— GUIDE —

Assignment 5: Classify Math Publications

AI-2 Systems Project (Summer Semester 2025)

Jan Frederik Schaefer

Friedrich-Alexander-Universität Erlangen-Nürnberg, Department Informatik

This document is intended to help you solve the assignment "Assignment 5: Classify Math Publications" [AS]. You do not have to read it, but we do recommend to at least take a look at the tips and common issues.

1 A few tips

1. There are many parameters you can change and (especially without much experience) it is hard to tell, what effect an adjustment has. It can help to take notes on the performance of different configurations to inform future experiments.
2. Take a look at the data in the various preprocessed forms.
3. To see the formulas visually, you can place them in an HTML file.

2 Common issues

1. Not submitting the solutions for the test dataset as specified in the *Evaluation* section.

3 Tutorial for getting started

3.1 Step 1: Preprocessing the formulas

Neural networks require numerical input, so our first task is to convert the MathML formulas to a numerical representation.

For this, we will tokenize the formulas. A similar thing is done in natural language processing, where texts are tokenized into words. For example, the sentence "This is a sentence." could be tokenized into the list of words `["This", "is", "a", "sentence", "."]`. Tokens are sometimes further simplified e.g. by lowercasing them, which results in fewer unique tokens (“This” and “this” would be the same token).

There are different ways to tokenize MathML formulas, and we leave it up to you to come up with a good way (or better, try several ways and see what works best). Tips:

- You can use an XML parser to parse the MathML formulas and then work with the resulting tree to create the tokens.
- Take a look at the output of your tokenizer to see if it makes sense.
- Think about what information is important and what can be ignored.
- It’s important to have a consistent tokenization across all formulas.
- Usually, many tokens occur only once or twice in the dataset. It can be helpful to group these tokens together. For example, you could replace all tokens that occur only once with a special token `"<UNK>"`.

Next, we can convert the tokens to numbers from 1 to n. Every token is then represented by a number, and the formula is represented by a list of numbers. To make all lists the same length, we can pad them with zeros – or truncate them if they are too long.

These lists of numbers will be the input to the neural network.

3.2 Step 2: Building the neural network

For this task, we will use keras, which offers a popular high-level API for building neural networks. This tutorial will not cover all the details and you are encouraged to take a look at the documentation and tutorials on the keras website [Ker].

The following code snippet is based on a keras example for text classification [KTC], but some modifications and simplifications have been made.

```
import os
os.environ["KERAS_BACKEND"] = "jax"
import keras
import numpy as np

# Example input/output data
X = np.array([[2, 1, 3, 3, 1, 0, 0], [1, 3, 2, 2, 1, 1, 0]])
Y = np.array([[0, 1], [1, 0]])

# Create a model
```

```
model = keras.models.Sequential(
    [
        keras.layers.Input(shape=(None,)),
        keras.layers.Embedding(input_dim=4, output_dim=2),
        keras.layers.Dropout(0.3),
        keras.layers.Conv1D(3, 2, padding="valid", activation="relu"),
        keras.layers.Conv1D(3, 2, padding="valid", activation="relu"),
        keras.layers.GlobalMaxPooling1D(),
        keras.layers.Dense(3, input_dim=2, activation='relu'),
        keras.layers.Dropout(0.3),
        keras.layers.Dense(2, activation='sigmoid'),
    ]
)

# Compile and train the model
model.compile(loss='binary_crossentropy', optimizer='adam', metrics=['accuracy'])
model.fit(X, Y, epochs=100, batch_size=2)

# Test the model
X2 = np.array([[2, 1, 3, 3, 1, 0, 0], [1, 3, 2, 1, 2, 1, 0]])
prediction = model.predict(X2)
```

To run the code, you will need to install keras and jax, e.g. by running pip install keras jax.

After a few imports, the input and output data are defined. Then the model is defined and trained. In the end, the model is used to make predictions on new data.

The interesting part is the model definition. You can find a detailed explanation of the different layers in the keras documentation, but here are the key ideas:

- The Embedding layer is used to map the input tokens to vectors. The input_dim parameter is the number of unique tokens, and the output_dim parameter is the dimension of the output vectors.
- The Dropout randomly sets a fraction of the input units to zero during training. This helps to prevent overfitting.
- The Conv1D layer is a 1D convolutional layer. It iterates over the input and applies a filter to each window of the input. This is useful for finding patterns in sequences. If you are not familiar with convolutional neural networks, you might want to read up

on them to get a better understanding.

- The GlobalMaxPooling1D layer takes the maximum value of each feature over the sequence. That way, the sequence is reduced to a single vector.
- The Dense layers are fully connected layers (i.e. the most basic type of neural network layer).

The parameters of the layers will have to be adjusted to fit the formula dataset (this way, the network would be way too small for the actual task). You may also want to experiment with different architectures, by adding/removing layers.

References

[AS] *Assignment 5: Classify Math Publications*. URL: https://kwarc.info/teaching/AISysProj/SS25/assignment-2.5.pdf.

[Ker] *Getting started with Keras*. URL: https://keras.io/getting_started/ (visited on 01/24/2025).

[KTC] *Text classification from scratch*. URL: https://keras.io/examples/nlp/text_classification_from_scratch/ (visited on 01/24/2025).