

***School of Computer Science and Engineering***

***CZ4041 Machine Learning***

***Machine Learning Group Project***

***Group 35***

**Contents**

1. Group Members 3
2. Literature Review 4
   1. Binary Relevance (BR) X
   2. Multilabel K-nearest neighbours (MLKNN) X
   3. Classifier Chain X
   4. Binary Relevance k-nearest neighbours (BRKNN) X
   5. Multilabel Adaptive Resonance Associative Map (MLARAM) X
   6. Label Powerset X
   7. RAndom K-labELsets (RAKEL) X
   8. Majority Voting X
   9. Embedding X
3. Implementation of state-of-the-art methods X
   1. Binary Relevance (BR) X
   2. Multilabel K-nearest neighbours (MLKNN) X
   3. Classifier Chain X
   4. Binary Relevance k-nearest neighbours (BRKNN) X
   5. Multilabel Adaptive Resonance Associative Map (MLARAM) X
   6. Label Powerset X
   7. RAndom K-labELsets (RAKEL) X
   8. Majority Voting X
   9. Embedding X
4. Proposed Method X
   1. Basic Neural Network X
   2. Custom Loss Function X
5. Comparison and Analysis X
6. Conclusion X
7. References X
8. **Group Members**
9. Myat Nyein Soe U1

Roles:

* Implement scikit-multilearn stock classifiers
* Implement basic neural network
* Hyperparameter tuning

1. Lim Jia Yao Christopher U1721955L

Roles:

* Implement basic neural network
* Implement novel loss function for neural network
* Literature review
* Report writing

1. Kwek Jia Ying U1

Roles:

1. Ng Kang Loon U1

Roles:

1. **Literature Review**

Recently, research in multilabel classification is gaining a lot of interest as more and more categorization problems that needs to be solved are complex in nature and instances do not belong to a single class most of the time. Such examples include the categorization of music, text, images, videos, etc. Multilabel classification is difficult as the number of labels to be predicted differs greatly even within the same dataset. Some instances could belong to only 1 class while others could belong to many classes. Thus, much research has been conducted to find fast and effective methods to perform multilabel classification.

The most common strategies that are used to solve multilabel classification problems are problem transformation and problem adaptation. Problem transformation involves transforming the problem domain from one complicated problem into many simpler problems. In the case of multilabel classification, the problem is broken down into multiple single label classifications and their outputs are combined to produce the final output. One classic example of this is the binary relevance method in multilabel classification literature. Problem adaptation involves adapting algorithms from other domains to fit multilabel classification. One example of this is by extending the classical k-nearest neighbours (KNN) classifier to accept multilabel data, which is then called multilabel k-nearest neighbours (MLKNN). On top of these usual strategies, convolutional neural networks (CNNs) with varying architectures have also been used where the problem domain includes data which have some sort of spatial relationship, such as images and text.

In this report, we will focus mainly on data that are non-image and non-text in nature. Each dataset will have a fixed number of input features as well as a fixed number of output labels. The datasets that we will be using are from the Knowledge Extraction Evolutionary Learning (KEEL) multilabel datasets [1].

In this section, we will review some of the most common approaches to multilabel classification, as well as a few novel approaches taken by researchers.

* 1. **Binary Relevance**

Arguably the most classic method to solving any multilabel problem is the binary relevance method. As mentioned above, this method is using the problem transformation strategy, or the divide-and-conquer strategy as well. As typical classifiers are only able to predict for one output given a set of inputs, the idea behind binary relevance is to then divide the problem of predicting for many labels, into predicting for one label, many times. Theoretically, this works perfectly well in the scenario that all the labels are independent from each other. However, that is rarely the case for any type of multilabel classification as labels tend to have hidden correlations.

Binary Relevance is however, still a high-level strategy and there is still a need to decide the underlying classifier to use for each label.

* 1. **Multilabel K-nearest neighbours (MLKNN)**

In [2], Zhang and Zhou proposed an extension of the traditional K-nearest neighbours (KNN) algorithm for multilabel classification problems. The basic idea for this method is that for every instance, it’s k-nearest neighbours will be identified and used to generate the labels for the given instance. This is done by utilizing maximum a posteriori (MAP) estimation, which is a Bayesian-based approach to estimating a distribution and model parameters that best explains an observed dataset [3].

