extracted by Charton et al. (2014), in different neural models utilizing pretrained embeddings. Our results show that all of the models that use linguistic features outperform their counterpart models that only use pretrained embeddings. The best performing model uses pretrained CamemBERT embeddings as input and a Convolutional Neural Network (CNN) as the hidden layer, and uses additional linguistic features. Adding the linguistic features to this model improves its performance by 4.5% and 11.4% in terms of micro and macro F1 scores, respectively, leading to state-of-the-art results and an improved classification of the rare classes.

Keywords: Embeddings, Neural Classification, Linguistic Features, Cooking Recipe Classification

1. Introduction

In this paper, we present the neural models we developed to revisit the DEFT (Defi Fouille de Texte) 2013 (Grouin et al., 2013) shared task, using state-of-the-art embedding approaches and deep architectures. The DEFT 2013 shared task addressed the general task of mining cooking recipes in French through several classification and information extraction tasks. Task 1, which is the focus of this paper, addressed the classification of recipes into four levels of difficulty (*Very Easy*, *Easy*, *Fairly Difficult*, and *Difficult*). The dataset is based on a collaborative website where individual users can post their own recipes. Due to this collaborative nature, the dataset raises several challenges, including the subjectivity of the labels and the use of varied vocabulary and grammar. Moreover, the dataset is highly imbalanced with approximately 90% of the samples belonging to the *Very Easy* and *Easy* classes, and only around 8% and less than 1% of the samples with a *Fairly Difficult* and *Difficult* label, respectively. These challenges sparked our interest and curiosity as to how the performances of traditional and state-of-the-art approaches would compare in a classification task. Additionally, even though the dataset revolves around food preparation, the challenge of an imbalanced dataset can be encountered in a variety of applications and real-world use cases.

To address the challenge of a high class imbalance, we experimented with pretrained embeddings as input features, and various deep architectures. We also experimented with adding the linguistic features extracted by Charton et al. (2014) to the deep models and measured per-class results to see if our approach could improve the performance on rare classes.

This paper is structured as follows: The next section presents a literature review of research in the area of text classification including both traditional and neural approaches. Section 3 describes the dataset and the task in detail. Section 4 provides a detailed description of our methodology and the classification architectures we used throughout our experiments. In Sections 5 and 6, we discuss the results of our experiments. Finally, in Section 7, we draw conclusions and highlight future work.

2. Related Work

Recipes and food classification and analysis have been the topic of interest of many research studies and applications. Kicherer et al. (2018) performed automatic recipe multilabel classification and feature analysis for a dataset of 5,000 noisy recipes from the web and 87 predefined classes. Su et al. (2014) and Naik and Polamreddi (2015) addressed recipe classification into cuisines based on ingredients using support vector machine (SVM) (Cortes and Vapnik, 1995; Joachims, 1998). The general task of text classification can be found in various use-cases, such as filtering (e.g. spam filtering), information retrieval (e.g., document classification, probabilistic retrieval models), and sentiment analysis (Aggarwal and Zhai, 2012; Wang and Manning, 2012; Yang et al., 2016). These tasks vary from binary to multi-class, to multi-label classifications. Traditional text classification approaches, such as support vector machine (SVM) (Cortes and Vapnik, 1995; Joachims, 1998) and Naive Bayes (Manning et al., 2008; McCallum and Nigam, 1998), rely heavily on feature selection and feature engineering. The most commonly used features are usually human-designed (Lai et al., 2015) such as bag-of-words, n-grams, part-of-speech, and phrases.

In general, linear classifiers (such as linear SVM), do not share parameters across features and classes (Joulin et al., 2016). Research suggested several solutions to this problem, such as the use of multi-layer neural networks (Joulin et al., 2016; Collobert and Weston, 2008). The rise of pretrained embeddings in recent years, combined with deep architectures, has offered powerful solutions for text classification (Lai et al., 2015).

Difficulty Level	**Train**		**Development**		**Test**	
	# of Samples	**Percentage**	**# of Samples**	**Percentage**	**# of Samples**	**Percentage**
Very Easy	5569	50.2%	1393	50.2%	1132	49.0%
Easy	4601	41.5%	1151	41.5%	968	41.9%
Fairly Difficult	855	7.7%	213	7.7%	189	8.2%
Difficult	64	0.6%	16	0.6%	20	0.9%
Total	11089	100.0%	2773	100.0%	2309	100.0%

Table 1: Statistics of the DEFT 2013 - Task 1 datasets. The training and development datasets were originally released as the training dataset by the DEFT 2013 organizers.

fastText (Bojanowski et al., 2017) and BERT (Devlin et al., 2019) have been shown to be scalable, leading to state-of-the-art performance in many NLP tasks (Joulin et al., 2016). Dai and Le (2015) trained Long Short Term Memory (LSTM) (Hochreiter and Schmidhuber, 1997) and Recurrent Neural Networks (RNN) on a variety of document classification tasks and found that state-of-the-art performance can be achieved with careful hyperparameter tuning. However, despite the popularity of deep models and their state-of-the-arts results in many tasks, many issues still need to be investigated. Wang et al. (2017) found that the classification of short texts can be challenging due to the texts' occasional deviation from accepted syntax, their lack of context, and the added ambiguity resulting from it. To address this challenge, they proposed a model which makes use of both explicit and implicit representations. To have access to *explicit representations*, they first use a knowledge base to extract the top 10 concepts in a sample. To create *implicit representations* of samples, they use pretrained word embeddings which are kept static and character embeddings which are learned during the training of the model. As their deep architecture, they use a CNN with two different branches, one of which is used for the training of character embeddings, and another that takes as input the explicit representation alongside the pretrained word vectors and keeps them frozen. In the end, the outputs of both branches are concatenated and used for the final classification. Using this setup, their model could significantly improve state-of-the-art results on short text classification, showing that a combination of traditional and novel approaches can be effective in a challenging classification task. Furthermore, little research in the area of deep learning has focused on the challenges of handling an imbalanced distribution of labels over data, although many real-world applications deal with highly imbalanced datasets (Johnson and Khoshgoftaar, 2019). Following the same approach as Wang et al. (2017) in using a mixture of traditional and neural approaches, we investigate the effectiveness of adding linguistic features, extracted by Charton et al. (2014), to deep models in addressing the class imbalance challenge in recipe classification.

