Project Report

1. Motivation and Problem Statement

The EurLex data set is a collection of multilingual documents about European Union law. These documents are indexed according to different categorizations and allows for multiple search facilities. The database constitutes a very challenging multilabel scenario due to the high number of possible labels.

a. Preprocessing of the dataset to obtain a collection of useful and relevant documents.
    - Need to filter out the documents based on our language of interest i.e. documents in English.
    - Remove the documents with any discrepancy or missing data.
b. Finding out suitable classification techniques for multilabel classification of the documents and train a model based on these concepts.
c. Comparison of the outcomes of applied classification techniques on test data based on their evaluation metrics.
d. Re-evaluation and cross validation of the data set to achieve an unbiased result..
e. Visualisation of the data along the process for ease of understanding.

2. Data Set

Given data set was the collection of legislative documents of European union from EUR-Lex 2006 data set created by TU Darmstadt. There were 19940 documents and 3953 unique labels named under EUROVOC Descriptor in each document. The input data for our model was the text data including the document title in each of the documents and the target attribute was their respective EUROVOC descriptor.

The average number of labels per documents was 5.31, while on an average, a label occurs in 26 documents. The count of labels occurring in a given document, ranges from 1 to 1253. The frequency distribution for each label sorted in increasing order of their frequencies can be seen in the below graph. As seen from the graph, the data set was highly imbalanced with respect to the data labels. To balance out the data set to some extent, we have ignored 1955 documents with label counts of five or less. Also, we found, out of 19940 documents, 592 documents either had missing descriptors or missing text. We removed these files from our input data set for training our models.

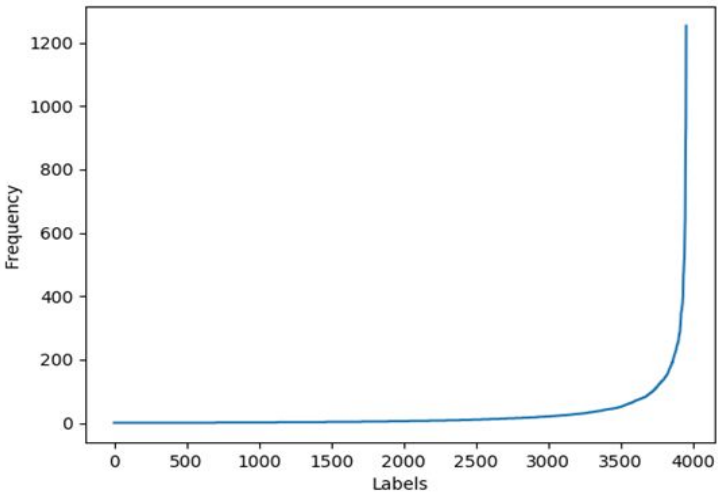

Figure: Label – Frequency Distribution

3. Concept

a. **Binary Relevance(BR):** In Binary relevance method for multi-label classification an ensemble of single-label binary classifier is trained. Each classifier in the ensemble is trained for one class. These classifiers predict the membership or non-membership of one class. Then union of output class labels of all these classifiers is taken as the multi-label output. This approach ignores possible correlations between class labels. Example: X is an independent feature and Y's are the target variables.

X	y1	y2	y3	y4
x1	0	1	1	0
x2	1	0	0	0
x3	0	1	0	0
x4	1	0	0	1
x5	0	0	0	1

In BR this can be seen as four different single class classification as shown (each color represents a different classifier).

X	y1	X	y2	X	y3	X	y4
x1	0	x1	1	x1	1	x1	0
x2	1	x2	0	x2	0	x2	0
x3	0	x3	1	x3	0	x3	0
x4	1	x4	0	x4	0	x4	1
x5	0	x5	0	x5	0	x5	1

b. **ML-KNN:** ML-KNN means multi label K-nearest neighbor. This algorithm is based on the popular K-nearest neighbors (KNN) algorithm. For each test instance in the training set, we first find out its neighboring K instances and then the label set of these K neighbors. We use the maximum a posteriori (MAP) principle to find out the label set of the test instance in consideration.

c. **Transformation to Data Set:**
   - **Tokenization:** Transformation of a text into individual tokens, which can be further processed. Tokenizer uses bag of words approach and hence loses information about the order of the words.
   - **Stop word removal:** Stop words are removed from the text to increase the performance. Moreover stopwords do not have discriminative power to discriminate documents from one another. The stop words lists Eur Lex dataset was given along with the dataset and that list is used to remove the stop words from the given Eur Lex dataset.
   - **Stemming:** Stemming is the processing of morphological variants of a word and transforming those to root\base word. Snowball stemmer is used for stemming in our dataset.

