News Article Classification

Lauren Contard, Archit Datar
Yue Li, Robert Lumpkin, Haihang Wu

1 Introduction and Problem Statement

1.1 Introduction and Description of Dataset

COVID-19 has influenced us all from every single perspective. However, people in the U.S. have shown distinct ways in response to COVID-19. Democrats are more likely than Republicans to wear masks and keep social distancing. Republicans are more likely than Democrats to blame China and focus on the economic impacts. What could be some of the drivers that lead to polarization in people's attitudes and behaviors in response to COVID-19? This study aims to explore whether and how mainstream news media in the U.S. plays a role in the polarization of public's attitudes and behaviors in response to COVID-19.

To answer this question, we analyzed the content of news articles about elite communication about COVID-19. Elite communication, such as White House briefings and politicians' Tweets, has been found to increase the polarization between left-leaning and right-leaning public in terms of attitudes and behaviors in response to COVID-19. However, it is still unknown whether news media amplify or narrow down the effects of elite communication on polarization. We collect news articles from ten news media, including newspapers (i.e., New York Times, Washington Post, Wall Street Journal, and USA Today) and cable network news (i.e., ABC News, NBC News, CBS News, CNN, Fox News, and MSNBC News).To obtain the news articles, we search the keywords in the GDELT database that monitors the world's broadcast, print, and web news from every country in more than 100 languages. We used two sets of keywords to capture the news articles about the White House responses to COVID-19. The first set of keywords were used to query White House, including "white house", "trump", "coronavirus task force", "press secretary", "McEnany", "vice president", and "pence". The second set of keywords were used to query press briefing, including "white house briefing", "coronavirus briefing", "press conference", "task force briefing", "news briefing", "press briefing", "daily briefing", "weekly briefing", and "news conference". Both sets of keywords were used in searching for articles that mentioned "covid" or "coronavirus" at least three times. The GDELT database returned 8588 news articles that met these criteria from January 20, 2020 to September 29, 2020. Part of these articles that were captured by the keywords are not relevant to the topic of this project and therefore were removed from the list.

We then used the Python package "Scrapy" to scrape the full text of these articles. We prepared the full text datasets at the paragraph-level. We randomly selected 290 paragraphs and manually labeled them based on the Extended Parallel Processing Model (EPPM) [14, 13]. The EPPM framework predicts that in order to effectively persuade people to adopt a recommended health practice, the messages need to invoke certain levels of perception of both threats and efficacy. Therefore, we manually labeled our samples into the following categories: (1) threats/impacts, (2) responses/actions, (3) severity, (4) susceptibility, (5) self-efficacy, (6) external efficacy, (7) response efficacy, (8) public health, (9) economy, (10) education, (11) political evaluation, (12) racial conflict, (13) international relations/foreign policies, (14) positive, (15) negative. All the 15 labels are marked as either 0(No) or 1(Yes). These 15 labels are not mutually exclusive. One paragraph usually has multiple labels, which is the central problem that we are trying to address in this project.

1.2 Problem Statement

There are three essential problems associated with our datasets. First, we have a multi-label classification problem. The paragraphs usually belong to more than one category. Existing methods for multi-label classification fall into two categories: (1) problem transformation methods, and (2) algorithm adaption methods [12]. Problem transformation methods transform the multi-label classification problem into multiple single-label classification problems and then apply single-label classification algorithms. The adapted algorithms, on the other hand, extend specific learning algorithms in order to handle multi-label data directly. We adapted two learning algorithms, k-nearest neighbors (k-NN) and neural networks, to handle the multi-label problem of our dataset. Second, our data has high dimensions. We essentially used words as our features to train our models, which results in thousands of distinct words. The high dimensions can cause problems with KNN based methods. Thus, we used two types of dimension reduction methods to handle this issue: linear and non-linear. Third, the order of the features can be considered in the classification. Traditional bag-of-words models in natural language processing (NLP) do not consider the order of words. The bag-of-words models not only lead to a high dimensional and highly sparse feature vector, but also lose tons of information in the order of words. Therefore, we used two types of features to train our classifiers: one considered the order of words and one does not.

Based on the three problems above, we proposed the following research questions.

RQ1: Which type of algorithms performs better in the multi-label classification problem: naive methods or novel methods?

