Lab 2: Multi-label Learning

INF581: Advanced Topics in Artificial Intelligence

1 Introduction

This lab deals with analyzing and building predictive models for multi-labelled data. We will use data from the Internet Movie Database[1] (IMDb).

1.1 Requirements

Python with Numpy, Matplotlib, ScikitLearn, Pandas, and NetworkX libraries.

2 Setup

The file `imdb.csv.gz` contains a vector representation of genres (i.e., labels), and a raw text summary of each film. The raw text summary can be converted to a bag of words representation by running `run_preprocessing.py`, which produces a new file `imdb_vectorized.csv.gz`. It should take 2-5 minutes.

Note: It could be suggested to use `music.csv` for testing, since it is much faster. Note that in some of the figures below, tasks are illustrated using this Music dataset, where pieces of music are associated with emotions.

3 The Tasks

You will find that the file `run_tasks.py` already contains code to load the data and split it into `X` and `Y` matrices.

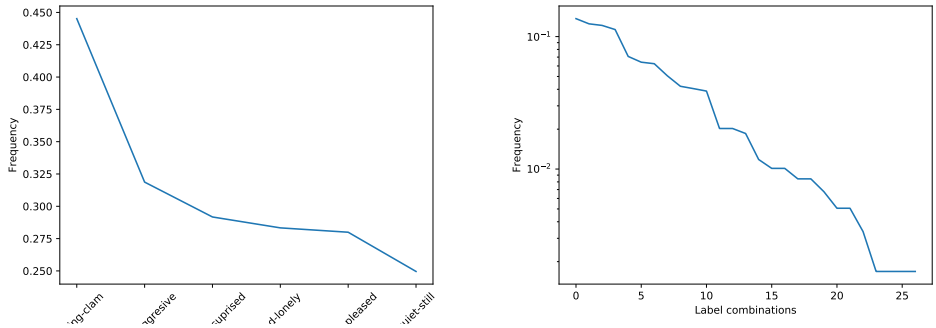
We will first explore the data to get an idea of which multi-label learning algorithms would be most appropriate to employ in this context.

Task 1

Calculate some basic statistics and plots on the multi-label dataset (in particular, the `Y` matrix, where each row stores a binary vector of label relevances (one column per label)):

- *Label cardinality*: the average number of relevant labels per example
- *Label density*: the proportion of relevant labels to non-relevant labels (inverse sparsity)
- The number of *unique label vectors* (which label combinations are found in the data)
- The *relevance frequency* of each label (the proportion of relevant examples to each label)
- Plot labels vs their frequency of occurrence, in decreasing order. (This is shown in Figure 1a for the Music dataset).

[1] `http://imdb.com/`



(a) (b)

Figure 1: The horizontal axis shows the labels (Figure 1a) and label vectors (Figure 1a) and the vertical axis shows the corresponding proportion in the training set (in log scale for the combinations).

- Plot label *combinations* (unique label vectors) vs frequency, in decreasing order. (Already shown for the Music dataset in Figure 1b).

We next take a look at the relationships among the labels. First, **marginal dependence** among the labels (ignoring the input space). We can make a simple analysis of linear dependence with a correlation matrix.

Task 2

- Create a pairwise correlation matrix of *the top 10 most common labels* and visualise it as a heatmap (a function `make_heatmap` is provided in `utils.py`).
  Hint: NUMPY provides `corrcoef`.
- Turn the matrix into a graph (`make_graph` is provided in `utils.py`).

See Figure 2 for examples on the Music data.

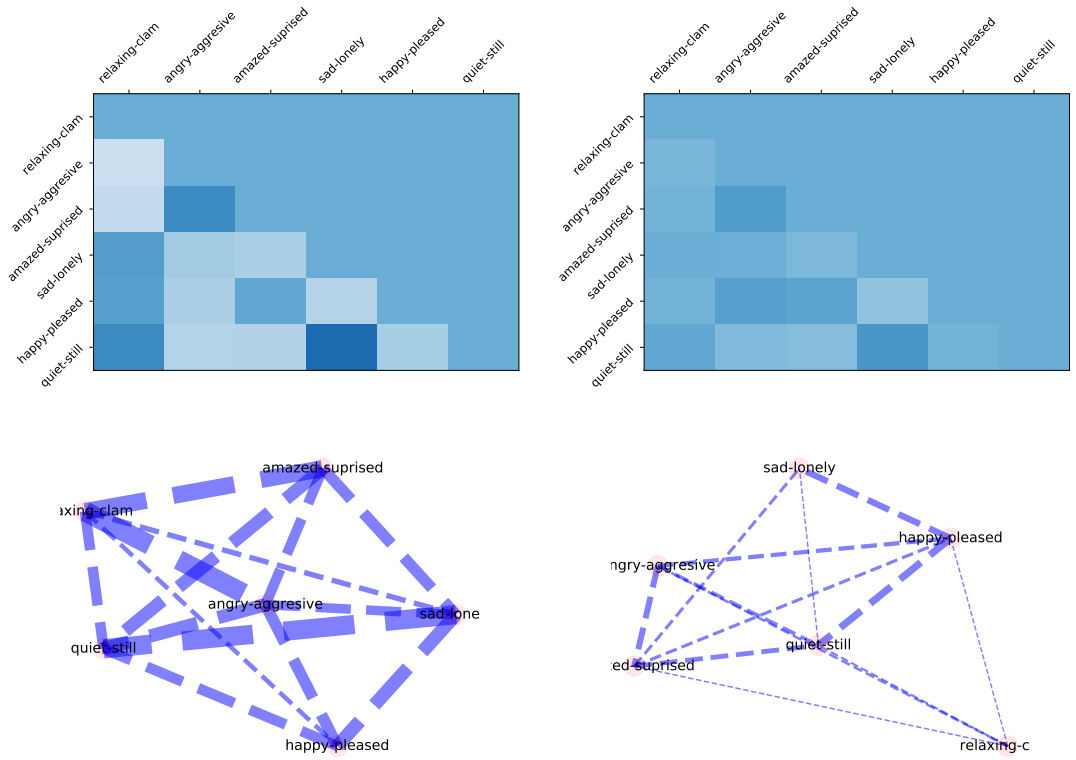
The following code shows how to initialize and train a multi-label classifier (namely, the *binary relevance* method of independent classifiers), provided by `binary_relevance.py`.

```
h = BR()
h.fit(X,Y)
Y_pred = h.predict(X_test)
```

