IDIAP COMMUNICATION REPORT

DEEP LEARNING OF CHARISMA

Daniel Carron

Idiap-Com-03-2020

Version of MAY 13, 2024

Centre du Parc, Centre du Parc, Rue Marconi 19, CH - 1920 Martigny
T +41 27 721 77 11 F +41 27 721 77 12 info@idiap.ch www.idiap.ch

IDIAP COMMUNICATION REPORT

DEEP LEARNING OF CHARISMA

Daniel Carron

Idiap-Com-03-2020

AUGUST 2020

Centre du Parc, Centre du Parc, Rue Marconi 19, CH - 1920 Martigny
T +41 27 721 77 11 F +41 27 721 77 12 info@idiap.ch www.idiap.ch

Deep Learning of Charisma

submitted on August 11, 2020

for the obtention of the title of Master in Artificial Intelligence
by

Daniel Carron

student number 14-863-070

Presented to:

Dr. Philip N. Garner, project supervisor
Alexandre Nanchen, company supervisor
Olivier Bornet, project coordinator

Preface

This project was motivated by personal interest into natural language processing and understanding, a predominant field nowadays if we consider the amount of textual information being shared online on a daily basis. The subject of charisma is an interesting one to learn about, even though it can be frightening to imagine what the effects of a charisma detection system could do if deployed on a wide scale. It is better to view it as a tool that could help people by improving their confidence, teaching them how to express their ideas clearly and inspiring others.

On a technological level, it was a great opportunity to learn about state-of-the-art machine learning methods and the various applications of attention. It has been an arduous task, having no prior experience regarding machine learning, but a very satisfying one overall.

This would not have been made possible without the support and guidance of my supervisors at the Idiap Research Institute and my office colleagues who took time to understand and explain difficult concepts, as well as share their ideas with me.

Martigny, 15 June 2020 D. C.

Abstract

The study of charisma in the field of leadership studies has been steadily growing after it was shown to be a good predictor of success in the workplace and political setting. In order to study what charisma is, one must be able to define character traits that can be observed and quantified. In this paper, we will take a look at elements of speech that are used to measure how charismatic a speaker is. Modern studies have defined a total of nine Charismatic Leadership Tactics with metaphors, lists or contrasts being a few of them. Until now, these elements of speech were marked by hand by trained human annotators from transcribed speeches they wished to analyze. This is a long and costly process that takes a toll on the person executing it and is prone to bias.

Here, we propose some methods using machine learning to perform the classification of text into these different classes. We explore state-of-the-art models and analyze their efficiency in performing this classification task. We begin by building a simple classifier with recurrent networks and linear projections which performs some amount of classification but can be largely improved on. We then introduce the mechanism of attention which greatly improves the performance by being able to focus on relevant words in the input sentence. Taking advantage of attention by increasing the number of heads looking at the sentences doesn't have a noticeable effect on the results. Lastly, we use transformers which are said to attain state-of-the-art performance in text processing tasks and see that a BERT model slightly improves on our simple attention model.

Contents

Preface i

Abstract iii

List of Figures vii

List of Tables ix

List of abbreviations xi

1 Introduction 1
- 1.1 Original setting . . . 2
- 1.2 Contributions . . . 2

2 Background 5
- 2.1 Overview . . . 5
  - 2.1.1 Word embedding . . . 5
  - 2.1.2 Neural Network . . . 5
  - 2.1.3 Classifier . . . 5
  - 2.1.4 Evaluation . . . 6
- 2.2 Implementation . . . 6
  - 2.2.1 Training the word embedding model . . . 6
  - 2.2.2 Creating the dictionary . . . 6
  - 2.2.3 Training the deep charisma model . . . 7
  - 2.2.4 Evaluating the deep charisma model . . . 7

3 Dataset 9

4 Baseline model 11

5 Original model 17

6 Original model modified word2vec embeddings 23

7 Original model with fasttext embeddings 25

8 Pytorch attention 27

9 Multihead attention **31**

10 Multihead attention with duplicated input **33**

11 Multihead attention with linearly increased input **35**

12 BERT model **37**

13 Conclusion **41**

A Appendix **43**

Bibliography **81**

Curriculum Vitae

List of Figures

2.1 Representation of the inputs and outputs of `train_word_embedding.py` . . . 7
2.2 Representation of the inputs and outputs of `compute_dictionary.py` 7
2.3 Representation of the inputs and outputs of `train_deep_charisma.py` 8
2.4 Representation of the inputs and outputs of `check_model.py` 8

4.1 Diagram of the baseline model 13

5.1 Basic attention model 19
5.2 Plot of the attention scores for the list class with punctuation 21
5.3 Plot of the attention scores for the list class on a sentence with no label, with punctuation 21

8.1 Multihead attention model 28

12.1 BERT model 39

A.1 Complementary information about dataset labelling 45
A.2 Results of the baseline model with punctuation 46
A.3 Results of the baseline model without punctuation 47
A.4 Results of the original attention model with punctuation 48
A.5 Results of the original attention model without punctuation 49
A.6 Results of the attention model with better embeddings and with punctuation . 50
A.7 Results of the attention model with better embeddings and without punctuation 51
A.8 Results of the attention model using fasttext embeddings with punctuation . . 52
A.9 Results of the attention model using fasttext embeddings without punctuation 53
A.10 Results of the multihead attention model using a single head with punctuation 54
A.11 Results of the multihead attention model using a single head without punctuation 55
A.12 Results of the multihead attention model for joint training using a single head with punctuation 56
A.13 Results of the multihead attention model for joint training using a single head without punctuation 57
A.14 Results of the multihead attention model using 2 heads with punctuation . . . 58
A.15 Results of the multihead attention model using 2 heads without punctuation . 59
A.16 Results of the multihead attention model using 4 heads with punctuation . . . 60

A.17 Results of the multihead attention model using 4 heads without punctuation . 61
A.18 Results of the multihead attention model using 8 heads with punctuation . . . 62
A.19 Results of the multihead attention model using 8 heads without punctuation . 63
A.20 Results of the multihead attention model with duplicated input using 2 heads with punctuation 64
A.21 Results of the multihead attention model with duplicated input using 3 heads with punctuation 65
A.22 Results of the multihead attention model with duplicated input using 4 heads with punctuation 66
A.23 Results of the multihead attention model with duplicated input using 5 heads with punctuation 67
A.24 Results of the multihead attention model with duplicated input using 6 heads with punctuation 68
A.25 Results of the multihead attention model with duplicated input using 7 heads with punctuation 69
A.26 Results of the multihead attention model with duplicated input using 8 heads with punctuation 70
A.27 Results of the multihead attention model with linearly projected input using 2 heads with punctuation 71
A.28 Results of the multihead attention model with linearly projected input using 3 heads with punctuation 72
A.29 Results of the multihead attention model with linearly projected input using 4 heads with punctuation 73
A.30 Results of the multihead attention model with linearly projected input using 5 heads with punctuation 74
A.31 Results of the multihead attention model with linearly projected input using 6 heads with punctuation 75
A.32 Results of the multihead attention model with linearly projected input using 7 heads with punctuation 76
A.33 Results of the multihead attention model with linearly projected input using 8 heads with punctuation 77
A.34 Results of the BERT model with punctuation 78
A.35 Results of the BERT model without punctuation 79

List of Tables

3.1 Examples of dataset sample, showing only the first columns 10
3.2 Correspondence of class number to class name and sample count 10

4.1 Results of the baseline model with punctuation 14
4.2 Comparison of the results of the baseline model with and without punctuation 15

5.1 Comparison of the baseline and attention model with punctuation 20
5.2 Comparison of the baseline model and the attention model with/without punctuation 20

6.1 Comparison of the original embeddings and the new embeddings with/without punctuation 24

7.1 Comparison of the word2vec and fasttext embeddings with/without punctuation 26

8.1 Comparison of the custom attention and pytorch attention with/without punctuation 29
8.2 Comparison of the individual training and joint training with/without punctuation 30

9.1 Results of the Pytorch attention with different number of heads with/without punctuation 32

10.1 Results of the Pytorch attention with duplicated inputs of the attention with punctuation 34

11.1 Results of the Pytorch attention with linearly projected inputs of the attention with punctuation 36

12.1 Comparison of the single head attention results and the BERT results with/without punctuation 38

A.1 List of options in the config.ini file 44

List of abbreviations

Abbreviation	Meaning
BERT	Bidirectional Encoder Representations from Transformers
CBOW	Continuous Bag Of Words
CLT	Charismatic Leadership Tactic
GRU	Gated Recurrent Unit
LSTM	Long Short-Term Memory
NLP	Natural Language Processing
RNN	Recurrent Neural Network
ROC	Receiver Operating Characteristic

1 Introduction

In order to understand what charisma is and the impact it can have on our lives, we can think of the outcome of political elections. In particular, we can take the case of the 2012 US presidential election. At the time, an economic model augmented using charisma was able to determine the victory of Barack Obama against Mitt Romney [1]. The same study has also shown that a person with high charisma is more likely to attain key leadership positions and affect the performance of the institutions they belong to. It is therefore a good predictor of success.

Charisma is not a straightforward quantity to measure. Until recently, charisma has mostly been evaluated using subjective and imprecise metrics, based on outcomes or antecedents rather than being independent from its effects [2]. Newer measures of charisma are more precise and define it as a values-based, symbolic, and emotion-laden leader signaling [3; 4; 5]. This consists in analyzing methods of symbolic communication (e.g., use of metaphors, anecdotes, contrasts, lists etc.) as well as the substance of a vision (e.g., collective feelings, values defended, goals communicated etc.). These markers are referred to as charismatic leadership tactics (CLTs). Despite the fact that scientific literature made great progress in that regard, it is difficult to execute large practical studies on the subject due to the way data is collected. Annotating text with these labels is traditionally done by hand, which is time consuming, costly and can suffer from cognitive bias on the annotator's part. To address these issues, we wish to develop a system capable of exacting these metrics automatically.

Approaching the problem from a machine learning perspective, this can be viewed as a text classification task, where the input sentence can belong to none, one or multiple charismatic tactics. Using the Bayesian approach, a conditional probability can be assigned to a class membership depending on the sequence of words considered. This answers the question "What is the probability that this sequence belongs to this class?"

In the field of natural language processing and speech processing, language modelling is usually performed using word embedding techniques and recurrent neural networks (RNNs) [6; 7]. The goal of word embedding is to map a continuous sequence into a high dimensional

continuous vector space. An RNN model is able to learn long range dependencies in that vector space, much better than traditional n-gram models. More recently, RNNs are being replaced by transformers, which are stacks of encoder-decoder layers using attention mechanisms [8]. This attention is good at focusing on the relevant parts of the input sequence for each element in the output.

State-of-the-art text classification relies mostly on the encoder of transformers in order to express the input sentence as a single state vector. This in turn is used as the input of a standard classifier such as a feed-forward network.

In this thesis, we test different models that can be used to classify sentences into the different CLTs as described by Antonakis et al [1; 3]. The following sections give an overview of the methods used to classify sentences and the dataset used. In chapter 4 and onward, each model is then detailed and the results shown and compared. Finally, we conclude by exploring the possibilities for future work in the field of classification of charisma.

1.1 Original setting

Thankfully, it was not necessary to build the system from the ground up as first system used to classify sentences into their corresponding CLTs has been provided at the start of the project [9] and a live demo showing its effectiveness has been shown at the Swisstext 2019 event. This was used as a base onto which improvements could be made. Results of a baseline model were also provided but not the model itself. We decided to create our own baseline model and confirm the results, as well as those of the provided model to ensure the received data was correct and to understand in depth how the system works.

1.2 Contributions

In the course of this project, numerous models have been implemented and tested to validate our hypotheses. First, that the baseline model composed of a simple recurrent network followed by a feedforward layer is able to perform some amount of classification of the sentences into their corresponding CLTs. We observed the influence of punctuation on the performance of the models. We then added an attention layer and show that it improves the performance by focusing on relevant words for the class being tested. We tried using multihead attention in order to improve the results by having multiple attention heads extract more information from the sentences and see that it has no significant effect. Finally, we replaced our encoder by a BERT model followed by an attention layer and observe a slight improvement in the results.

Various additions and fixes have been made to the original implementation, which are enumerated here:

- Verification and modification of the word embedding process

- Addition of the Fasttext word embedding technique
- Addition of the multihead attention technique
- Possibility to perform a joint training of the classes
- Modifications to the files used to train the system on the computation grid
- Addition of the BERT model

2 Background

The aim of our system is to classify sentences into corresponding CLT classes based on their content and meaning. A sequence of steps must be devised in order to achieve that goal. We give here a general overview of the process which will be detailed in chapter 4, along with the different model implementations.

2.1 Overview

2.1.1 Word embedding

Our model will need numerical values as input in order to perform operations on them, compute the loss and output results after being trained. As the dataset is composed of sentences in text form, they must first be transformed into numbers. This process is called word embedding and consists in transforming a textual representation of a word into a high dimensional vector containing numbers by using a language model.

2.1.2 Neural Network

The word embeddings go through a neural network that will try to learn how the sentences relate to each class. Recurrent units are used as they are efficient at accumulating information across sequences. Using a loss function that computes how similar the output of the model and the reference labels from the dataset are from each other, the model will tune its parameters to obtain better results after each iteration.

2.1.3 Classifier

Finally, the output of the model must be transformed to indicate whether the input sentence corresponds to a particular class or not. A classifier composed of a feedforward layer trained and a sigmoid function is used, that will output a probability for the sentence to belong to a

class. It is possible afterwards to threshold those values depending on the accuracy wanted from the system.

2.1.4 Evaluation

Evaluation metrics are necessary to know how well the model performs. We have chosen to use the receiver operating characteristic (ROC) curves, which are drawn using true and false positive rates as they can be computed easily from our outputs and allow to produce graphs that indicate whether the model is good or bad at first glance. An ideal curve representing perfect classification would follow the left and top axis, whereas a random classification would look like a diagonal curve. We compute the area under the ROC curve to have a numerical value corresponding to our results, going from 0 to 1.

2.2 Implementation

A modular approach was adopted so that different parts of the system can easily be replaced and improved without the need of changing others. For example, it is easy to change how the data is loaded from the dataset or the design of the the model by replacing the corresponding files without the need to modify the number of function parameters or return values in other modules. This also allows for quick and easy parameterization of the model from a configuration file. Appendix A.1 details the available configuration options.

2.2.1 Training the word embedding model

`train_word_embedding.py` takes care of training a word embedding model, generating embeddings and saving them to a file that can be loaded back later. As the dataset is stored into a .csv file, an iterator over that file is created to fetch the sentences and tokenize them. Then, a pretrained word model can be loaded if one is available and has been specified. Using a model that has been pretrained on a big dataset has the advantage of being more accurate and faster to converge than simply training the model on our relatively small dataset. This results in a dictionary of words and their corresponding vectors which is saved to be reused in the other modules.

2.2.2 Creating the dictionary

`compute_dictionary.py` creates a structure that adds complementary information on the words found in the dictionary. It starts by loading the embeddings generated previously. Then, using an iterator over the datasets, it checks that all the words present in our sentences have a corresponding embedding. As the embeddings have been generated using a pretrained model and our training set it is possible that new words not seen previously are encountered, for example in our test set or real-life situations. If that is the case, the vector is initialized

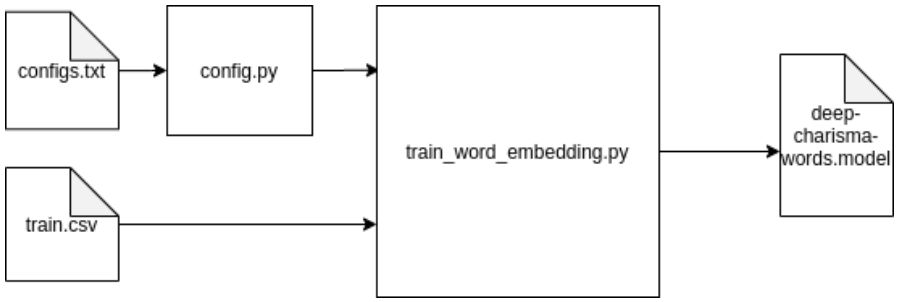

Figure 2.1 – Representation of the inputs and outputs of `train_word_embedding.py`

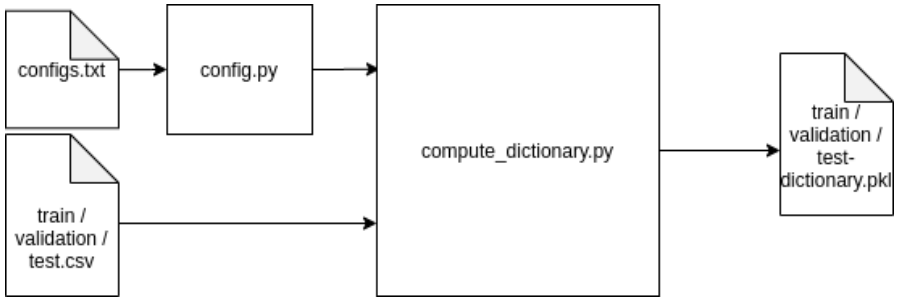

Figure 2.2 – Representation of the inputs and outputs of `compute_dictionary.py`

randomly. Each word is assigned an index and has a count initialized to zero. The dictionary is saved to be used later.

2.2.3 Training the deep charisma model

`train_deep_charisma.py` constitutes the main part of our system. For each sentence in our dataset, we create a new dictionary containing the talk index, sentence index, the words of the sentence, the corresponding embeddings and the target labels. The following process is then applied for each epoch. The model is set in training mode and batches of embeddings are given to it. Prediction scores are given in return, which indicate the probability for a sentence to belong to a class. A loss function computes how similar these results are to the target labels and the weights of the model are adjusted based on that loss. Once all training sentences were given to the model, it is changed into evaluation mode and the process is repeated using the evaluation set so that we can finetune the hyperparameters. The total loss is compared to the one of the previous epoch and if it is lower, the model is saved, effectively keeping the best model generated.

2.2.4 Evaluating the deep charisma model

`check_model.py` is the last step in the pipeline and allows us to see how the model handles unseen data. The embeddings from the test set are fed through the model while in evaluation mode, much like in the evaluation process but for a single epoch. Applying thresholds on the

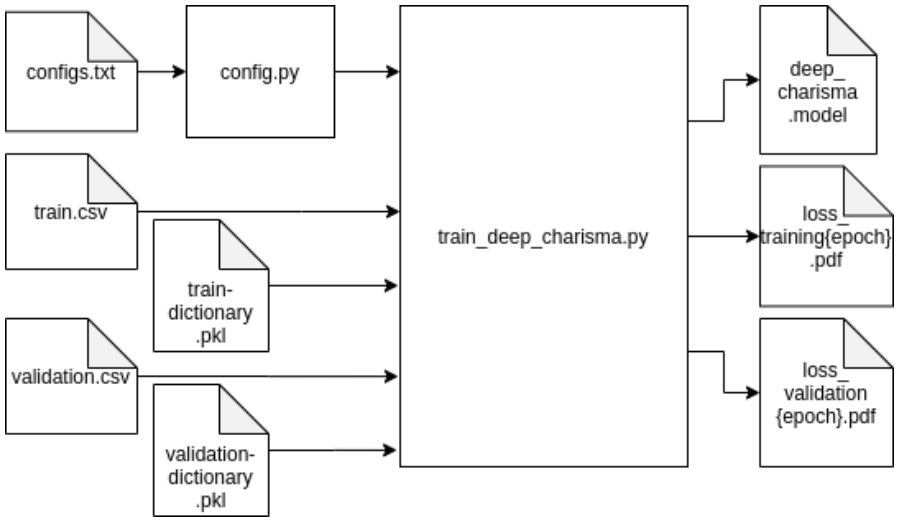

Figure 2.3 – Representation of the inputs and outputs of `train_deep_charisma.py`

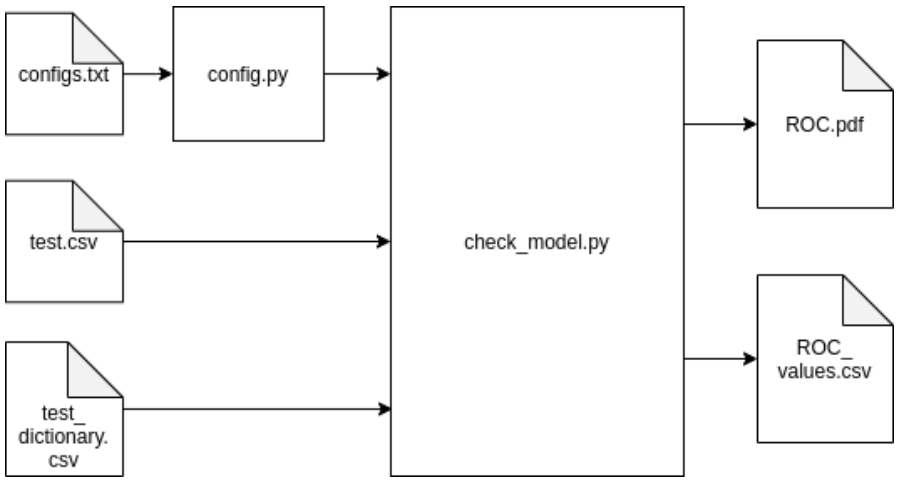

Figure 2.4 – Representation of the inputs and outputs of `check_model.py`

results control whether we consider the result to belong to the target class or not. Comparing those thresholded results to the target scores allows us to compute true/false positive and true/false negative rates, which are used to plot ROC curves and other useful metrics such as the F1 score, precision and recall. The area under the ROC curve is used to evaluate how well the model performs. Other visualization tools permit to combine the results for multiple classes into a single file, average the results obtained across several trainings of the model in order to show how reliable these are, as the neural network has a small element of randomness to it. We can also plot the effect of the attention on the sentences.

3 Dataset

Our dataset is composed of a random sample of 240 TED talk [1] transcriptions, with a total of 31600 sentences. All the talks were given in English, 130 of which were delivered in the United States while the rest span across 18 countries. The sentences were labeled by three human annotators that have been trained to recognize CLTs in speech.The talks have been randomly split into train, validation and test sets of size 19446, 6311 and 5843 respectively.

Nine CLTs have been defined, which can be divided in two categories:

-Framing and creating a vision:

- **Metaphors**: simplify the message, trigger an image, make it easy to remember
- **Rhetorical questions**: create an intrigue and interest in knowing the answer
- **Stories** and anecdotes: trigger an image, create identification with the protagonists, distill a message into a moral
- **Contrasts**: define the vision in terms of what it should or should not be and focus attention on the mes- sage
- **Lists**: provide “proof”, focus attention, and show completeness

-Engendering substantive statements:

- **Moral conviction**: makes value systems clear and provides moral justification for the mission
- **Sentiments of the collective**: putting into words and images what followers feel; shows the similarities of the leader with the followers and closes the psychological distance of the leader with the followers

[1] https://www.ted.com/

- **Expectations**: shows leader ambition, aligns efforts towards the achievement of goals
- **Confidence**: raises self-efficacy belief, a key determinant of performance

Each annotator labeled the sentences with either 1 or 0 for each leadership tactic, indicating whether the sentence belongs to a tactic or not. We consider that a sentence belongs to a class if at least one of the three annotators defined it as such. Table 3.1 shows a few samples from our training set.

talk_id	id_sentence	sentence	R1_all	R2_all	R3_all	R1_collective	R2_collective	R3_collective	R1_contrast
1	107	What's this?	1	1	1	0	0	0	0
1	108	This is not a transport[...]."	1	1	1	0	0	0	1
1	109	And I used to be bothered by [...]	0	0	0	0	0	0	0

Table 3.1 – Examples of dataset sample, showing only the first columns

The additional "all" columns indicate the number of labels each annotator defined per sentence.

In this document, results often refer to classes by a number, which corresponds to the order by which they appear in the dataset. Table 3.2 shows to which classes these numbers correspond to, as well as the number of samples per class in the training set:

Class number	Class name	Count
0	Collective	547
1	Contrast	1976
2	Goal (Expectations)	395
3	Goals2 (Confidence)	509
4	List	4208
5	Metaphor	1933
6	Moral	615
7	Question	2502
8	Story	1884

Table 3.2 – Correspondence of class number to class name and sample count

We notice that the collective, Goal, Goals2 and Moral classes are under-represented compared to the other ones. It is possible this will have an impact on the results as the model might not have enough samples to learn from.

Appendix A.1 gives some more information about how the sentences have been labelled by the annotators.

4 Baseline model

First of all, we need to create a baseline model that represents what can be done with a typical neural network. Our expectation with this model is not to obtain perfect accuracy but to have a preliminary set of results that can be compared to our future improved models. It is also a good way to check that all parts of the system work as intended. Namely, that the datasets are loaded properly, the words are converted to embeddings, the model is able to train and output results. We can also see if the data visualization tools are suited for the task and give us meaningful information.

Architecture

We are first using a word embedding layer to transform the words of our sentences into a high dimensional vector representation. This step is necessary as we need some kind of numerical representation of our input to process.

We are first using Google's word2vec[1] [10; 11] to obtain embeddings. To understand how word2vec works, two concepts must be explained.

A language model is a probabilistic representation over a sequence of words. In a grammatically correct and meaningful sentence, it can be assumed that the probability of a word to appear depends one the ones that are present before it in the sequence. Let's take as an example the sentence "The dog was running in the _". It is clear that the missing word depends on the content of the preceding words "dog" and "running". This restricts which word is likely to appear next. A dog running in a park or a garden makes sense, whereas "sky" or "tree" has a very low probability to appear. These probabilities can be learned by analyzing an extensive collection of texts. In an n-gram model, the probability of a word depends on the n words that precede it:

$$P(x_i|x_{i-(n-1)},...,x_{i-1}) \tag{4.1}$$

[1] https://code.google.com/archive/p/word2vec/

with x_i being an input word.

Word2vec has different ways of creating a language model. In the continuous bag of words (CBOW) representation, the surrounding words are used to predict the missing word. In the continuous skip-gram representation, it is the missing word that is used to predict the surrounding words.