RQ2: Within the adapted algorithm methods, which one of the algorithms performs better in the multi-label classification problem: k-NN or neural networks?

RQ3: How can we effectively implement linear and non-linear dimensions reduction in the multi-label classification problem?

RQ4: Which type of the features perform better in the multi-label classification problem: with or without considering the order of words?

2 Methods

2.1 Text Pre-processing

We preprocessed the paragraph-level text data in two ways to get the features for our models. In the first approach we preprocessed the text without considering the order of words. Specifically, we removed punctuation, tokenized the words to unigrams or compounds, removed stopwords, and transformed the words to their word stems. We then calculated the term frequency-inverse document frequency (tf-idf) of each word. We finally got 2094 dimensional tf-idf vectors that served as our features for the KNN based and feed-forward neutral network models. In the second approach we preprocessed the text considering the order of words. Specifically, we kept the order of words by generating sequences of integers from processed tokens: urls, punctuation, and stopwords were removed.

2.2 KNN Methods – Binary Relevance and ML-KNN

Solutions to the general multi-label learning problem include a wide variety of approaches. Arguably, the most intuitive among these is what's referred to as "binary relevance". This approach works by decomposing the multi-label learning task into a number of independent binary learning tasks (one per class label) [8]. Binary Relevance methods are often criticized in the literature because of their label independence assumption, producing a potential weakness of ignoring correlations among labels [5]. In this project, we compare a binary relevance KNN model to a novel, bayesian KNN based approach: ML-kNN.

The ML-kNN (Multi-label k nearest neighbors) model is derived from the traditional k nearest neighbors (kNN), except for the multi-label case. While the goal of the traditional kNN algorithm is to predict whether class of the test sample based on the classes of its k nearest neighbors, the goal of ML-kNN is to predict multiple classes based on the classes of the k nearest neighbors of the test point. For the unseen data point, its nearest neighbors are identified. Then, based on the number of neighboring instances belonging to each possible class, maximum a posteriori (MAP) principle is utilized to determine the label set for the unseen instance.

ML-kNN can be used in a variety of problems, such as, text categorization [7], functional genomics [2], and in image classification.[1]

In a typical multi-label learning problem, we have a domain of instances (χ) and a finite set of labels ($\mathcal{Y} = \{1, ..., Q\}$). Given $x \in \chi$ and its associated $Y \subseteq \mathcal{Y}$, $\vec{y}_x$ is the category vector for x such that (for all $\ell \in \mathcal{Y}$) $\vec{y}_x(\ell) = 1$ if $\ell \in Y$. Otherwise, $\vec{y}_x(\ell) = 0$.

The basic concept of ML-kNN is to estimate the probability that a test instance (t) has a label (ℓ) given the number of nearest neighbors ($\vec{C}_t(\ell)$ out of $N(x)$) that have label ℓ. Let E_j^ℓ ($j \in \{1, ..., K\}$) denote the event that, among the K nearest neighbors of t, there are exactly j instances which have label ℓ. Further, let H_0^ℓ denote the event that test instance t does not have a label ℓ and let H_1^ℓ denote the event that it does have label ℓ. Further, let $\vec{y}_t$ be the vector containing the predicted labels at test instance t. Then, the predicted value for the ℓ^{th} label is given as:

$$\vec{y}_t(\ell) = \underset{b\in\{0,1\}}{\operatorname{argmax}} P\left(\mathrm{H}_b^\ell | E_{\vec{C}_t(\ell)}^\ell\right), \quad \ell \in \mathcal{Y} \tag{1}$$

Using Bayes theorem, this can be written as:

$$\vec{y}_t(\ell) = \underset{b\in\{0,1\}}{\operatorname{argmax}} \frac{P\left(\mathrm{H}_b^\ell\right) \cdot P\left(E_{\vec{C}_{t(\ell)}}^\ell | \mathrm{H}_b^\ell\right)}{P\left(E_{\vec{C}_t(\ell)}^\ell\right)} \tag{2}$$

Since $E_{\vec{C}_t(\ell)}^\ell$ is independent of H_b^ℓ, this can be equivalently written as:

$$\vec{y}_t(\ell) = \underset{b\in\{0,1\}}{\operatorname{argmax}} P\left(\mathrm{H}_b^\ell\right) \cdot P\left(E_{\vec{C}_{t(\ell)}}^\ell | \mathrm{H}_b^\ell\right) \tag{3}$$

The details of the derivation of $P\left(\mathrm{H}_b^\ell\right)$ and $P\left(E_{\vec{C}_{t(\ell)}}^\ell | \mathrm{H}_b^\ell\right)$ are beyond the scope of this report and are provided in [15].

