Text Mining Final Assignment

Ernest Vanmosuinck
3210359
s3210359@umail.leidenuniv.nl

Pengxu Zheng
2917211
s2917211@vuw.leidenuniv.nl

January 2022

1 Introduction

In this project, we aim to answer the question: can open-source machine learning libraries have good performances on classifying text from political election manifestos? The task was modeled from the research by Verberne et al. [1], where the datasets were extracted and employed. We preprocessed the text data extracted from XML documents and investigated the performance of multi-label classifiers available on the open-source Machine Learning library *scikit-learn*, which were all applied to different multi-label problem transformation methods. We then compared the performance of each datasets and methods, and applied the best performing classifier to all leftover manifestos.

2 Background/Related work

The task of properly assigning a topic or theme to documents falls under the task of document/text classification, a form of supervised learning when the targets are labeled. In our case, the election manifestos have been assigned more than one theme, so this project falls into the task of *multi-label* classification [2]. The task of categorizing political election data is a very important one, as it can help the general public get a faster understanding of the political ideologies that parties support. This further supports the use of mutli-label text classification for categorizing political manifestos. In 2003, Pouliquen et al. introduced a multi-label classifier trained on European legislation documents [3]. The authors developed a language-independent system which maps raw documents into the multilingual conceptual thesaurus named `EUROVOC` to conduct automatic text annotation, which to some extent share a similarity to the research question that we proposed in this project. As stated in the introduction, this project is modeled after the research by Verberne et al. [1], in which they evaluated the performance of the Linguistic Classification System [4] on unseen election manifestos when trained on older manifestos.

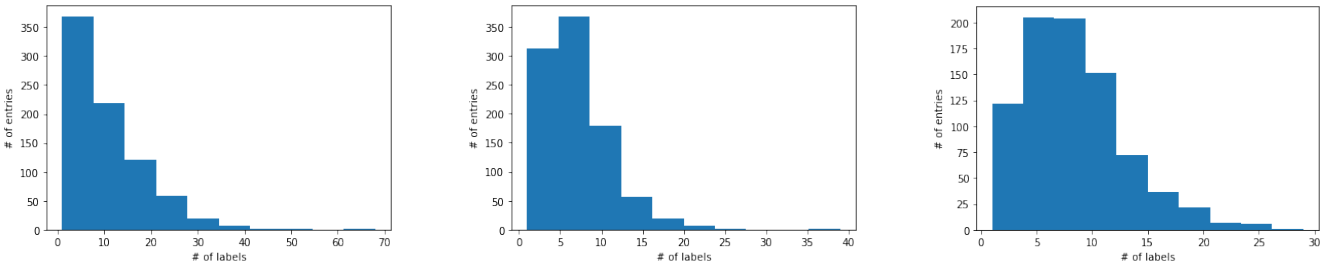
3 Data

For this project, we will be working with three different collections of Dutch election manifestos, which exist digitally in the forms of PDFs. Luckily for us, those texts have already been converted to

the XML formats for us by the authors of the research paper, Verberne et al. [1]. We will be working with the election manifestos of years 1986, 1994 and 1998. The manifestos are grouped by parties and each paragraph is assigned a set of themes. It is worth mentioning that the paragraphs did not all have the same number of themes assigned to them, as can be seen in figure 1, spread among the three manifestos. Most paragraphs contained 0-10 different themes. A statistical description of this data is can be examined in table 1.

Year	# Word	Voc. size	# Texts	# Themes	# Words/text	# Themes/text
1986	290,942	32,836	797	214	373	10.5
1986	269,270	32,499	951	210	284	6.9
1986	244,697	31,162	826	218	302	8.3

Table 1: Statistics on Classification Data



(a) 1986 Manifesto (b) 1994 Manifesto (c) 1998 Manifesto

Figure 1: Labels count histogram on the different datasets

The hierarchy of the XML files has been annotated as follows:

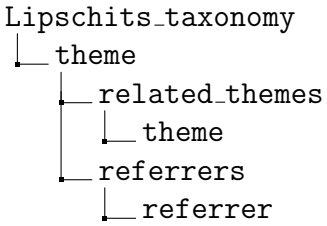


Figure 2: Hierarchy of the processed XML files

At the **theme** level, the ID of the theme, namely, the word itself, is shown. Under the **theme** level, tags **related_themes** and **referrers** collect all relevant information regarding this specific theme word. If for a specific theme there exists related themes, then the related themes will be listed under the **related_themes** category. Though in practice this **related_themes** category was observed to be empty for the most of the time. An example of the structure can be viewed in figure 3, taking the theme "`armoedebestrijding`" as root.