The second concept is that of negative sampling. Instead of trying to predict the missing word using probabilities, we take the words and estimate whether they are neighbours or not. This effectively turns the problem into a regression model, which is much faster.

The embeddings are first initialized randomly and are then trained using the language model and negative sampling. As a result, embeddings with a close meaning align themselves in similar dimensional spaces.

To be more efficient in the way we generate our embeddings, we augment them by using a pretrained model that used the Google news database as a corpus. This allows for more coherent word vectors and reduces the risk of encountering out-of-vocabulary words.

We use the word embeddings as inputs of a recurrent neural network. RNNs are good at modelling sequential data. They differ from regular feed-forward networks by having a hidden state that keeps some information about the previous state of the network. This means that each state in the output sequence has some knowledge of the entirety of the previous information, or in our case, the previous words in the sentence. In practice, a recurrent network tends to forget information the longer the sequence is. This is the vanishing gradient or short memory problem which is inherent to how back-propagation works. One way to solve this issue is to add gates represented by tanh and sigmoid functions that regulate the flow of information. Depending on the implementation, these are called LSTM [12] or GRU networks.

The recurrent network acts as an encoder that compresses the sentence into a sequence of states representing the words by accumulating information as it "reads" the sentence. This means that the hidden states will have some knowledge of the words that precede them in the input sentence. That can be further enhanced by using bidirectional RNNs [13], that will also read the sentence from the end and give some knowledge of what words follow the current state. An output state of the bidirectional RNN is then the concatenation of the forward and backward information.

The forward and backward states of the RNN then go through a classifier composed of a fully connected layer and a sigmoid to obtain probabilities as output. The classes are trained independently from each other, resulting in 9 different models. When loading the labels from the dataset we consider the current class against the others, which is a binary classification task. Therefore, the performance of the model would have to be greater than a random classification of 50% in order to be useful.

Figure 4.1 shows a high level representation of the baseline model.

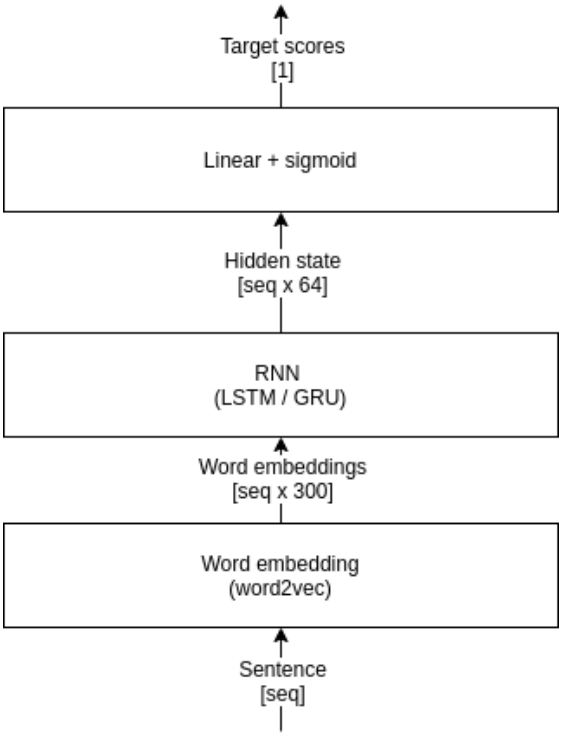

Figure 4.1 – Diagram of the baseline model

For our experiments, we use the word2vec implementation from the gensim library[2] with the GoogleNews-vectors-negative300 pretrained model. The pytorch library[3] is used for the other parts of our model. Our recurrent network is a three layers deep, bidirectional LSTM, although a GRU can also be used and performs just as well. Hidden dimensions of the input correspond to those of the word embeddings, 300, and we define 64 dimensions as output, which represent 32 forward and backward states concatenated together. We train the model on batches of 10 samples, randomly selected from the training set. The training is done for a duration of 50 epochs but only the epoch with the best results during evaluation is considered when saving the model. Binary cross entropy is used to compute the loss and we chose Adam with a dropout of 0.43 as the optimizer. These hyperparameters were chosen using grid search, providing the best results. The same hyperparameters ared in the subsequent experiments except if stated otherwise.

Each experiment is run ten times and the results averaged as there can be some variations in the classification outputs used to compute our evaluation metrics.

[2]https://radimrehurek.com/gensim/
[3]https://pytorch.org/

Baseline results

Table 4.1 shows the results of the baseline model. It can be observed that simply using a encoder of type LSTM and a feed-forward network as classifier is able to reach a rather good performance. As can be expected, performance is class dependant. The best classified class is the questions class. As the punctuation has been kept in the input of our system, we are confident that the recurrent network is looking at the interrogation mark at the end of sentences to help with the classification as that is the common element between them and finds itself always at the same position. Furthermore, RNNs are better at keeping information located at the end of sentences. Other punctuation such as commas might also help in the classification, mainly of the list class.

Class number	Class name	Area under ROC
0	Collective	0.76
1	Contrast	0.80
2	Goal	0.77
3	Goals2	0.73
4	List	0.76
5	Metaphor	0.70
6	Moral	0.81
7	Question	0.99
8	Story	0.85

Table 4.1 – Results of the baseline model with punctuation

We are confident that punctuation such as question marks and commas have an impact on the performance of the model. This begs the question, what is the performance on sentences that have been stripped of their punctuation? This point is important and comes into play if the system were to be used in real life deployment. In our current experimental setting, the text has carefully been transcribed. This might not be the case in real life. Transcription is an arduous and time consuming task that can be subject to interpretation. Therefore, punctuation can be omitted either by accident or personal preference. On top of that, we could imagine a situation where the input text has not been transcribed by a person but by a speech-to-text system, which may or may not be reliable in regards to punctuation. That is why conducting a second experiment using models trained on sentences stripped of their punctuation might give another insight into the model's performance, such as how much lexical content is actually inferred.

Most of the results obtained in table 4.2 show only slight variations in performance, aside from the list and questions class. This confirms our assumption that the recurrent network is relying heavily on commas and question marks for these two classes, but also indicates that it is able to learn some other information related to the meaning of the sentences.

Class number	Class name	Punctuation	No punctuation
0	Collective	0.76	0.77
1	Contrast	0.80	0.79
2	Goal	0.77	0.75
3	Goals2	0.73	0.71
4	List	0.76	0.70
5	Metaphor	0.70	0.68
6	Moral	0.81	0.82
7	Question	0.99	0.88
8	Story	0.85	0.83

Table 4.2 – Comparison of the results of the baseline model with and without punctuation

5 Original model

We now wish to find methods that would be able to improve the results of the baseline model. Specifically, we wish to implement an attention mechanism and we hypothesize it will improve the results, being able to focus more strongly on particular elements of the sentences that are proper to each class.

Architecture

This first attention model has been provided at the start of the project. It is the same as the baseline model but with an attention layer of the type described in [14] added between the recurrent network and the classifier.

In the case of language processing, attention is a mechanism that indicates, for each word in the sentence, which other words are related to it. As an example, let's take the following sentence:

The animal didn't cross the street because it was tired.

It is clear to us that "it" refers to the animal because a street, in the literal sense, cannot be tired. Therefore, when evaluating the word "it", a great importance is put on "animal", or in other words we pay more attention to it.

We can compare this to how a search engine works. There is some data (the input of our model) from which we want to extract samples that are relevant to our question (query). We can ask "What is a metaphor?" and the search engine orders the results of the search so that the first couple of samples returned answer our question.

This is a simple concept to understand, but the issue is that we don't know how to ask a question directly on embeddings, as they are a series of numbers that have no meaning to us. The trick is to make the network learn what a metaphor is by showing it the results we expect, the target labels. The query is therefore initialized randomly using a uniform distribution and trained with the other model parameters in the hopes it will be able to formulate the question

we don't know how to ask.

The attention works as a decoder on the encoded outputs of the recurrent network.

The decoder produces a sequence of outputs defined by

$$P(y_i|y_1,...,y_{i-1},\mathbf{x}) = g(y_{t-1},s_t,c) \tag{5.1}$$

where y_{t-1} is the output of the attention for the previous time step and s_i is a hidden state of the decoder. c is called the context vector and is a weighted sum of the outputs of the encoder.

$$c = \sum_{j=1}^{T_x} \alpha_{ij} h_j \tag{5.2}$$

with α being the weight matrix. This weight matrix is defined by a softmax function

$$\alpha_{ij} = \frac{\exp(e_{ij})}{\sum_{k=1}^{T_x} \exp(e_{ik})} \tag{5.3}$$

e defines the alignment model which indicates the relevant words in the input sentence for each output state s:

$$e_{ij} = a(s_{i-1},h_j) = v_a^T \tanh(W_a s_{i-1} + U_a h_j) \tag{5.4}$$

The alignment model with weight matrices v_a, W_a and U_a is represented by a feed-forward network trained with the rest of the model.

The goal of our system is to know the relevant words to a particular class. Therefore, the output of our the attention is not a sequence of states s of equal length as the input sentence but a single vector representation.

The attention will output a vector of high dimension for each sentence, which corresponds to the class we are training on. To get the probability for that sentence to belong to a class, we train a linear classifier to reduce the number of dimensions to a single value, then apply a sigmoid function.

Figure 5.1 shows a diagram of what this original model looks like.

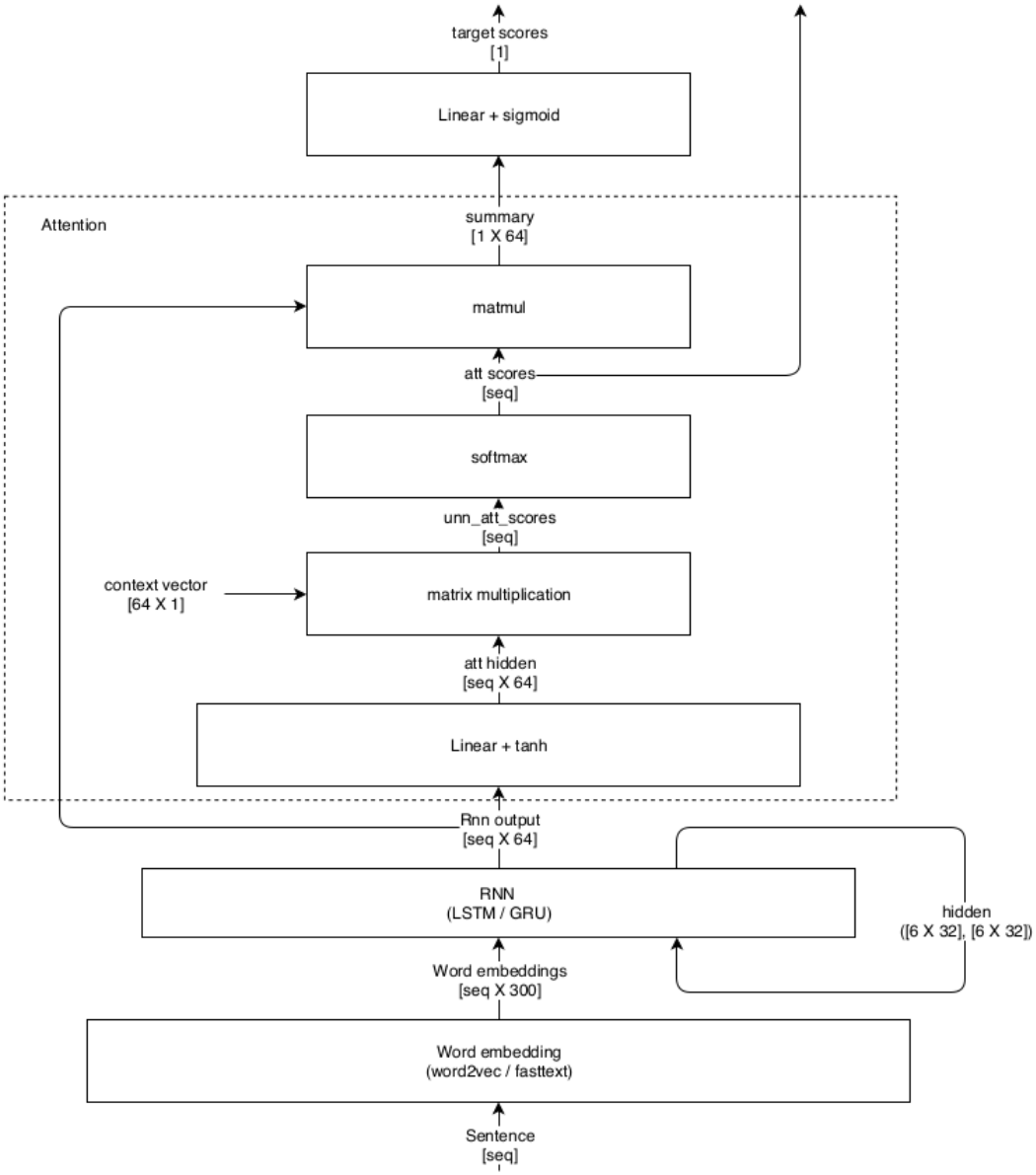

Figure 5.1 – Basic attention model

Results

The results obtained with a simple attention model confirm our hypothesis that it is able to generally improve the performance. Table 5.1 compares the results of our attention model with those of the baseline.

Class number	Class name	Baseline	Attention
0	Collective	0.76	0.86
1	Contrast	0.80	0.86
2	Goal	0.77	0.84
3	Goals2	0.73	0.84
4	List	0.76	0.80
5	Metaphor	0.70	0.75
6	Moral	0.81	0.86
7	Question	0.99	0.99
8	Story	0.85	0.89

Table 5.1 – Comparison of the baseline and attention model with punctuation

On average the area under the ROC improves by 0.04 to 0.11 depending on the class.

We train the model again on sentences that have had their punctuation removed to see how the attention handles this situation.

Class number	Class name	Baseline	Attention
0	Collective	0.76/0.77	0.86/0.87
1	Contrast	0.80/0.79	0.86/0.86
2	Goal	0.77/0.75	0.84/0.85
3	Goals2	0.73/0.71	0.84/0.85
4	List	0.76/0.70	0.80/0.71
5	Metaphor	0.70/0.68	0.75/0.74
6	Moral	0.81/0.82	0.86/0.88
7	Question	0.99/0.88	0.99/0.87
8	Story	0.85/0.83	0.89/0.89

Table 5.2 – Comparison of the baseline model and the attention model with/without punctuation

To have a better idea of how the attention is able to improve the results by focusing on relevant words, we can plot the attention weights corresponding to a few sample sentences. Figure 5.2 demonstrates how the attention of the list class can focus on parts of the sentence containing a repetition of commas and gives them more weight.

In sentences that aren't lists such as in figure 5.3, the attention still focuses on punctuation and joining words. This could explain why the accuracy of the list class is lower than we expect

it as sentences could be misclassified.

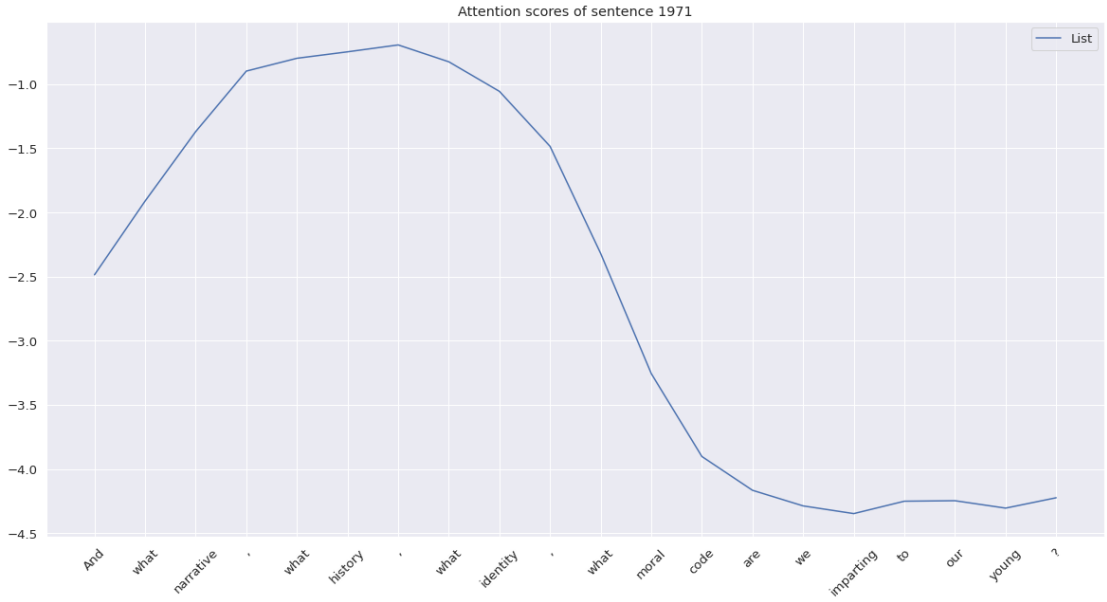

Figure 5.2 – Plot of the attention scores for the list class with punctuation

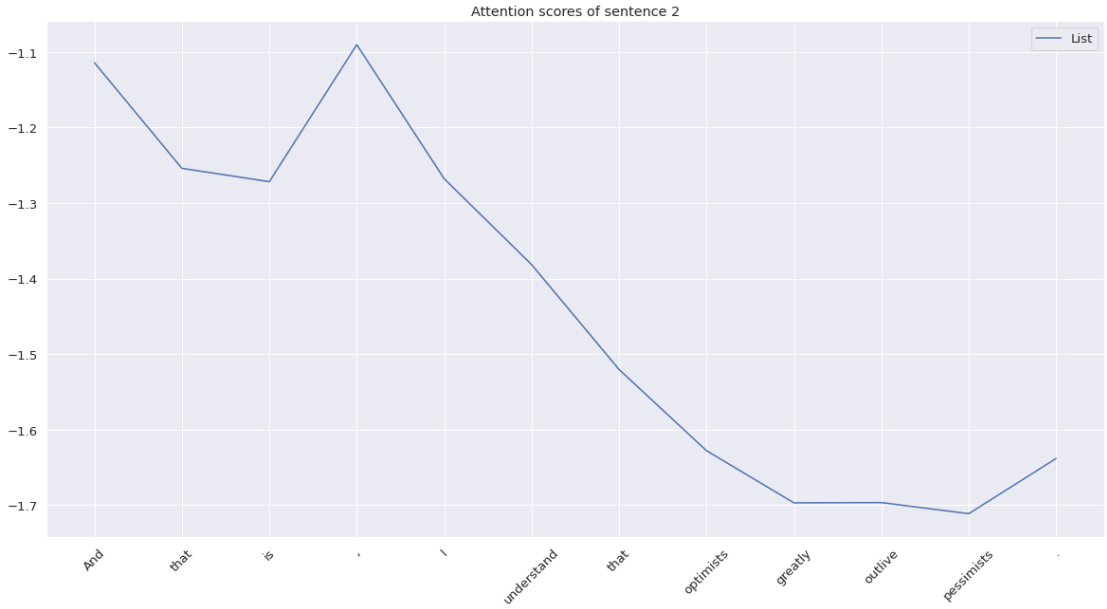

Figure 5.3 – Plot of the attention scores for the list class on a sentence with no label, with punctuation

6 Original model modified word2vec embeddings

One concern with the provided model was that the embeddings might not have been generated properly before going into the recurrent network. This was a great concern, as it might result in classifications that are entirely erroneous. It was therefore decided that the code concerning the generation of embedding would have to be looked at first.

Architecture

The word embedding process allows for a pretrained file to be loaded. We also need to add the words present in our training set to the pretrained ones. The original way this process was done is the following:

1. Train the word embedding model on the words from our training set
2. Load and add the pretrained embeddings, replacing the ones that are also present in our training dataset.

This is done under the assumption that a pretrained model has vectors that are better than the ones trained on our small set, so replacing our custom vectors with pretrained ones should be more accurate. However, the process described has major flaw. By training the model before loading the pretrained vectors, there is no guarantee that both sets of vectors are aligned in the vector space.

To remedy that issue, the process has been changed to perform the following:

1. Load the words from our training set
2. Load and add the pretrained vectors
3. Train the word embedding model

This way, the training should align both sets of vectors in a manner that makes sense in the vector space.

Results

It appears that this more correct way of computing word embeddings doesn't have any real effect on the results. This could be explained by the fact that the embeddings are mostly taken from the pretrained model and very few are newly encountered in our dataset. In any case, this method makes more sense and will be used from now on.

Class number	Class name	Original embeddings	New embeddings
0	Collective	0.86/0.87	0.86/0.84
1	Contrast	0.86/0.86	0.86/0.85
2	Goal	0.84/0.85	0.84/0.84
3	Goals2	0.84/0.85	0.78/0.79
4	List	0.80/0.71	0.79/0.73
5	Metaphor	0.75/0.74	0.76/0.76
6	Moral	0.86/0.88	0.87/0.87
7	Question	0.99/0.87	0.99/0.92
8	Story	0.89/0.89	0.88/0.89

Table 6.1 – Comparison of the original embeddings and the new embeddings with/without punctuation

7 Original model with fasttext embeddings

Originally, the system only supported word2vec to generate word embeddings. It produced acceptable results but other, more recent types of word embedding techniques might change the way our network behaves and produce better results. It was decided to add the fasttext method[1], developed by facebook, to the list of available embedding techniques. Instead of looking at whole words, fasttext reads sub-words, or character n-grams. For example, the word "apple" using character 3-grams would be represented as <ap, app, ppl, ple, le>. This has the advantage of being able to better handle rare and out-of-vocabulary words than word2vec is.

Architecture

The architecture is the same as in the previous section, only with the word embedding layer using fasttext instead of word2vec, still from the gensim library.

As for the pretrained embeddings, we are now using a model trained on the English wikipedia[2]. The disadvantage is that this model is significantly larger and slower to load than our previous Google pretrained model.

Results

Fasttext results in a slight increase of the classification performance in most classes, but at the cost of slower training of the word embeddings and bigger file sizes. We decide to keep using word2vec embeddings for the following experiments as the results of fasttext are not enough to make up for the disadvantages.

[1] https://fasttext.cc/
[2] https://fasttext.cc/docs/en/pretrained-vectors.html

Class number	Class name	Word2vec embeddings	Fasttext embeddings
0	Collective	0.86/0.84	0.85/0.89
1	Contrast	0.86/0.85	0.87/0.87
2	Goal	0.84/0.84	0.88/0.85
3	Goals2	0.78/0.79	0.85/0.84
4	List	0.79/0.73	0.80/0.75
5	Metaphor	0.76/0.76	0.78/0.79
6	Moral	0.87/0.87	0.89/0.88
7	Question	0.99/0.92	0.99/0.92
8	Story	0.88/0.89	0.90/0.90

Table 7.1 – Comparison of the word2vec and fasttext embeddings with/without punctuation

8 Pytorch attention

At this point in our experiments, we noticed that we could modify the original code provided to make extensive use of the pytorch library and be more efficient in how the model is trained.

Architecture

One of the biggest change in our code is the implementation of the attention layer. The attention layer in the provided attention model from the beginning of the project was built from the ground up and not using a dedicated library. This cause a great deal of confusion as the comments were not very descriptive, the variable names not respecting the attention paper notations and some tricks were necessary to obtain the correct matrix shapes. Since that first model has been created, an implementation of the attention mechanism has been added in Pytorch. In an effort to have reliable, efficient and clean code, it has been decided to replace the custom attention layer by the Pytorch implementation. Pytorch doesn't currently contain a simple attention layer, but provides a multihead attention class. Multihead attention is a technique used to try and learn higher-level information from the input sentences by increasing the number of attentions looking at it. In this model, each attention applied is called a head. It works by taking the inputs of the attention attention, meaning the output of the LSTM and the query, projecting them linearly n times into different vector spaces, then applying attention on these n different heads. Finally, the output of each head is concatenated into a single matrix and linearly projected to the desired output shape.

Figure 8.1 shows how multihead attention works.

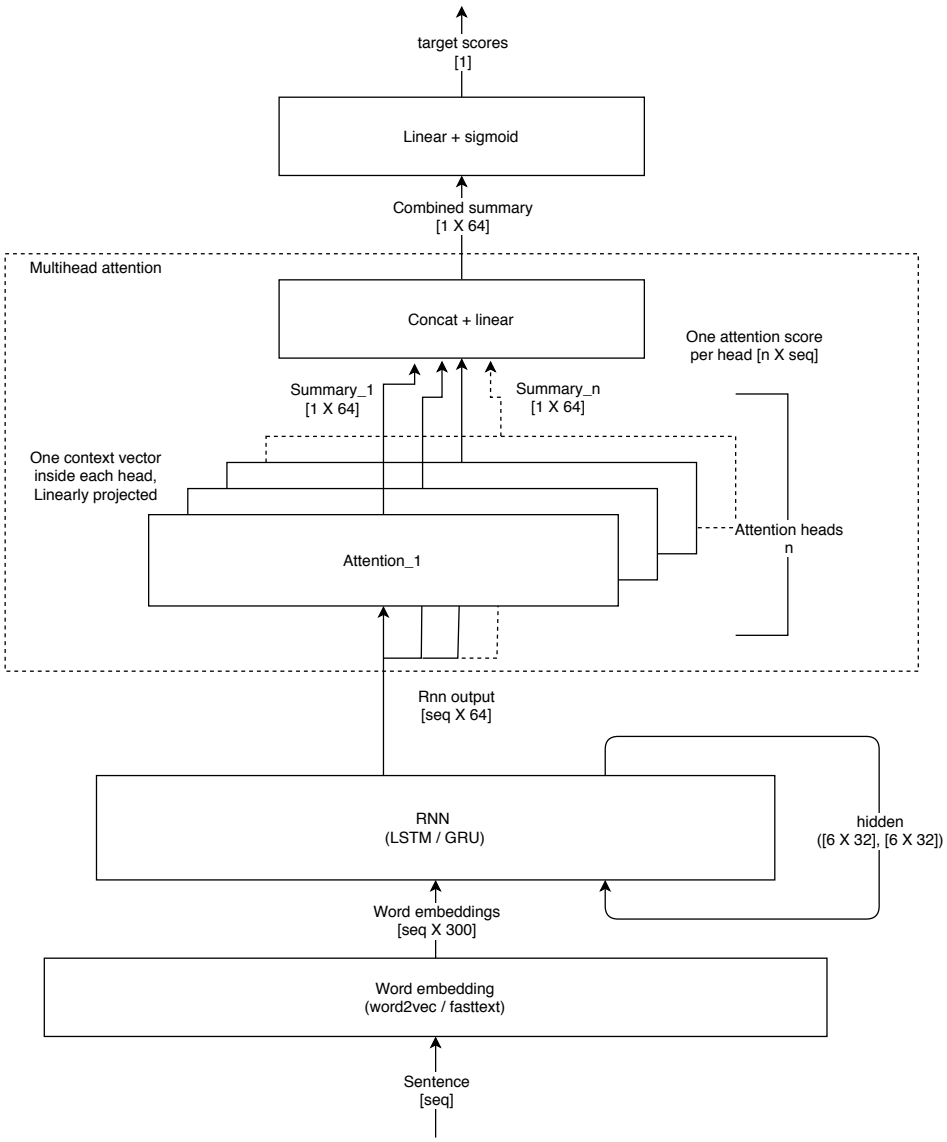

Figure 8.1 – Multihead attention model

For now, we only wish to use the regular attention to make sure we obtain the same results as before. This is done by using a single head in the multihead model.