2.3 Linear Dimension Reduction (PCA)

The available data for our project is high dimensional; it has a few hundred data points and a few thousand features. Such data can have significant generic disadvantages such as being pront to overfitting. More specifically, for a kNN-type model, this can cause problems because the distance the between points is considered Euclidian. In such a scenario, having an extremely high dimensional feature space causes the distances between points to be fairly similar, as these distance vector components are partitioned across many dimensions. [10]

Thus, dimensionality reduction was deemed beneficial. We performed the traditional principal components analysis (PCA) to understand if the data could be represented in fewer linear combinations.

The standard PCA method partitions the total variance of the data along various linear combinations of the features.

2.4 Nonlinear Dimension Reduction (ANN Autoencoder)

In addition to reductions in dimension due to PCA, we also implement an ANN autoencoder (see section 2.4 for an introduction to ANNs). Autoencoders can learn data projections with suitable dimensionality and sparsity limitations that are more useful than other fundamental methods such as PCA, which only allow for linear data representations [6].

This nonlinear dimension reduction is done by by training a feed forward (FF) neural network to perform the identity mapping, where the network inputs are reproduced at the output layer. The network contains an internal "bottleneck" layer (containing fewer nodes than input or output layers), which forces the network to develop a compact representation of the input data, and two additional hidden layers [4]. Look to the diagram to the right, for a visualisation of this architecture.

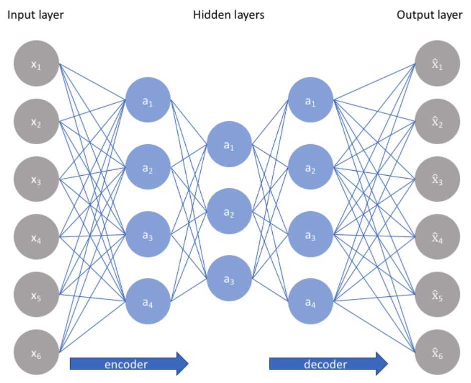


Figure 1: Autoencoder architecture.

The particular network that we trained had three hidden layers, as in the diagram. The first, second and third hidden layers are of dimensions 128, 64, and 128, and use the activations tanh, ReLu, and sigmoid, respectively. Training was performed using Adam optimization, MSE loss, and over 400 epochs. After training, the generated encodings were used to repeat our model fitting procedures for both the binary-relevance KNN and ML-KNN algorithms.

2.5 Artificial Neural Networks (Feed-Forward & Recurrent)

Classification approaches utilizing different artificial neural networks are also utilized in our project. Inspired by biological nervous systems, neural networks date back to the first half of the 20^{th} century with works such as those by McCulloch and Pitts, which could model simple logical operations [11]. Since most subsequent work in the following two decades centered around single layer networks, the power of neural networks was restricted to linearly separable problems. This excluded the possibility of learning even simple functions like XOR, which required a second layer [3]. In the early 1980s, research on neural networks resurged largely due to successful learning algorithms for multi-layer neural networks and are used today for various tasks such as computer vision, associative memory, representation learning, NLP, etc..

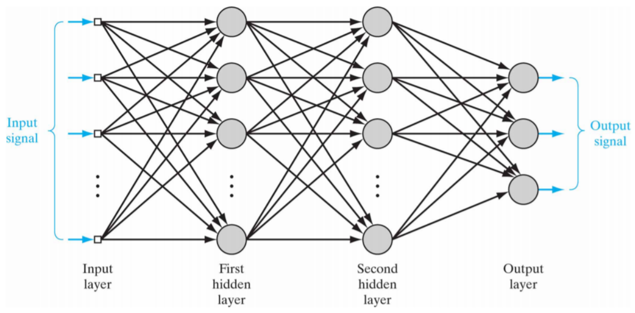


Figure 2: Feed forward network diagram.

