Text Classification in Natural Language Processing Using Neural Networks

Using neural networks to create a natural language processing model that classifies news articles into four different categories

Mila Pourali

Vanier College

360-HTB-VA: Research Methods for the Sciences

Professor Julie Plante

Dr. Joshua Wyatt Smith

May 11, 2021

Abstract

Natural language processing is a subfield of artificial intelligence that processes human languages for computers to understand. Text classification is one of many tasks done in this field. Using neural networks containing multiple layers and features, supervised machine learning can train a nonlinear model to classify text. This research paper details on the training of *spaCy*'s 'nlp' pipeline with its small English language model to perform multiclass classification of news articles into four different categories: "World News", "Sports News", "Business News", "Science-Technology News". The trained model does not overfit the data and it has an accuracy of 91% on previously unseen data. The performance of the model could be further improved through the exploration of larger language models, lengthier training sessions, and further hyperparameter optimisations.

Introduction

Amongst the various rapidly evolving fields of computer science, natural language processing (NLP) has seen a lot of development come its way through the advancements of neural networks. During the past decade, the world has seen a rise in virtual assistants like Siri, Google, and Cortana, highly specific search engine predictions like auto-complete and auto-correct, as well as translation services such as Google Translate. NLP plays a huge role in everyday lives. Text classification is one such subcategory of tasks in NLP. This type of task has numerous useful applications for businesses like sentiment-analysis over various social media platforms, or archival organisation of numerous documents. In this research project, we propose a methodology to categorize news articles into four different categories using neural networks.

Bibliographical Review

NLP relies on computers to complete tasks. However, computers do not understand natural languages the way that humans do. Computers understand numbers. In order for the machine learning model to understand the input at a very basic level, datasets containing textual information are transformed using features that are carefully engineered. For example, paragraphs are broken down into sentences, which are then “tokenized” (Eisenstein, 2018, p. 19) to break them down into words. These words can be represented using “one-hot encodings” (ElDen, 2019) in a matrix of size nxn such as in Figure 1:

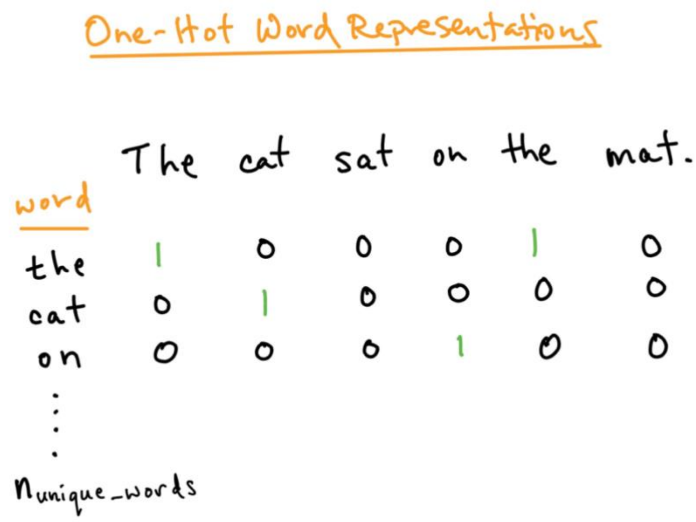

Figure 1 - Example of one-hot encoding representations (ElDen, 2019).

Using this same example, by letting x_j be the count of a word j in the text document above, the sum of all occurrences of each j can then be represented by the column matrix $x = [2, 1, 1, 1, 1]^T$ (T indicates to take the transpose of the 1x5 row matrix and turn it into a 5x1 column matrix (Nicholson, 2019)). This is called a bag-of-words representation, which can be used to predict a label using scores called weights assigned to each specific word. Bag-of-words representation is used in linear text classification (Eisenstein, 2018, p. 13). Linear text classification implies that the entire dataset of labels can be separated by a straight line, as represented in Figure 2:

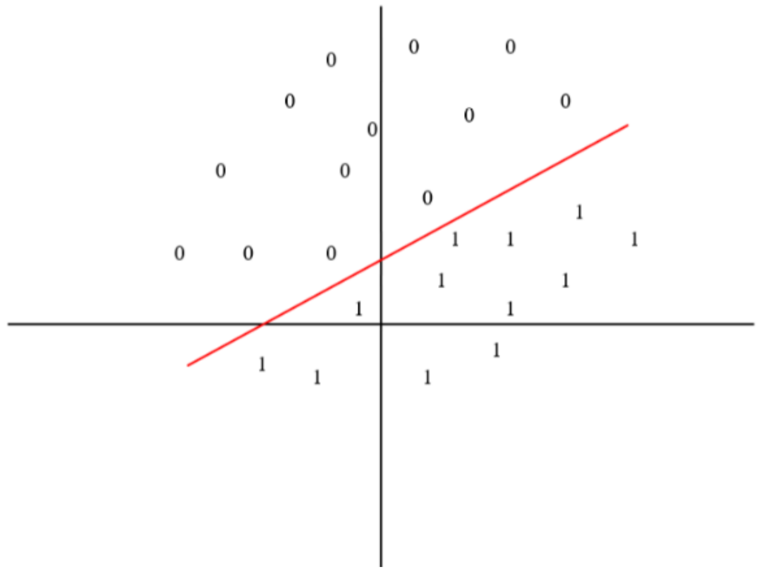

Figure 2 - Example of linear separation with binary labels ($y \epsilon \{0, 1\}$) classification algorithm.

Some datasets, however, cannot be separated linearly. For visualization purposes, Figure 3 represents a non-linearizable dataset:

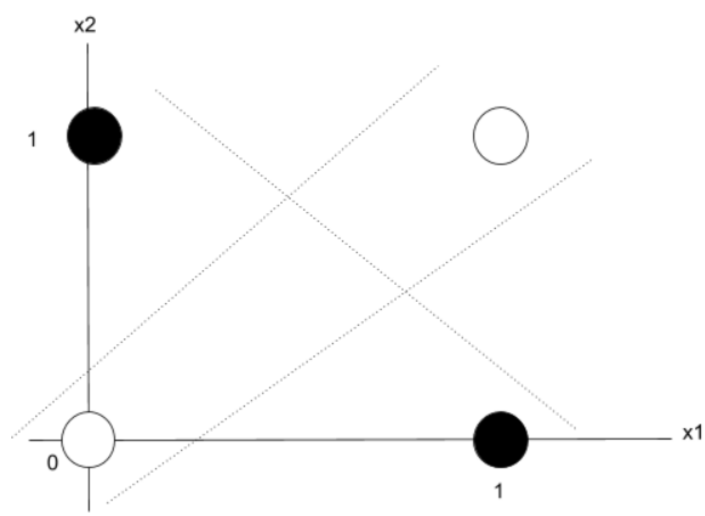

Figure 3 - Example of non-linearizable dataset with binary label (Addison, 2019).