Task 3

Finish the code in the `get_errors` function in `utils.py` file, to return an $N \times L$ matrix containing the errors of each prediction (e.g., error matrix $\mathbf{E}$ where $\mathbf{E}_{jk} = [\![y_j^{(i)} = h_j(\mathbf{x}_i)]\!]$, i.e., simple accuracy, where $y_j^{(i)}$ the j-th label (column) of the i-th instance (row)). You can use cross-fold evaluation to ensure that you get a vector of errors for each and every instance.

Then, repeat Task 2 but on the *error* correlation matrix rather than the label correlation matrix,



N.B. Under different random seeds, the graphs may have a different layout. It may be possible to perceive that conditional dependence is weaker, and should be clear that marginal and conditional dependence are related but not equivalent.

Figure 2: View of marginal dependence (left) and conditional dependence (right).

thus producing the *conditional* correlation matrix. Try using both logistic regression and decision trees as the base model and see if there is any difference.

Task 4

Build the label powerset method (in `label_powerset.py`), i.e., a multi-*class* classifier where each class represents a particular combination of labels/binary vector. Your `predict` function should return an $N \times L$ matrix where each row is

$$\hat{\mathbf{Y}}_{i,:} = \underset{\mathbf{y} \in \mathcal{Y}}{\operatorname{argmax}} P(\mathbf{y}|\mathbf{x}_i) = [\hat{y}_1^{(i)}, \ldots, \hat{y}_L^{(i)}] \in \{0,1\}^L$$

i.e., the i-th example/row contains the MAP estimate vector/combination of labels most likely given input $\mathbf{x}_i$. The pmf P is learned as part of your model (use logistic regression). Initially, the set $\mathcal{Y}$ contains all combinations which you observed in the training data.

For datasets such as IMDB, $\mathcal{Y}$ will be too large (inference time will be infeasibly large). Indeed, the upper limit is $|\mathcal{Y}| = \min(N, 2^L)$. Consider, therefore, only the $n = 100$ most frequent combinations (i.e., $|\mathcal{Y}| = n = 100$). That means that at training time you will discard some instances.

Hint: You should be able to re-use some code that you already wrote, i.e., the `count_combinations`

function.

Then compare your label powerset classifier to BR under Hamming loss, and 0/1 loss (use cross-fold validation as you implemented earlier in `get_errors`), and running time. Do results make sense? Try for higher and lower values for n, and observe the run-time vs predictive performance tradeoff.

Hint: ScikitLearn has implementations of these loss metrics, or you can make use of the `get_errors` function you wrote earlier.

We have nice little representation of knowledge. Not only about a classification task, but about several human concepts and the relationships among them. A relatively simple agent could make use of these for tasks other than just labelling films.

Task 5

For this task, you must use the IMDB data.

Take your classifier of the previous task and implement also the `predict_proba` function. As a probabilistic multi-class problem, you will have a distribution $P(\mathbf{y}|\mathbf{x})$ over all $\mathbf{y} \in \mathcal{Y}$. You can use this to obtain a vector of approximate marginal probabilities such that the i-th row of the matrix you return from this function contains the vector

$$[P(y_1 = 1|\mathbf{x}_i), \ldots, P(y_L = 1|\mathbf{x}_i)]$$

Train your classifier on the *full* dataset, and write the model to disk. Now, just for fun, a subjective evaluation: Write a brief summary for a story ('creative writing') in a file called `my_summary.txt`. Then run the script `run_extra.py`, which creates a new test instance based on your text, loads your model (from file `h.dat`) and prints out the marginal probabilities for each label. Does the labelling of your model make sense?