* 1. **Classifier Chain**
  2. **Binary Relevance k-nearest neighbours (BRKNN)**
  3. **Multilabel Adaptive Resonance Associative Map (MLARAM)**
  4. **Label Powerset**
  5. **RAndom K-labELsets (RAKEL)**
  6. **Majority Voting**
  7. **Embedding**

1. **Implementation of state-of-the-art methods**

In this section, we will elaborate on the parameters and specific conditions on how we implement each multilabel classification method.

* 1. **Binary Relevance**
  2. **Multilabel K-nearest neighbours (MLKNN)**
  3. **Classifier Chain**
  4. **Binary Relevance k-nearest neighbours (BRKNN)**
  5. **Multilabel Adaptive Resonance Associative Map (MLARAM)**
  6. **Label Powerset**
  7. **RAndom K-labELsets (RAKEL)**
  8. **Majority Voting**
  9. **Embedding**

1. **Proposed Method**

After reviewing multiple algorithms, we have proposed our own method to tackle the multilabel classification problem. In this section, we will provide more details on our implementation.

* 1. **Basic Neural Network**
  2. **Custom Loss Function**

The most common loss function used for neural network-based multilabel classification is binary cross entropy, which is another name for logarithmic loss. The loss function is as shown below:

Where is the ground truth label of the ith data point, is the predicted label of the ith data point, and N represents the total number of data points. From the equation, we can see that when is 1, is added to the summation term, and conversely, when is 0, is added to the summation term. We can also see that when , the term added to the summation term is log(1) = 0, and when is far from , the term added to the summation term tends to negative infinity, which will later be negated.

Prior to the loss function, the output of the final layer would typically go through a sigmoidal activation function in order to limit the output between 0 and 1, as the loss function expects a probability distribution for .

There are major shortcomings in using this loss function for multilabel classification, especially so if the labels are imbalanced, where there are a lot more 0s than 1s in the labels. As the loss weightage for false positives and false negatives are the same, given that the number of 0s typically far exceed the number of 1s in any given dataset, using simple binary cross entropy will cause the neural network to learn to predict 0s most of the time as the loss will be very low. In extreme cases, even if the neural network predicts every single label as a 0, it will still achieve a decently high hamming accuracy and a very low loss.

Due to the aforementioned shortcomings, we have decided to implement a custom loss function instead. Our custom loss function is split up into 2 components: a main propensity loss function and a secondary exact-match hinge filter. The propensity loss function is inspired by [] and [] aims to deal with label imbalance. First, a propensity score is calculated for every label based on its frequency of occurrence in the known (training) dataset, then this propensity value is inserted into a modified hamming loss. The equation for calculating each label’s propensity is modelled after a sigmoidal function and is given by:

Where is the propensity score for label ℓ, A and B are application specific constants (in the general case, it is recommended for A=0.55 and B=1.5 as mentioned in []), N is the total number of data points, and is the number of data points which contain the label ℓ.

The modified hamming loss is given by:

Where N is the total number of data points, L is the total number of labels, is the propensity score for label j, is the ground truth value of the ith data point and the jth label, is the predicted value of the ith data point and the jth label,

and

Note that this equation has been slightly modified as compared to the referenced equation (4) of []. This is because the referenced paper proposes a flawed equation, in which the binary selector will always reduce to 1 and thus becoming irrelevant.

We can see that this loss function discriminates each label independently and applies a different weightage to each label based on the number of occurrences of each label from the term . When , the binary selector term will become 0 which will in turn cause the loss for that label to be 0 as well, which is desirable. When , the loss for that label will be equal to the squared difference of the ground truth label and the predicted value, then scaled by the propensity of that label.

The secondary exact-match hinge filter is applied on the result of the above hamming loss, and what it does is as long as all the rounded off predicted values of all labels match the ground truth value, i.e:

Where is an exclusive NOR Boolean function which returns 1 when x=y and 0 otherwise. This term acts as a hinge that will allow the loss to propagate only when there is at least 1 incorrectly classified label. If all L labels are classified correctly (after rounding the predicted values off to either 0 or 1), then the final loss will be 0.

Therefore, the final loss function is:

1. **Comparison and Analysis**
2. **Conclusion**
3. **References**

[1] <https://sci2s.ugr.es/keel/multilabel.php>

[2] <https://cs.nju.edu.cn/zhouzh/zhouzh.files/publication/pr07.pdf>

[3] <https://machinelearningmastery.com/maximum-a-posteriori-estimation/>

[4] <https://www.kdd.org/kdd2016/papers/files/rfp0261-jainA.pdf>

[5] <http://ceur-ws.org/Vol-2126/paper10.pdf>