d. **Feature Selection:** We have opted two approaches for feature selection.
   - **TF-IDF:** This method of feature selection evaluates how important a word is to a document in a collection or corpus. The importance increases proportionally to the number of times a word appears in a document but is inversely proportional to the number of documents in which the word appears.
   - **Term Frequency:** The frequency of tokens in each document is taken into consideration. This data is then transformed into term frequency matrix for further processing by the model.

e. Comparison of two Multi label classification approaches:

	ML-KNN	**Binary Relevance**
1	ML- KNN approach for multi label classification takes into account the correlation between labels.	Binary Relevance approach does not take into account the correlation between labels while classifying.
2	This approach is not as simple as Binary Relevance hence not that popular.	This approach is very easy to implement hence it is quite popular.
3	ML KNN is a lazy learner model i.e. it does not actually build a model by learning to be later used.	Binary relevance method learns a model first and the uses that for further classification.

4. Implementation

a. Tools and Libraries Used

Sr. no.	Library Name	Version	Use
1.	Natural Language Toolkit	3.4.1	To tokenize and stem words.
2.	Urlib	-	To read HTML or HTM urls.
3.	BeautifulSoup	4.7.1	To parse content from an HTML webpage.
4.	scikit-learn	0.21.2	a) To vectorise text content. (TF-IDF/TF) b) To import classifiers. c) To split the data in K folds. d) To find the evaluation metrics
5.	scikit-multilearn	0.2.0	To work with multilabel classification problem.
6.	scipy	1.3.0	To work with sparse matrices.

b. Workflow

The source code is divided into four functions. Each function represents the certain process that's done on the data.

TextTrain():

Input: none

Output: X - a sparse matrix.

Use: The function goes through all the .html (or .htm) files kept in a given folder. Empty and corrupt files were skipped over. Stores the text content (including heading) for each file. Then vectorises, using either TF-IDF or Term Frequency. Conceptually this returns a term-document matrix, with weight depending on the vectoriser used. Returns this matrix, named as X. This is a sparse matrix.

DescTrain():

Input: none

Output: Y - a sparse matrix.

Use: The function goes through all the .html (or .htm) files kept in a given folder. Empty and corrupt files were skipped over. Labels which occur less than 5 times in dataset, were not considered. Stores the descriptors from each file. Makes a sparse matrix marking presence (or absence) of a descriptor in a file.This is the sparse matrix, named Y, that is returned.

DataSplit(X,Y)

Input: X,Y - sparse matrices

Output: X_train, X_test, y_train, y_test - sparse matrices

Use: This function performs K Folds Cross Validation. Here the data gets split into K folds (K=5); testing is performed K times, where successively 1 fold is used for testing in each iteration and remaining K-1 folds are used for training. This function returns four sparse matrices: X_train, X_test, y_train and y_test.

Classification(X_train, X_test, y_train, y_test)

Intput: X_train, X_test, y_train, y_test - sparse matrices

Output: none

Use: This function take the data splits, fits a classifier to them and then predicts descriptors (y_hat) for each file. We used two classification approaches, Binary Relevance with Naive Bayes classifier and Multilabel K Nearest Neighbour (MLkNN), with K=20. Comparing the prediction (y_hat) and the ground truth (y_test), we get evaluation metrics like Micro F1-score, Macro F1-score, Micro-recall, Macro-recall, Micro-precision, Macro-precision and Hamming loss.

5. Evaluation

a. Evaluation Metric

A confusion matrix for binary classification looks like:

Predicted \ Actual	Actual Positive	Actual Negative
Predicted Positive	True Positive (TP)	False Positive (FP)
Predicted Negative	False Negative (FN)	True Negative (TN)

Then Precision, Recall and F1-score are defined as following:
Precision (P) = TP / (TP + FP)
Recall (R) = TP / (TP+ FN)
F1-score = 2*P*R/ (P + R)

These metrics have to be updated to adequately handle the multilabel scenario. In multilabel classification, for each label one could create a confusion matrix and hence could also calculate these metrics. Now the next logical step is how would we average these individual metrics for each label, to somehow comment in general about the classification.

One of the easier way, is to assume that data is not skewed and that all the labels present in the data are equally important. This assumption leads to the understanding that we could directly take an average of all the individual metrics. This approach is called the **macro-average**.Whereas a **micro-average** will aggregate the contributions of all classes to compute the average metric.

Therefore, formula for micro-averages will look like following, where 'C', is the set of labels.