More recently, there has been a wave of nonlinear classification using neural networks. Eisenstein's *Natural Language Processing* (2018) describes feedforward neural networks as pertinent to the creation of two-step classifiers. It lets x be the text, y be the label, and $z =$

$[z_1, z_2, \ldots, z_{K_z}]^T$ where each z is the labeled feature of a training dataset. The first step in creating such a network is to use x to predict z using a logistic regression classifier to establish $p(z_k \mid x)$ for each element k in the set of the matrix z (p. 48). This establishes the conditional probability of z_k given that x occurs (Kinney, 1996, p. 14). The second step is to use z to predict y to establish $p(y \mid z)$ using the same type of classifier. This creates the diagram in Figure 4, also known as a neural network:

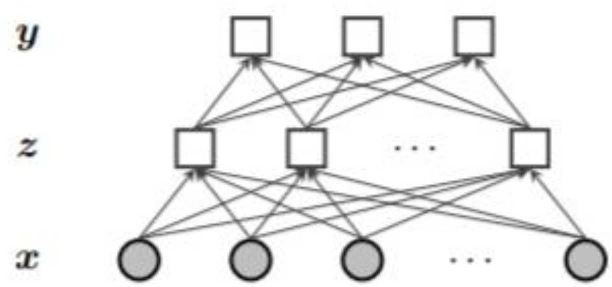

Figure 4 - Neural network representation (Eisenstein, 2018, p. 49).

There can be multiple layers of features in between x and y, and the layers can also be hidden, meaning that they are not directly observable and labeled in the training dataset (Eisenstein, 2018, p. 50). It also means that the parameters of each node are not manipulated directly. In one hidden layer, there could be a "set of latent features" (p. 50), meaning that each feature needs to be activated using an activation function that transforms the input. If there are multiple hidden layers, then the output of one layer (suppose it goes through an activation function) becomes the input for the following layer, defined by $\sum_{j=1}^{V} \theta_{j,k}^{(x \to z)} x_j$ where the word j becomes the vector $\theta_j^{(x \to z)}$ which is the embedding. The incorporation of the word j as the bag-of-words vector x_j approaches the hidden feature z_k (p. 53). Popular activation functions are the sigmoid function, the tanh function, and the rectified linear unit.

An example of a loss function that deep learning models use is the negative conditional log-likelihood in order to establish how well the model is performing:

$$-\mathcal{L} = -\sum_{i=1}^{N} \log p\left(y^{(i)} \middle| x^{(i)};\, \theta\right)$$

This computes the negative value of the summation over the entire dataset of the logarithmic value of the probability of the label y given the text x which both depend on the set of parameters θ (Eisenstein, 2018, p. 52).

Backpropagation is an algorithm that relies on the process of differentiating “the loss with respect to each parameter of the model” (p. 55) using the Chain Rule and combining it with sequencing and caching. The Chain Rule says that if the loss at an instance i ($\ell^{(i)}$) and the embedding vector of column n of the input layer weights matrix $\Theta^{(x\rightarrow z)}$ ($\theta_{n,k}^{(x\rightarrow z)}$) are both differentiable at z_k, then

$$\frac{\partial \ell^{(i)}}{\partial \theta_{n,k}^{(x\rightarrow z)}} = \frac{\partial \ell^{(i)}}{\partial z_k} \frac{\partial z_k}{\partial \theta_{n,k}^{(x\rightarrow z)}} = \frac{\partial \ell^{(i)}}{\partial z_k} \times f'\left(\theta_k^{(x\rightarrow z)} \cdot x\right) \times x_n$$

(Stewart, 2016, p. 198), where f is an activation function, and in a case where the loss does not depend on a specific feature z_k, then its derivative will be roughly equal to 0 (Eisenstein, 2018, p. 55). If that’s the case, then the model knows to stop learning. According to Eisenstein, “backpropagation computes the gradients with respect to all the variables [z, y] [and of the loss] except the inputs, and propagates these gradients backwards to the parameters [and its immediate parents, in the case of the loss]” (p. 55-56) where gradients are indications of “the direction of greatest change of a function of more than one variable” (LibreTexts, 2019). In basic terms, during the training process, because most neural networks tend to be feedforward, if the model makes a wrong prediction, in theory, it would have to restart the entire learning process from the very start. However, this can be extremely time-consuming and often costly, especially when dealing with CNNs and bigger-scale interconnected neural network structures. This is when backpropagation becomes useful. Instead of starting over, the model can simply backpropagate the errors to its nodes, which will then adjust their respective parameters accordingly, such as their weights and threshold values, in order to make a better forward prediction.

There exist many types of text classification problems in natural language processing. One such example is sentiment analysis where labels tend to be binary in the form of positive or negative sentiment (however, more recent work aims at achieving finer granularity and nuance). Although sentiment analysis through a lexicon can avoid machine learning by simply comparing the number of positive and negative words in each text document to figure out its label (bag-of-words method), it is very limiting because natural languages are extremely convoluted when it comes to synonyms, double-meanings, metaphors, irony, sarcasm, negation, context, and so much more (Eisenstein, 2018, p. 70). That is why using neural networks to train a model for text classification tends to provide much more complexity in the analysis, using more than just bag-of-words in order to classify the text more accurately.

This project consists of using the free and open-source library *spaCy* (Honnibal & Montani, spaCy 2, 2017) to further train a pre-existing natural language processing model (through a pipeline called “TextCategorizer” which is implemented with hidden layers of neural networks) that classifies news articles into the four following categories: “World News”, “Sports News”, “Business News”, “Science-technology News”. The model will undergo supervised machine learning using two datasets (training and testing). The model will learn to recognize patterns from the training dataset and infer conclusions when facing datasets that it has not previously encountered. Natural language processing models can be optimized using the following equation:

$$\hat{y} = argmax\, \Psi\, (x, y\, ;\, \theta)$$

where the parameter $y \epsilon \Upsilon(x)$ represents the output and $x \epsilon \chi$ represents the input, and θ is the vector representation of the parameters for the model that is represented by Ψ. The predicted output symbolized by ŷ will give a number between 0 and 1 which will determine the article's likeliness of belonging to each specific category. Ideally, the number should be closer to 1 for a reliable result, rather than around 0.5. For this specific project, the input consists of various news articles and the output consists of the labeled categories of news articles. The training dataset of labeled news categories can be represented by $\{\, (x^i, y^i)\, \}_{i=1}^{120\ 000}$ (Eisenstein, 2018, p. 7-8).