For our project, we tried implementing both FF networks and recurrent neural network (RNN) architectures. Perceptrons are the building block for FF networks. A perceptron has an input layer of neurons and an output neuron. Weighted connections exist between every input neuron and the output neuron. A forward pass through the network consists of multiplying the values of the input neurons by the value of their connections' weight, summing and then applying some non-linearity (which we call the activation function). Feed forward networks are typically built by stacking perceptron units vertically and horizontally. All of our FF networks utilize the same architectures; the first hidden layer is of size 32 with ReLU activation, dropout regularization with drop probability of 0.5, and an output layer of size 13 (the number of labels) with sigmoid activations. The FF architectures were trained on 2094 dimensional tf-idf vectors.

While FF networks display incredible expressive power, RNNs are a popular adaptation for NLP problems because they are uniquely suited for sequence processing. This is due to their hidden unit connections with shared weights, which allow for information from previous and/or future states to influence the current state. This nice feature also leads to the exacerbation of the "gradient exploding/-vanishing" problem during training. Many methods have been proposed to overcome this, one of which is the "gated" architecture we use, known as the Long-Short-Term-Memory (LSTM) architecture.

On the reduced dataset, all of our RNN models utilize the same bidirectional LSTM architecture. On the full dataset, all of our models utilize the same single-directional LSTM architecture. In both the bidirectional and single-directional LSTM's, we used hidden states of size 16 followed by a dense output layer with sigmoid activations.The RNNs were trained on padded sequences of integers corresponding to sequences of the processed tokens from the paragraphs; urls, punctuation, and stop words were all removed.

All of our networks (both FF and RNN) utilize the same optimization algorithm (Adam optimization – a variant of gradient descent), however, training was performed, using two different loss functions. The standard binary cross entropy loss function was used, in addition, to the more novel BPMLL (back prop for multilabel learning) loss. The BPMLL loss function requires instances to have at least one label. Thus, on the one hand, we trained both cross entropy and BPMLL models on such a reduced dataset in addition to training just cross entropy models on the full dataset.

The BPMLL loss function aims to leverage correlations between labels by evaluating the error as a function of pairwise errors between labels. Namely, the loss function is given by:

$$E = \sum_{i=1}^{m} E_i = \sum_{i=1}^{m} \frac{1}{|Y_i||\overline{Y}_i|} \sum_{(k,l)\in Y_i \times \overline{Y}_i} \exp(-(c_k^i - c_l^i))$$

where $c_j^i = c_j(x_i)$ is the output of the network on x_i on the j^{th} class. The back propagation algorithm is derived in the same manner is for a cross entropy or MSE loss. Details are ommitted here, but can be found in [16].

2.6 Threshold Function Learning

In both of their papers, introducing the BPMLL and ML-KNN algorithms, Zhang & Zhou also describe a common method for learning threshold functions. Perfect classification, using a constant threshold requires two conditions: (1) Logit values corresponding to labels included in an instance's label set be separated from logit values corresponding to labels not in an instance's label set. And (2) this separation be around some constant value (usually either 0.5 or 0). Learning a threshold function aims to relax the second condition. Namely, we fit a linear regression model to learn threshold values from the logit outputs of our models. For more details, see [16] & [15].

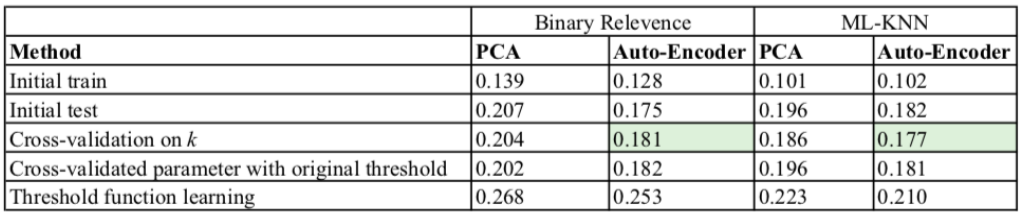
3 Results

3.1 ML-KNN Results

Performing a PCA analysis, we found that 90 percent of the variance could be captured with 165 principal components. In order to reduce the data further, we considered the maximum absolute difference (among all labels) between the means of positive and negative principal component values. The largest value of this difference was 3.0, so we consider 0.3 as a cutoff. There were 103 principal components with maximum differences below 0.3, which were disregarded, leaving 62 principal components (from 2094 initial features).