3. Dataset and Task

In this work, we use the DEFT 2013 dataset from Task 1 (Grouin et al., 2013) to develop and evaluate our models. The dataset consists of cooking recipes taken from Marmiton (`www.marmiton.org`); a French collaborative website for sharing cooking recipes. When users submit a recipe on this website, they must choose one of the four predefined difficulty levels: *Very Easy*, *Easy*, *Fairly Difficult*, or *Difficult* that corresponds to their perceived complexity of the recipe being submitted. The goal of this classification task is to correctly assign a difficulty level to a cooking recipe given its title, ingredients, and instructions.

The DEFT 2013 - Task 1 training and testing datasets consist of 13,862 and 2,309 French recipes, respectively, labeled with their difficulty level. In order to validate the model, we set aside 20% of the training data and used it as the development dataset. The remaining 80% of the original training data was used for training the model.

Table 1 summarizes statistics of the training, development, and test datasets. As shown in Table 1, the distribution of the labels in all datasets is highly imbalanced. Approximately 50% and 41% of the samples belong to the *Very Easy* and *Easy* classes, respectively. That leaves merely around 8% of the data belonging to the *Fairly Difficult* class, and less than 1% of the data belonging to the *Difficult* class.

4. Methodology

In order to evaluate the use of neural classification on the DEFT 2013 dataset, we performed two sets of experiments. The first one involves using different deep architectures with pretrained embeddings as input; while in the second, a combination of pretrained embeddings and extracted linguistic features are utilized for classification.

The overall model architecture is shown in Figure 1. Details of the models are presented below.

4.1. The Input Layer

Each recipe includes a title and a preparation section. In order to feed the recipes to the model, first, the title and the preparation sections of each recipe are concatenated. These samples are then fed to an embedder.

As shown in Figure 1, the embedder takes in a sample as input, tokenizes it based on the type of embedding to be produced (e.g. contextual versus non-contextual), then outputs a dense vector representation for each token.

To create these vector representations, three different types of embedders are used:

fastText The fastText embedder (Bojanowski et al., 2017) is used to output 300-dimensional pretrained embeddings. fastText embeddings are trained based on the skip-gram model proposed by Mikolov et al. (2013). These embeddings are created by taking into account the morphology of words, instead of treating them as distinct units. Therefore, using this method, each word is represented as a sum of the representations of the character N-grams

Model	Hyperparameters
CNN-fastText	300 unigram, 200 bigram, 100 trigram, and 100 4-gram filters, with max pooling
GRU-fastText	two layers of 128 bidirectional GRUs, with attention
LSTM-fastText	one layer of 64 bidirectional LSTM units, with attention
CNN-BERT	300 unigram, 200 bigram, 100 trigram, and 100 4-gram filters, with max pooling
GRU-BERT	one layer of 64 bidirectional GRUs, with attention
LSTM-BERT	one layer of 64 bidirectional LSTM units, with attention
Transformer-fastText	three layers of 300 transformer units with 6 attention heads
CNN-CamemBERT	250 bigram and 50 trigram filters, with max pooling

Table 2: Hyperparameters used to train the models not using linguistic features.

Model	Hyperparameters
CNN-fastText	250 bigram convolution filters, with max pooling
GRU-fastText	one layer of 64 bidirectional GRUs, with attention
LSTM-fastText	one layer of 64 bidirectional LSTM units, with attention
CNN-BERT	250 bigram convolution filters, with max pooling
GRU-BERT	one layer of 64 bidirectional GRUs, with attention
LSTM-BERT	one layer of 64 bidirectional LSTM units, with attention
Transformer-fastText	three layers of 300 transformer units with 6 attention heads
CNN-CamemBERT	400 unigram filters, with max and average pooling

Table 3: Hyperparameters used to train the models utilizing linguistic features.

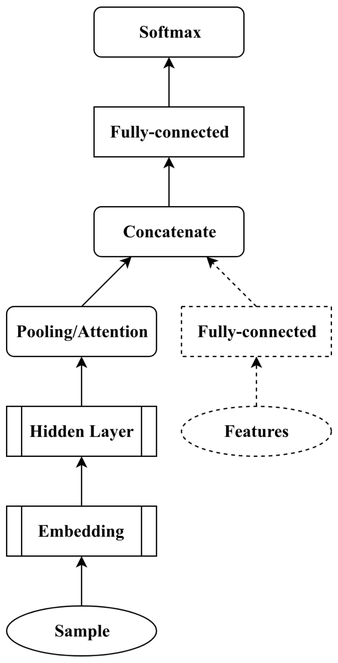

Figure 1: Overall architecture of the model.

constituting it, giving us the opportunity to experiment with models that focus on N-grams in a sample to perform the classification. The *flair* (`https://github.com/zalandoresearch/flair/`) package was used to extract the pretrained fastText embeddings.

BERT In addition to fastText, a BERT embedder is used to produce pretrained BERT embeddings for the tokens in each sample. Proposed by Devlin et al. (2019), BERT embeddings are trained using a transformer architecture that takes both left and right contexts into account, allowing us to experiment with contextual embeddings as well as non-contextual ones (i.e., fastText). We use the BERT base multilingual cased version which has been pretrained on the cased Wikipedia text for 104 languages, using the *first* pooling operation (i.e., using only the initial sub-word's embedding), and working with the last layer of the pretrained model, which results in a dense 768-dimensional representation for each token.