Methodology

The platform Google Colaboratory (Google, 2021) was used to write the code in a Jupyter notebook that served to train the model for news articles classification. The programming language used in this project is Python 3 (3.6.9) (Van Rossum & Drake, 2009). The news articles training and testing datasets were provided by the online Kaggle community and are originally sourced from AG News (AG, 2020). The code for the project was written by adapting two online examples (Ligade, 2018; llefebure, 2018). When training, 10 epochs were performed. This minimised the chances of overfitting the model. The optimal number of epochs is determined after monitoring the loss of the training dataset. During the training, the training data is separated into two subcategories: 80% of it is used for the actual training and the other 20% is used for validation. The data is randomized during the training due to *spaCy*'s internal shuffling and a random 20% of this data is used as the validation dataset. All of this is done in order to avoid overfitting the model. For further details, refer to the annexes for the complete code.

The textual data is preprocessed using *spaCy*'s 'nlp' pipeline, a convolutional neural network (CNN) with multiple layers of features and is implemented with a small English language model. Numerous language features are applied to the texts. Each of these features is a pipeline component. First, the texts are tokenized, meaning sentences and paragraphs are broken down into smaller units such as words and punctuation. Then, they go through a part-of-speech tagger. This component establishes whether each word in the textual data is a noun, verb, pronoun, determinant, etc.

	text	lemma	pos	tag	dep	shape	is_alpha	is_stop	is_punctuation
0	Wall	Wall	PROPN	NNP	compound	Xxxx	True	False	False
1	St.	(St.,)	PROPN	NNP	compound	Xx.	False	False	False
2	Bears	(Bears,)	PROPN	NNPS	ROOT	Xxxxx	True	False	False
3	Claw	(Claw,)	PROPN	NNP	dobj	Xxxx	True	False	False
4	Back	(back,)	ADV	RB	advmod	Xxxx	True	True	False
5	Into	(into,)	ADP	IN	prep	Xxxx	True	True	False
6	the	(the,)	DET	DT	pobj	xxx	True	True	False
7	Black	(Black,)	PROPN	NNP	nmod	Xxxxx	True	False	False
8	(	((,)	PUNCT	-LRB-	punct	(	False	False	True
9	Reuters	(Reuters,)	PROPN	NNP	intj	Xxxxx	True	False	False
10	)	(),)	PUNCT	-RRB-	punct	)	False	False	True
11	Reuters	(Reuters,)	PROPN	NNP	nmod	Xxxxx	True	False	False
12	-	(-,)	PUNCT	HYPH	punct	-	False	False	True

Figure 5 – Decomposition of text example for one news article from the training dataset with its first twelve elements.

The texts also go through a syntax parser, a component that establishes the relationships between words and builds a syntax tree, as illustrated in Figure 6.

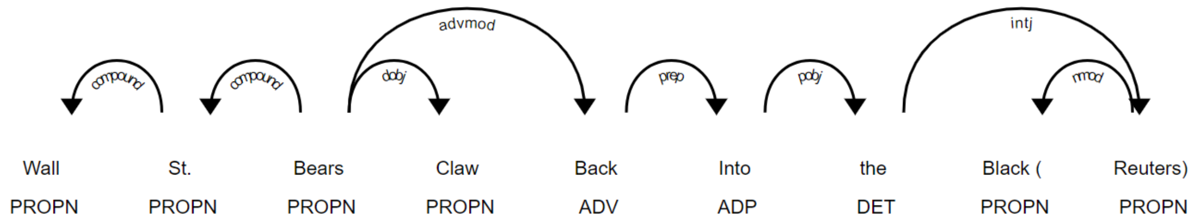

Figure 6 - Syntax tree.

The 'NER' is a component that serves the purpose of recognizing name-entities such names of people, organisations, locations, etc., as show in Figure 7.

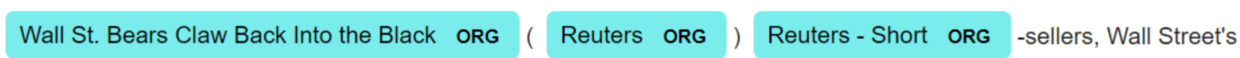

Figure 7 - NER example.

The texts also go through the process of lemmatization which is a component that transforms the words into their dictionary form. For example, the word "shaken" becomes "shake", such that it can be recognized under a more general form. This is particularly useful when it comes to verb conjugations. In the preprocessing of the textual data, there was also a removal of stop words (using *spaCy*'s stop word inventory). Stop words are words that do not significantly change the meaning of a sentence. Some examples of such words are "and", "is", and "the".

The 'textcat' pipeline used to train the model by further feeding it with news article data is a pre-existing model that has been built using linear bag-of-words and neural networks (Honnibal & Montani, Text classification architectures, 2021). The neural network of the 'textcat' pipeline is constructed using a 'tok2vec' model, which is one that transforms word tokens into word vectors, and uses backpropagation to save its prediction for each batch of data it is fed. That way, the next and previous components in the 'nlp' pipeline can use backpropagation to use the same weights of the model (Honnibal & Montani, Tok2Vec, 2021). Thus, we make use of "transfer-learning".

Results

Figure 8 - Distribution of training dataset with classes 1, 2, 3, 4.