The ANN autoencoder method (described above) for nonlinear dimension reduction was also performed. This was specified to produce a reduced data set of 62 features as well, to allow direct comparisons.

The naive binary relevance classification was performed on both reduced data sets. Cross-validation was used to select $k = 3$ nearest neighbors, and threshold function learning was used as well. The same process was then carried out using ML-KNN classification. The test Hamming loss for each step, for both methods and data sets, is displayed below.



	Binary Relevence		ML-KNN	
Method	**PCA**	**Auto-Encoder**	**PCA**	**Auto-Encoder**
Initial train	0.139	0.128	0.101	0.102
Initial test	0.207	0.175	0.196	0.182
Cross-validation on *k*	0.204	0.181	0.186	0.177
Cross-validated parameter with original threshold	0.202	0.182	0.196	0.181
Threshold function learning	0.268	0.253	0.223	0.210

Figure 3: Hamming loss for KNN classification methods

For both methods, we achieved better classification with the data reduced by the auto encoder method than by PCA. This suggests that the nonlinear dimension reduction was able to capture more information in the same number of features.

ML-KNN achieved a lower Hamming loss than the naive KNN, as expected. However, this difference (0.177 vs. 0.181) was not as great as expected. While ML-KNN was able to use correlations between labels, this may not have been as relevant as we expected. This is something that could be investigated in further work.

Finally, the learned threshold function did not improve prediction. This also occurred with neural network classification (see below), and is discussed further in section 4.

3.2 Artificial Neural Network Results

As some instances in the full dataset lack labels, the 'Reduced Dataset' is created by removing these instances without any labels. The full dataset and reduced dataset are then used to train and validate four networks: Feed Forward network with cross entropy loss(CE FF), Feed Forward network with BPMLL (BPMLL FF), Recurrent neural network with Bidirectional (single directional on full dataset) LSTM and cross entropy loss(CE RNN), and Recurrent neural network with Bidirectional (single directional on full dataset) LSTM and BPMLL(BPMLL RNN). We also train the neural network using 3 different training rates: 0.01, 0.001, 0.0001. For each learning rate, the validation set hamming loss is plotted against epochs as shown in figure 4 and 5.

Figure 4 shows the training history of CE FF and CE RNN using full dataset and constant threshold. From the left and middle sub figures, it can be shown that CE FF can achieve lower hamming loss than CE RNN. For the right sub figure, though the CE FF has higher hamming loss than CE RNN, CE FF seems to have decreasing trend of hamming loss as opposed to the plateau trend of CE RNN at 100 epoch, meaning that it is highly possible that CE FF will have lower hamming loss than CE RNN if the training continues. The effect of learning rate on training is also clear from Figure 4; smaller learning rate leads to slower training process. But from figure 1, it is unclear whether smaller learning rate can reduce the hamming loss finally.

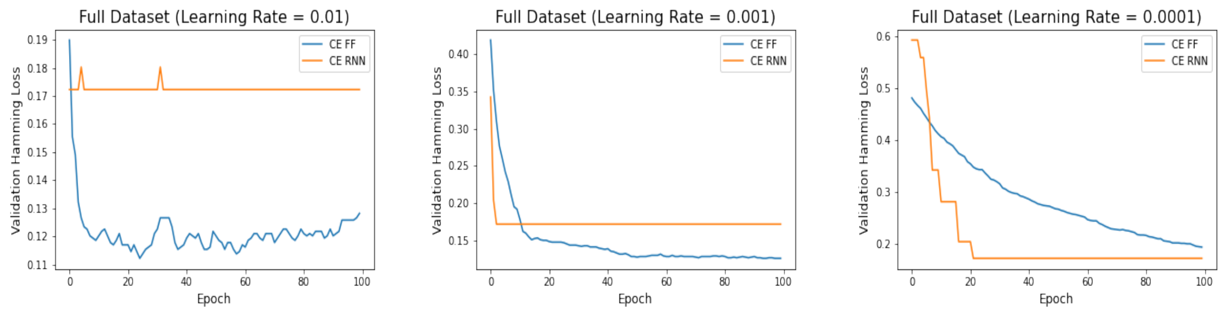


Figure 4: 2 models trained by full dataset

To compare with constant threshold, Table 3.1 gives the validation set hamming loss after 100 epochs of training, using a learned threshold function, for each network with 3 different learning rates. No significant improvement in hamming loss has been observed for both 2 networks after learned threshold function is used.