Another change was to properly pad the sentences in our input batches using dedicated Pytoch functions, as that wasn't being done before. This helps make the computation in the LSTM more efficient. We also make use of the Dataloader class to get batches from our custom datasets.

In addition, a few parameters that were fixed in the code are now loaded from the configuration file and some minor bugs have been fixed.

Results

Seeing that the new implementation of the attention using pytorch perform just as well as our previous model ensures that the changes we made make sense and don't change the workings of our system.

Class number	Class name	Custom attention	Pytorch attention
0	Collective	0.86/0.84	0.84/0.87
1	Contrast	0.86/0.85	0.86/0.85
2	Goal	0.84/0.84	0.84/0.86
3	Goals2	0.78/0.79	0.83/0.80
4	List	0.79/0.73	0.80/0.75
5	Metaphor	0.76/0.76	0.79/0.76
6	Moral	0.87/0.87	0.88/0.85
7	Question	0.99/0.92	0.99/0.92
8	Story	0.88/0.89	0.89/0.89

Table 8.1 – Comparison of the custom attention and pytorch attention with/without punctuation

Joint training

Instead of training all classes separately, we would like to perform a joint training of the classes. This is motivated by the fact that we expect some amount of correlation between the classes, charisma being a combination of multiple elements that might influence each other. Being able to look at them at the same time might allow the network to establish relations that would increase the results.

This is done by increasing the dimensions of the query vector to match the number of classes and training the model on all classes rather than on the current target class against the non-targets.

Results

Contrary to our expectations, the results are worse than previously obtained. The attention is not as specialized as it used to be and has to adapt itself to maximize the classification results across all classes. However, we could try performing a joint training of a smaller number of classes at once. It would provide a better indication of class independence. This has not been done as some amount of brute-forcing would be necessary and the computation time would be too great to train all combinations of classes.

Class number	Class name	Individual training	Joint training
0	Collective	0.84/0.87	0.66/0.69
1	Contrast	0.86/0.85	0.76/0.71
2	Goal	0.84/0.86	0.76/0.72
3	Goals2	0.83/0.80	0.71/0.66
4	List	0.80/0.75	0.74/0.66
5	Metaphor	0.79/0.76	0.63/0.66
6	Moral	0.88/0.85	0.80/0.77
7	Question	0.99/0.92	0.99/0.85
8	Story	0.89/0.89	0.76/0.81

Table 8.2 – Comparison of the individual training and joint training with/without punctuation

We will keep experimenting on single-class training as it appears to provide the best classification results.

9 Multihead attention

Now that multihead attention has been added to our system, we decide to take full advantage of it. By increasing the number of heads, the attention should be able to gather more semantic meaning from the sentences, resulting in a better comprehension of what constitutes a CLT class. We expect a better classification output, especially in the more abstract classes such as metaphors, morals and goals.

Architecture

Increasing the number of attention heads is similar as projecting the inputs of the attention a certain number of times across different spaces, applying attention on each one of the projections in parallel and concatenating the output.

We therefore set the number of heads in our attention module to some higher value and train our model to see the evolution of the results.

Results

Increasing the number of heads in the multihead attention model doesn't seem to have any significant effect on the results. After further inspection of the model, it was found that we are missing one step in order to apply multihead attention properly. This is explained in the next section.

Class number	Class name	1 head	2 heads	4 heads	8 heads
0	Collective	0.84/0.87	0.83/0.85	0.86/0.81	0.82/0.79
1	Contrast	0.86/0.85	0.85/0.86	0.86/0.86	0.85/0.85
2	Goal	0.84/0.86	0.85/0.85	0.84/0.84	0.86/0.87
3	Goals2	0.83/0.80	0.85/0.80	0.79/0.82	0.77/0.82
4	List	0.80/0.75	0.80/0.75	0.81/0.76	0.81/0.75
5	Metaphor	0.79/0.76	0.78/0.78	0.77/0.77	0.77/0.76
6	Moral	0.88/0.85	0.87/0.86	0.86/0.87	0.86/0.86
7	Question	0.99/0.92	0.99/0.92	0.99/0.91	0.99/0.92
8	Story	0.89/0.89	0.89/0.88	0.89/0.89	0.89/0.89

Table 9.1 – Results of the Pytorch attention with different number of heads with/without punctuation

10 Multihead attention with duplicated input

The reason why we only tested to set the number of heads to powers of two in the previous model is due to an error showing up when using any other number of heads. Pytorch expects the dimensionality of the model to be a multiple of the number of heads. In practice, it takes the input and splits it n times, one for each head. A side effect of this is that each head only works on a fraction of the available information. To be able to gather more information about the sentences, it would make sense that each head would have access to the complete information. We therefore take the output of the LSTM and duplicate it n times. This also allows us to use any number of heads we wish to.

Architecture

The model is similar as in the previous section but the outputs of the LSTM are duplicated by the number of heads along the hidden dimension. Changes are made in the definitions of the multihead attention and classifier to accommodate a greater number of dimensions.

Results

Applying multihead attention on bigger vectors doesn't seem to have much impact on the results. It even looks like we obtain slightly worse ones when using more than five heads.

Our reasoning with this method was that each head could have access to the whole sentence. After more research on the workings of multihead attention, we see that this is not the proper way to use it. We address the correct way to apply multihead attention in the next experiment.

Class number	Class name	1 head	2 heads	3 heads	4 heads	5 heads	6 heads	7 heads	8 heads
0	Collective	0.84	0.85	0.80	0.84	0.82	0.80	0.80	0.85
1	Contrast	0.86	0.86	0.85	0.86	0.85	0.86	0.86	0.85
2	Goal	0.84	0.87	0.87	0.81	0.86	0.81	0.83	0.85
3	Goals2	0.83	0.81	0.79	0.74	0.80	0.80	0.80	0.80
4	List	0.80	0.80	0.80	0.79	0.80	0.80	0.81	0.80
5	Metaphor	0.79	0.79	0.79	0.77	0.78	0.79	0.77	0.76
6	Moral	0.88	0.85	0.88	0.89	0.88	0.82	0.87	0.88
7	Question	0.99	0.99	0.99	0.99	0.99	0.99	0.99	0.99
8	Story	0.89	0.88	0.88	0.89	0.85	0.87	0.89	0.87

Table 10.1 – Results of the Pytorch attention with duplicated inputs of the attention with punctuation

11 Multihead attention with linearly increased input

Instead of duplicating the hidden dimensions, the proper method is to apply a linear projection on the outputs of the recurrent network to obtain vectors of a higher dimension. One issue in our simple multihead attention model was that the 64 hidden dimensions were being divided by the number of heads, meaning that each head would only have a very small amount of information to work with as we add more of them.

Architecture

The duplication function from the previous model is replaced by a feedforward network of dimension 64 to num_heads * 64, giving each head 64 dimensions to work with, no matter the number of heads.

Results

The results are rather similar to the previous model, without much improvement. In general, is seems that we are not able to improve on the basic attention model by using more attention heads. We will next try to use more recent models that are said to be performing state-of-the-art classification.

Class number	Class name	1 head	2 heads	3 heads	4 heads	5 heads	6 heads	7 heads	8 heads
0	Collective	0.84	0.85	0.86	0.81	0.83	0.83	0.81	0.87
1	Contrast	0.86	0.85	0.85	0.85	0.87	0.87	0.85	0.85
2	Goal	0.84	0.81	0.87	0.83	0.80	0.83	0.86	0.81
3	Goals2	0.83	0.79	0.79	0.82	0.80	0.79	0.79	0.78
4	List	0.80	0.78	0.80	0.80	0.80	0.80	0.80	0.80
5	Metaphor	0.79	0.78	0.76	0.75	0.76	0.78	0.76	0.77
6	Moral	0.88	0.86	0.88	0.84	0.86	0.86	0.87	0.89
7	Question	0.99	0.99	0.99	0.99	0.99	0.99	0.99	0.99
8	Story	0.89	0.89	0.90	0.90	0.89	0.90	0.88	0.88

Table 11.1 – Results of the Pytorch attention with linearly projected inputs of the attention with punctuation

12 BERT model

Since the attention mechanism was first published and became wildly used in a variety of NLP and speech tasks, new state-of-the-art models have been developed to take full advantage of it. One such model is the transformer. Described for the first time in the aptly named paper "Attention is All You Need" [8], the transformer gets rid of the recurrent network and relies entirely on the attention mechanism. At a high level, it is still a encoder-decoder module that takes an input sentence, creates a compressed representation of it and decodes it into another sequence.

Architecture

The encoder and decoder of the transformer are actually composed of stacks of encoder and decoders of equal number. Those are rather similar to each other. An encoder layer is made of a multihead self-attention layer followed by a feed-forward layer. Self-attention is a bit different from the attention we have been using until now. Instead of the query being a hidden state of the decoder, it is a linear projection of the input. This implies that the output of the attention is of the same length of the input sentence. The self-attention and feed-forward layers possess residual connections used in layer normalization [15].

The decoder works the same way except for an added encoder-decoder attention layer after the normalized self-attention layer. The output of the final encoder stack is used as input by every encoder-decoder attention layer in the decoder.

Finally, a linear layer followed by a softmax creates the outputs of the transformer.

Here, we decided to use a particular implementation the the transformer called the Bidirectional Encoder Representations from Transformers, or BERT. As the name states, it only uses the encoder stack of the transformer. It is used to create dense representations of the the input sentence, which are supposedly more effective than word2vec embeddings and usable in a greater variety of tasks. Rather than a new model, it BERT is considered a learning strategy. It is first pre-trained on data that doesn't require labelling. We can then use this pretrained

model and fine-tune it to fit our application.

As was the case with the recurrent network in our previous models, the transformer returns a sequence of length equal to that of the input that needs be reduced. To solve that, we take the hidden state of the last layer of decoder and apply our previous attention on it to obtain a single representation of the input. This again is used by the feed-forward and sigmoid layer we used until now. BERT effectively replaces the word embedding layer as well as the recurrent network.

We must also use the provided BERT tokenizer to ensure the inputs are in the same format BERT was trained on and contains the proper start and end special tokens.

We use the default BERT implementation from the Huggingface[1] library and the "bert-base-cased" pretrained model and tokenizer.

Results

The BERT model performs noticeably better in every class, be it with or without punctuation, with the exception of the collective class without punctuation.

Class number	Class name	Basic attention	BERT
0	Collective	0.84/0.87	0.86/0.83
1	Contrast	0.86/0.85	0.89/0.88
2	Goal	0.84/0.86	0.88/0.88
3	Goals2	0.83/0.80	0.86/0.85
4	List	0.80/0.75	0.84/0.81
5	Metaphor	0.79/0.76	0.83/0.82
6	Moral	0.88/0.85	0.90/0.90
7	Question	0.99/0.92	0.99/0.97
8	Story	0.89/0.89	0.92/0.91

Table 12.1 – Comparison of the single head attention results and the BERT results with/without punctuation

[1] https://huggingface.co/

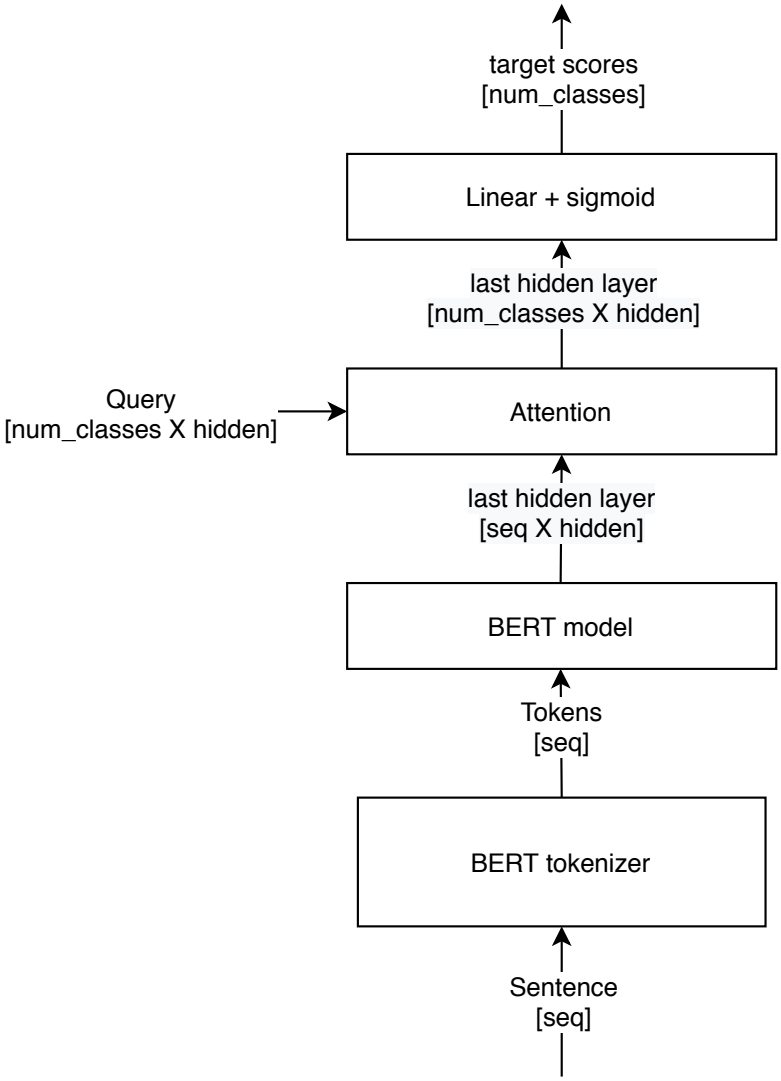

Figure 12.1 – BERT model

13 Conclusion

The hypothesis that metrics of charisma can be inferred using deep learning techniques has been demonstrated. Attention only slightly improves the results but gives meaningful insight into what constitutes some of the classes. Metaphors appear to be the most difficult to classify, which can be explained by the fact that they carry information at a higher semantic level. Surprisingly, all models have some difficulty recognising lists even when punctuation is used.

A concern with the dataset used was the under-representation of some classes, particularly the collective, goals, goals2, and moral. However, the classification results are rather satisfying. Having access to a different and bigger dataset would prove difficult, as annotating text is very time consuming and costly. It also requires people that have been trained for the task of recognizing CLTs.

Performing a joint training of the classes produces worse results than training the classes separately. This could indicate the classes are not correlated and are robust building blocks used to define the core of what defines charisma. The joint training has also been performed for the multihead attention models but has been omitted in this document as the results were similar as those in chapter 8. A joint training using the BERT model could prove to be interesting but has not been done due to implementation and time issues.

The BERT model proved to be the most efficient at performing the task, whether it be in the classification accuracy or the speed of the training. There is now a great number of transformers released and it would be interesting to try more recent ones that might further improve our model, such as RoBERTa [16], ALBERT [17] or XLNET [18].

The presented system is a precursor to a bigger charisma rating system. We are currently only able to label classes according to the predefined CLTs but there isn't any grading of the speaker being done. This would require a discussion with the people involved with the research on charismatic leadership as simply counting the number of CLTs present in a speech would certainly not provide an accurate result.

During this project, we exclusively worked on transcriptions of speeches that were given orally.

As transcribed text is not the most convenient to work with in real-life situations, it has been considered to add a text-to-speech module that would enable someone to speak and being graded on the fly. Furthermore, recording speech would enable us to take into consideration charisma indicators other than lexical information alone, such as pace, intonation and pitch. Continuing on that line of thought, using video recording would let us capture posture, mannerisms and facial expressions which would give further insight into the mechanics of charisma.

A Appendix

Configuration name	Type	Description
Classes	int, list of ints	Classes to train
ClassName	string	Class name of the model, required for dynamic loading)
Dropout	float	Dropout
EmbeddingType	string	Type of word embedding, word2vec or fasttext
FasttextBin	string	Path to fasttext pretrained model
LearningRate	float	Learning rate
MinWordCount	int	Minimal number of instances of a word that must be found to keep it
ModelName	string	Name of the model
NbEpochs	int	Number of training epochs
NbThresholds	int	Number of thresholds for ROC curve
Num_Heads	int	Number of attention heads
Optimizer	string	Optimizer used
OutputAttentionWeights	bool	Dump the attention weights to a file during evaluation
OutputRnnOut	bool	Dump the output of the RNN to a file during evaluation
Param_HiddenSize	int	Number of hidden dimensions in the RNN
Param_LossFunction	string	Loss function used
Param_NbRNNs	int	Number of recurrent layers
Param_RNNType	string	Type of RNN used, LSTM or GRU
PlotEER	bool	Plot Equal Error Rates
PlotF1Score	bool	Plot F1 scores
PlotMaxPrecision	bool	Plot max precision
PlotMaxRecall	bool	Plot Max Recall
PlotNbCols	int	Number of columns in the pdf
PlotROC	bool	Plot ROC curves
Pretrained	bool	Indicates if we are using a pretrained model or not
TrainBatchSize	int	Batch size
TrainEpochs	int	Number of training epochs for the word embedding
UsePunctuation	bool	keeps or removes punctuation in the input sentences
WeightDecay	float	Weight decay
Word2VecBin	string	Path to word2vec pretrained model

Table A.1 – List of options in the config.ini file

Train set

Class number	Class name	Total coder 1	Total coder 2	Total coder 3	Min	Max	Only first	Only second	Only third	First and second	Second and third	First and third	At least one	At least two	All three
0	Collective	220	110	155	110	220	111	68	92	13	7	40	346	75	15
1	Contrast	891	325	713	325	891	358	49	179	48	26	317	1170	584	193
2	Goal	77	142	78	77	142	27	99	36	9	21	8	213	51	13
3	Goals2	69	229	36	36	229	30	184	11	25	10	3	274	49	11
4	List	1861	1037	1910	1037	1910	423	72	449	55	68	547	2462	1518	848
5	Metaphor	854	485	352	352	854	489	177	72	128	43	105	1143	405	129
6	Moral	252	110	72	72	252	179	61	23	24	6	26	337	74	18
7	Question	1450	1364	1225	1225	1450	130	85	4	128	29	71	1569	1350	1122
8	Story	732	354	631	354	732	327	109	277	40	41	202	1108	445	162

Validation set

Class number	Class name	Total coder 1	Total coder 2	Total coder 3	Min	Max	Only first	Only second	Only third	First and second	Second and third	First and third	At least one	At least two	All three
0	Collective	62	60	46	46	62	23	38	22	2	6	8	106	23	7
1	Contrast	289	99	260	99	289	127	19	90	16	16	109	424	188	47
2	Goal	39	47	40	39	47	15	32	23	3	8	6	90	20	3
3	Goals2	24	88	17	17	88	11	72	6	8	6	3	108	19	2
4	List	636	351	662	351	662	153	19	180	23	19	172	860	508	294
5	Metaphor	350	150	117	117	350	224	50	20	38	5	39	426	132	50
6	Moral	98	50	47	47	98	67	32	26	12	5	14	158	33	2
7	Question	502	450	445	445	502	37	10	4	34	8	35	525	474	397
8	Story	230	120	219	120	230	104	46	96	13	12	63	383	137	49

Test set

Class number	Class name	Total coder 1	Total coder 2	Total coder 3	Min	Max	Only first	Only second	Only third	First and second	Second and third	First and third	At least one	At least two	All three
0	Collective	59	44	39	39	59	26	26	20	4	2	10	95	23	7
1	Contrast	287	110	253	110	287	102	20	57	10	8	113	382	203	72
2	Goal	36	58	28	28	58	13	43	18	8	1	4	92	18	5
3	Goals2	34	104	17	17	104	16	84	5	11	2	1	127	22	8
4	List	711	380	680	380	711	164	19	130	23	22	212	886	573	316
5	Metaphor	274	150	101	101	274	160	63	22	40	6	36	364	119	37
6	Moral	92	34	25	25	92	71	18	6	6	3	11	120	25	5
7	Question	388	360	349	349	388	30	13	0	16	7	18	408	365	324
8	Story	251	131	218	131	251	107	52	80	17	9	74	383	154	54