CamemBERT Finally, we experiment with the pretrained CamemBERT embeddings (Martin et al., 2019). CamemBERT is a French version of the BERT model and is pretrained on the French portion of the multilingual Oscar corpus (Ortiz Suárez et al., 2019). Extracting features from the last layer of the pretrained CamemBERT model leads to a 768-dimensional representation for each token. The pretrained CamemBERT model provides the opportunity to measure the effect of using contextual embeddings that are trained exclusively on French data.

4.2. The Hidden Layer

As shown in Figure 1, the dense vectors created by the input layer are fed to the hidden layer. We experiment with four different types of hidden architectures: a CNN (LeCun et al., 1999) which works on N-grams in a sample by processing N consecutive token embeddings separately, bidirectional Gated Recurrent Units (GRU) (Cho et al., 2014) and a bidirectional LSTM (Hochreiter and Schmidhuber, 1997) that process token embeddings from first to last in a forward pass and from last to first in a backward pass, and finally, a transformer encoder (Vaswani et al., 2017) which creates contextualized token representations using multi-head attention.

4.3. The Pooling/Attention Layer

When a CNN is used in the hidden layer, first the Concatenated Rectified Linear Unit (CReLU) activation function is applied on the output of the hidden layer. Its ouput is then passed to a pooling layer.

In the case of the bidirectional GRU, the bidirectional LSTM, and the transformer, an attention mechanism (Bahdanau et al., 2014) is used instead of pooling. The attention mechanism works by calculating weights for different time-steps (tokens), allowing it to give more weight to tokens that can play a more important role in the final classifications. These weights are generated by the following process: First, a feed-forward layer is applied on the output of the hidden layer for a time-step (of size N), mapping it to a scalar (of size 1). Equation 1 shows how the scalar v_t for a particular time-step t is calculated. y_t is the output of the hidden layer for that time-step and w stands for the weights in the single feed-forward layer.

$$v_t = y_t \times w \tag{1}$$

When the scalars are calculated for all time-steps, they are concatenated and a softmax activation function is applied on the resulting vector to create the vector of weights ω, using Equation 2:

$$\omega = Softmax([v_1, v_2, v_3, \ldots, v_n]) \tag{2}$$

Finally, the weight vector, generated in the previous step, is used by the attention mechanism to calculate a weighted-average of the outputs of the hidden layer for all time-steps in a sample text. Equation 3 shows how the attention mechanism functions by multiplying the output of the hidden layer at time-step t (y_t) by the corresponding generated weight (ω_t), then summing over the results of this multiplication for time-steps *1* to *n*.

$$Attention = \sum_{t=1}^{n} y_t \omega_t \tag{3}$$

4.4. Addition of linguistic features

In order to measure the effect of adding linguistic features to the deep neural models, we experimented with the set of linguistic features extracted by Charton et al. (2014) specifically for this task. The following features are extracted for each recipe:

1. The number of words that make up the title, an integer.
2. The number of words that make up the preparation section, an integer.
3. The number of ingredients, an integer.
4. The cost associated with the meal that can be *cheap*, *average*, or *fairly expensive*. This feature is input as an ordinal number (1-3).
5. The presence (or absence) of 22 discriminative words which belong to the *fairly difficult* class, represented as 22 binary features. The extraction and selection process of these words is explained in detail by Charton et al. (2014).
6. The presence (or absence) of 48 discriminative trigrams extracted from the preparation section of the samples, represented as 48 binary features.
7. The number of verbs belonging to three discriminative verb families present in the preparation section, represented as three integers (for a detailed explanation of the verb families and how they are extracted, see (Charton et al., 2014)).

The extraction of the above features leads to the creation of feature vectors of size 77. These feature vectors are first passed to a one-layer feed-forward neural network, whose output is concatenated to the output of the attention/pooling layer and mapped to a vector of size 4 (representing the 4 classes) by another feed-forward layer. Finally, a softmax activation function produces the probability distribution over the classes.

4.5. Model Optimization

In order to train the neural networks, the AdamW optimizer (Loshchilov and Hutter, 2019) (a modified version of the Adam optimization method proposed by Kingma and Ba (2015)) with a weight decay coefficient of 0.02 was used. The learning rate was set to 10^{-3} for all models. However, for models with CNN as the hidden layer, the initial learning rate of 10^{-3} was set to 10^{-4} after two epochs of training.

As loss function, multi-class cross-entropy was used with class weights, to handle the imbalanced distribution of the labels in the dataset, assigning more penalty to errors which were made on less frequent classes compared to more frequent ones. For the models that did not make use of linguistic features, the class weights were calculated proportional to the inverse of the number of samples in each class (in the training dataset), resulting in weights of 0.0104, 0.0126, 0.0680, 0.9089 for the *Very Easy*, *Easy*, *Fairly Difficult*, and *Difficult* classes respectively. For models utilizing linguistic features, the class weights were set to 0.1, 0.1, 0.2, 0.6 for the *Very Easy*, *Easy*, *Fairly Difficult*, and *Difficult* classes, respectively.

The mini-batch size was set to 32 and in order to have a uniform sequence length in each batch, zero-padding was used. Samples with similar lengths were placed in the same batch to minimize the amount of padding.

In the models using fastText embeddings as input, only the first 300 tokens of the samples were used. In the models with BERT embeddings, the samples were limited to their first 100 tokens. A dropout layer with a rate of 0.2 was applied on the output of the *Concatenate* layer in Figure 1 to reduce overfitting. Finally, gradient clipping with a norm of 0.5 was used to prevent the exploding gradient problem (Pascanu et al., 2012).

4.6. Overall Training Process

Each model was trained for 20 epochs and the model parameters were saved after each epoch of training. The optimal model parameters were chosen from the epoch in which the best micro score on development data was achieved. This model was then evaluated on the test dataset. The hyperparameters used for the training of each model are presented in Tables 2 and 3.