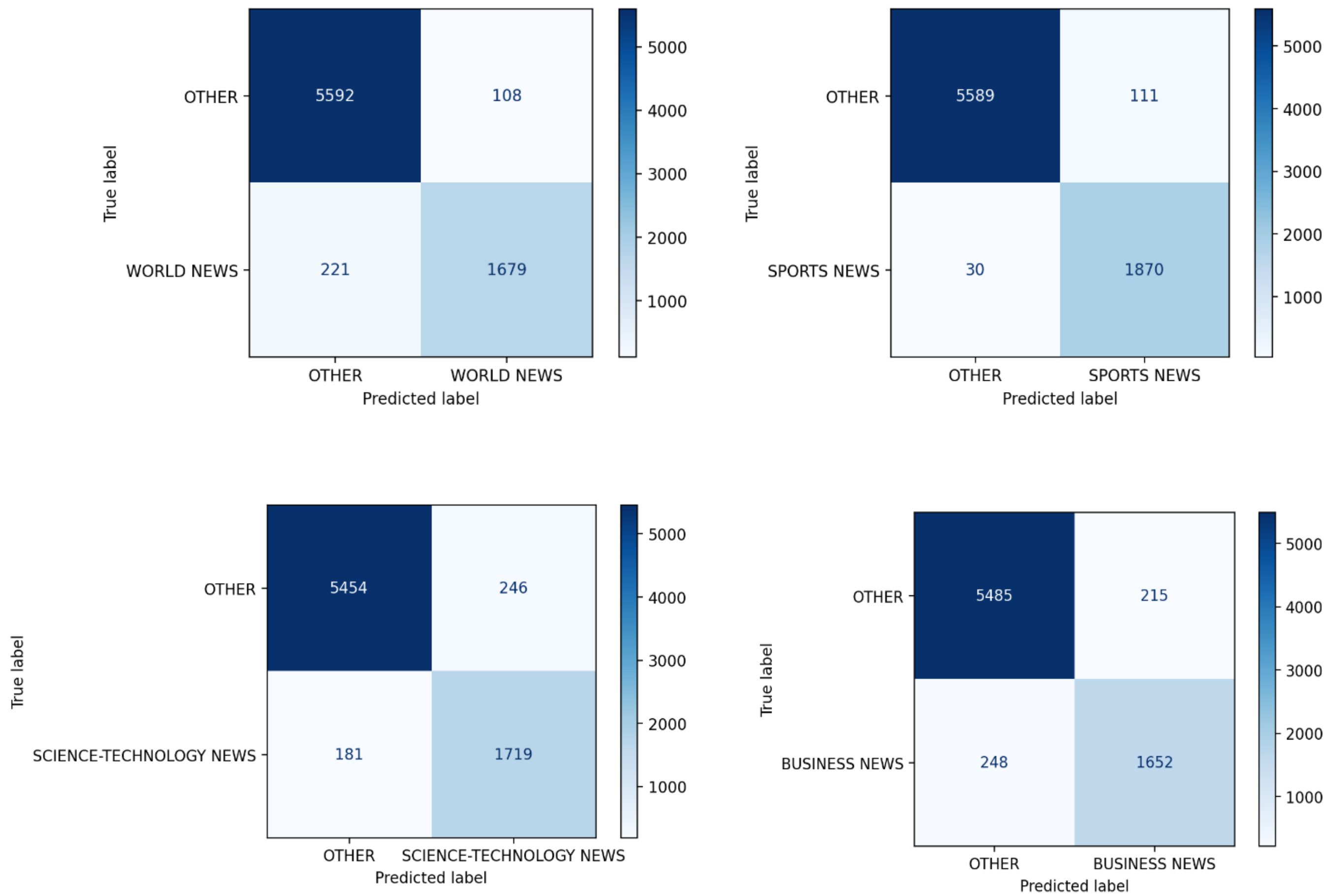

Figure 9 - Confusion matrices for the testing dataset (One class vs All other classes).

The training dataset consists of a total of 120 000 news articles. As can be observed on Figure 8, each class had the same number of articles and therefore, the data had an even distribution and there was little to no bias towards one class during the training. Figure 9 shows the confusion matrices for each class when it is compared to all three other classes at once after the model has been trained. This is a visualization of true positives, true negatives, false positives, and false negatives.

Table 1

Classification Report for the Training Dataset

Class	Precision	Recall	F1-Score	Support
WORLD	0.97	0.91	0.94	30 000
SPORTS	0.96	0.99	0.98	30 000
BUSINESS	0.93	0.92	0.92	30 000
SCIENCE-TECH	0.91	0.95	0.93	30 000
	ACCURACY		0.94	120 000

Table 2

Classification Report for the Testing Dataset

Class	Precision	Recall	F1-Score	Support
WORLD	0.94	0.88	0.91	1900
SPORTS	0.94	0.98	0.96	1900
BUSINESS	0.88	0.87	0.88	1900
SCIENCE-TECH	0.87	0.90	0.89	1900

ACCURACY	0.91	7600

Various metrics can be used to evaluate how well the model performs. The precision is a measurement of how accurate the model is at predicting true positives (TP) compared to the total number of positives predicted for a specific class. It is calculated using:

$$precision = \frac{TP}{TP + FP}$$

where FP denotes false positives. In Table 2, the precision score of the model on the testing dataset (i.e., unseen data) is 87% for the "Science-Technology" class, and only goes up for the other classes. These high values indicate that the model returns more relevant (TP) than irrelevant results (FP). Taking the science-technology category, for example, 87% of the news articles that the model classifies as "Science-Technology" are actually science-technology news articles, while 13% of those classified into this category are truly another one. The class with the highest precision is a tie between the "World" and "Sports" classes, meaning that they have the least number of false positives.

The recall is a measurement of how well the model is able to predict true positives (TP) amongst all the predictions that are labelled positive for a specific class. It can be calculated using:

$$recall = \frac{TP}{TP + FN}$$

where FN denotes false negatives (Shmueli, Multi-Class Metrics Made Simple, Part I: Precision and Recall, 2019). As seen in Table 2, the recall score for "Business" class is 87%, and higher for the other classes. These high values mean that the model is able to predict most of the relevant results. For instance, amongst all the articles that should be classified as “Business”, the model is able to classify 87% of them correctly. The class with the highest recall score is “Sports”, where it has the lowest number of false negatives, or in other words, the lowest number of incorrectly classified sports news articles.

The F1-score is the harmonic average of the precision and the recall. A harmonic average is defined by the following series:

$$\frac{n}{\sum_{i=0}^{n} \frac{1}{x_i}}$$

where n is the number of items being averaged together. In the case of the F1-score, the harmonic average is calculated as follows:

$$\frac{2}{\frac{1}{recall} + \frac{1}{precision}} = 2 \cdot \frac{precision \cdot recall}{precision + recall}$$

The F1-score penalizes wrong predictions more severely (Shmueli, Multi-Class Metrics Made Simple, Part II: the F1-score, 2019). In Table 2, the F1-scores are greater than 88%. The macro-averaged F1-score, which is simply an arithmetic average, becomes $\frac{0.91+0.96+0.88+0.89}{4} = 0.91$. The accuracy of the trained model is 91%.

By comparing Tables 1 and 2, we can see that the classification reports of the training and the testing dataset are similar. This means that the model can predict nearly as accurately on unseen data as it can on seen data, thus making it a well-performing model. Since the accuracy of the model on the testing dataset is only 3% lower than its accuracy on the training dataset, the model is unlikely to be overtrained. Another indication of overfitting of the model is the receiver operating characteristic curve (ROC) for each class plotted against all others.