Figure A.1 – Complementary information about dataset labelling

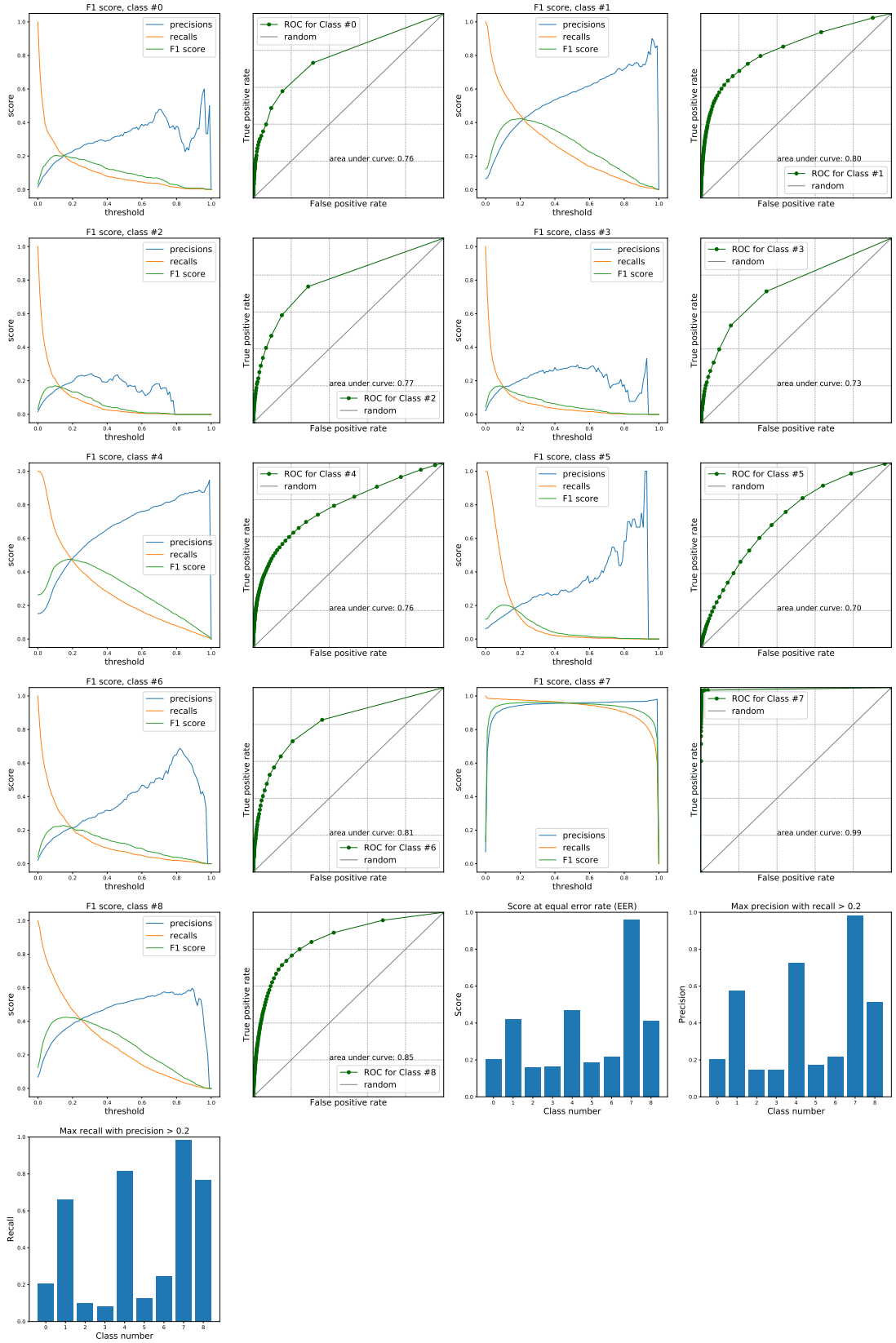

Figure A.2 – Results of the baseline model with punctuation

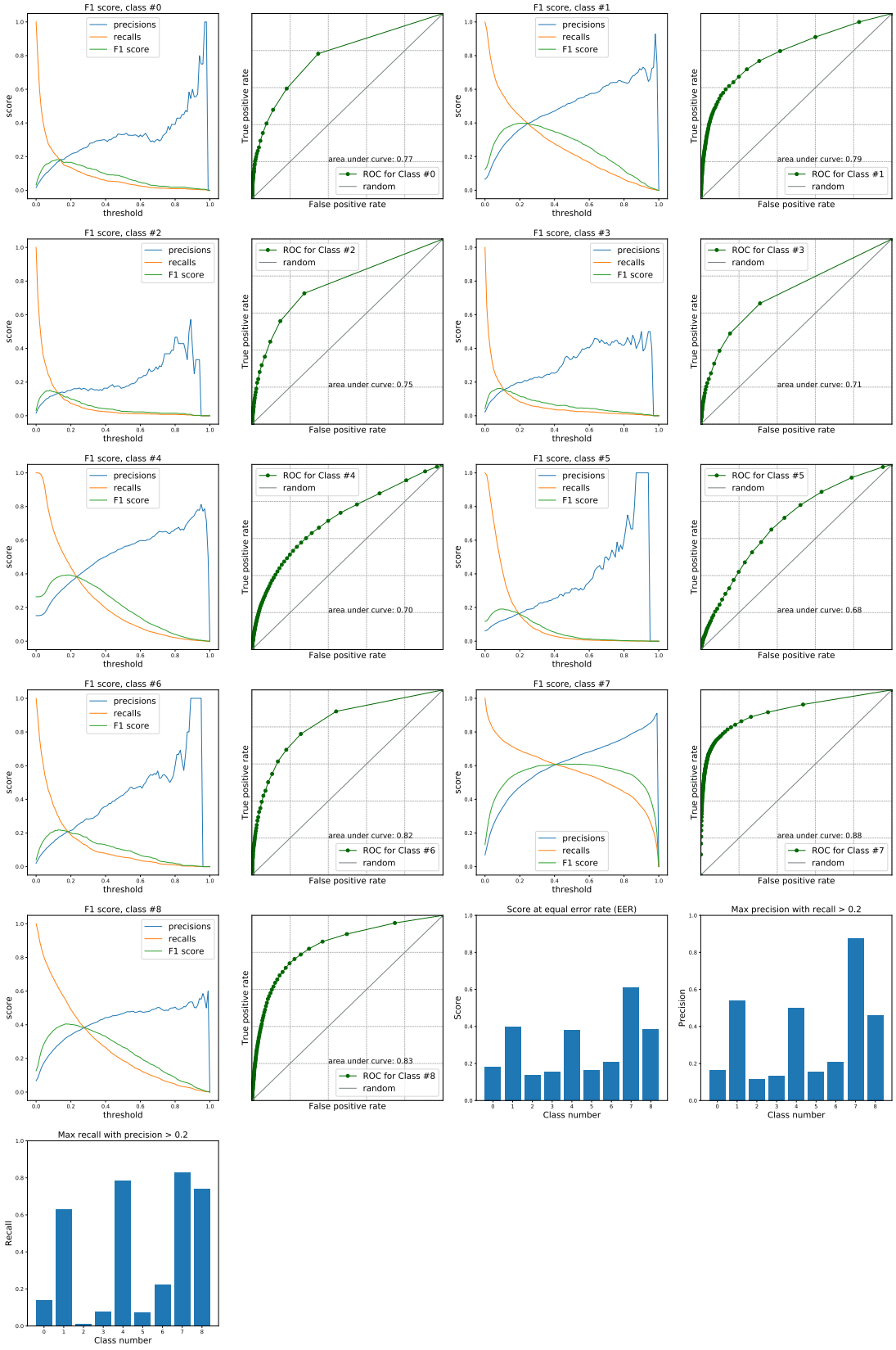

Figure A.3 – Results of the baseline model without punctuation

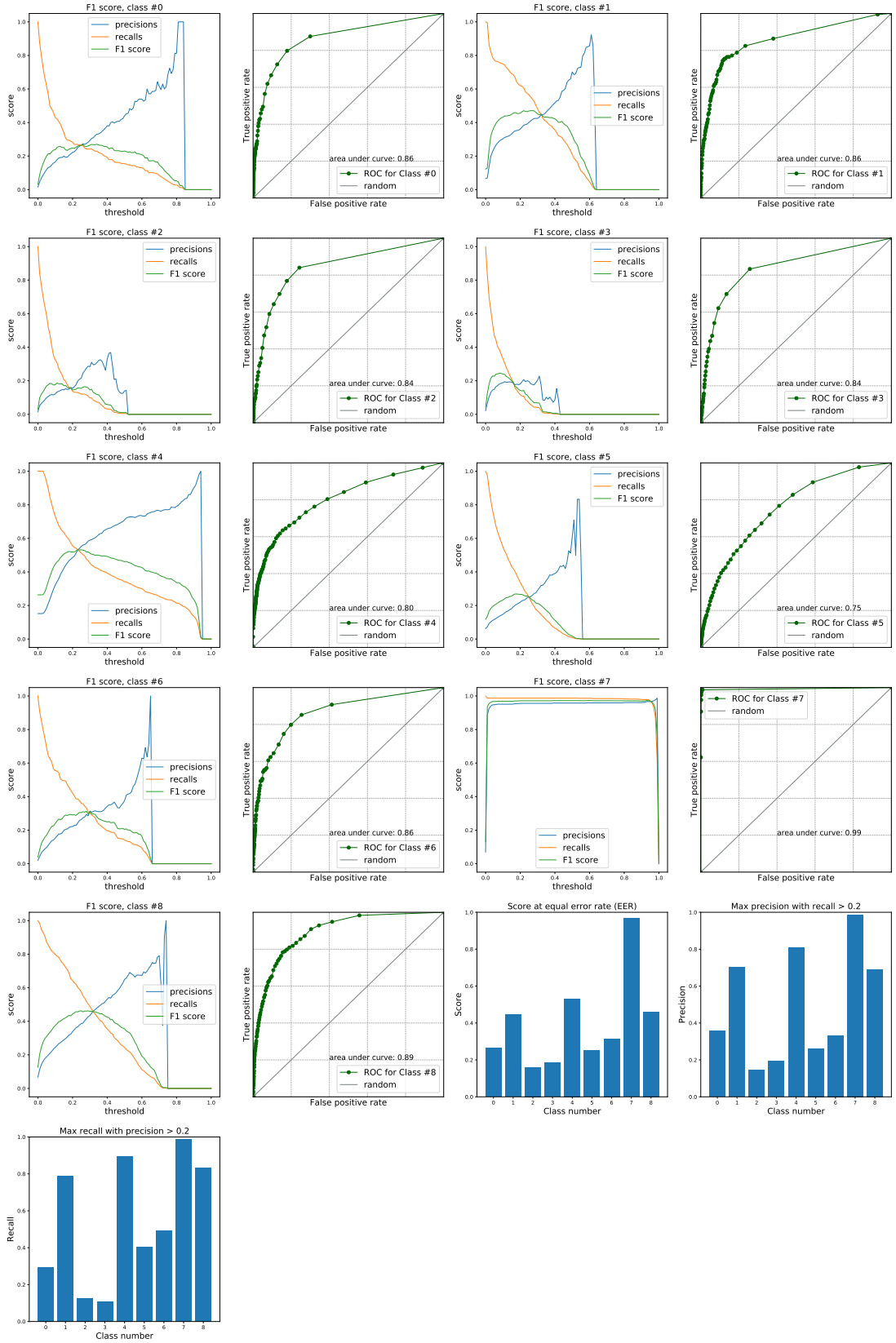

Figure A.4 – Results of the original attention model with punctuation

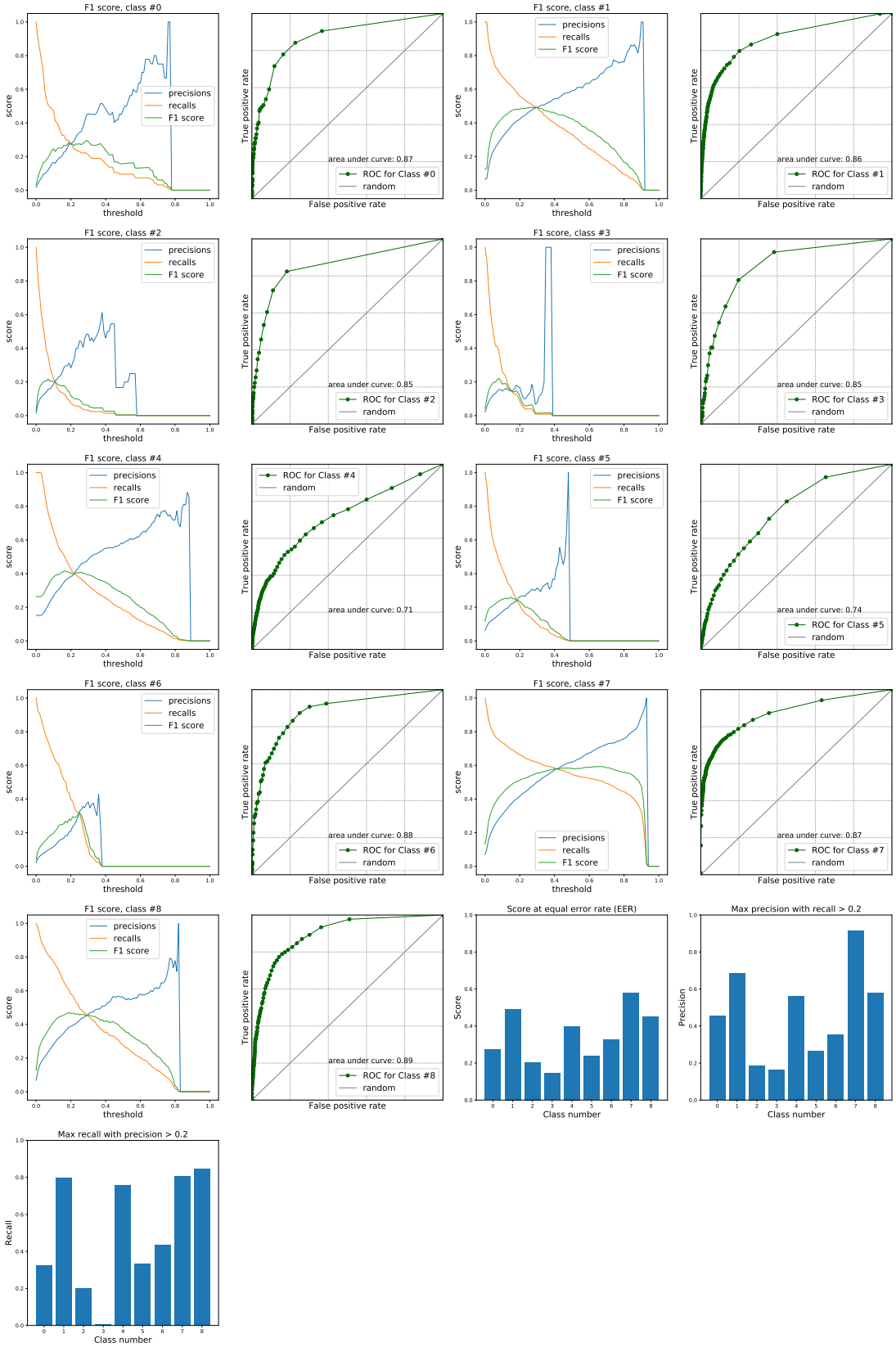

Figure A.5 – Results of the original attention model without punctuation

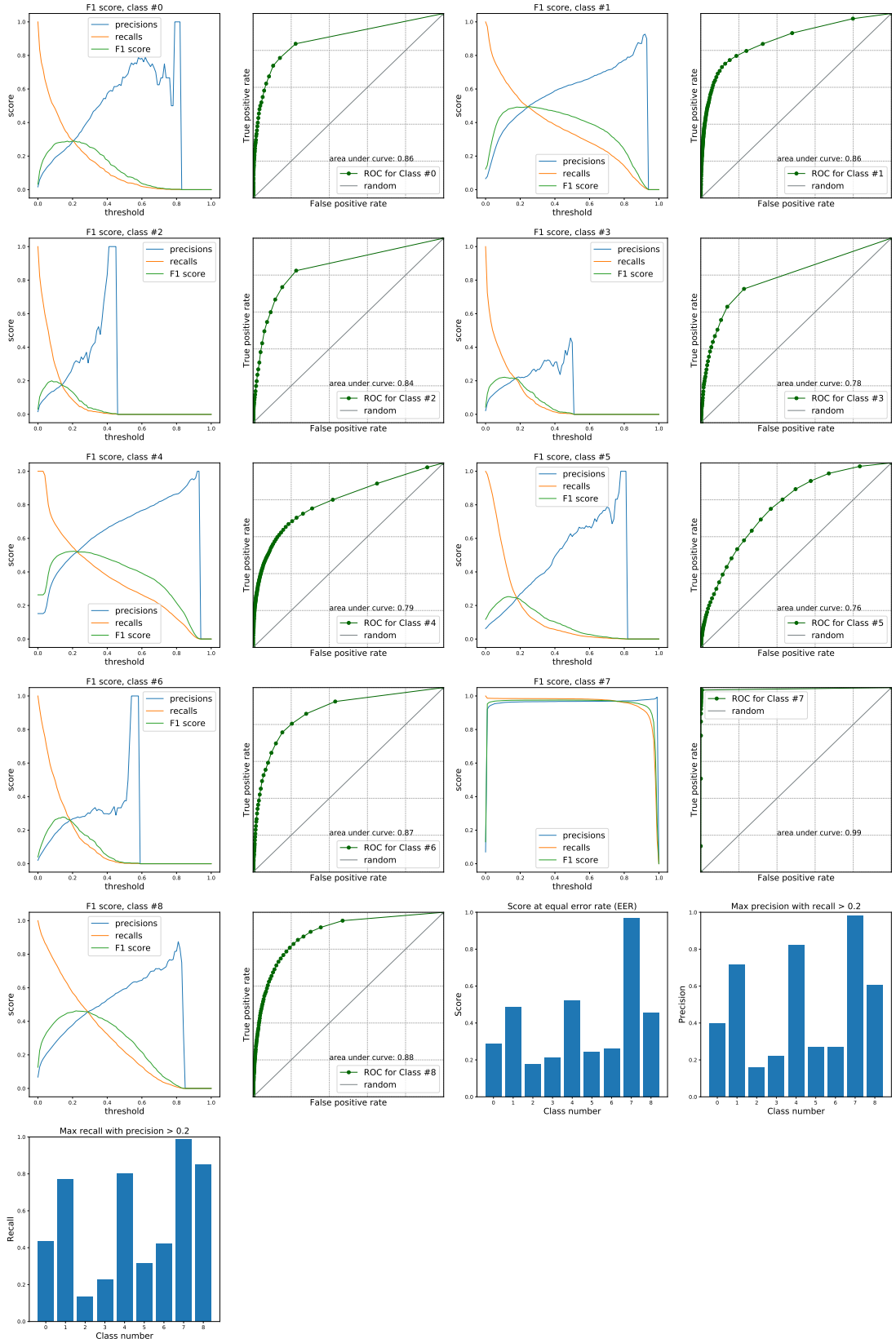

Figure A.6 – Results of the attention model with better embeddings and with punctuation

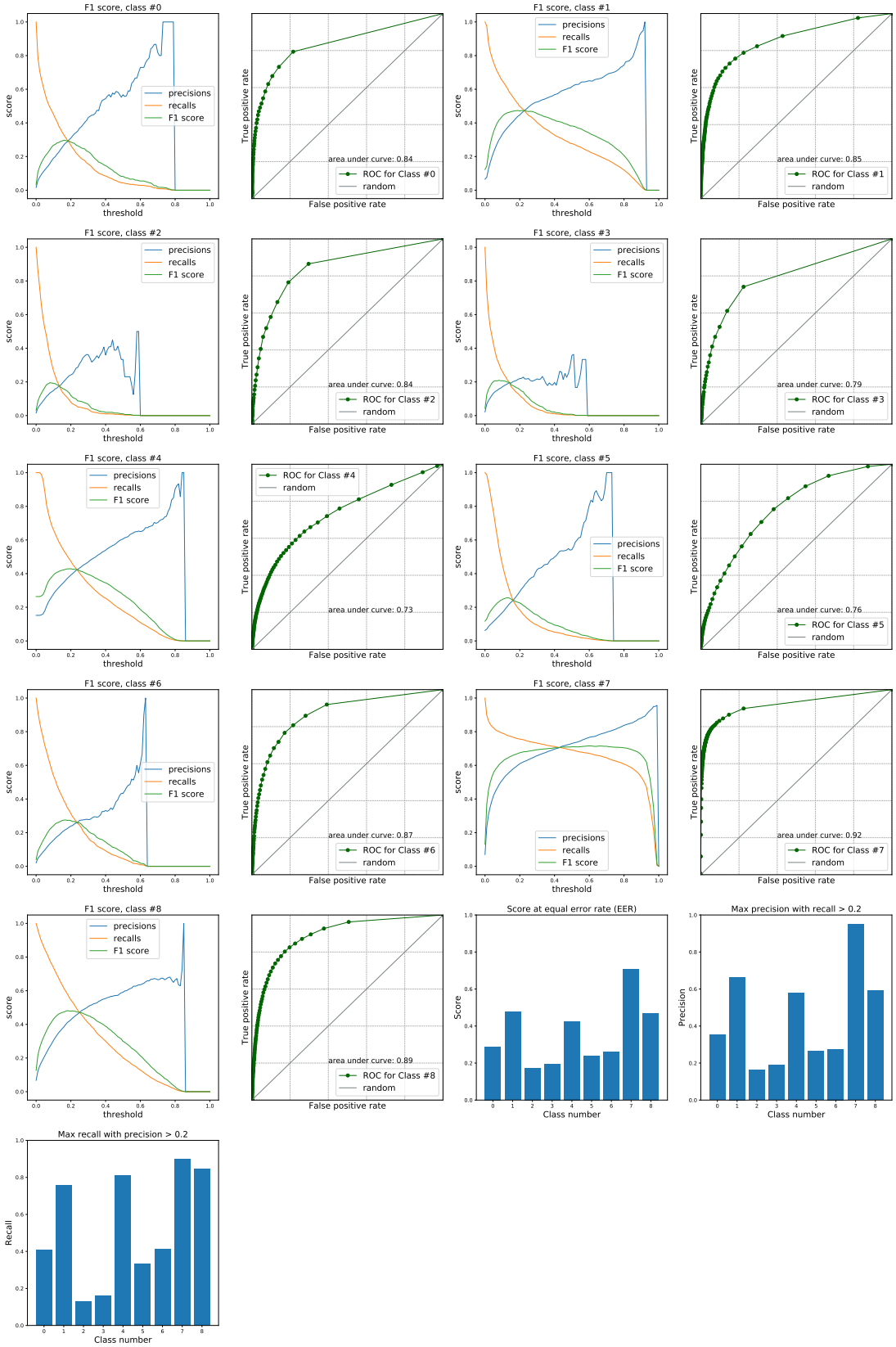

Figure A.7 – Results of the attention model with better embeddings and without punctuation

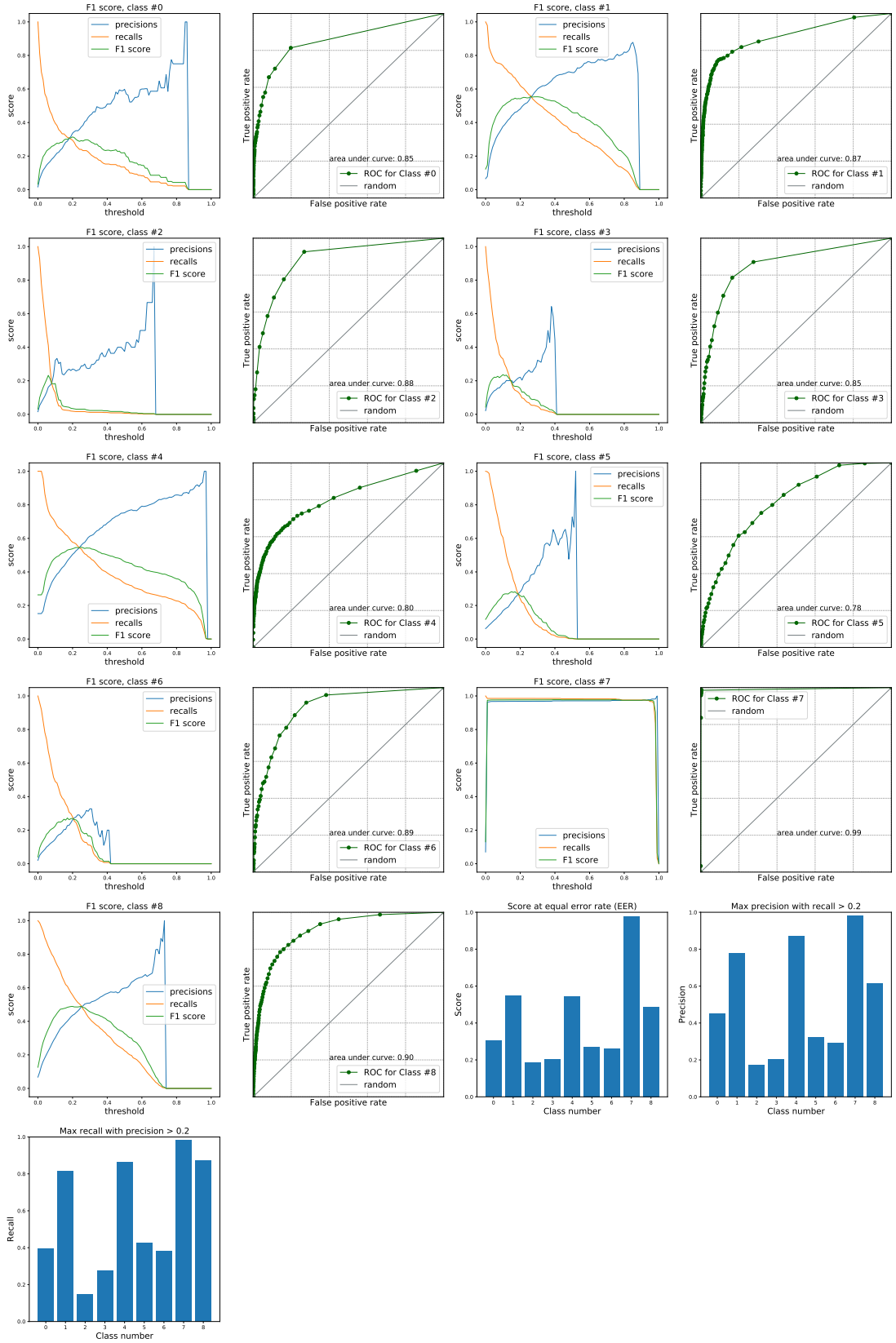

Figure A.8 – Results of the attention model using fasttext embeddings with punctuation

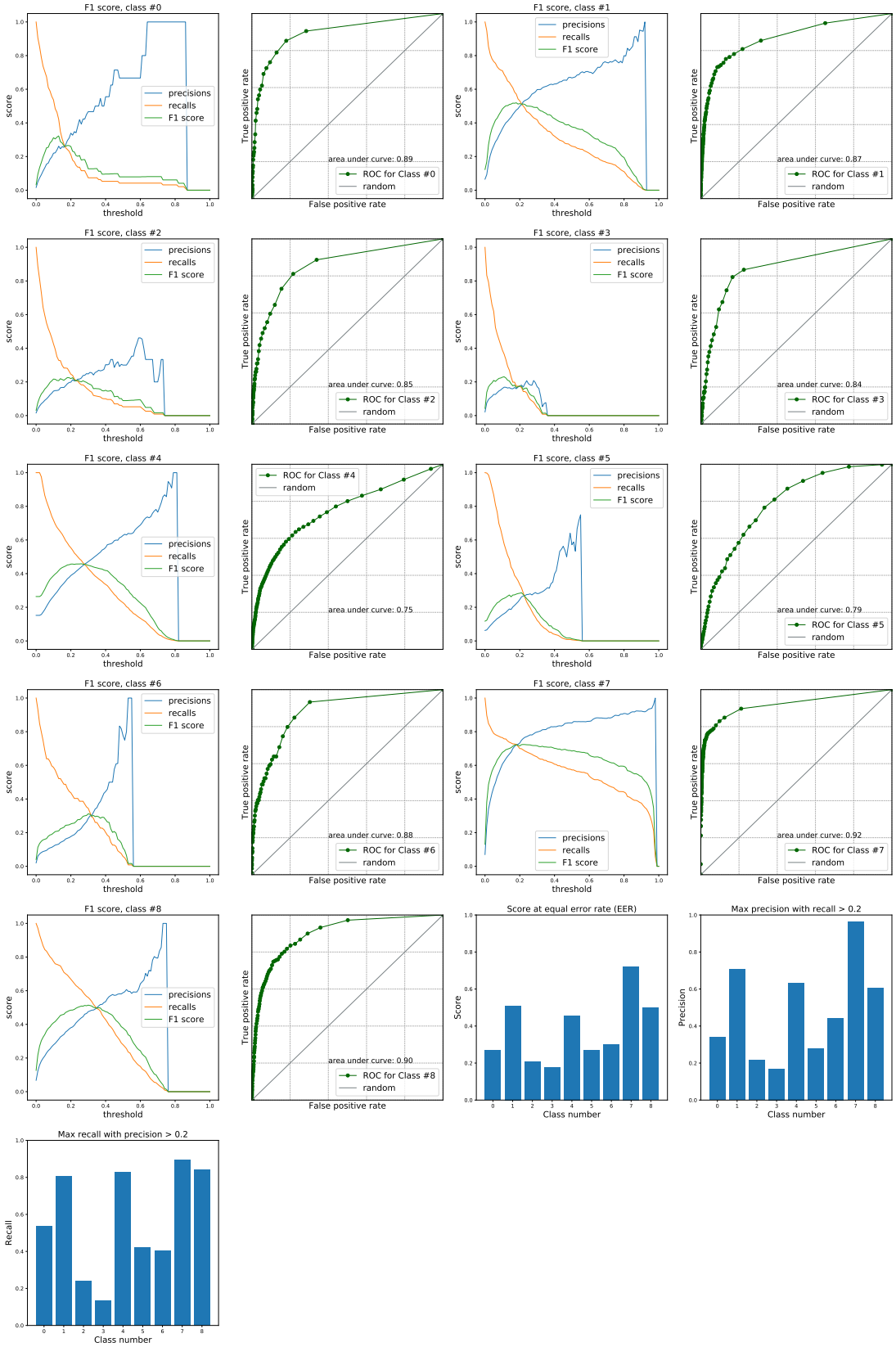

Figure A.9 – Results of the attention model using fasttext embeddings without punctuation