4.7. Fine-tuning BERT

As an additional experiment, the BERT base multilingual cased model was fine-tuned. To do so, the AdamW optimizer with a weight decay rate of 0.01 was used.

Model	Development				Test			
	Micro F1-P-R	Macro F1	Macro P	Macro R	Micro F1-P-R	Macro F1	Macro P	Macro R
CNN-fastText	60.9	**42.7**	**44.2**	41.3	**59.6**	**45.3**	**50.8**	**40.9**
GRU-fastText	57.0	35.0	35.2	34.8	57.6	34.3	34.5	34.1
LSTM-fastText	56.1	40.8	**44.2**	38.0	54.2	34.6	34.2	34.9
CNN-BERT	**61.5**	39.3	41.1	37.6	58.8	37.7	39.4	36.1
GRU-BERT	59.9	38.5	38.5	38.4	58.0	36.8	37.0	36.7
LSTM-BERT	58.8	42.2	44.1	40.4	56.9	37.3	38.8	36.0
Transformer-fastText	58.6	35.5	37.8	33.4	58.4	41.0	48.5	35.5
BERT (fine-tuned)	56.9	39.3	42.9	36.2	55.9	36.0	36.8	35.3
CNN-CamemBERT	60.9	**42.7**	43.4	**42.0**	59.3	38.6	39.9	37.3

Table 4: Performance of different deep architectures with pretrained embeddings as input, on the development and test data

Model	Development				Test			
	Micro F1-P-R	Macro F1	Macro P	Macro R	Micro F1-P-R	Macro F1	Macro P	Macro R
CNN-fastText	64.7	**51.7**	**68.6**	41.5	63.6	49.4	**66.9**	39.2
GRU-fastText	64.4	39.8	42.8	37.2	63.4	39.4	42.4	36.8
LSTM-fastText	64.0	39.6	44.9	35.5	59.9	36.4	40.3	33.2
CNN-BERT	64.5	49.1	60.0	41.6	62.0	47.3	59.3	39.3
GRU-BERT	65.8	41.7	45.5	38.5	63.1	39.3	42.1	36.8
LSTM-BERT	64.5	45.5	55.5	38.5	62.3	43.3	54.8	35.8
Transformer-fastText	63.1	39.8	42.9	37.2	61.6	39.4	42.6	36.7
CNN-CamemBERT	**66.4**	50.3	58.5	**44.2**	**63.8**	**50.0**	62.0	**42.0**

Table 5: Performance of different deep architectures using pretrained embeddings and additional linguistic features on the development and test data

Cyclical learning rate (Smith, 2017) with a maximum value of 10^{-5} was chosen, using the fastAI implementation (`https://docs.fast.ai/callbacks.one_cycle.html`). Weighted cross-entropy was used as loss function and the mini-batch size was set to 32. Finally, samples were limited to their first 128 tokens before being fed to the model. Fine-tuning was stopped at the epoch for which the lowest validation error was recorded. The results of the fine-tuned BERT model can be found in Table 4.

5. Results

Micro-average evaluation is used as the primary metric to evaluate the performance of the models. Since the micro-average scores are calculated on all 4 classes, the micro-average F1 score is mathematically equal to micro-average precision and micro-average recall.

Furthermore, macro F1, precision, and recall are used as additional evaluation metrics. In the context of the DEFT 2013 shared task, the macro F1 score is calculated using Equation 4 (Grouin and Forest, 2012), using macro precision and macro recall (Sokolova and Lapalme, 2009). Macro precision and macro recall are computed using Equations 5 and 6, respectively, with n representing the number of classes.

$$macro\ F1 = \frac{2 \times macro\ precision \times macro\ recall}{macro\ precision + macro\ recall} \quad (4)$$

$$macro\ precision = \frac{\sum_{i=1}^{n} \frac{(true\ positives_i)}{(true\ positives_i + false\ positives_i)}}{n} \quad (5)$$

$$macro\ recall = \frac{\sum_{i=1}^{n} \frac{(true\ positives_i)}{(true\ positives_i + false\ negatives_i)}}{n} \quad (6)$$

In this paper, both macro and micro F1 scores are used in order to analyze the results and compare them with the results of systems participating to DEFT 2013.

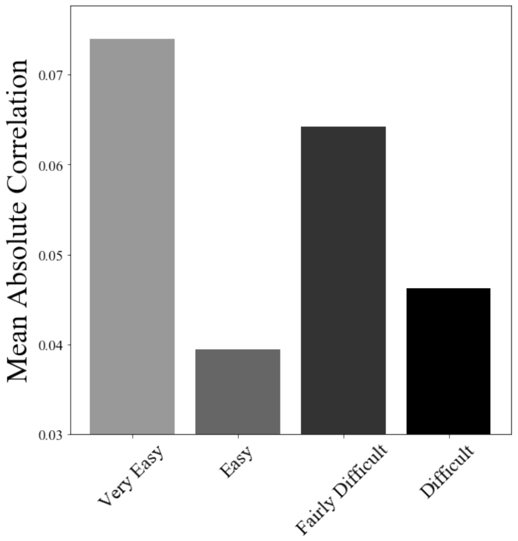

Figure 2: Mean absolute correlation of the linguistic features with different classes.

Table 4 presents the performance of different models which utilize only pretrained embeddings, in terms of micro score (F1, precision, and recall) and macro scores on both development and test data.

Table 5 shows the performance of different models that use pretrained embeddings and linguistic features to perform the classification. As shown in Tables 4 and 5, in terms of micro evaluation, all of the models which utilize linguistic features outperform their counterpart models that only use pretrained embeddings. The effect of adding linguistic features is discussed in further detail in Section 6.