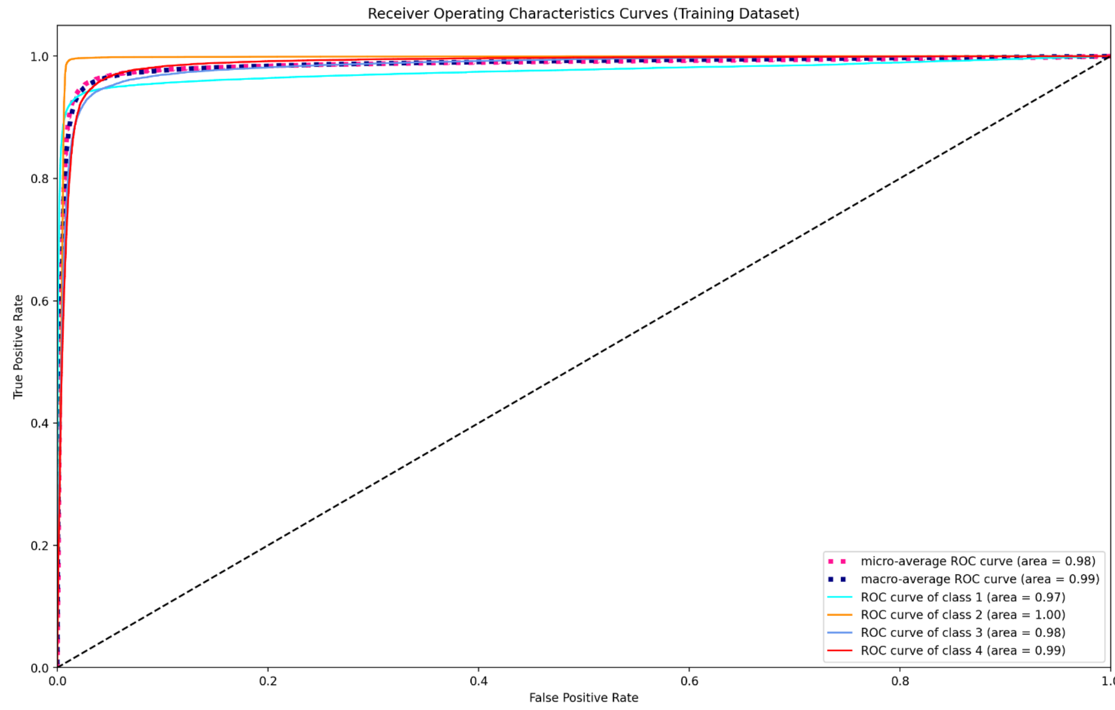

Figure 10 - ROC curves for the training dataset.

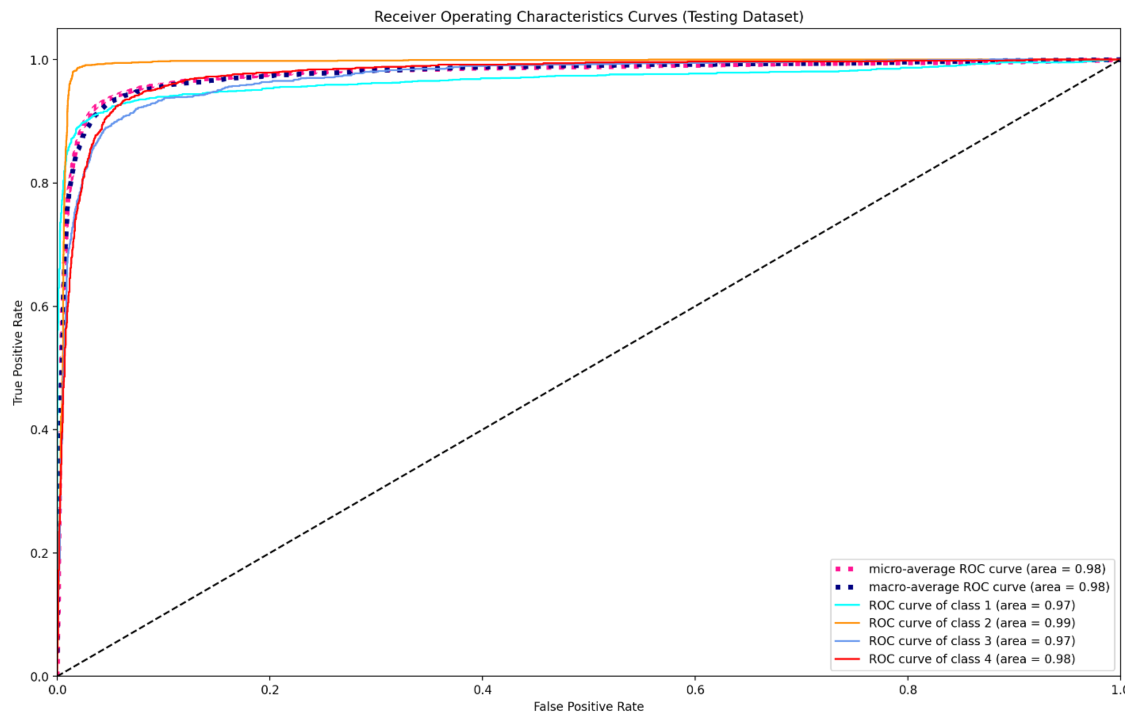

Figure 11 - ROC curves for the testing dataset.

The ROC curve is the result of plotting the rate of true positive predictions against the rate of false positive predictions by the model. The rate of true positives is also called the recall score. The rate of false positives (FP) is calculated using the following equation:

$$FPR = \frac{FP}{TN + FP}$$

which serves to indicate that amongst all the news articles whose labels are the other three classes, what proportion of them were classified incorrectly. For example, it calculates what proportion of the news articles that were either “Sports”, “Business”, or “Science-Technology”, actually got predicted as “World”. The area under the curve (AUC) is a measurement of how well the model is able to tell the difference between one against all the other classes when it is making predictions, meaning it is a measure of the separability of the data the model receives. In Figures 10 and 11, the dotted straight line represents the location on the graph where the AUC = 0.5, meaning that the model would not be able to distinguish one class against all others, or in other words, every time it would make a prediction for that class, it would likely be a random guess. If the curves were underneath the dotted line, it would mean that the model would have a higher chance at predicting incorrectly for this class than it would correctly (Google, 2020). In Figure 11, the calculated AUC values of every class (against all other classes, at once) are 0.97, 0.99, 0.97, 0.98. As a more overall metric, the macro-averaged AUC value can be calculated from:

$$AUC_{macro} = \frac{\sum_{i=1}^{n} AUC_i}{i}$$

where n is the number of different categories. This averaging method gives an equal weight to each class. In the case of the trained model, this is an appropriate metric to use because there is no significant class imbalance, be it in the training dataset, or the testing one. That is the reason why the micro-averaged AUC score, which takes into consideration the data distribution for the presence of any class imbalance, is nearly the same as the macro-averaged one (Vaughan, 2021).