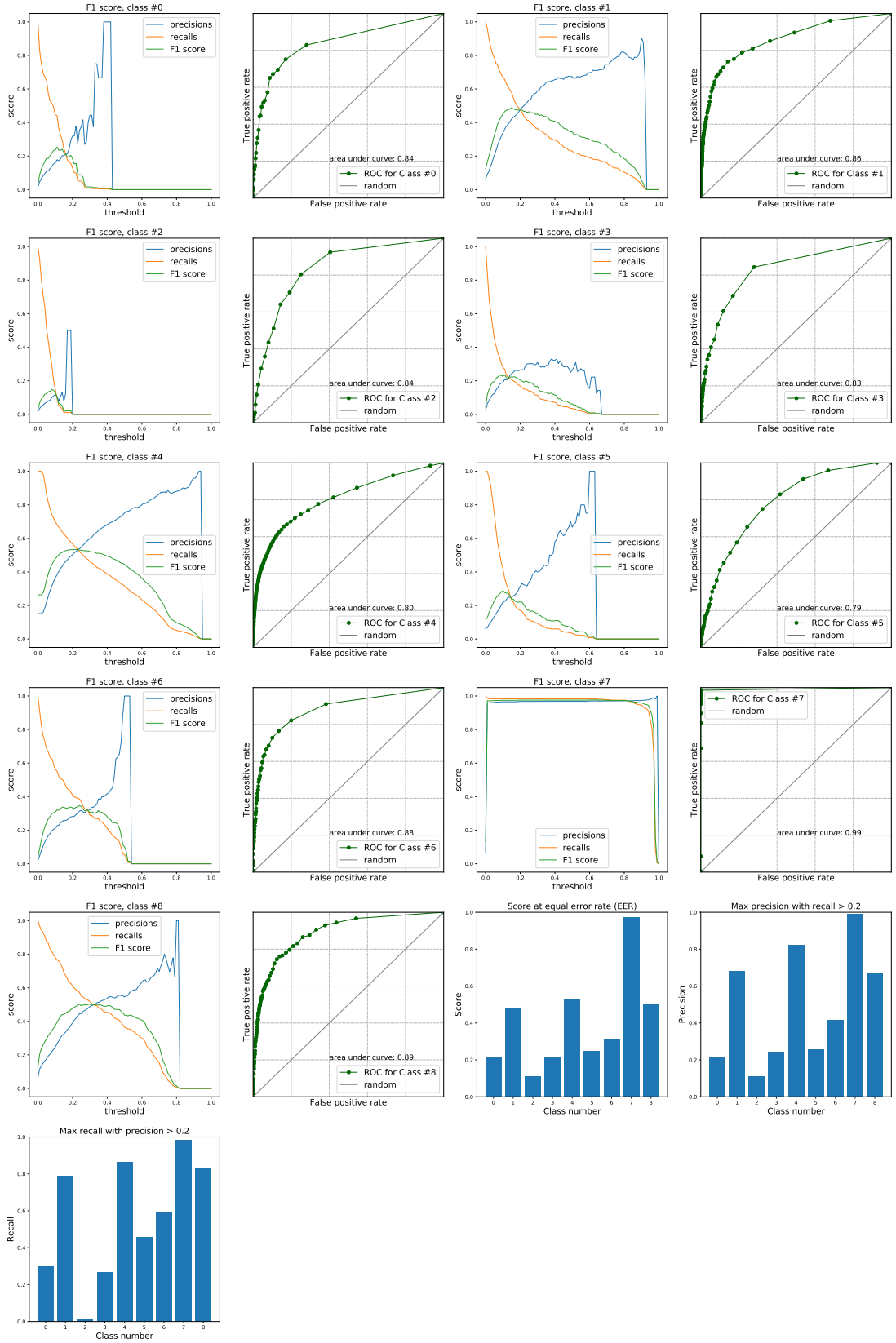

Figure A.10 – Results of the multihead attention model using a single head with punctuation

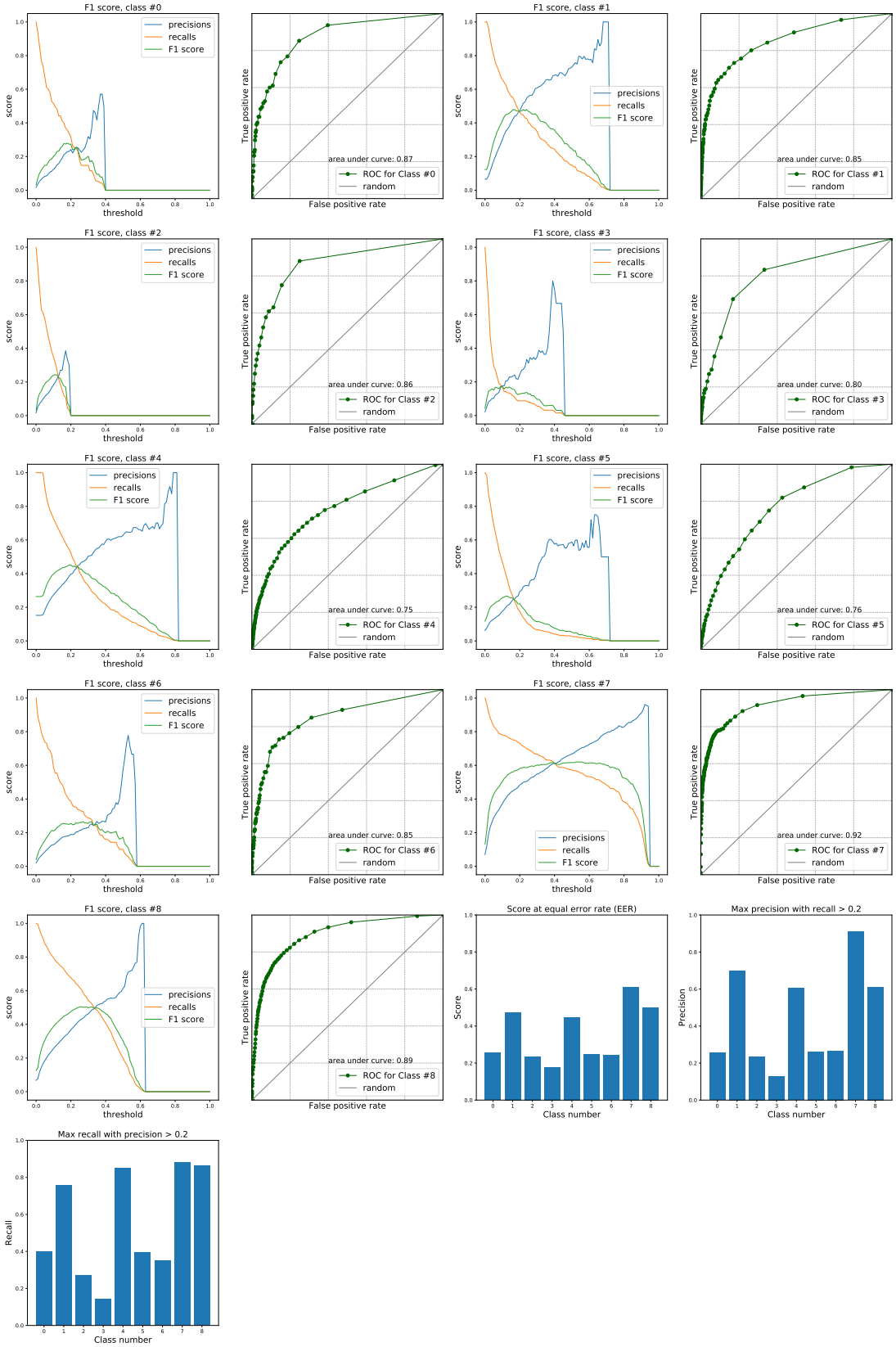

Figure A.11 – Results of the multihead attention model using a single head without punctuation

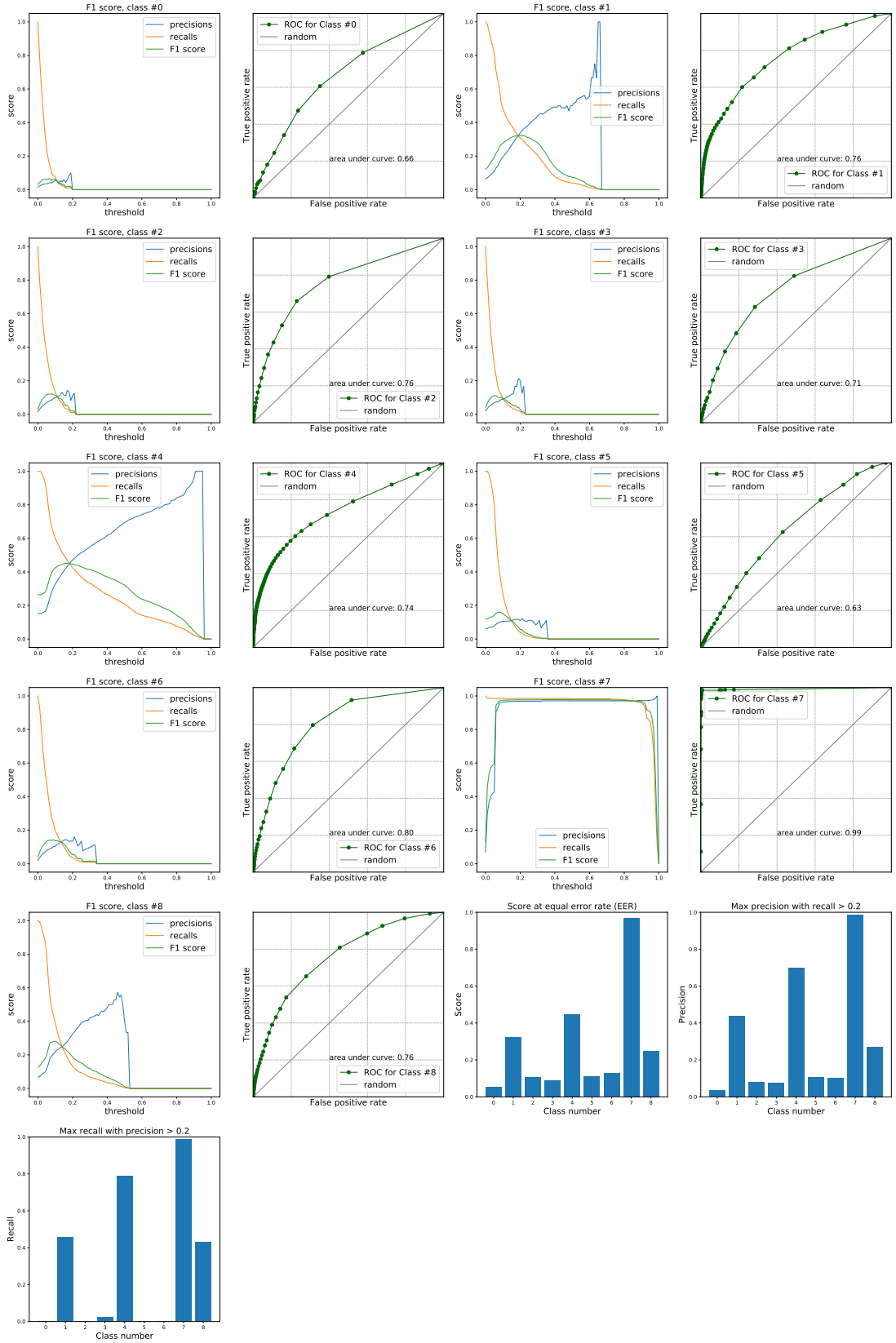

Figure A.12 – Results of the multihead attention model for joint training using a single head with punctuation

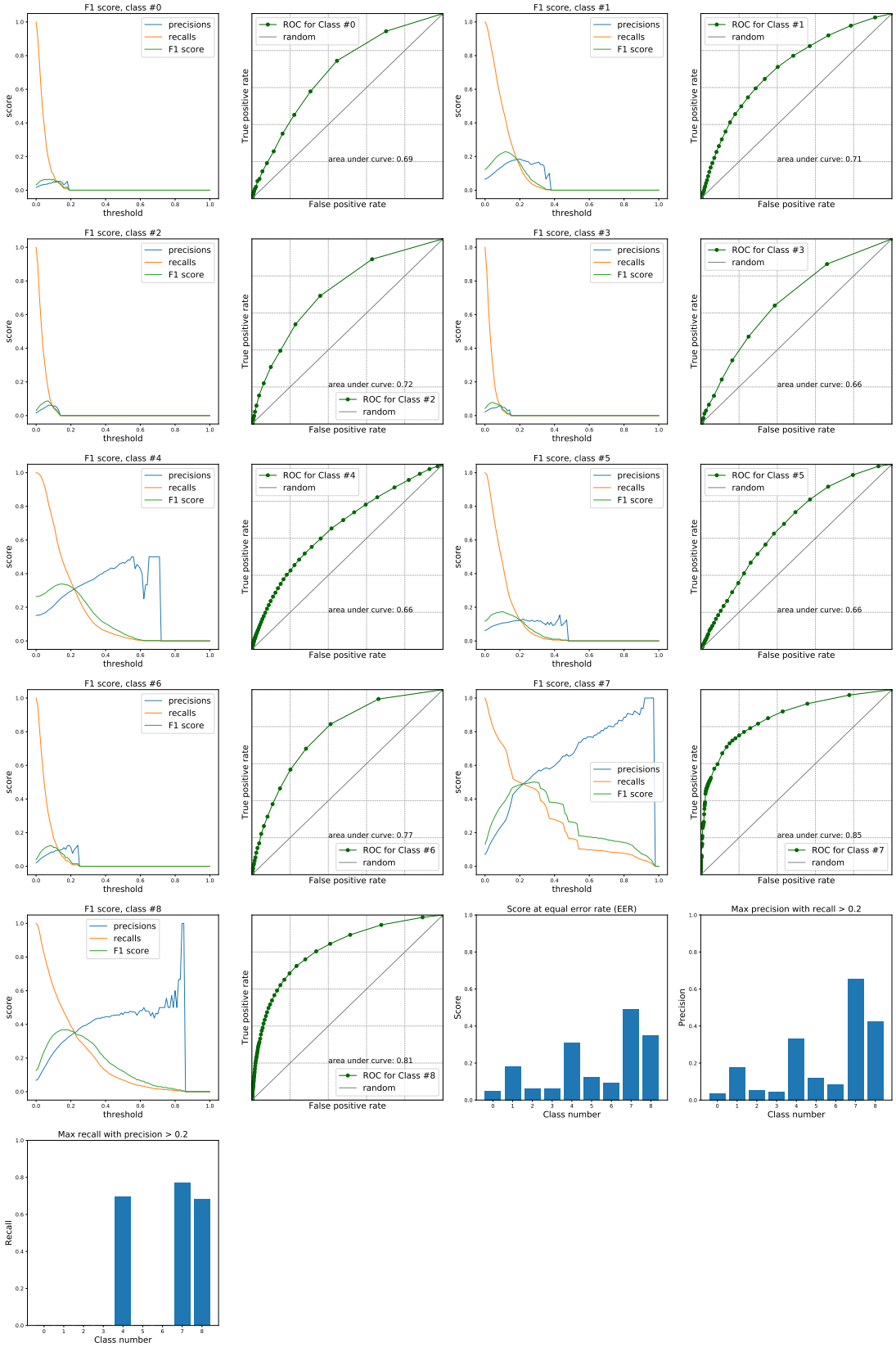

Figure A.13 – Results of the multihead attention model for joint training using a single head without punctuation

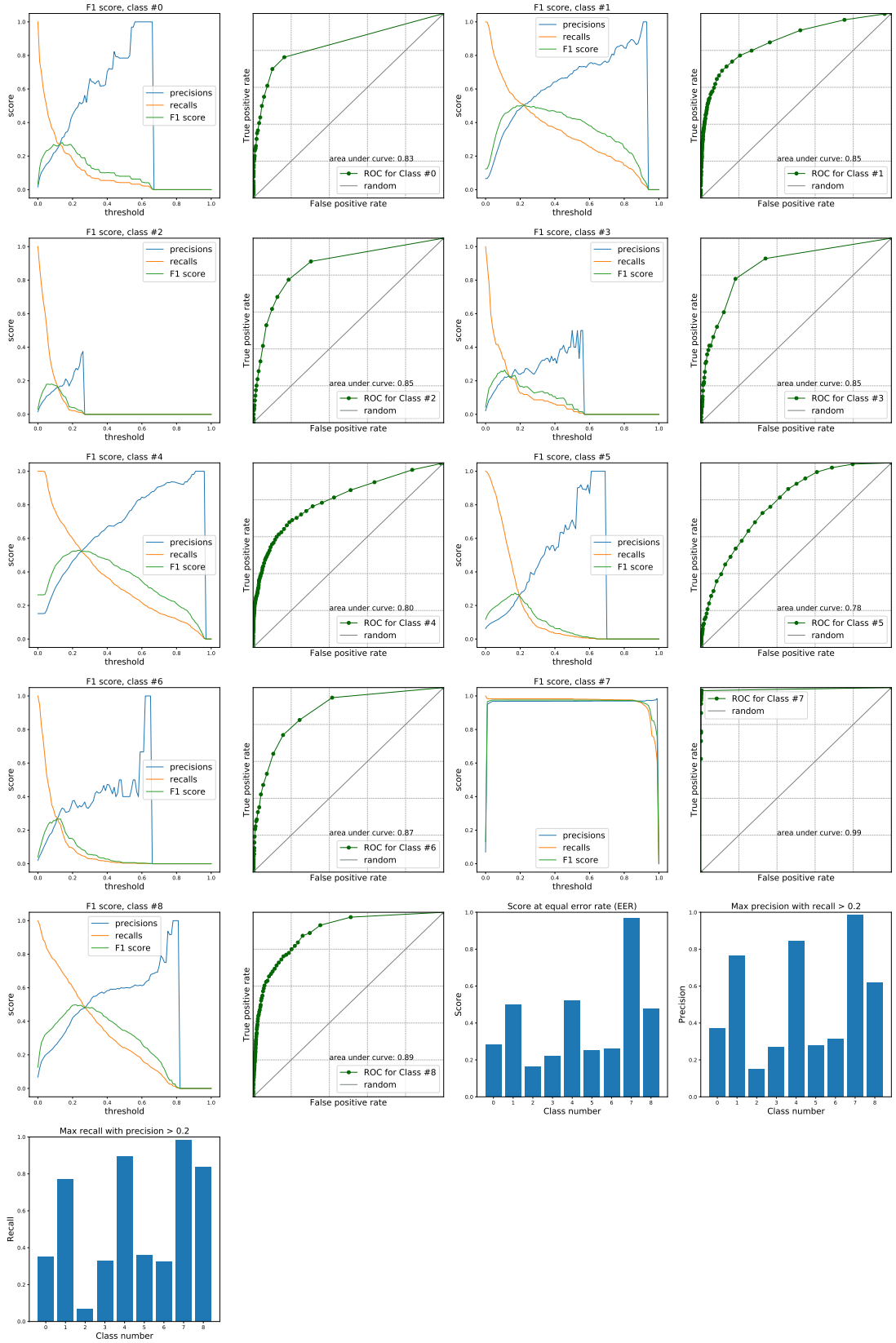

Figure A.14 – Results of the multihead attention model using 2 heads with punctuation

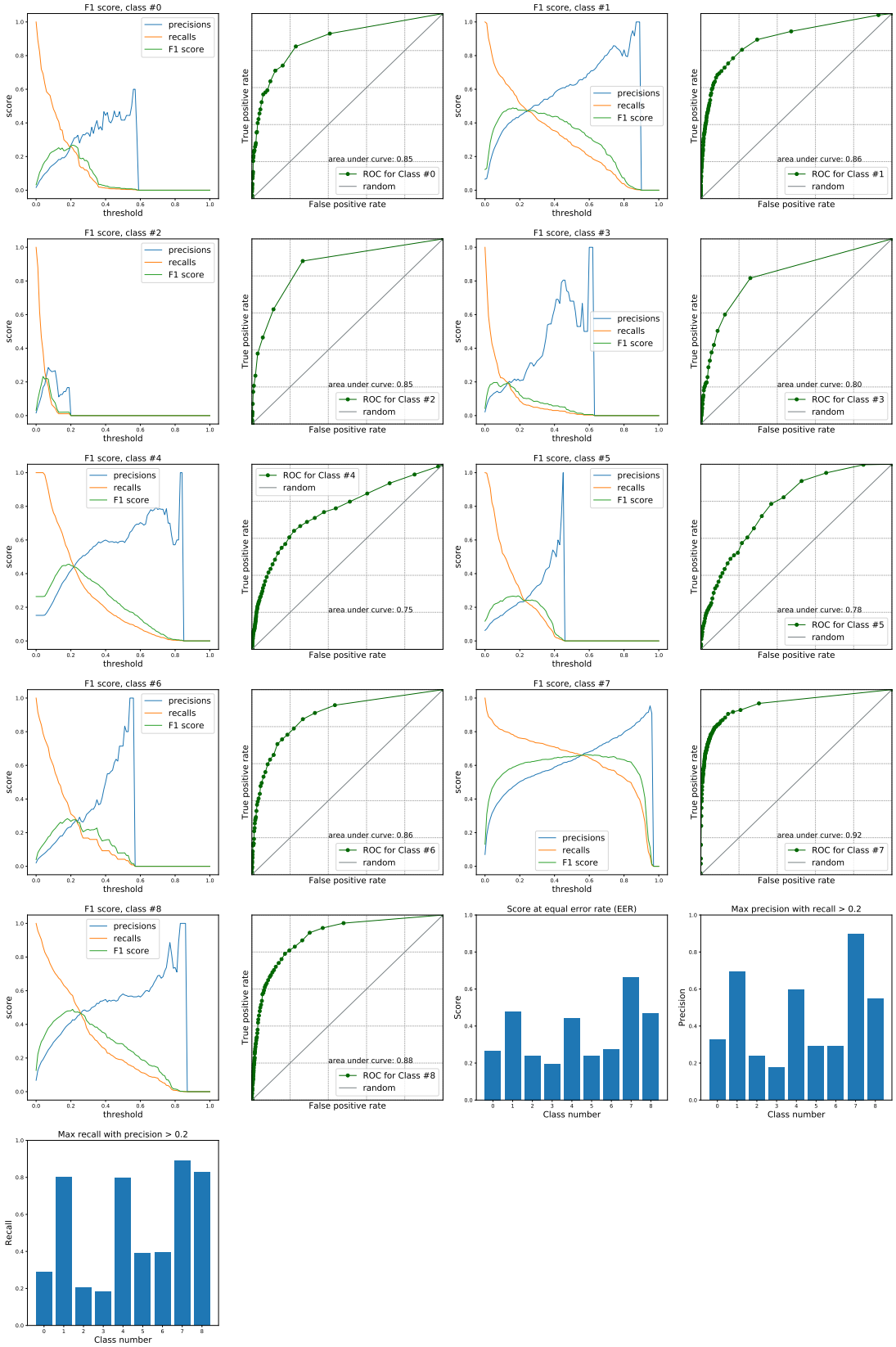

Figure A.15 – Results of the multihead attention model using 2 heads without punctuation

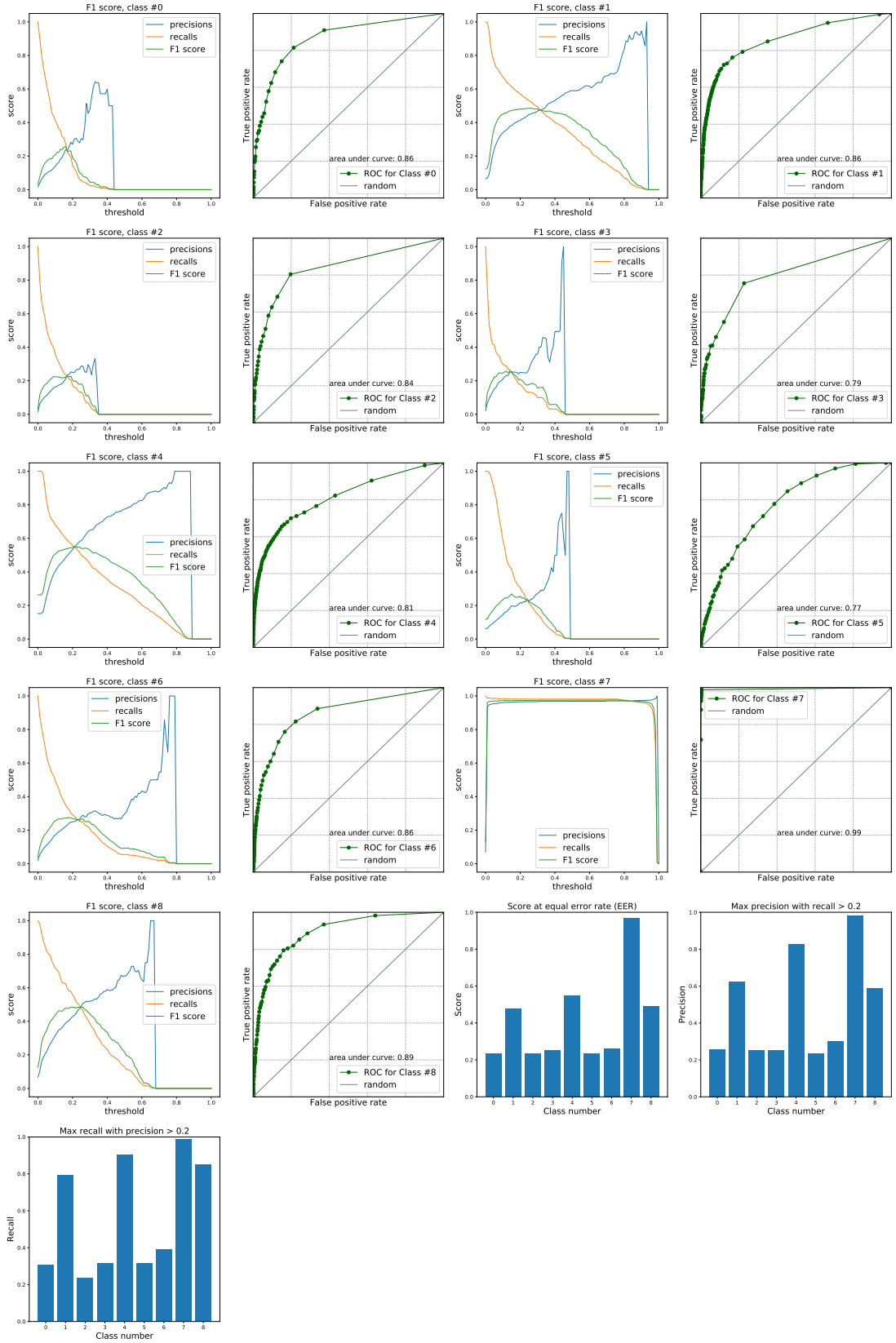

Figure A.16 – Results of the multihead attention model using 4 heads with punctuation