For comparative purposes, the results achieved by the top 3 systems (Charton et al., 2014; Collin et al., 2013; Bost et al., 2017) at DEFT 2013 are presented in Table 6. As

	Micro F1-P-R	Macro F1	Macro P	Macro R
First System (Charton et al., 2014)	62.5	48.4	68.2	37.5
Second System (Collin et al., 2013)	61.2	45.1	52.4	39.5
Third System (Bost et al., 2017)	59.2	45.3	63.3	35.3

Table 6: Results of top 3 participating systems on the test data of DEFT 2013 - Task 1

Model	**Development**				**Test**			
	Very Easy	**Easy**	**Fairly Difficult**	**Difficult**	**Very Easy**	**Easy**	**Fairly Difficult**	**Difficult**
CNN-fastText	72.4	60.6	24.6	22.2	**72.4**	58.8	23.3	9.5
GRU-fastText	72.7	59.7	19.0	0.0	**72.4**	58.3	18.3	0.0
LSTM-fastText	**74.6**	52.6	13.8	0.0	71.9	47.0	10.2	0.0
CNN-BERT	72.0	60.8	24.9	21.0	70.0	58.1	22.5	17.4
GRU-BERT	73.9	61.1	22.6	0.0	72.0	58.1	18.9	0.0
LSTM-BERT	71.4	**62.7**	2.7	20.0	69.5	**60.5**	4.0	9.1
Transformer-fastText	70.4	60.0	21.4	0.0	69.3	58.0	22.6	0.0
CNN-CamemBERT	74.0	61.8	**27.0**	**27.2**	72.2	59	**25.2**	**25.0**
DEFT Top System (Charton et al., 2014)	72.0	57.4	23.2	12.4	71.7	56.2	18.8	9.5

Table 7: Per class results, in terms of F1 score, achieved by different deep architectures using pretrained embeddings and additional linguistic features on development and test data

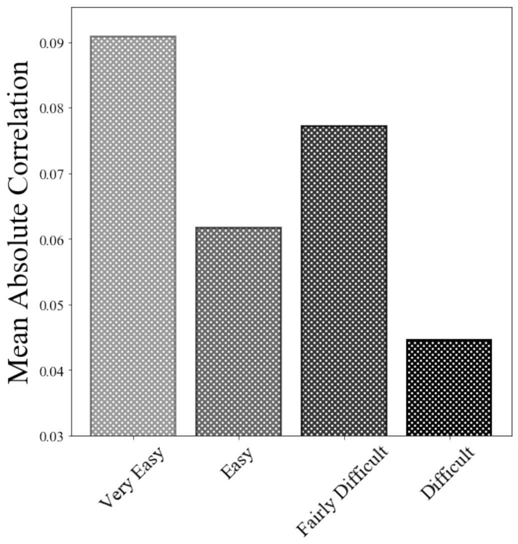

Figure 3: Mean absolute correlation of the neural features extracted by the CamemBERT-CNN model (trained without the linguistic features) with different classes.

Table 5 shows, our best model (CNN-CamemBERT with features) outperforms the top performing model by Charton et al. (2014) at DEFT 2013 with a micro F1 score of 63.8% versus 62.5% and a macro F1 score of 50.0% versus 48.4%. Finally, for further analysis, Table 7 shows F1 scores of the models that utilize pretrained embeddings and linguistic features for each difficulty class, along with the results of the top performing system by Charton et al. (2014) at DEFT 2013, showing that the CNN-CamemBERT model with features performs well on individual classes, especially, in the case of less frequent ones. It should be noted that in order to obtain the development results and validate the model, Charton et al. (2014) used five fold cross-validation.

6. Discussion

In this section, we first inspect the correlation between the linguistic features and the difficulty classes. Then, we turn our attention to the change in the performance of each model after the addition of the linguistic features. Finally, we take a closer look at the performance of the models on different classes.

6.1. Correlation of Features with Classes

In order to have a better sense of the effect of adding the linguistic features to the models, we first focus on the correlation between the discriminatve features with the difficulty classes. Pearson correlation is used to calculate the correlation between each feature and each class, separately (through binarizing the labels, i.e. setting the labels from the target class equal to 1, and the rest of the labels equal to 0). Since the resulting value can be either positive or negative and in both cases signaling a correlation, the absolute of the computed correlation values is used. Finally, for each class, an average is computed over all of the correlation values of different features with that class.

Figure 2 shows the mean absolute correlation between the linguistic features and the classes.

As shown in Figure 2, although the *Fairly Difficult* and *Difficult* classes are the least frequent ones, the mean absolute correlation between the linguistic features and the classes is at its minimum for the *Easy* class, with approximately 41% of samples belonging to it. This demonstrates that the extracted linguistic features can be assumed to be in favor of the less frequent classes despite the small number of samples belonging to those.

As a comparison, the mean absolute correlation between the neural features extracted by the CNN-CamemBERT model before the addition of the linguistic features and the different classes has been computed (using the same method explained above for computing the correlation values between the linguistic features and the classes). The mentioned neural features are the vectors output by the *Concatenate* layer in Figure 1, before the last fully-connected layer.