An AUC value of 0.98 indicates that the trained model is very good at separating one class from all the others during its classification process. Comparing the ROC curves for the training dataset (seen data) and the ones for the testing dataset (unseen data) is an appropriate cross-validation method to tell if the model is overfit. Given that the macro-averaged AUC values are 0.99 for the training dataset and 0.98 for the testing dataset, the difference between the two is 0.01. Since the difference is very low, it means that the model is unlikely to be either overfit or underfit.

If the model was overfitting the data it was fed, one method to reduce overfitting is to use k-fold cross-validation. This would involve splitting the entire shuffled data into k number of groups, and then take one of the groups as the testing dataset. All the other groups would each serve as a training dataset, and the model would be trained in $k - 1$ different ways. At the end, each trained model would be tested on the held-out dataset and through comparison of their evaluation metrics, the best model would be picked (Brownlee, 2018).

Conclusion

Though in general, text classification is pertinent for businesses to analyze public sentiment, news article classification has its own specific purposes. Article classification can be useful in fields of legal investigation such as police, detective, or prosecution work to find out public information very efficiently. It is also useful to media outlets because it allows them a faster way to achieve a better organization of their news websites for public consumption. Overall, using a convolutional neural network with numerous language features to train a pre-existing natural language processing *spaCy* model (‘nlp’) to classify news articles into four different categories

(“World”, “Business”, “Science-Technology”, “Sports”) results in a 91% accuracy with a very low likelihood of overfitting given the calculated AUC values. In terms of limitations within the field of natural language processing, it is important to take into consideration the challenges that still exist with such a model. For instance, it is a given that there might as well be an infinite number of possible names for real companies that are created on a daily basis. No matter how well this model is trained, it will have a harder time recognizing the name of every single official company that exists and will therefore ignore obvious clues in its classification. Another type of issue is rooted in the rising creative freedom authors express by foregoing capitalization, where “Apple” and “apple” will not refer to the same thing. These are both examples of where the ‘parser’ and ‘NER’ features of the neural network could use some perfecting, in future experiments. The percentage of accuracy could be improved by training the ‘nlp’ pipeline with *spaCy*’s larger English language models that have pre-existing word vectors over a higher number of iterations.

Bibliography

Addison, A. (2019, July 22). *Linear Separability and the XOR Problem*. Retrieved from Automatic Addison: https://automaticaddison.com/linear-separability-and-the-xor-problem/

AG. (2020). AG News Classification Dataset. Retrieved from https://www.kaggle.com/amananandrai/ag-news-classification-dataset

Brownlee, J. (2018, May 23). *A Gentle Introduction to k-fold Cross-Validation.* Retrieved from Machine Learning Mastery: https://machinelearningmastery.com/k-fold-cross-validation/

Eisenstein, J. (2018, November 13). Natural Language Processing. MIT Press. Retrieved from https://github.com/jacobeisenstein/gt-nlp-class/blob/master/notes/eisenstein-nlp-notes.pdf

ElDen, I. S. (2019, October). *KDnuggets*. Retrieved from Introduction to Natural Language Processing (NLP): https://www.kdnuggets.com/2019/10/introduction-natural-language-processing.html

Google. (2020, February 10). *Classification: ROC Curve and AUC.* Retrieved from Machine Learning Crash Course: https://developers.google.com/machine-learning/crash-course/classification/roc-and-auc

Google. (2021). *Google Colaboratory*. Retrieved from Google Research: https://colab.research.google.com/

Honnibal, M., & Montani, I. (2017). spaCy 2. Retrieved from SpaCy: https://spacy.io/

Honnibal, M., & Montani, I. (2021). *Text classification architectures*. Retrieved from spaCy: https://spacy.io/api/architectures#textcat

Honnibal, M., & Montani, I. (2021). *Tok2Vec*. Retrieved from spaCy: https://spacy.io/api/tok2vec

Kinney, J. J. (1996, December 17). Probability: An Introduction with Statistical Applications. Wiley.

LibreTexts. (2019, June 23). *UCD Mat 21C: Multivariate Calculus*. Retrieved from LibreTexts: https://math.libretexts.org/Courses/University_of_California_Davis/UCD_Mat_21C%3A_Multivariate_Calculus

Ligade, P. (2018, October 24). *Text Classification using SpaCy*. Retrieved from Kaggle: https://www.kaggle.com/poonaml/text-classification-using-spacy

llefebure. (2018, October 27). *Headline Classification: SpaCy vs. Keras*. Retrieved from Kaggle: https://www.kaggle.com/llefebure/headline-classification-spacy-vs-keras

Nicholson, W. K. (2019). Linear Algebra With Applications. Retrieved from https://lila1.lyryx.com/textbooks/OPEN_LAWA_1/marketing/Nicholson-OpenLAWA-2019A.pdf

Shmueli, B. (2019, July 2). *Multi-Class Metrics Made Simple, Part I: Precision and Recall.* Retrieved from towards data science: https://towardsdatascience.com/multi-class-metrics-made-simple-part-i-precision-and-recall-9250280bddc2

Shmueli, B. (2019, July 3). *Multi-Class Metrics Made Simple, Part II: the F1-score.* Retrieved from towards data science: https://towardsdatascience.com/multi-class-metrics-made-simple-part-ii-the-f1-score-ebe8b2c2ca1

Stewart, J. (2016). Single Variable Calculus: Early Transcendentals. *8*. CENGAGE Learning.

Van Rossum, G., & Drake, F. (2009). Python 3 Reference Manual. Scotts Valley, CA: CreateSpace.