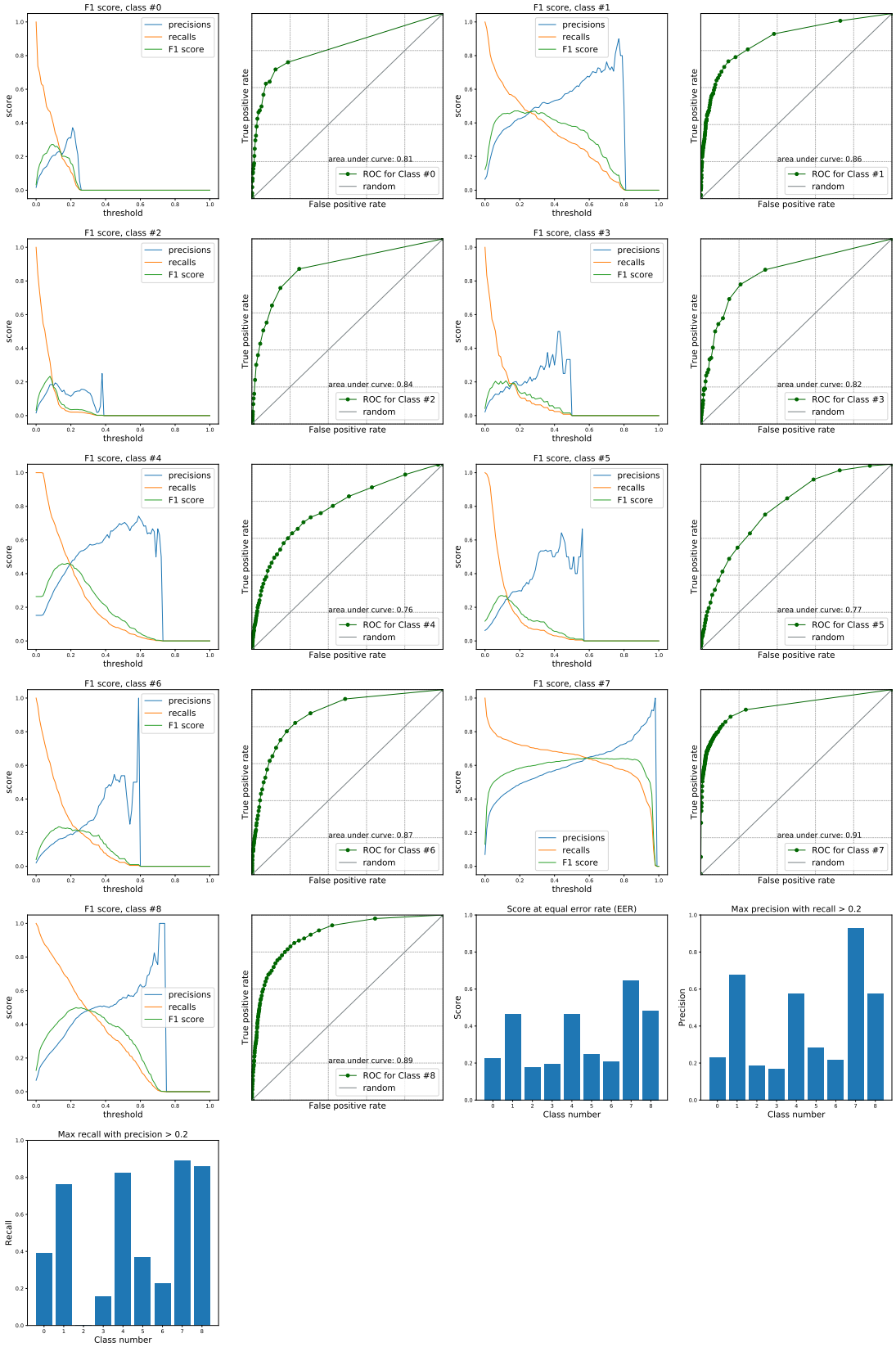

Figure A.17 – Results of the multihead attention model using 4 heads without punctuation

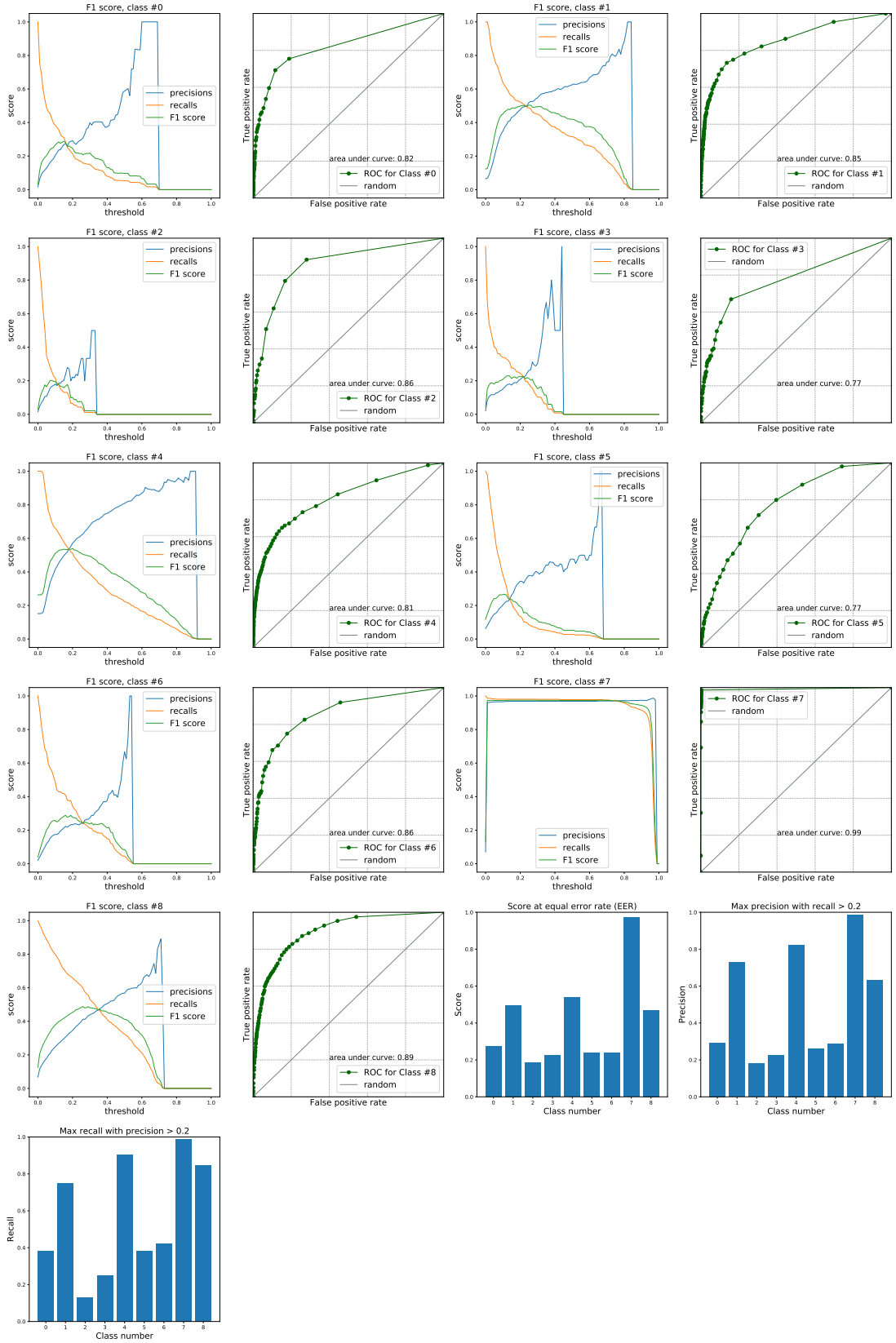

Figure A.18 – Results of the multihead attention model using 8 heads with punctuation

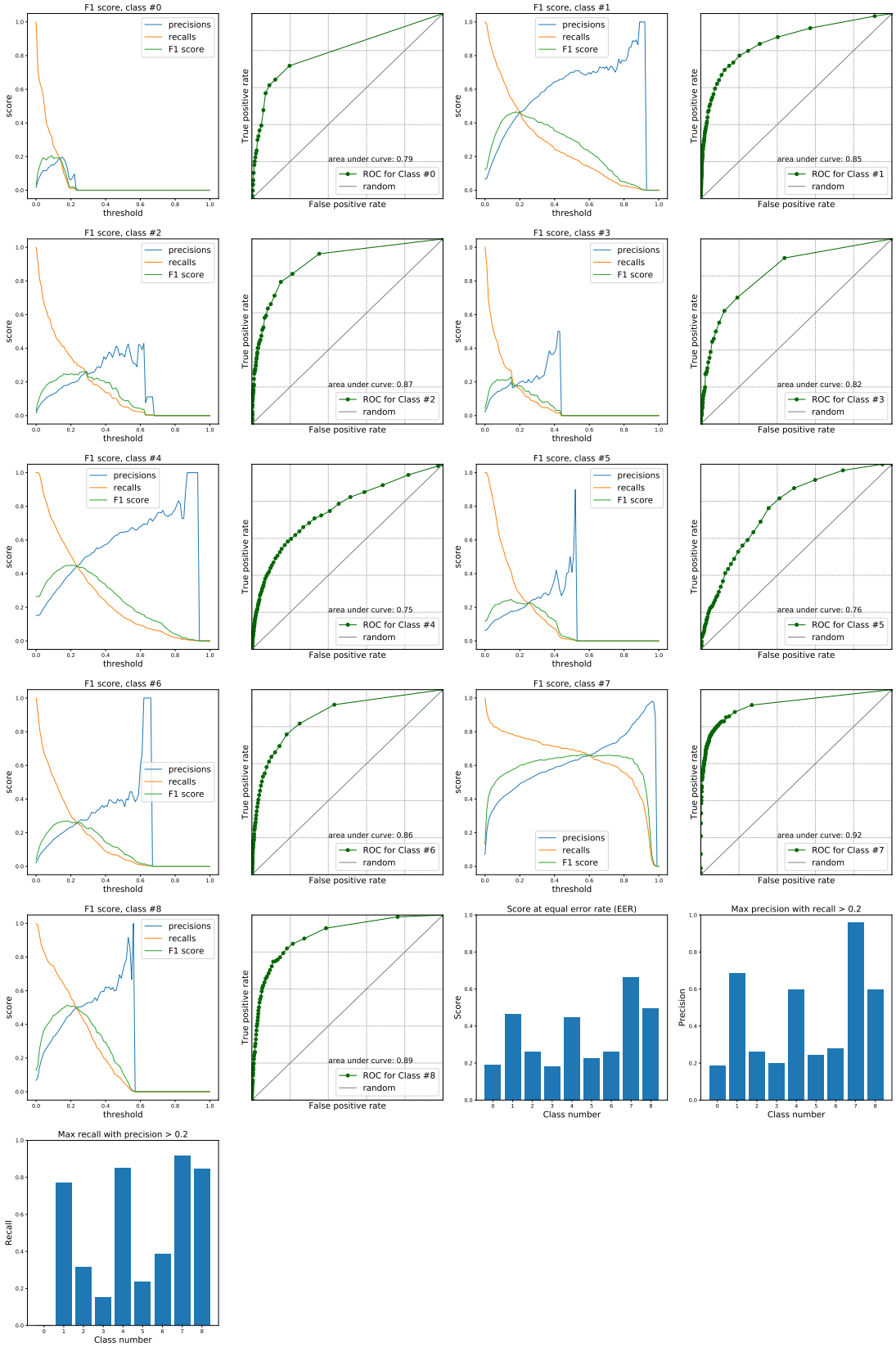

Figure A.19 – Results of the multihead attention model using 8 heads without punctuation

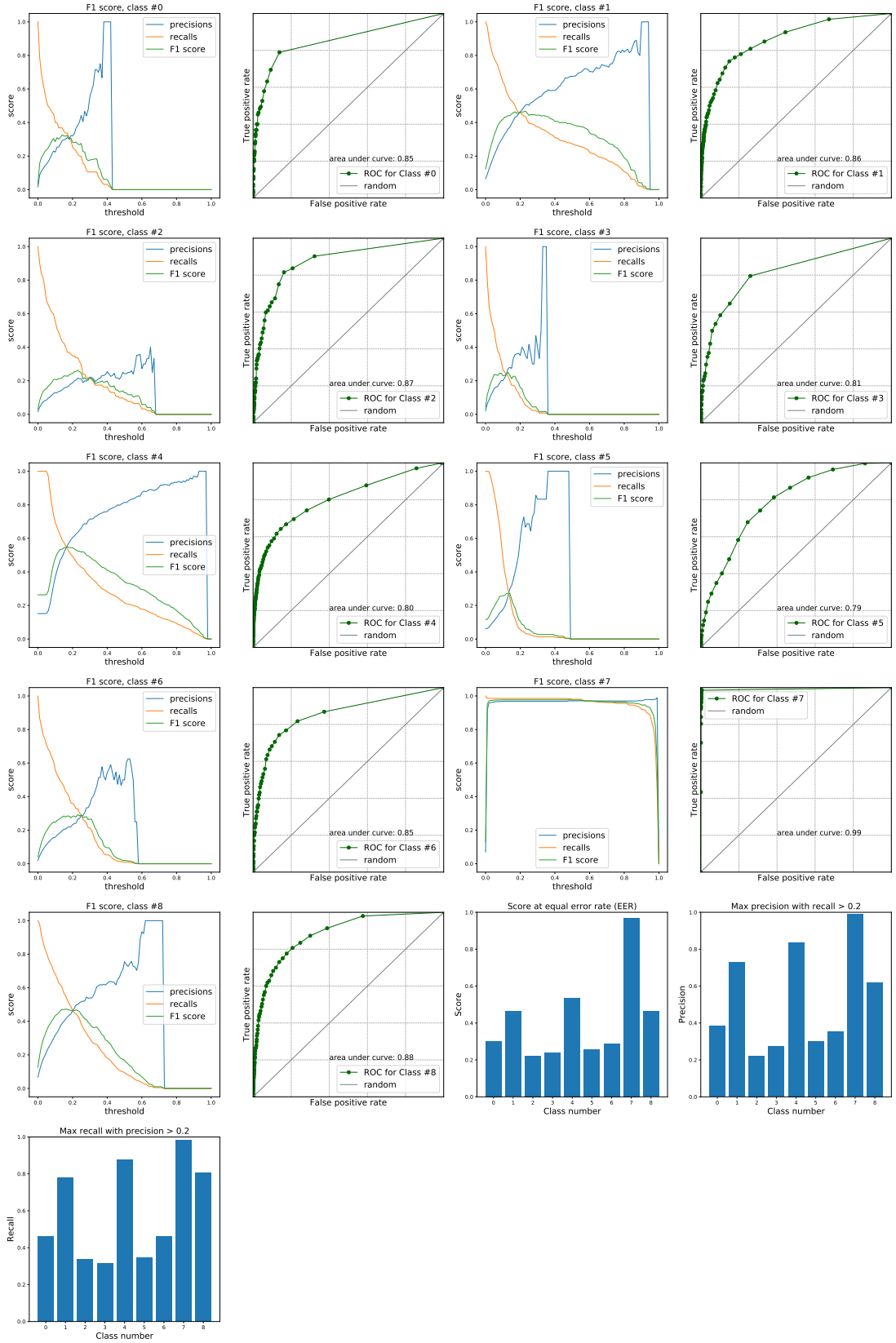

Figure A.20 – Results of the multihead attention model with duplicated input using 2 heads with punctuation

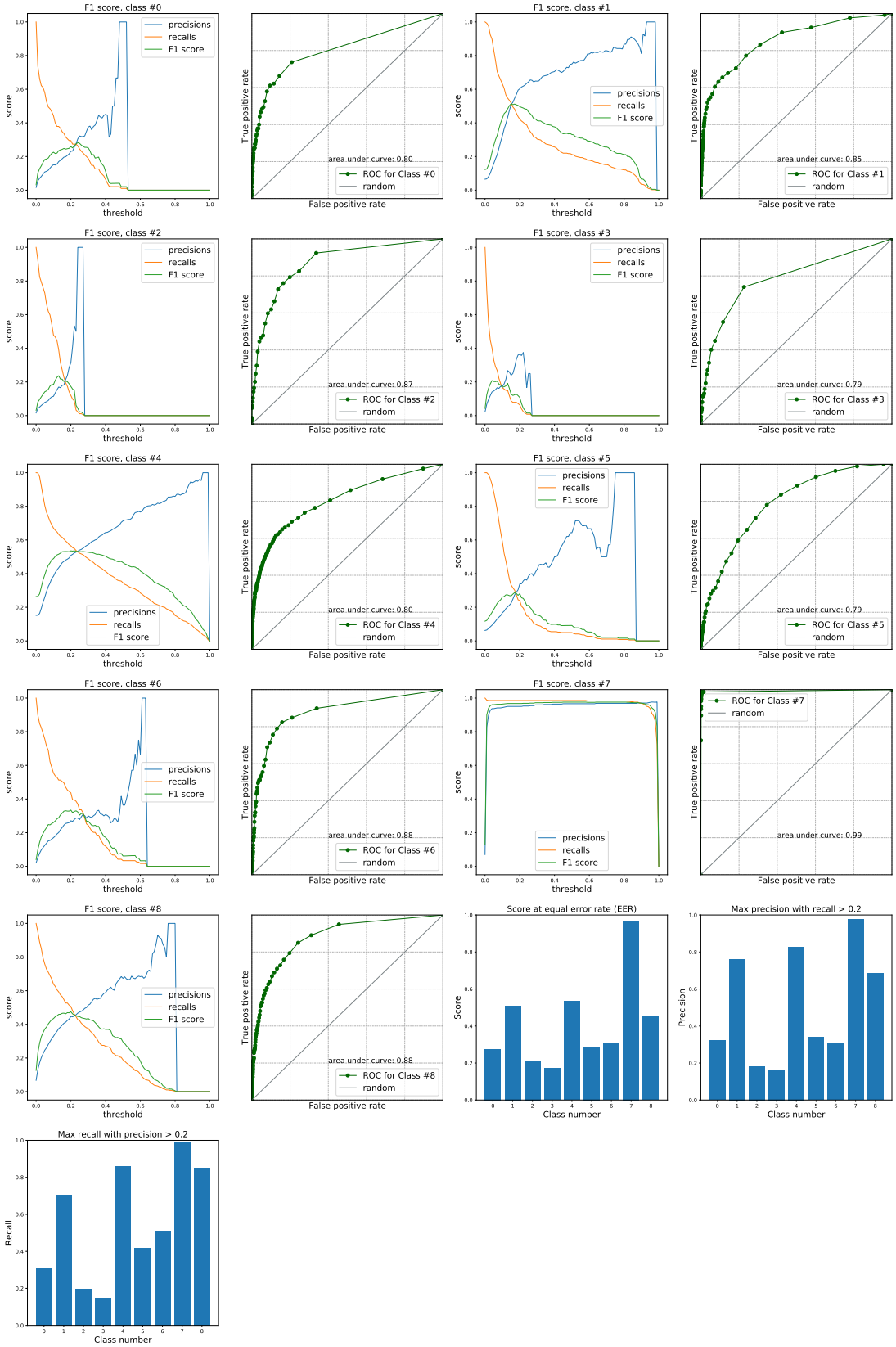

Figure A.21 – Results of the multihead attention model with duplicated input using 3 heads with punctuation

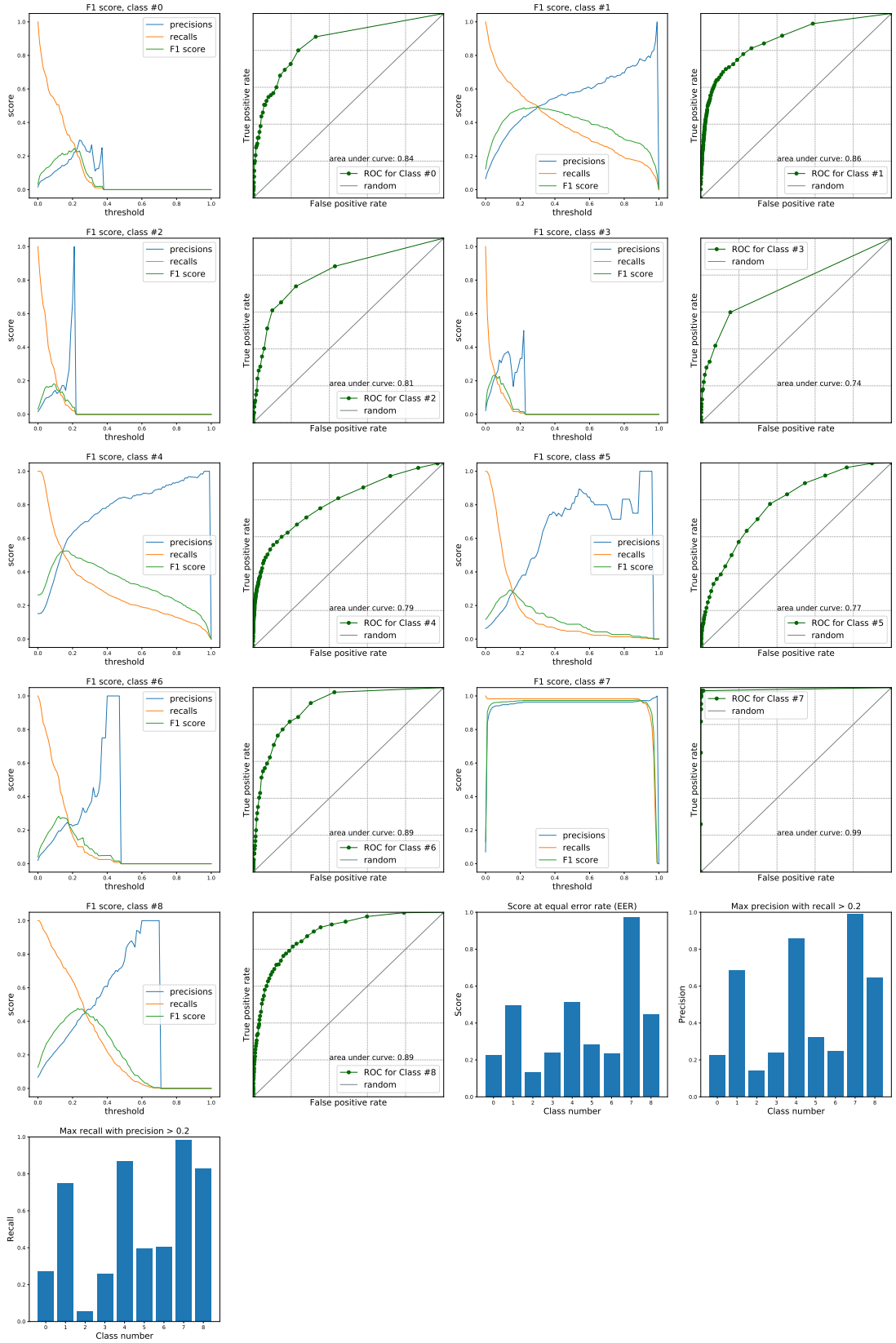

Figure A.22 – Results of the multihead attention model with duplicated input using 4 heads with punctuation