- Microaveraging Precision $Prc^{micro}(D) = \frac{\sum_{c_i \in C} TPs(c_i)}{\sum_{c_i \in C} TPs(c_i) + FPs(c_i)}$

- Microaveraging Recall $Rcl^{micro}(D) = \frac{\sum_{c_i \in C} TPs(c_i)}{\sum_{c_i \in C} TPs(c_i) + FNs(c_i)}$

Image: Micro-Averaging

And similarly formula for macro-averages will look like:

. Macroaveraging Precision $Prc^{macro}(D) = \frac{\sum_{c_i \in \mathcal{C}} Prc(D, c_i)}{|\mathcal{C}|}$

. Macroaveraging Recall $Rec^{macro}(D) = \frac{\sum_{c_i \in \mathcal{C}} Rcl(D, c_i)}{|\mathcal{C}|}$

Image: Macro-averaging

One other measure that makes overall comment on the multilabel classification is **Hamming Loss**. It basically counts the fraction of disagreements between predicted labels and the ground truth.

b. Evaluation Strategy

The values of evaluation metric for single split of training and testing set, should not be readily accepted. These values could be misleading. To rule out the possibility of a favourable spilt; one generalises the evaluation metric values by taking average over K folds. This method is called **K Fold Cross Validation**, where the original dataset is divided into K equal folds. Of the K subsamples, a single fold is retained as testing set, and the remaining K−1 subsamples are used as training data. The K results, one from each iteration of validation, can then be averaged to produce a single estimation.

In the project we implemented K Fold Cross Validation, with K=5 and found all the above mentioned evaluation metrics.

c. Parameters and Results

There are two sets of parameters we work with:
Set 1: Vectorisation of text data, either using Tf-idf term weighting or using term frequency.
Set 2: Classification approach, either using MLkNN (k=20) classifier or using BR with Naive Bayes classifier.

Following are the results of classifiers with vectorization approach. These evaluation metrics were generalised using 5 Fold Cross Validation.

Vectorisation \ Classifier	MLkNN(k=20)	BR with NB
Tf-idf	Micro F1-score: 0.292 Macro F1-score: 0.039 Micro recall: 0.188 Macro recall: 0.04 Micro precision: 0.687 Macro precision: 0.041 Hamming Loss: 0.013	Micro F1-score: 0.198 Macro F1-score: 0.03 Micro recall: 0.116 Macro recall: 0.028 Micro precision: 0.722 Macro precision: 0.037 Hamming Loss: 0.013
Tf	Micro F1-score: 0.255 Macro F1-score: 0.036 Micro recall: 0.162 Macro recall: 0.038 Micro precision: 0.599 Macro precision: 0.038 Hamming Loss: 0.013	Micro F1-score: 0.165 Macro F1-score: 0.023 Micro recall: 0.092 Macro recall: 0.021 Micro precision: 0.873 Macro precision: 0.028 Hamming Loss: 0.013

Using Micro-averaging values from above four settings, we plotted micro-precision against micro-recall. An ideal classifier would have a value of one for both micro-precision and micro-recall.

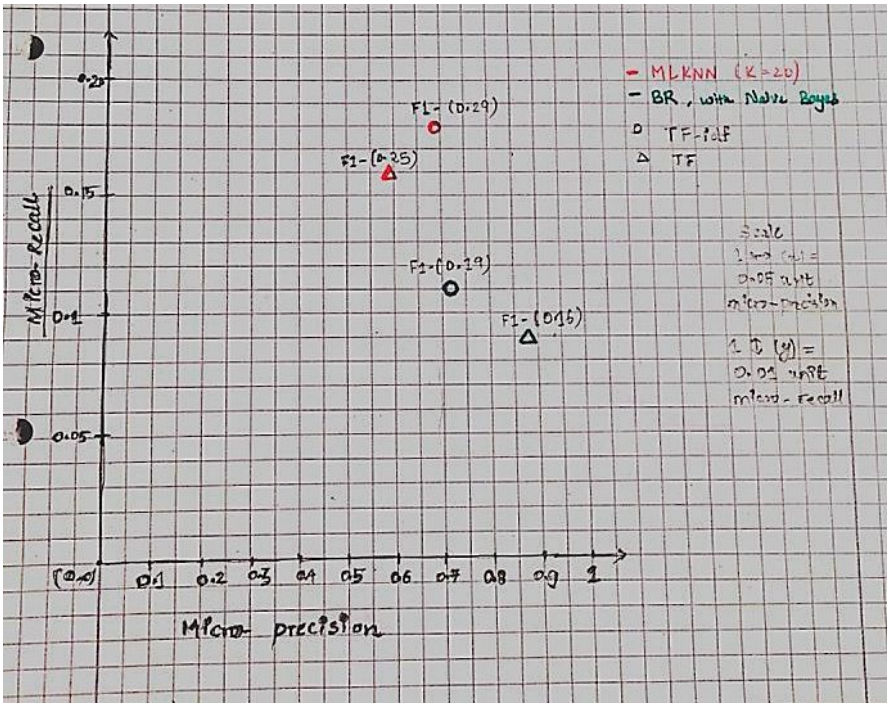

Figure: micro-precision vs micro-recall.