Figure 3 presents the mean absolute correlation between the neural features from the CNN-CamemBERT model and the

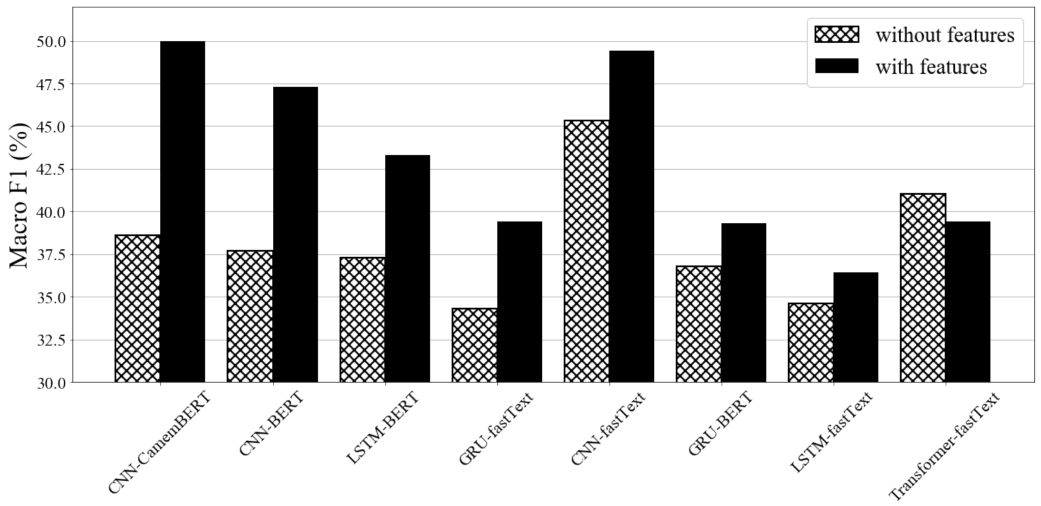

Figure 4: Macro F1 of each model before and after the addition of the linguistic features, sorted from the highest to the lowest amount of improvement.

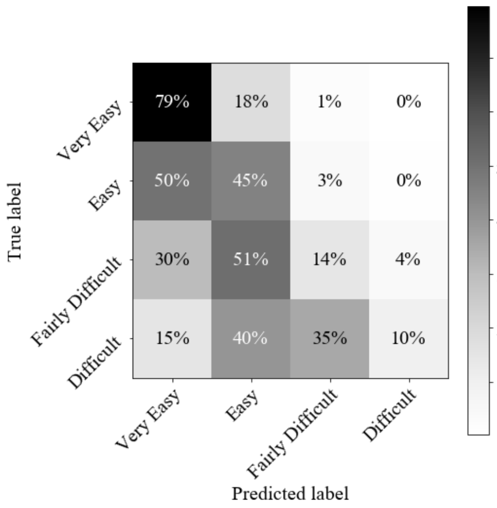

Figure 5: Confusion matrix of the predictions made by the CNN-CamemBERT model, without the use of linguistic features.

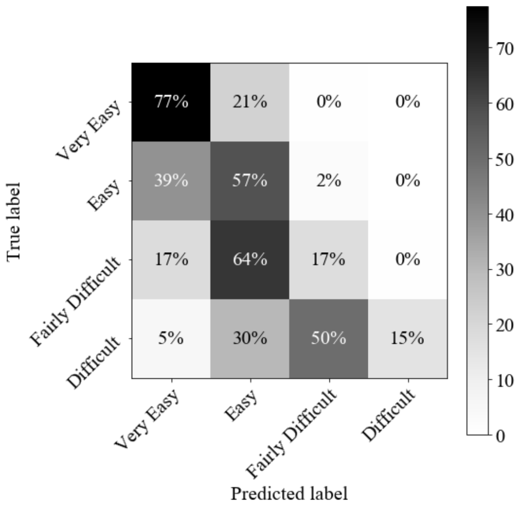

Figure 6: Confusion matrix of the predictions made by the CNN-CamemBERT model, using the linguistic features.

difficulty classes. As shown in Figure 3, the mean absolute correlation between the neural features and the *Fairly Difficult* class is the second highest after the *Very Easy* class, although the *Fairly Difficult* class includes less than 8% of the training samples. We hypothesize that in this case, the class weights have been helpful in training the network towards the extraction of features that are in favor of this class. However, this is not the case for the *Difficult* class as the least amount of mean absolute correlation has been calculated for this class, which is caused by the very small number of training samples belonging to it (less than 1%).

A comparison between Figures 2 and 3, shows how the linguistic features have been helpful in handling the imbalanced distribution of labels in the training data, especially for the *Difficult* class which is the least frequent one. We believe that this has been the reason behind the need for more balanced class weights (as mentioned in Section 4.5.) to achieve better results, when training the models that use the linguistic features.

6.2. Effect of Features on Performance

The help of the linguistic features in better handling the rare classes can explain the increase in the macro F1 after adding the said features to the system (see Figure 4); an increase which can be seen in the macro F1 score achieved by almost all of the models (excluding the Transformer-fastText model).

Looking at Figure 4 which shows the improvement in macro F1 for each model before and after utilizing the linguistic features, it can be observed that, in general, the models that use pretrained BERT-based embeddings (including CamemBERT) as input features, achieve a higher improvement in terms of macro F1 score after using the linguistic features. Knowing that, as explained in Section 4.4., the linguistic features are based on N-grams (and hence, non-contextual), We believe that this is due to the contextual nature of BERT-based embeddings, which can be better complemented by the linguistic features. This is not the case for fastText embeddings, as they are non-contextual by nature. Therefore, the non-contextual linguistic features may not add the same amount of advantage to the models using fastText embeddings in comparison to the models that use BERT and CamemBERT embeddings.

The greatest amount of increase in macro F1 can be seen in the case of the CNN-CamemBERT model (from 38.6% to 50%). When linguistic features were not used, this model was not the top performing one among our models (see Table 4). On the other side, when linguistic features were incorporated in the model, it achieved the best overall performance (see Table 5). This can lead to the hypothesis that the neural features that are extracted by the CNN-CamemBERT model, although not the most representative of the classes, can be complementary to the linguistic features, resulting in the top performance of the model.