Vaughan, D. (2021, March 26). *Multiclass averaging.* Retrieved from The Comprehensive R Archive Network: https://cran.r-project.org/web/packages/yardstick/vignettes/multiclass.html

Young, T., Hazarika, D., Poria, S., & Cambria, E. (2018, November 25). Recent Trends in Deep Learning Based Natural Language Processing. Retrieved from https://arxiv.org/pdf/1708.02709.pdf

Appendix I

Code – Training the Model and Plotting Figure 8

```
!pip install spacy -q
!python -m spacy download en_core_web_sm
import spacy
spacy.__version__  #should give version 2.2.4

import numpy as np
import matplotlib.pyplot as plt
from spacy.lang.en import English
import pandas as pd
from spacy.util import minibatch, compounding
import random
import matplotlib.pyplot as plt                    #for graphs
%matplotlib inline

spacy.require_gpu() #should give 'true'

#LOAD THE DATA
from google.colab import drive
drive.mount("/content/gdrive")
train = pd.read_csv(mila_training_data)    #total 120 000
test = pd.read_csv(mila_testing_data)      #total 7600

#check the data
train.info()
test.info()

#check for data distribution by plotting a graph
from google.colab import files
list(train["Class Index"]).sort()
child = plt.figure(figsize=(15,6), dpi=200)
train["Class Index"].value_counts().plot(kind='bar')
plt.title("Training Data Distribution")
child.savefig('train_data_distribution.png',  bbox_inches="tight")
files.download('train_data_distribution.png')
plt.show()

#TEXT PREPROCESSING
import string
from spacy.lang.en.stop_words import STOP_WORDS as stopwords
nlp = spacy.load('en_core_web_sm', disable=['parser', 'tagger', 'ner'])
#load the small english language model
symbols = " ".join(string.punctuation).split(" ") + ["-", "...", "”", "“",
".."]

# this function merges the columns "Title" and "Description" and creates a
new dataset with the labels and the texts
def merging(dataset):
    merged_columns = dataset["Title"] + " " + dataset["Description"]
```

```
    new_dataset = pd.concat([dataset["Class Index"],
merged_columns.rename('text')], axis=1)
    return new_dataset

train_data = merging(train)
test_data = merging(test)

# this is a cleanup function
def cleanup(docs, logging=False):
    texts = []
    counter = 1
    for doc in docs:
        if counter % 1000 == 0 and logging:
            print("Processed %d out of %d documents." % (counter, len(docs)))
        counter += 1
        doc = doc.strip().replace("\n", " ").replace("\r", " ").replace("\\",
" ")
        doc = nlp(doc, disable=['parser', 'ner'])
        tokens = [tok.lemma_.lower().strip() for tok in doc if tok.lemma_ !=
'-PRON-']
        tokens = [tok for tok in tokens if tok not in stopwords and tok not
in symbols]
        tokens = ' '.join(tokens)
        texts.append(tokens)
    return pd.Series(texts)


#clean both training and testing 'text' columns
train_data['clean text'] = cleanup(train_data['text'])
test_data['clean text'] = cleanup(test_data['text'])


train_data['tuples'] = train_data.apply(
    lambda row: (row['clean text'],row['Class Index']), axis=1)
train = train_data['tuples'].tolist()

# change the format of the labels to suit the
cat_dict = {1:"WORLD NEWS", 2:"SPORTS NEWS", 3:"BUSINESS NEWS", 4:"SCIENCE-
TECHNOLOGY NEWS"}

def build_label(categories):
    cats = {'WORLD NEWS': 0, 'SPORTS NEWS': 0, 'BUSINESS NEWS': 0, "SCIENCE-
TECHNOLOGY NEWS": 0}
    for category in categories:
        cats[category] = 1
    return cats

def load_data(limit=0, split=0.8):                                          #
parameters of the function
                                                                            #
parameters are optional bc there is an "=" sign
    training_data = train                                                     #
rename the data because you don't want to modify the original dataset
```

```
    np.random.shuffle(training_data)                                   #
shuffling bc when you do multiple epochs, you want to reduce overtraining
    training_data = training_data[-limit:]                               #
-limit because there will be manipulation of it later
    texts, labels = zip(*training_data)                                #
unpacks each (text, label) into a separate argument

    cats = [build_label([cat_dict[y]]) for y in labels]

    split = int(len(training_data) * split)
# splitting the data into 80% of train data
    return (texts[:split], cats[:split]), (texts[split:], cats[split:])


n_texts = 120000        # number of articles used

n_iter = 10             # number of epochs

if 'textcat' not in nlp.pipe_names:
    textcat = nlp.create_pipe('textcat')
    nlp.add_pipe(textcat, last=True)
else:
    textcat = nlp.get_pipe('textcat')

textcat.add_label('WORLD NEWS')
textcat.add_label('SPORTS NEWS')
textcat.add_label('BUSINESS NEWS')
textcat.add_label('SCIENCE-TECHNOLOGY NEWS')

print("Loading news data...")
(train_texts, train_cats), (dev_texts, dev_cats) = load_data(limit=n_texts)
print("Using {} examples ({} training, {} evaluation)"
      .format(n_texts, len(train_texts), len(dev_texts)))
train_data = list(zip(train_texts,
                      [{'cats': cats} for cats in train_cats]))

# change the format of the real labels
dev_cats_list=[]
for d in dev_cats:
  for key, value in d.items():
    if value == 1:
      dev_cats_list.append(key)

nlp.pipe_names #check that only the 'textcat' pipeline is being trained


#BEGINNING OF MODEL TRAINING

#get rid of other pipelines anyway, just to make sure
counter=0
other_pipes = [pipe for pipe in nlp.pipe_names if pipe != 'textcat']
with nlp.disable_pipes(*other_pipes):
    optimizer = nlp.begin_training()
```

```
    print("Training the model...")

    for i in range(n_iter):
        print('Epoch %d' % i)
        losses = {}
        # batch up the examples using spaCy's minibatch
        batches = minibatch(train_data, size=128)
        for batch in batches:
            texts, annotations = zip(*batch)

            #logging                                   #tracking what's going
on every epoch
            if counter % 100 == 0:                     #if remainder of counter
after being divided by 100 == 0
                print("Example input:")
                print("\t",texts[0])
                print("Example label:")
                print("\t",annotations[0])
            counter += 1
            nlp.update(texts, annotations, sgd=optimizer, drop=0.2,
                       losses=losses)
        with textcat.model.use_params(optimizer.averages):
            docs = [nlp.tokenizer(h) for h in dev_texts]
            test_pred = np.array(
                [sorted(doc.cats.items(), key=lambda x: -x[1])[0][0]
                 for doc in textcat.pipe(docs)])
            print('Test Acc: %.4f' %
                  (pd.Series(test_pred == dev_cats_list).sum() /
len(dev_cats_list)))


#quickly check with validation set to see if there are any problems with the
code
from sklearn.metrics import classification_report

spacy_y_pred = [sorted(doc.cats.items(), key=lambda x: -x[1])[0][0]
                for doc in nlp.pipe(dev_texts)]
spacy_y_pred
print(classification_report(dev_cats_list, spacy_y_pred))

#SAVE THE TRAINED MODEL
nlp.to_disk("/content/gdrive/My Drive/textcat_SMALL")
```