In contrast with constant constant thresholds, this time CE FF seems to have higher hamming loss than CE RNN for all 3 learning rate. It also seems that the learning rate of 0.001 has the lowest hamming loss.

Learning Rate	**CE FF**	**CE RNN**
0.01	0.302	0.238
0.001	0.179	0.173
0.0001	0.212	0.177

Table 3.1: Hamming Loss with Threshold Function Learning

All the 4 models are trained with the reduced dataset using constant threshold and the results are shown in Figure 5. Similar to the full dataset case, RNN in general has higher hamming loss than Feed-Forward for learning rate of 0.01 and 0.001. The same conclusion seems to hold for learning rate of 0.0001 if training epoches increase.

From the right and middle sub figures, for both RNN and Feed-Forward network, the hamming loss is quite close between BPMLL and cross entropy. In fact, the left sub figure shows BPMLL has higher hamming loss than cross entropy for both RNN and Feed-Forward network.

Also, smaller learning rate needs more training epochs for finale convergence.

Similarly, as shown in Table 3.2, the 4 models are also trained with Threshold Function Learning, but the hamming loss is not reduced for all 4 models. Also, no clear winner among 4 models is observed.

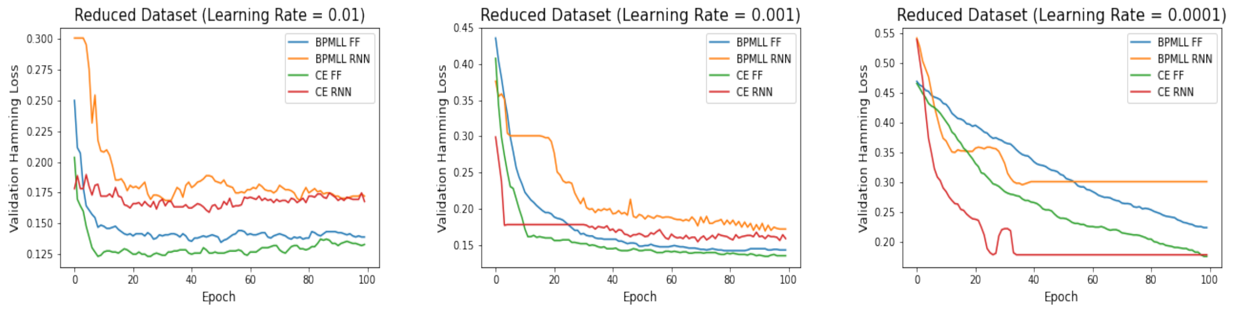


Figure 5: 4 models trained by reduced dataset

Learning Rate	**CE FF**	**BPMLL FF**	**CE RNN**	**BPMLL RNN**
0.01	0.1520979021	0.145979021	0.2447552448	0.2194055944
0.001	0.1853146853	0.2578671329	0.1791958042	0.20454545
0.0001	0.2071678322	0.1844405594	0.1896853147	0.2132867133

Table 3.2: Hamming Loss with Threshold Function Learning

4 Discussion & Conclusions

Evaluating our neural network classifiers, we find three interesting trends, in particular: (1) The FF networks networks generally tend to outperform the RNNs. (2) The networks trained using a cross entropy loss generally tend to outperform those trained using a BPMLL loss. And (3) Applying a learned threshold function did not seem to provide any notable improvements to prediction. We posit some reasons that might explain each of these observations.

While training the biderectional LSTM on the reduced dataset, we noticed that the hamming loss and cross entropy loss tends to fall very quickly, while the validation set hamming loss and cross entropy loss both stall or begin to increase after some epochs. This could be a sign that the network is overfitting. The RNN networks that we defined have more parameters to learn than the feed forward networks, which could partially explain this, given our limited amount of training data.

Both Zhang and Zhou's original paper as well as Nam et al. [9] provide insights that might partially explain why we saw no benefits from using BPMLL over CE loss. One such reason, is that the surface for the BP-MLL loss has plateaus in which gradient descent can be very slow in comparison with the cross-entropy. A second reason: while BPMLL is supposed to leverage correlations between labels, Nam et al. conjecture that these correlations also may cause overfitting. If groups of hidden units specialize in predicting particular label subsets that occur frequently in the training data, it will become harder to predict novel label combinations that only occur in the test set [9].