Scale: 1 x is 0.05 units of micro-precision; 1 y is 0.01 units of micro- recall.

For micro-F1 values, it was found that when dataset was vectorized using Tf-idf weighting, it outperformed when dataset was vectorized using term frequency weighting.

Similarly, the micro-F1 values for MLkNN(k=20) we superior to those of binary relevance used with naive bayes classifier. Interestingly though, MLkNN classifier performed slightly better than BR, in terms of micro-recall, while BR with NB classifier performed much better than MLkNN in terms of micro-precision (the graph has different scaling to two axis.)

6. Conclusion

We were able to solve the problem statement (that of dealing with a large dataset with multi label classification) up to an extent. We were able to predict the labels using classification algorithms like binary relevance and MLkNN, although sometimes the evaluation metric values were mediocre.

The two classifiers performed almost equally well. Though given that these data come from legal domain, where false negatives generally have higher cost than false positives, it would be appropriate to use a classifier that gives better values for (macro/micro) recall. Hence, MLkNN would be preferred, even though it only provides a slightly better value than BR with NB, in terms of recall.

Computing power was a bottleneck for our implementation. To access the data faster, strategy like storing .html files to .csv, could be looked into. Other ideas which were not implemented but could be used in future are - a) finding correlation between the labels b) exploring label data set using clustering algorithms in a high dimensional space c) finding correlation between the terms present in data d) dimensionality reduction of the text data set.

References

1. Image: Micro-Averaging: Deep Dive into multi-label classification web article. Kartik Nooney
2. Image: Macro-Averaging: Deep Dive into multi-label classification web article. Kartik Nooney
3. Ml-knn:A Lazy Learning Approach to Multi-Label Learning, Min-Ling Zhang, Zhi-Hua Zhou, National Laboratory for Novel Software Technology Nanjing University, China

Steps to run the code

There are certain things to be taken care of:

1. Directory Path: in line 49 and 94, system path of the directory containing the .html file should be replaced.
2. Url Path: in line 51 and 96, url path of the .html file should be replaced.

```
46  def TextTrain():
47  #FOR WORKING WITH DIRECTORY CONTAING FILES
48  #Take notice of url (and paths) that you give as vairable, these are local dependent
49      for filename in os.listdir("D:\\ATML Project Data\\RandomTrain"):
50          if (filename.endswith(".html") or filename.endswith(".htm")) and (filename not in BadFiles):
51                  url = str('file:///D:/ATML Project Data/RandomTrain/' + filename)
```

3. Parameter Setting: Uncomment the section and run as desired
   a. Tf-idf or Tf Vectoriser:

```
72        # # AT A TIME, ONLY ONE TYPE OF VECTORIZER SHOULD BE ACTIVATED.
73        # # ACTIVATE WHICHEVER TYPE OF VECTORISER YOU WANNA WORK WITH.
74
75        # #FOR MAKING SPARSE DOC-TERM MATRIX. WITH TF-TDF WEIGHTING
76        # vectorizer = TfidfVectorizer()
77        # X = vectorizer.fit_transform(corpustext)
78        # print(vectorizer.get_feature_names())
79
80
81        # # #FOR MAKING SPARSE DOC-TERM MATRIX. WITH Term frequency WEIGHTING
82        vectorizer = CountVectorizer()
83        X = vectorizer.fit_transform(corpustext)
84        # print(vectorizer.get_feature_names())
85
```

   b. Classifier: MLkNN or BR

```
125   #---------------------Defining Classifiers-------------#
126   # # AT A TIME, ONLY ONE TYPE OF CLASSIFIER SHOULD BE ACTIVATED.
127   # # ACTIVATE WHICHEVER TYPE OF CLASSIFIER YOU WANNA WORK WITH.
128
129
130       # MLKNN CLASSIFICATION, WITH K=20
131       # classifier = MLkNN(k=4)
132
133       #BR
134       classifier = BinaryRelevance(GaussianNB())
135
```

The code is currently running with parameters set as, Classifier: BR with NB and Vectoriser: Term frequency. Here, one could also change the number of nearest neighbour MLkNN classifier considers, by changing the value of K at line 131.

4. One can change the number of folds for K fold cross validation, at line 148.

```
147  #----------------------------K Fold Cross Validation----------------------#
148      kf = KFold(n_splits=5, shuffle= True)
149      for train_index, test_index in kf.split(X):
150          X_train, X_test, y_train, y_test = X[train_index], X[test_index], Y[train_index], Y[test_index]
```

5. Run the code using the following command: python MultiLabelClassifier.py