Appendix II

Code – Figure 9: Confusion Matrices

```
import numpy as np
from sklearn.metrics import multilabel_confusion_matrix

confmats = multilabel_confusion_matrix(test_cats_list, spacy_y_pred,
labels=['WORLD NEWS','SPORTS NEWS','BUSINESS NEWS','SCIENCE-TECHNOLOGY
NEWS'])

from sklearn.metrics import confusion_matrix, ConfusionMatrixDisplay
import matplotlib.pyplot as plt
from google.colab import files

disp1 = ConfusionMatrixDisplay(confusion_matrix=confmats[0],
display_labels=['OTHER', 'WORLD NEWS'])
disp1.plot(values_format=".5g", cmap='Blues')
disp1.figure_.savefig('ConfMat_WORLD.png',  bbox_inches="tight", dpi= 200)
files.download('ConfMat_WORLD.png')
disp2 = ConfusionMatrixDisplay(confusion_matrix=confmats[1],
display_labels=['OTHER', 'SPORTS NEWS'])

disp2.plot(values_format=".5g", cmap='Blues')
disp2.figure_.savefig('ConfMat_SPORTS.png',  bbox_inches="tight", dpi= 200)
files.download('ConfMat_SPORTS.png')
disp3 = ConfusionMatrixDisplay(confusion_matrix=confmats[2],
display_labels=['OTHER', 'BUSINESS NEWS'])

disp3.plot(values_format=".5g", cmap='Blues')
disp3.figure_.savefig('ConfMat_BUSINESS.png',  bbox_inches="tight", dpi= 200)
files.download('ConfMat_BUSINESS.png')
disp4 = ConfusionMatrixDisplay(confusion_matrix=confmats[3],
display_labels=['OTHER', 'SCIENCE-TECHNOLOGY NEWS'])

disp4.plot(values_format=".5g", cmap='Blues')
disp4.figure_.savefig('ConfMat_SCIENCE.png',  bbox_inches="tight", dpi= 200)
files.download('ConfMat_SCIENCE.png')
```

Appendix III

Code – Figures 10 and 11: ROC Curves

```
import numpy as np
import matplotlib.pyplot as plt
from itertools import cycle

from sklearn import svm, datasets
from sklearn.metrics import roc_curve, auc
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import label_binarize
from sklearn.multiclass import OneVsRestClassifier
from scipy import interp
from sklearn.metrics import roc_auc_score

n_classes = 4

# TRAINING DATASET #

# Compute ROC curve and ROC area for each class
fpr = dict()
tpr = dict()
roc_auc = dict()
for i in range(n_classes):
    fpr[i], tpr[i], _ = roc_curve(y_true[:, i], y_pred[:, i])
    roc_auc[i] = auc(fpr[i], tpr[i])

# Compute micro-average ROC curve and ROC area
fpr["micro"], tpr["micro"], _ = roc_curve(y_true.ravel(), y_pred.ravel())
roc_auc["micro"] = auc(fpr["micro"], tpr["micro"])
from google.colab import files
# First aggregate all false positive rates
all_fpr = np.unique(np.concatenate([fpr[i] for i in range(n_classes)]))

# Then interpolate all ROC curves at these points
mean_tpr = np.zeros_like(all_fpr)
for i in range(n_classes):
    mean_tpr += interp(all_fpr, fpr[i], tpr[i])

# Finally average it and compute AUC
mean_tpr /= n_classes

fpr["macro"] = all_fpr
tpr["macro"] = mean_tpr
roc_auc["macro"] = auc(fpr["macro"], tpr["macro"])

# Plot all ROC curves
ROC = plt.figure(figsize=(16,10), dpi= 200)
plt.plot(fpr["micro"], tpr["micro"],
         label='micro-average ROC curve (area = {0:0.2f})'
               ''.format(roc_auc["micro"]),
         color='deeppink', linestyle=':', linewidth=4)
```

```
plt.plot(fpr["macro"], tpr["macro"],
         label='macro-average ROC curve (area = {0:0.2f})'
               ''.format(roc_auc["macro"]),
         color='navy', linestyle=':', linewidth=4)

colors = cycle(['aqua', 'darkorange', 'cornflowerblue', 'r'])
for i, color in zip(range(n_classes), colors):
    plt.plot(fpr[i], tpr[i], color=color,
             label='ROC curve of class {0} (area = {1:0.2f})'
             ''.format(i+1, roc_auc[i]))

plt.plot([0, 1], [0, 1], 'k--')
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Receiver Operating Characteristics Curves (Training Dataset)')
plt.legend(loc="lower right")
ROC.savefig('ROC.png',  bbox_inches="tight")
files.download('ROC.png')

plt.show()

# TESTING DATASET #

# Compute ROC curve and ROC area for each class
fpr = dict()
tpr = dict()
roc_auc = dict()
for i in range(n_classes):
    fpr[i], tpr[i], _ = roc_curve(y_true[:, i], y_pred[:, i])
    roc_auc[i] = auc(fpr[i], tpr[i])

# Compute micro-average ROC curve and ROC area
fpr["micro"], tpr["micro"], _ = roc_curve(y_true.ravel(), y_pred.ravel())
roc_auc["micro"] = auc(fpr["micro"], tpr["micro"])
# First aggregate all false positive rates
all_fpr = np.unique(np.concatenate([fpr[i] for i in range(n_classes)]))

# Then interpolate all ROC curves at these points
mean_tpr = np.zeros_like(all_fpr)
for i in range(n_classes):
    mean_tpr += interp(all_fpr, fpr[i], tpr[i])

# Finally average it and compute AUC
mean_tpr /= n_classes

fpr["macro"] = all_fpr
tpr["macro"] = mean_tpr
roc_auc["macro"] = auc(fpr["macro"], tpr["macro"])

# Plot all ROC curves
```

```
ROC = plt.figure(figsize=(16,10), dpi= 200)
plt.plot(fpr["micro"], tpr["micro"],
         label='micro-average ROC curve (area = {0:0.2f})'
               ''.format(roc_auc["micro"]),
         color='deeppink', linestyle=':', linewidth=4)

plt.plot(fpr["macro"], tpr["macro"],
         label='macro-average ROC curve (area = {0:0.2f})'
               ''.format(roc_auc["macro"]),
         color='navy', linestyle=':', linewidth=4)

colors = cycle(['aqua', 'darkorange', 'cornflowerblue', 'r'])
for i, color in zip(range(n_classes), colors):
    plt.plot(fpr[i], tpr[i], color=color,
             label='ROC curve of class {0} (area = {1:0.2f})'
             ''.format(i+1, roc_auc[i]))

plt.plot([0, 1], [0, 1], 'k--')
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Receiver Operating Characteristics Curves (Testing Dataset)')
plt.legend(loc="lower right")
ROC.savefig('ROC.png',  bbox_inches="tight")
files.download('ROC.png')

plt.show()
```