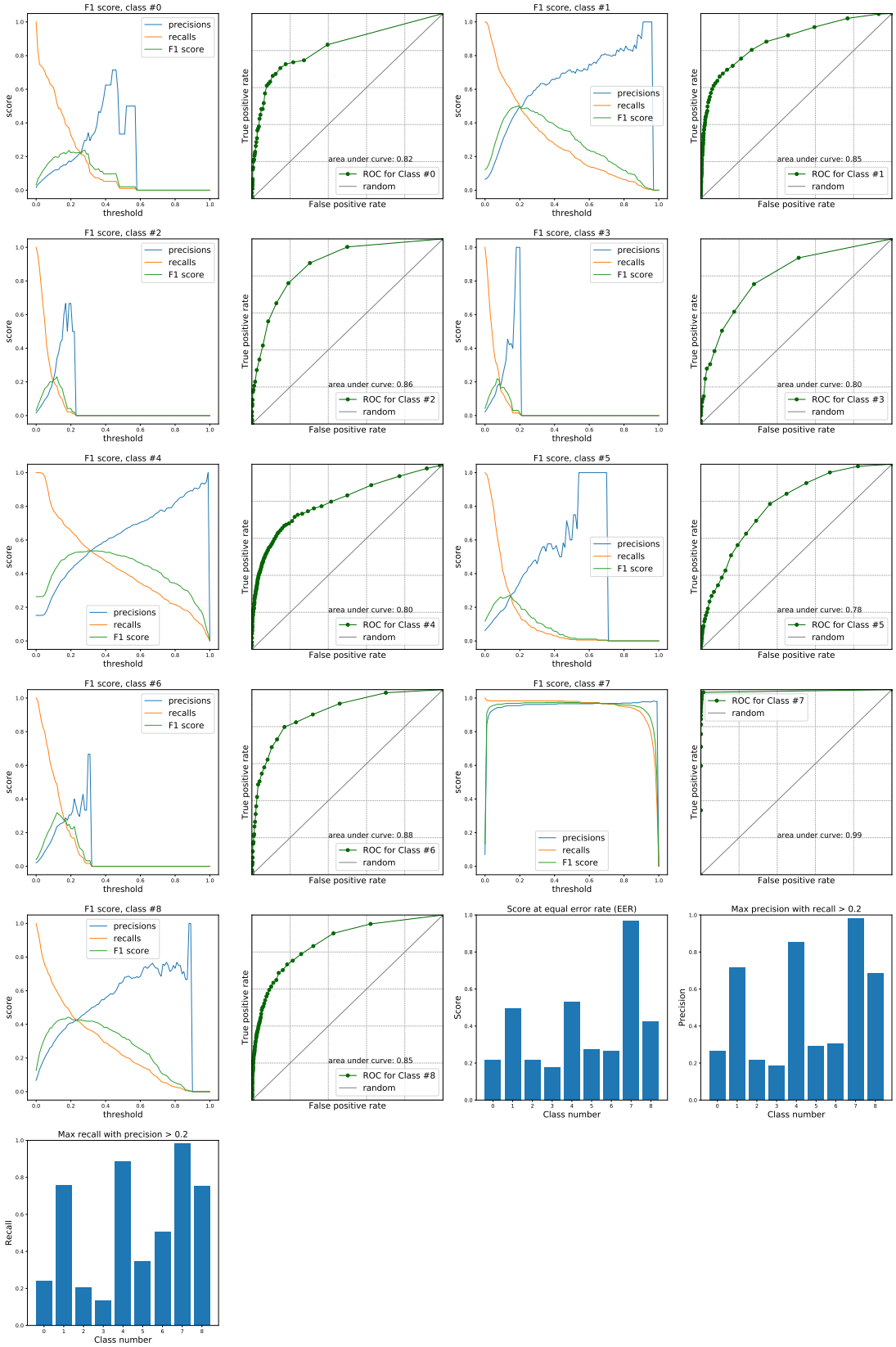

Figure A.23 – Results of the multihead attention model with duplicated input using 5 heads with punctuation

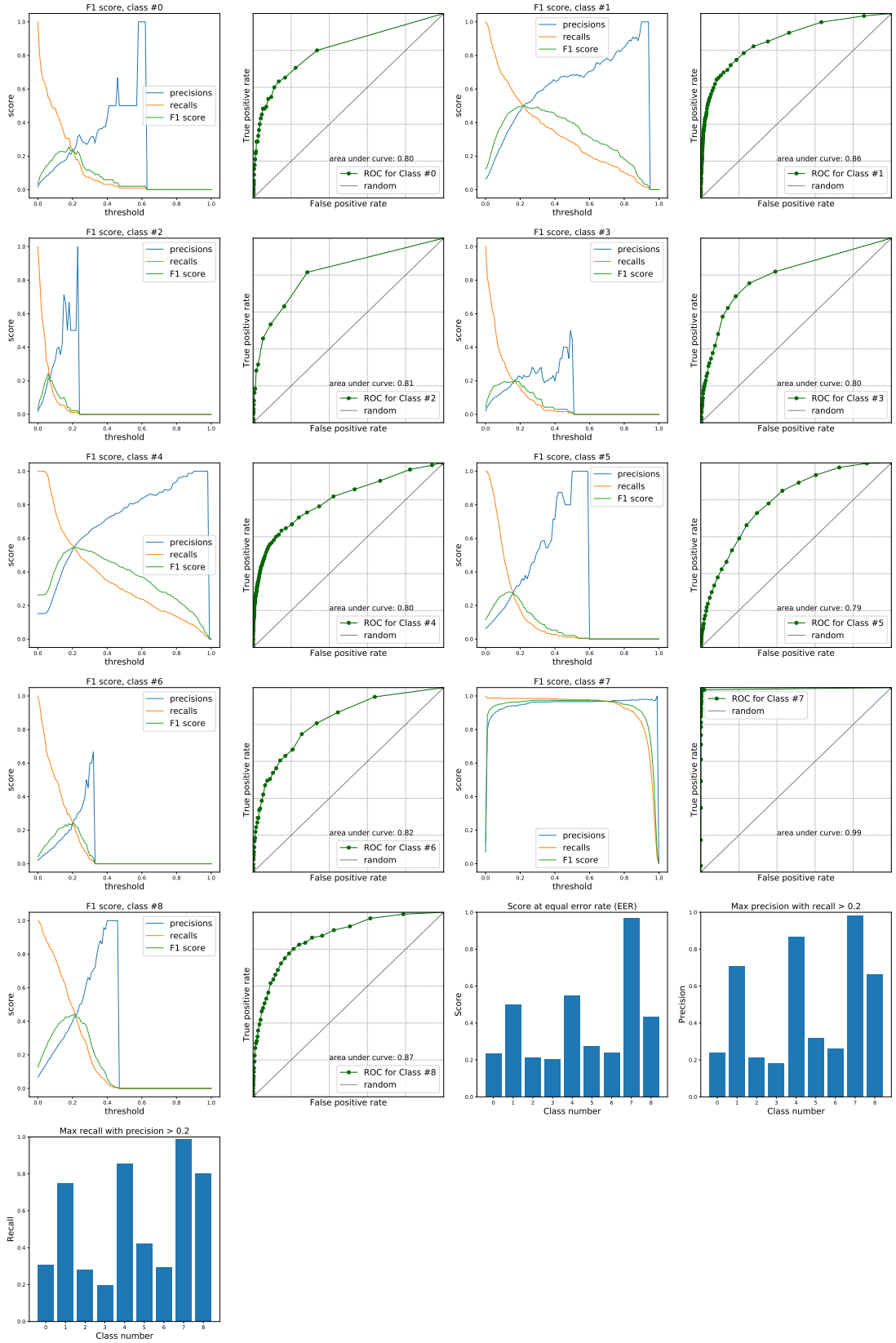

Figure A.24 – Results of the multihead attention model with duplicated input using 6 heads with punctuation

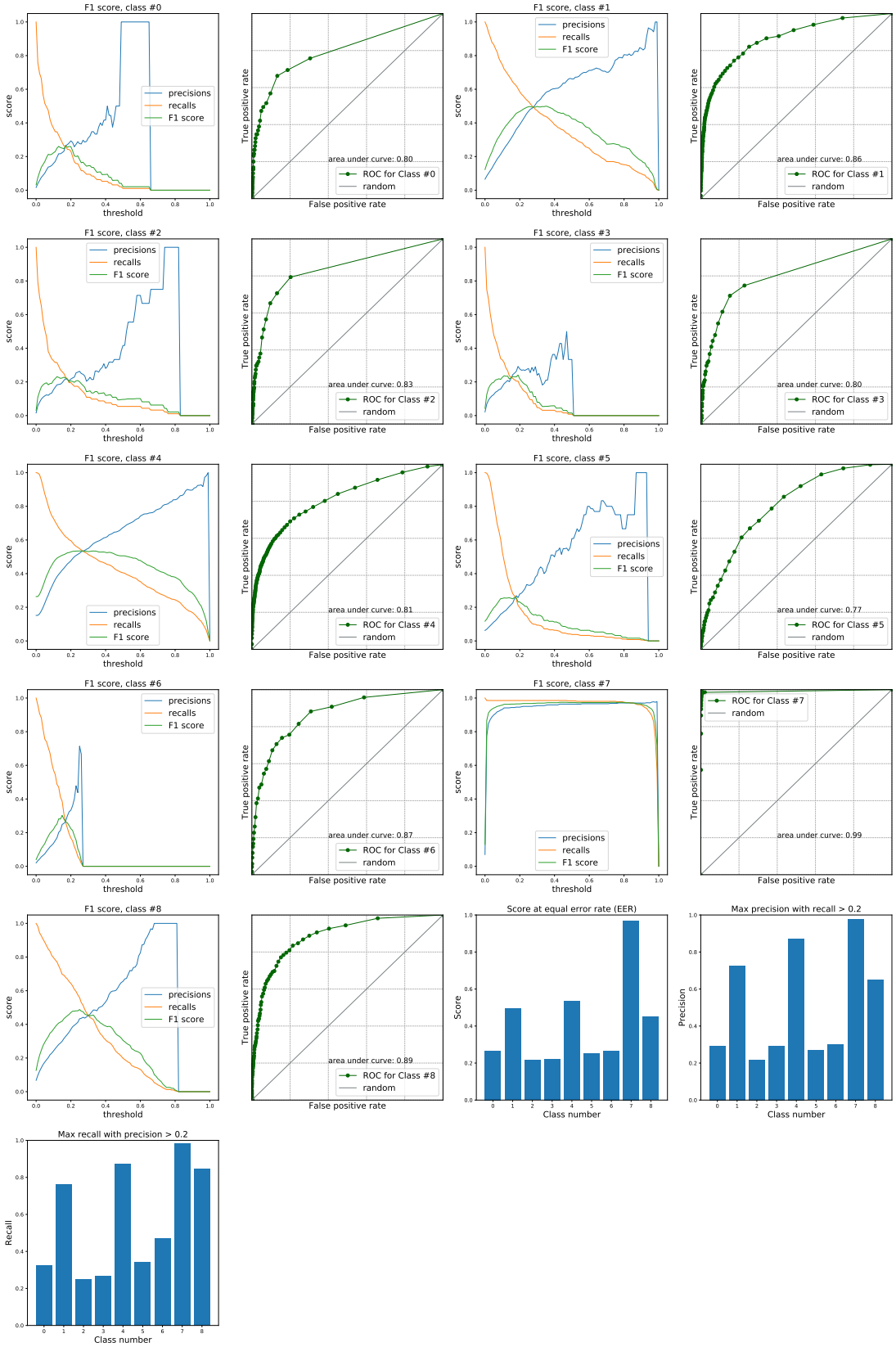

Figure A.25 – Results of the multihead attention model with duplicated input using 7 heads with punctuation

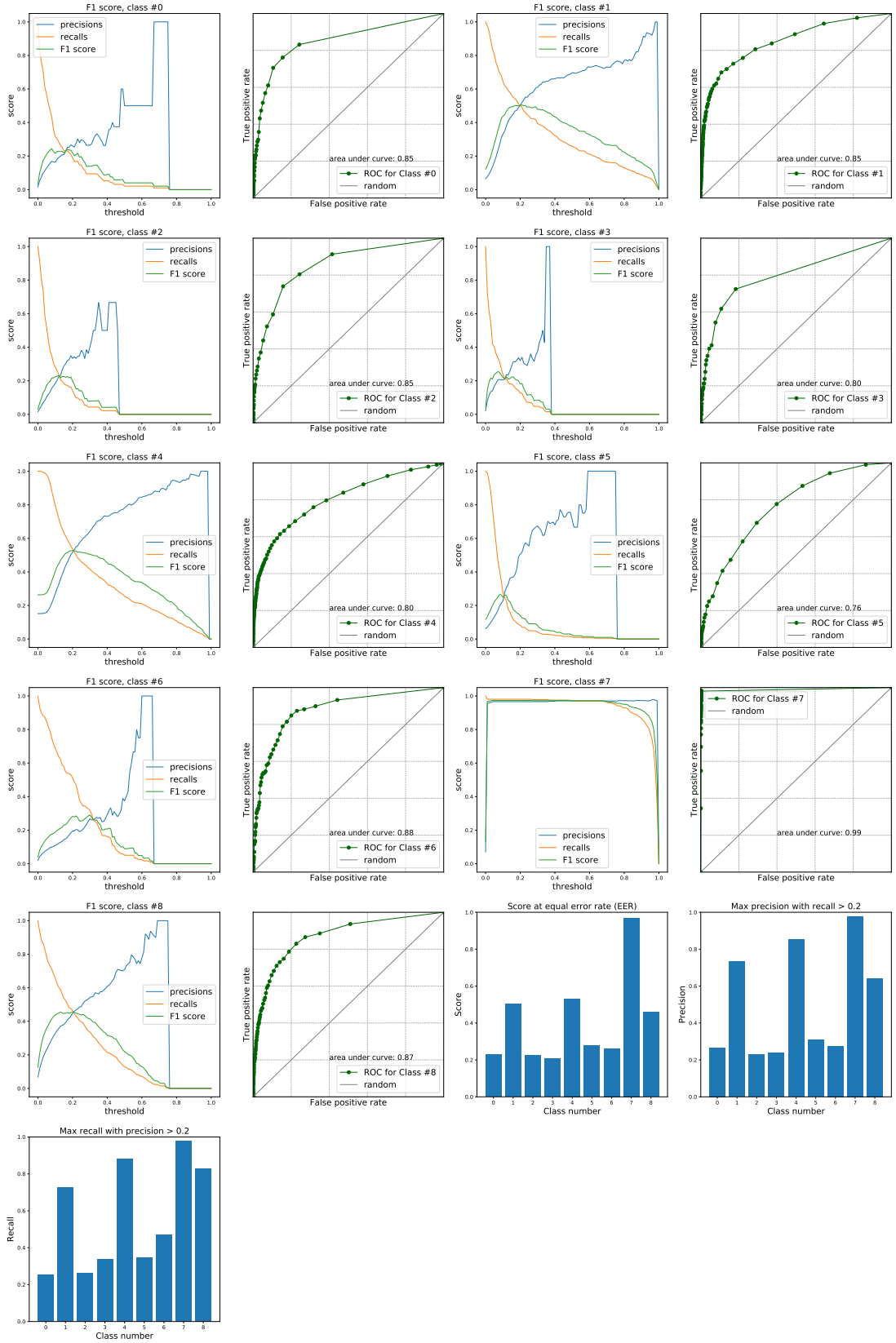

Figure A.26 – Results of the multihead attention model with duplicated input using 8 heads with punctuation

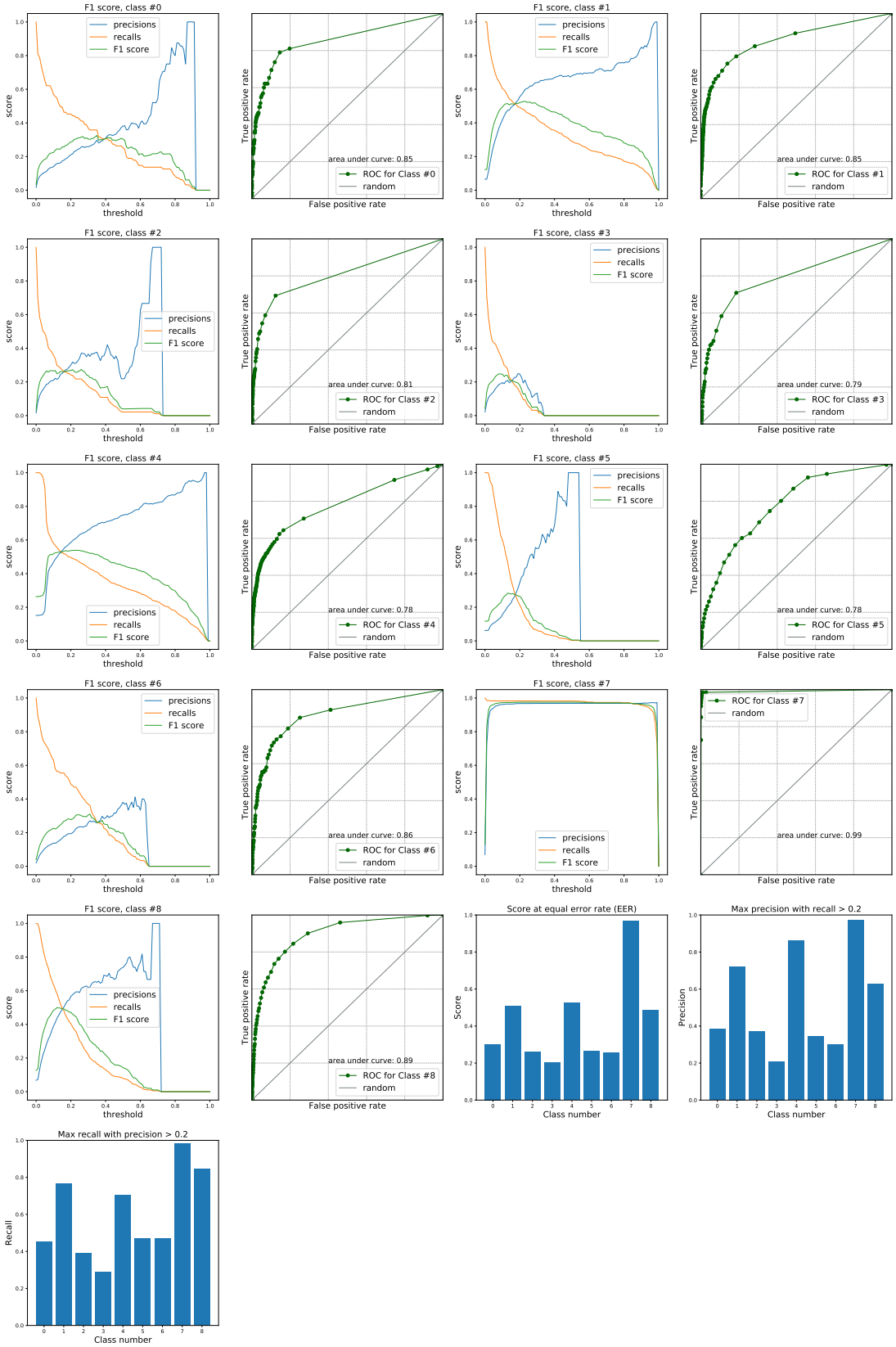

Figure A.27 – Results of the multihead attention model with linearly projected input using 2 heads with punctuation

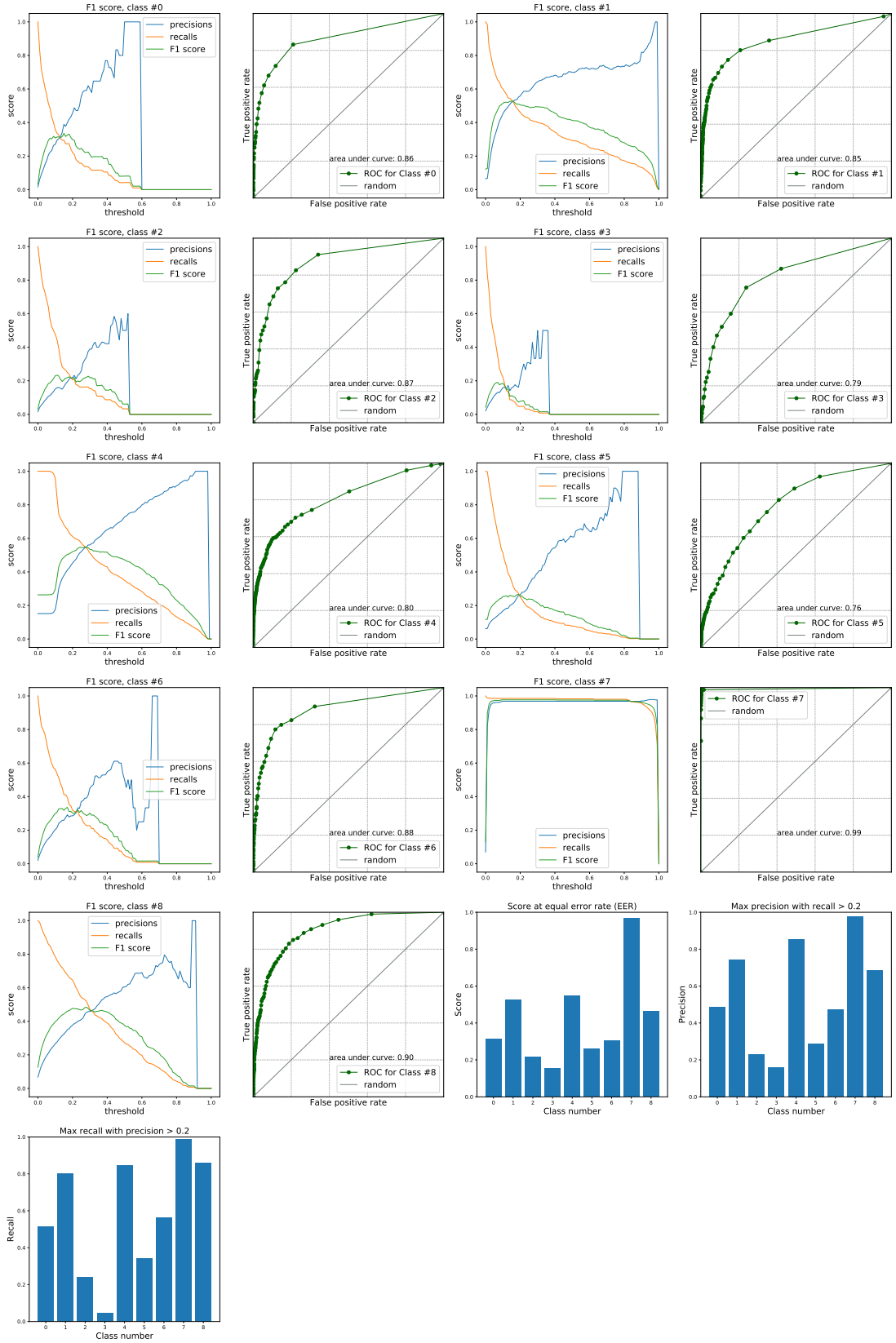

Figure A.28 – Results of the multihead attention model with linearly projected input using 3 heads with punctuation

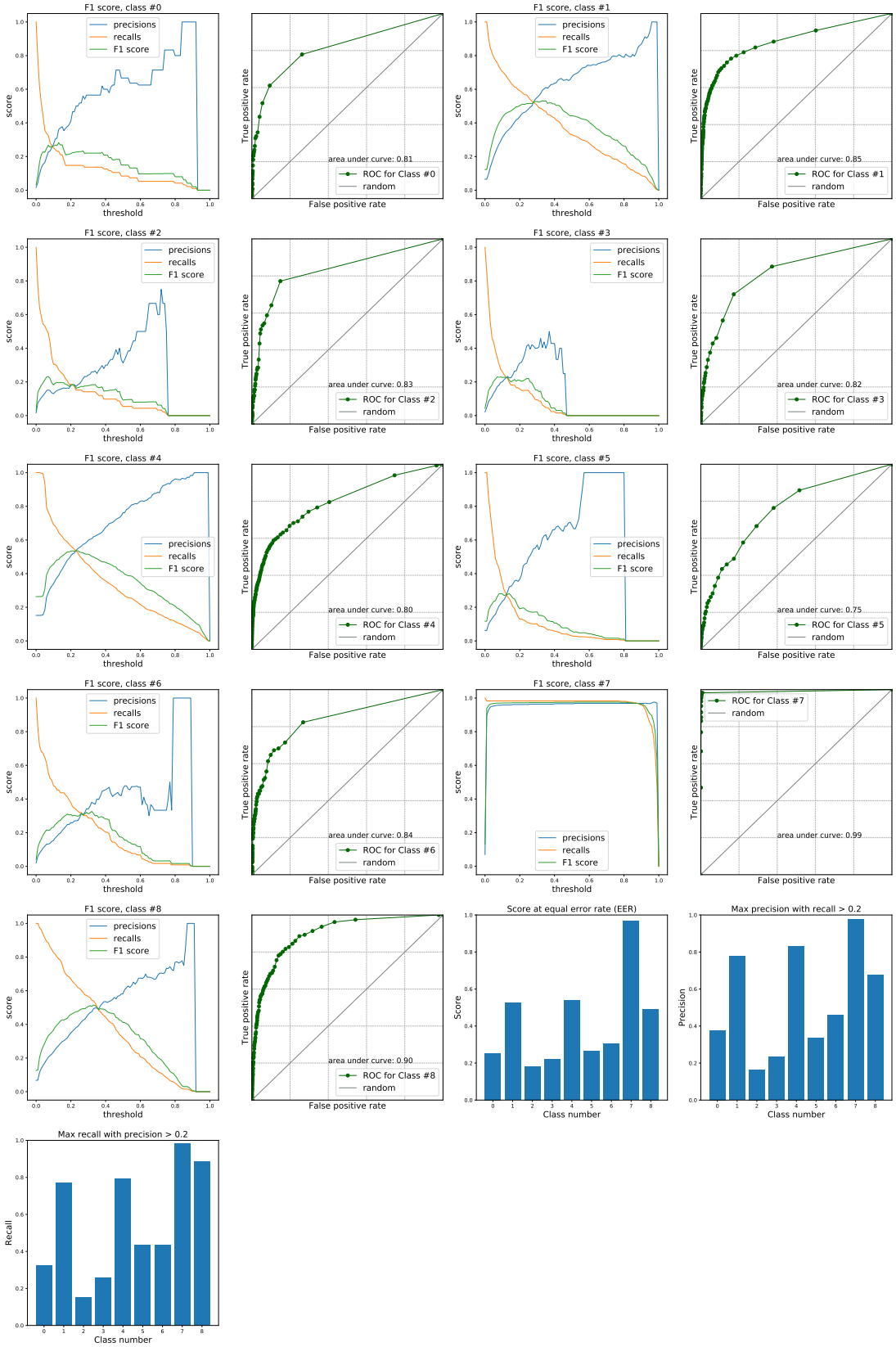

Figure A.29 – Results of the multihead attention model with linearly projected input using 4 heads with punctuation