6.3. Performance on Different Classes

Table 7 presents the results achieved on different classes by the models that use the linguistic features. As shown in Table 7, four models could achieve an F1 score above zero on the least frequent class, i.e. the *Difficult* class. Out of these four models, three had CNN as their deep architecture. Furthermore, looking at the results that were achieved on the *Fairly Difficult* class, it can be seen that three out of the four top-performing models on this class made use of CNN as their hidden layer. This shows the better capacity of the CNN to capture the information required to detect the two classes *Fairly Difficult* and *Difficult*, which have a significantly smaller number of training samples belonging to them. Finally, it can be seen in Table 7 that the performance of the CNN-CamemBERT model is significantly higher on the last two difficulty classes, compared to other models, resulting in the CNN-CamemBERT model achieving the highest macro F1 among all models (see Table 5).

Figures 5 and 6 present the confusion matrices of the predictions by the CNN-CamemBERT model, before and after the addition of the linguistic features, respectively. A comparison between the two matrices shows that although the performance on the *Very Easy* class drops after adding the features, a remarkably higher number of correct predictions are made on the *Easy* class when the model incorporates the linguistic features. In fact, the CNN-CamemBERT model that does not use the linguistic features is more likely to incorrectly tag a sample as *Very Easy* instead of *Easy*, as a result of the imbalanced distribution of labels over the training data. In the case of the *Fairly Difficult* class, when an incorrect prediction is made, the addition of the linguistic features to the model increases the chances of a *Fairly Difficult* sample to be incorrectly tagged as *Easy*, as opposed to being tagged as *Very Easy*, which can be deemed an improvement in detecting the difficulty degree of a sample recipe.

Finally, looking at the predictions on the *Difficult* class in Figure 5, it is interesting to see that the CNN-CamemBERT model that does not use the linguistic features is more likely to predict a *Difficult* sample as *Easy* instead of tagging it as *Fairly Difficult* (which is more similar to the true label in terms of degree) when making a mistake, thus showing a bias towards a more frequent class. However, this is not the case for the CNN-CamemBERT model that uses the linguistic features, as shown in Figure 6. When making an incorrect prediction on the *Difficult* class, this model is more likely to incorrectly tag a *Difficult* recipe as *Fairly Difficult*, rather than *Easy*, despite the outstandingly lower number of samples in the *Fairly Difficult* class compared to the *Easy* class. This can be seen as a sign of improvement in a classification task where the classes are not completely distinct and also proof of a lesser degree of bias towards the more frequent classes in this model.

7. Conclusion and Future Work

In this paper, we proposed a deep-learning-based classification model to revisit the first task of DEFT 2013. To develop the model, we experimented with pretrained fastText, BERT, and CamemBERT embeddings as input features, and a CNN, GRU, LSTM, and transformer encoder as hidden architecture. We also checked the effect of adding the linguistic features extracted by Charton et al. (2014) to the deep model and evaluated its performance using the DEFT 2013 - Task 1 dataset, which includes a collection of cooking recipes tagged with one of four difficulty levels, and the evaluation metrics specific to the shared task. The results achieved by different models before and after the addition of linguistic features showed that, in general, the performance improved after incorporating the features. The best performance on the test data was achieved with pretrained CamemBERT embeddings as input and CNN as the hidden layer, and with the addition of the linguistic features to the model.

The incorporation of the linguistic features in this model improved its performance by approximately 4.5% in terms of micro F1 and 11.4% in terms of macro F1 score, resulting in state-of-the-art results on the dataset. This setup also proved effective in better handling the imbalanced distribution of the labels over the dataset and lead to a better classification of the rare classes.

As future work, three possible directions can be proposed:

Firstly, it would be interesting to experiment with a mixture of contextual and non-contextual embeddings (for example, CamemBERT and fastText) as input features to the model, knowing that they can capture different information and could potentially act in a complementary manner to perform a better classification. Also, seeing that our best model used pretrained CamemBERT embeddings as input, we could experiment with models using different types of hidden layers, and CamemBERT embeddings as input. Finally, we could check the effect of fine-tuning CamemBERT embeddings and check if it results in a better performing classification model.

Reproducibility

To ensure reproducibility and comparisons between systems, our source code is publicly released as an open source software in the following repository: `https://github.com/cooking-classification/LREC2020`.

The data could be obtained by contacting the DEFT 2013 shared task organizers (see `https://deft.limsi.fr/2013/index.php?id=1&lang=en`

8. Bibliographical References

Aggarwal, C. C. and Zhai, C., (2012). *A Survey of Text Classification Algorithms*, chapter 6. Springer.

Bahdanau, D., Cho, K., and Bengio, Y. (2014). Neural machine translation by jointly learning to align and translate. *Computing Research Repository*, arXiv:1409.0473.

Bojanowski, P., Grave, E., Joulin, A., and Mikolov, T. (2017). Enriching word vectors with subword information. *Transactions of the Association for Computational Linguistics (TACL)*, 5:135–146.

Bost, X., Brunetti, I., Cabrera-Diego, L. A., Cossu, J.-V., Linhares, A., Morchid, M., Torres-Moreno, J.-M., El-Bèze, M., and Dufour, R. (2017). Systémes du LIA à DEFT 13. *arXiv preprint arXiv:1702.06478*.

Charton, E., Meurs, M.-J., Jean-Louis, L., and Gagnon, M. (2014). Using collaborative tagging for text classification: From text classification to opinion mining. *Informatics*, 1(1):32–51.

Cho, K., van Merrienboer, B., Gulcehre, C., Bahdanau, D., Bougares, F., Schwenk, H., and Bengio, Y. (2014). Learning phrase representations using RNN encoder-decoder for statistical machine translation. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP 2014)*, pages 1724–1734, Doha, Qatar, October.

Collin, O., Guerraz, A., Hiou, Y., and Voisine, N. (2013). Participation de orange labs à deft 2013. *Actes du neuvième DÉfi Fouille de Textes*, pages 67–79.

Collobert, R. and Weston, J. (2008). A unified architecture for natural language processing: Deep neural networks with multitask learning. In *Proceedings of the 25th International Conference on Machine Learning (ICML 2008)*, pages 160–167, Helsinki, Finland, July.