Lastly, we conclude that our models were able to effectively separate predicted logit values around a constant value of 0.5, since learning a threshold function seemed to provide no benefit. We note that, for networks trained on fewer than 100 epochs, with much larger loss scores, learning a threshold function did provide significant improvements. This indicates that, initially, training was able to separate true from false labels, but could not force the separation to be around 0.5 until trained further.

References

[1] Matthew R. Boutell et al. *Learning multi-label scene classification.* 2004.

[2] André Elisseeff and Jason Weston. "A Kernel Method for Multi-Labelled Classification". In: *Proceedings of the 14th International Conference on Neural Information Processing Systems: Natural and Synthetic.* NIPS'01. Vancouver, British Columbia, Canada: MIT Press, 2001, pp. 681–687.

[3] Simon Haykin. *Neural Networks and Learning Machines.* Pearson, 2009.

[4] Mark A. Kramer. "Nonlinear Principal Component Analysis Using Autoassociative Neural Networks". In: *AIChE Journal* 37.2 (1991), pp. 233–243. DOI: https://doi.org/10.1002/aic.690370209.

[5] Oscar Luaces et al. "Binary relevance efficacy for multilabel classification". In: *Progress in Artificial Intelligence* 1.4 (2012), pp. 303–313. DOI: 10.1007/s13748-012-0030-x.

[6] Mohamad Aljnidi Kadan Aljoumaa Maha Alkhayrat. "A comparative dimensionality reduction study in telecom customer segmentation using deep learning and PCA". In: *Journal of Big Data* 7.9 (2020). ISSN: 2196-1115. DOI: https://doi.org/10.1186/s40537-020-0286-0.

[7] Andrew Kachites McCallum. "Multi-label text classification with a mixture model trained by EM". In: *AAAI 99 Workshop on Text Learning.* 1999.

[8] Xu-Ying Liu Min-Ling Zhang Yu-Kun Li and Xin Geng. "Binary relevance for multi-label learning: an overview". In: *Frontiers of Computer Science* 12.2 (2018), pp. 191–202. DOI: 10.1007/s11704-017-7031-7.

[9] Jinseok Nam et al. "Large-Scale Multi-label Text Classification — Revisiting Neural Networks". In: *Machine Learning and Knowledge Discovery in Databases.* Ed. by Toon Calders et al. Berlin, Heidelberg: Springer Berlin Heidelberg, 2014, pp. 437–452.

[10] Lan Huong Nguyen and Susan Holmes. "Ten quick tips for effective dimensionality reduction". In: *PLoS computational biology* 15.6 (2019), e1006907.

[11] Gualtiero Piccinini. "The First Computational Theory of Mind and Brain: A Close Look at McCulloch and Pitts' *Logical Calculus of Ideas Immanent in Nervous Activity*". In: *Synthese* 141.2 (2004), pp. 175–215. DOI: 10.1023/B:SYNT.0000043018.52445.3e.

[12] Grigorios Tsoumakas and Ioannis Katakis. "Multi-label classification: An overview". In: *International Journal of Data Warehousing and Mining (IJDWM)* 3.3 (2007), pp. 1–13.

[13] Kim Witte. "Fear control and danger control: A test of the extended parallel process model (EPPM)". In: *Communications Monographs* 61.2 (1994), pp. 113–134.

[14] Kim Witte. “Putting the fear back into fear appeals: The extended parallel process model”. In: *Communications Monographs* 59.4 (1992), pp. 329–349.

[15] Min-Ling Zhang and Zhi-Hua Zhou. “ML-KNN: A lazy learning approach to multi-label learning”. In: *Pattern Recognition* 40.7 (2007), pp. 2038–2048. ISSN: 0031-3203. DOI: `https://doi.org/10.1016/j.patcog.2006.12.019`. URL: `https://www.sciencedirect.com/science/article/pii/S0031320307000027`.

[16] Min-Ling Zhang and Zhi-Hua Zhou. “Multilabel Neural Networks with Applications to Functional Genomics and Text Categorization”. In: *IEEE Transactions on Knowledge and Data Engineering* 18.10 (2006), pp. 1338–1351. DOI: `10.1109/TKDE.2006.162`.