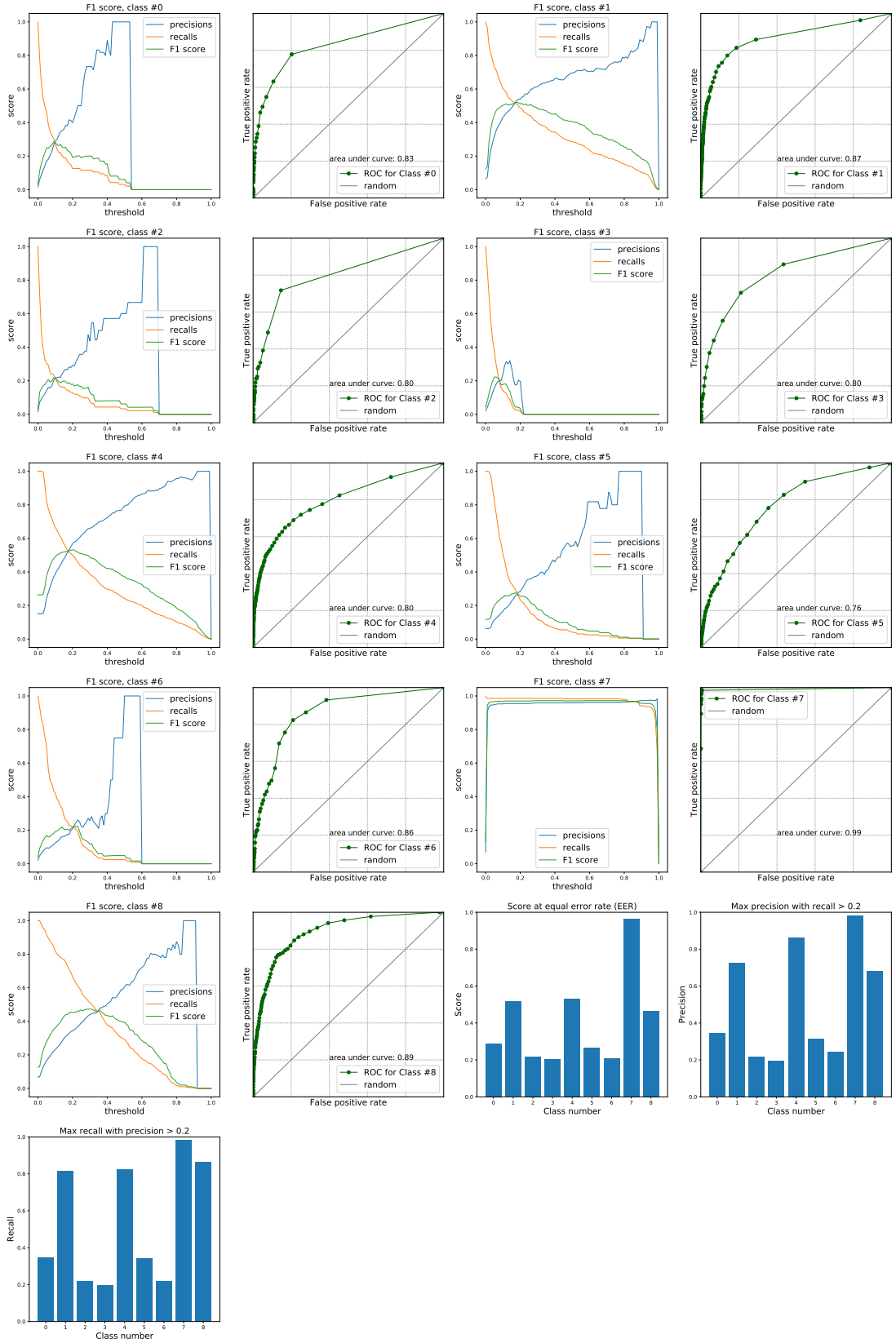

Figure A.30 – Results of the multihead attention model with 14 learly $_{p}roje ctedinputusing5headswithpunctuation$

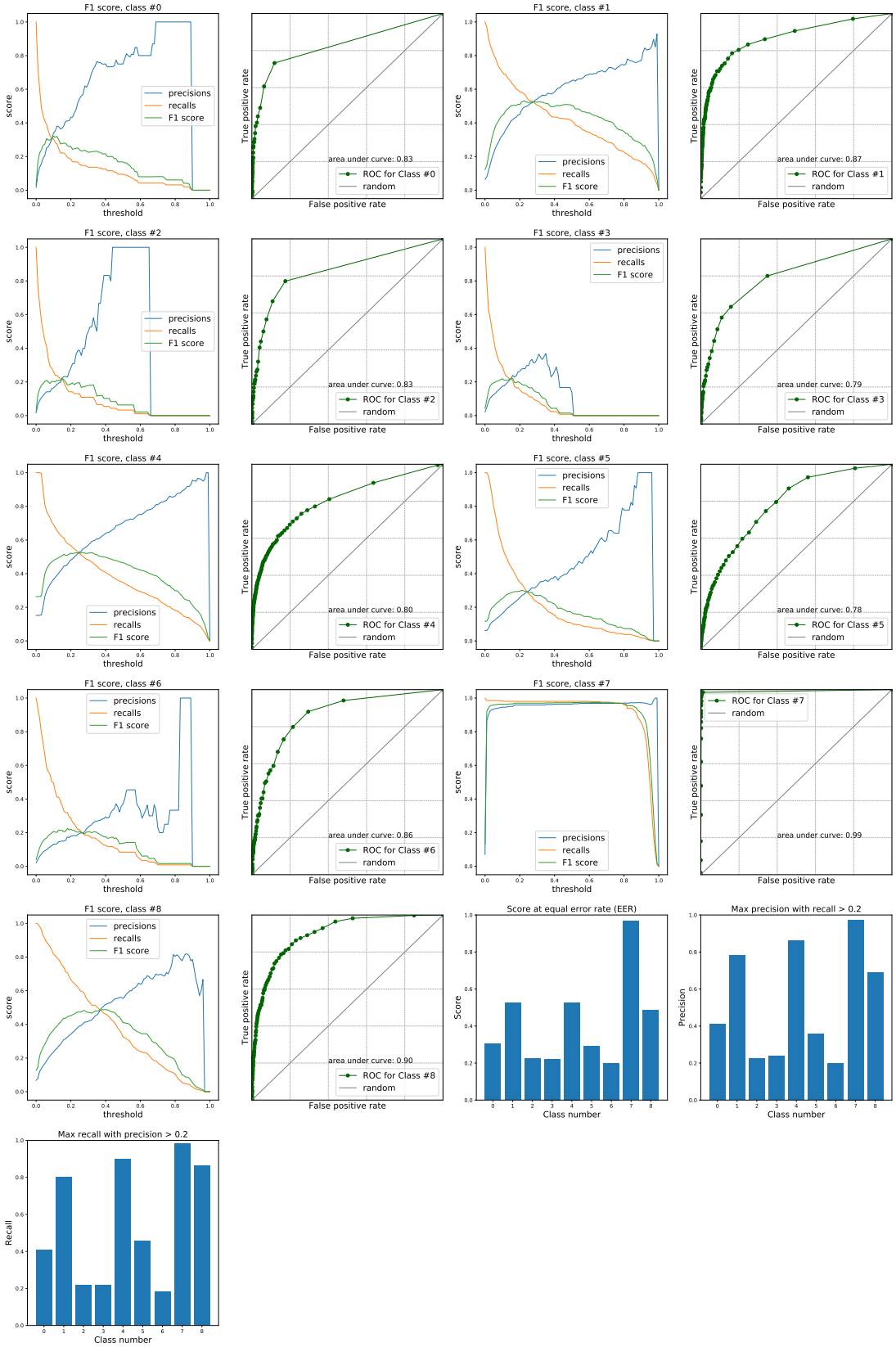

Figure A.31 – Results of the multihead attention model with linearly projected input using 6 heads with punctuation

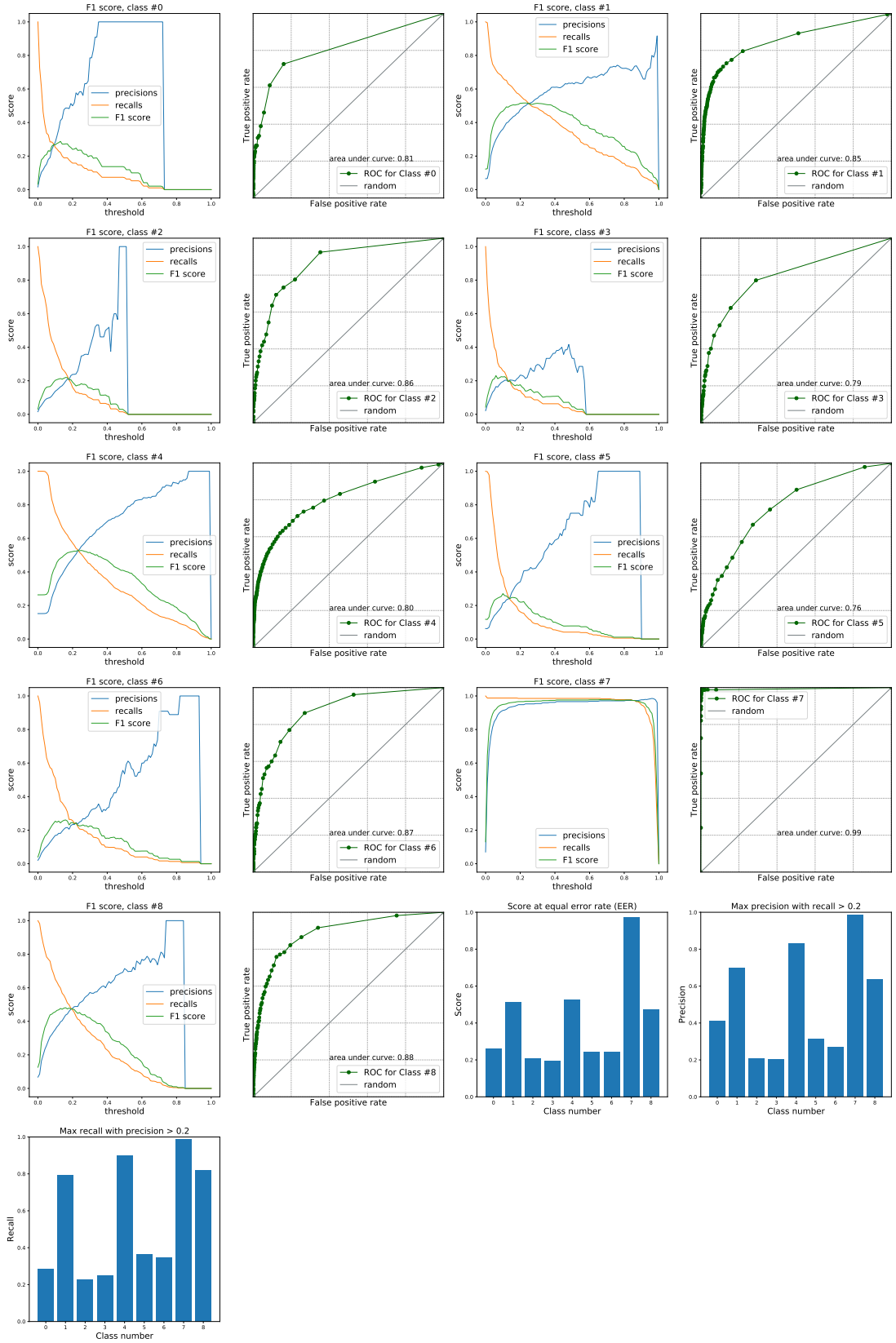

Figure A.32 – Results of the multihead attention model with linearly projected input using 7 heads with punctuation

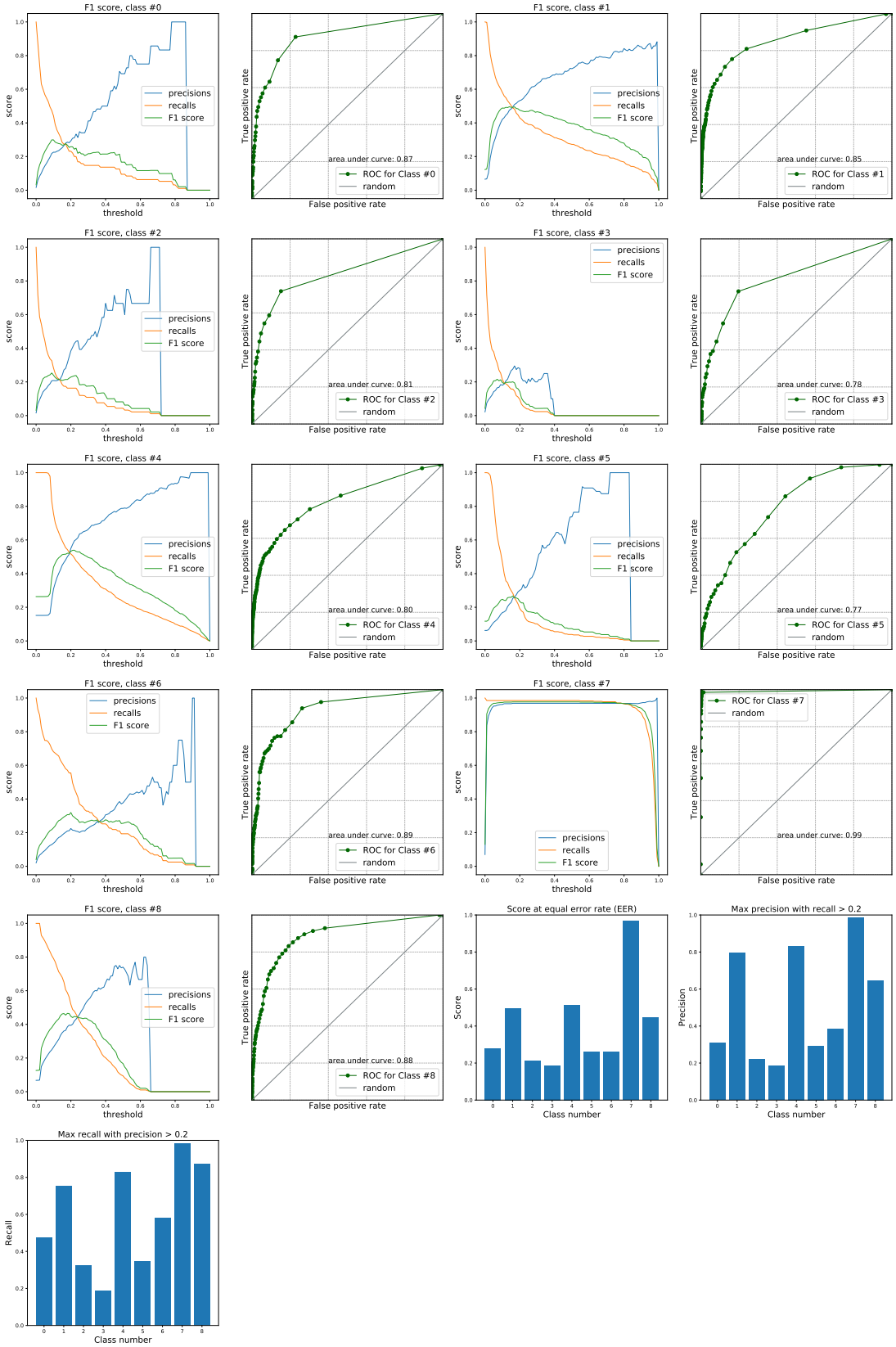

Figure A.33 – Results of the multihead attention model with linearly projected input using 8 heads with punctuation

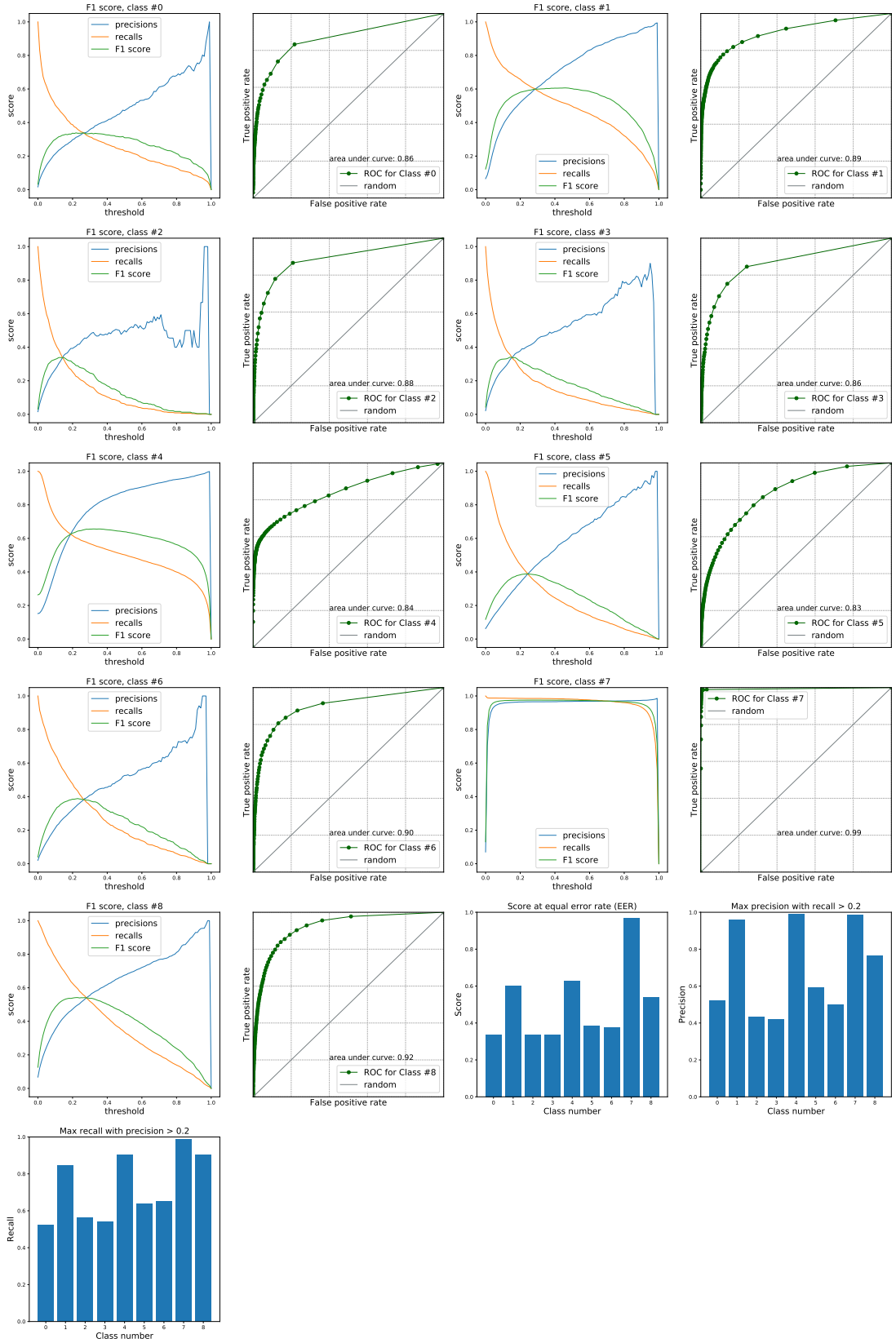

Figure A.34 – Results of the BERT model with punctuation

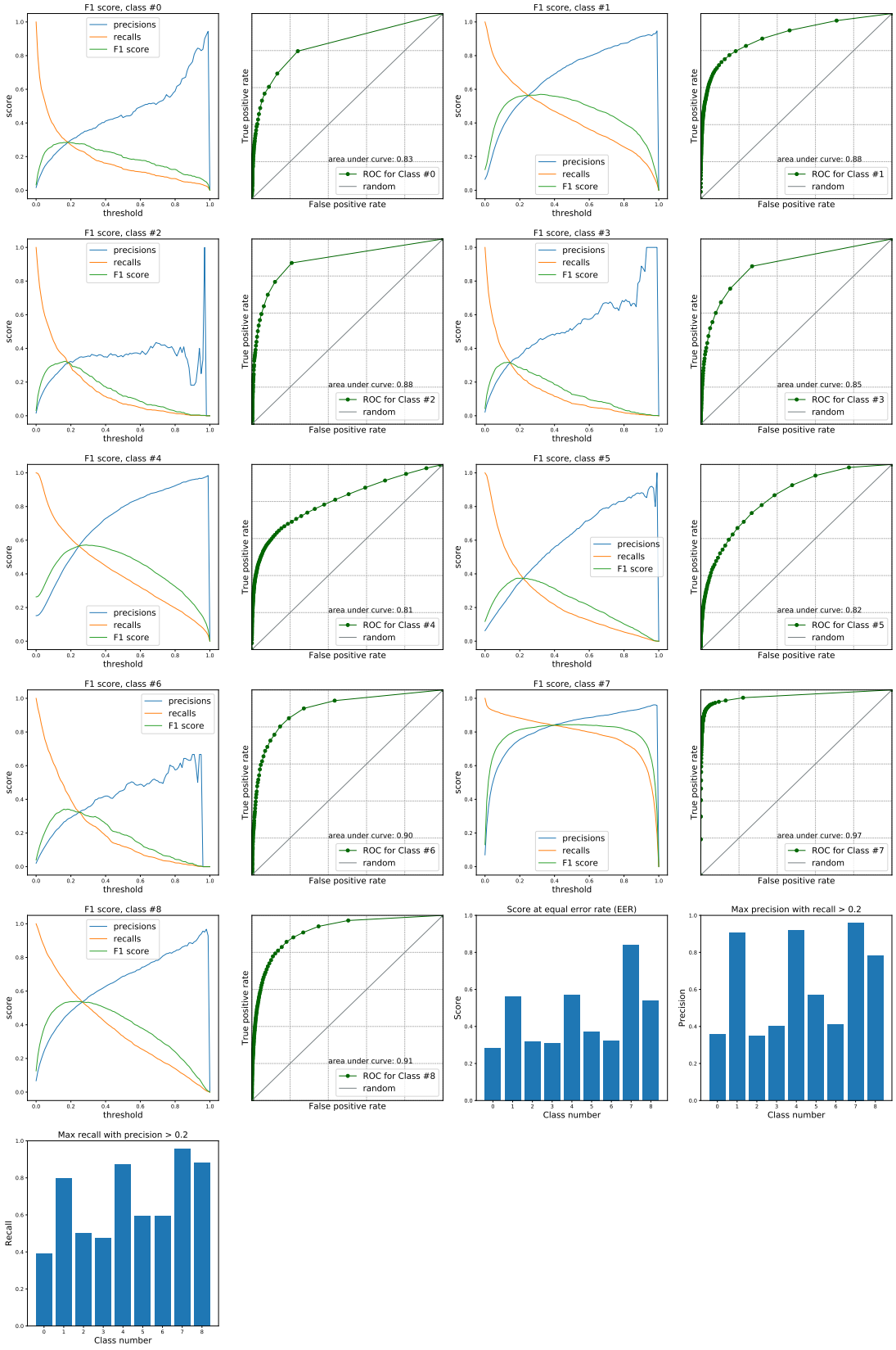

Figure A.35 – Results of the BERT model without punctuation

Bibliography

[1] Philippe Jacquart and John Antonakis. When does charisma matter for top-level leaders? effect of attributional ambiguity. *The Academy of Management Journal*, 06 2014.

[2] George C. Banks, Krista N. Engemann, Courtney E. Williams, Janaki Gooty, Kelly Davis McCauley, and Melissa R. Medaugh. A meta-analytic review and future research agenda of charismatic leadership. *The Leadership Quarterly*, 28(4):508 – 529, 2017. Charisma: New frontiers. A special issue dedicated to the memory of Boas Shamir.

[3] John Antonakis, Nicolas Bastardoz, Philippe Jacquart, and Boas Shamir. Charisma: An ill-defined and ill-measured gift. *Annual Review of Organizational Psychology and Organizational Behavior*, 3(1):293–319, 2016.

[4] John Antonakis. Charisma and the "new leadership". *In John Antonakis and David V. Day, editors, The Nature of Leadership*, page 56–81.

[5] John Antonakis, Marika Fenley, and Sue Liechti. Can charisma be taught? tests of two interventions. *Academy of Management Learning & Education*, 10(3):374–396, 2011.

[6] Tomas Mikolov, Martin Karafiát, Lukas Burget, Jan Cernocký, and Sanjeev Khudanpur. Recurrent neural network based language model. volume 2, pages 1045–1048, 01 2010.

[7] Tomas Mikolov, Martin Karafiát, Lukas Burget, Jan Cernocký, and Sanjeev Khudanpur. Recurrent neural network based language model. volume 2, pages 1045–1048, 01 2010.

[8] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need, 2017.

[9] Philip N. Garner, John Antonakis, Olivier Bornet, Dimitra Loupi, and Dominic Rohner. Deep learning of charisma. Idiap-RR Idiap-Internal-RR-47-2018, Idiap, July 2018.

[10] Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. Efficient estimation of word representations in vector space, 2013.

[11] Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg Corrado, and Jeffrey Dean. Distributed representations of words and phrases and their compositionality, 2013.

[12] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural Computation*, 9(8):1735–1780, 1997.

[13] M. Schuster and K.K. Paliwal. Bidirectional recurrent neural networks. *Trans. Sig. Proc.*, 45(11):2673–2681, November 1997.

[14] Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. Neural machine translation by jointly learning to align and translate, 2014.

[15] Jimmy Lei Ba, Jamie Ryan Kiros, and Geoffrey E. Hinton. Layer normalization, 2016.

[16] Yinhan Liu, Myle Ott, Naman Goyal, Jingfei Du, Mandar Joshi, Danqi Chen, Omer Levy, Mike Lewis, Luke Zettlemoyer, and Veselin Stoyanov. Roberta: A robustly optimized bert pretraining approach, 2019.

[17] Zhenzhong Lan, Mingda Chen, Sebastian Goodman, Kevin Gimpel, Piyush Sharma, and Radu Soricut. Albert: A lite bert for self-supervised learning of language representations, 2019.

[18] Zhilin Yang, Zihang Dai, Yiming Yang, Jaime Carbonell, Ruslan Salakhutdinov, and Quoc V. Le. Xlnet: Generalized autoregressive pretraining for language understanding, 2019.