Cortes, C. and Vapnik, V. (1995). Support vector networks. *Machine Learning*, 20:273–297.

Dai, A. M. and Le, Q. V. (2015). Semi-supervised sequence learning. In C. Cortes, et al., editors, *Advances in Neural Information Processing Systems (NIPS 28)*, pages 3079–3087. Curran Associates, Inc., December.

Devlin, J., Chang, M.-W., Lee, K., and Toutanova, K. (2019). BERT: Pre-training of deep bidirectional transformers for language understanding. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies (ACL/HLT 2019)*, pages 4171–4186, Minneapolis, Minnesota, June.

Grouin, C. and Forest, D. (2012). *Expérimentations et évaluations en fouille de textes: Un panorama des campagnes DEFT*. Lavoisier.

Grouin, C., Paroubek, P., and Zweigenbaum, P. (2013). DEFT2013 se met à table: présentation du défi et résultats. In *Actes de DEFT*, Les Sables-d'Olonnes, France, 21 juin. TALN.

Hochreiter, S. and Schmidhuber, J. (1997). Long short-term memory. *Neural Computation*, 9(8):1735–1780.

Joachims, T. (1998). Text categorization with support vector machines: Learning with many relevant features. In *Proceedings of 10th European Conference on Machine Learning (ECML 1998)*, Chemnitz, Germany, April.

Johnson, J. M. and Khoshgoftaar, T. M. (2019). Survey on deep learning with class imbalance. *Journal of Big Data*, 6(1):27.

Joulin, A., Grave, E., Bojanowski, P., and Mokolov, T. (2016). Bag of tricks for efficient text classification. *Computing Research Repository*, arXiv:1607.01759.

Kicherer, H., Dittrich, M., Grebe, L., Scheible, C., and Klinger, R. (2018). What you use, not what you do: Automatic classification and similarity detection of recipes. *Data & Knowledge Engineering*, 117:252–263.

Kingma, D. P. and Ba, J. (2015). Adam: A method for stochastic optimization. In *Proceedings of the 3rd International Conference on Learning Representations (ICLR 2015)*, San Diego, CA, USA, May.

Lai, S., Xu, L., Liu, K., and Zhao, J. (2015). Recurrent convolutional neural networks for text classification. In *Proceedings of the 29th AAAI Conference on Artificial Intelligence (AAAI 2015)*, Austin, Texas, USA, January.

LeCun, Y., Haffner, P., Bottou, L., and Bengio, Y. (1999). Object recognition with gradient-based learning. In *Shape, contour and grouping in computer vision*, pages 319–345. Springer.

Loshchilov, I. and Hutter, F. (2019). Decoupled weight decay regularization. In *Proceedings of the 3rd International Conference on Learning Representations (ICLR 2019)*, New Orleans, Louisiana, USA, May.

Manning, C. D., Raghavan, P., and Schútze, H., (2008). *Text Classification and Naive Bayes*, chapter 13. Cambridge University Press.

Martin, L., Muller, B., Suárez, P. J. O., Dupont, Y., Romary, L., de la Clergerie, É. V., Seddah, D., and Sagot, B. (2019). Camembert: A tasty french language model. *arXiv preprint arXiv:1911.03894*.

McCallum, A. and Nigam, K. (1998). A comparison of event models for naive bayes text classification. In *Proceeding of the AAAI workshop on learning for text categorization*, pages 41–48.

Mikolov, T., Sutskever, I., Chen, K., Corrado, G. S., and Dean, J. (2013). Distributed representations of words and phrases and their compositionality. In C. J. C. Burges, et al., editors, *Advances in Neural Information Processing Systems (NIPS 26)*, pages 3111–3119. Curran Associates, Inc.

Naik, J. and Polamreddi, V. (2015). Cuisine classification and recipe generation. *Online: http://cs229. stanford. edu/proj2015/233 report. pdf*.

Ortiz Suárez, P. J., Sagot, B., and Romary, L. (2019). Asynchronous Pipeline for Processing Huge Corpora on Medium to Low Resource Infrastructures. In *Proceedings of the 7th Workshop on the Challenges in the Management of Large Corpora (CMLC 2019)*, Cardiff, United Kingdom, July.

Pascanu, R., Mikolov, T., and Bengio, Y. (2012). Understanding the exploding gradient problem. *Computing Research Repository*, arXiv:1211.5063v1.

Smith, L. N. (2017). Cyclical learning rates for training neural networks. In *2017 IEEE Winter Conference on Applications of Computer Vision (WACV)*, pages 464–472. IEEE.

Sokolova, M. and Lapalme, G. (2009). A systematic analysis of performance measures for classification tasks. *Information processing & management*, 45(4):427–437.

Su, H., Lin, T.-W., Li, C.-T., Shan, M.-K., and Chang, J. (2014). Automatic recipe cuisine classification by ingredients. *UbiComp 2014 - Adjunct Proceedings of the 2014 ACM International Joint Conference on Pervasive and Ubiquitous Computing*, pages 565–570, September.

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, Ł., and Polosukhin, I. (2017). Attention is all you need. In *Advances in neural information processing systems (NIPS 2017)*, pages 5998–6008.

Wang, S. and Manning, C. (2012). Baselines and bigrams: Simple, good sentiment and topic classification. In *Proceedings of the 50th Annual Meeting of the Association for Computational Linguistics (ACL 2012)*, pages 90–94.

Wang, J., Wang, Z., Zhang, D., and Yan, J. (2017). Combining knowledge with deep convolutional neural networks for short text classification. In *International Joint Conference on Artificial Intelligence (IJCAI)*, pages 2915–2921.

Yang, Z., Yang, D., Dyer, C., He, X., Smola, A., and Hovey, E. (2016). Hierarchical attention networks for document classification. In *Proceedings of the 2016 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies (NAACL-HLT 2016)*, pages 1480–1489.