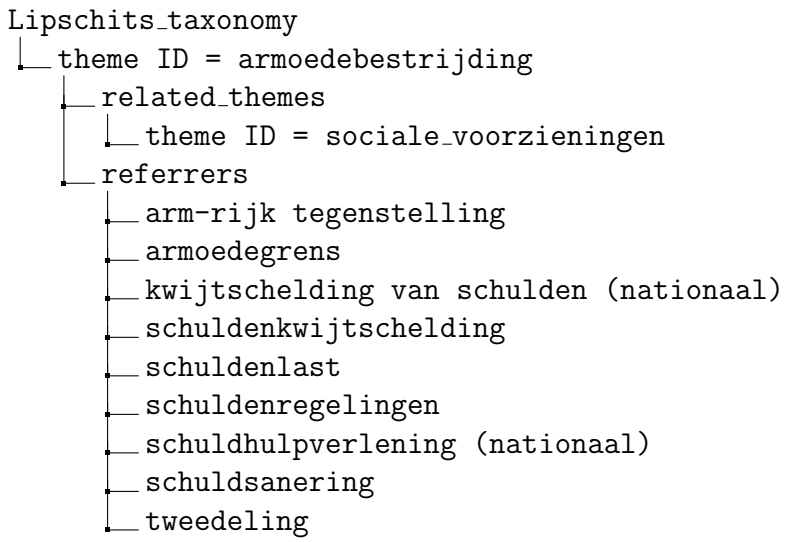


Figure 3: A fully populated theme example from year 1998's XML file

3.1 Preprocessing

The text data contained in the XML documents contained quite a bit of noise, most likely due to OCR errors. Those errors were handled along with the preprocessing after the data was extracted from the XML documents. For this project, we employed the open-source library `spacy` [5]. `spacy` offers a wide range of functions to preprocess text data in a variety of languages, including Dutch. There is also the open-source library `NLTK` [6], which offers the same functionalities. We have decided to use spacy, as we have concluded that it was more user friendly and faster to use [7].
We applied lower casing to the text, followed by a removal of any punctuation and digits. The punctuation was taken from the `string` library, and includes the following: !"#$%&'()*+,-./:;>=?@[\]^_`{|}~. We then tokenize the text and keep the lemmas of all tokens. Lemmatization is the process of keeping the canonical form of a set of words.
We then applied a Vectorizer to change the text data to transform it to a sparse vector by computing the tf-idf value of each term in that paragraph. The tf-idf value of a term is computed by multiplying the term frequency (number of times term occurs in a document) with the inverse document frequency (number of documents divided by the number of documents in which the term occurs). The vector is normalized with 'L2' normalization, and the `n_gram_range` parameter is set to (1,3).

Finally, the theme labels had to be modified to be used by classifiers to make predictions. To solve this issue, the data was transformed to binary representation using `MultiLabelBinarizer`, meaning it will transform the list of string labels into a list of binary representation of said topics. The `MultiLabelBinarizer` takes the list of labels and creates a list of binary representation for said list over the list of all labels occurring in the dataset. An example of this process can be viewed in figure 4 with dummy data.

	labels	a	b	c	d	e
0	[a, b, c]	1	1	0	0	1
1	[c, e]	0	0	1	0	1
2	[a]	1	0	0	0	0
3	[b, c, e]	0	1	1	0	1

Figure 4: Multi-label Binarization on dummy data

4 Methods

When looking at the task of multi-label classification, there are multiple existing approaches available. In this project, we will be focusing on three: Binary Relevance, Classifier Chaining and Label Powerset. All approaches are easily implementable with the open-source library `scikit-multilearn` [8]. The main idea is to transform the multi-label classification problem into binary and multi-class classification problems with each approach being discussed in the following subsections.

4.1 Binary Relevance

Binary relevance is considered as the most intuitive approach to transform a multi-label classification problem with `k` labels into `k` binary classification problems [9]. In our implementation, we introduced scikit-learn's `BinaryRelevance` transformer from `skmultilearn.problem_transform` to perform problem transformation along with the six classifiers chosen for the multi-label classification task.

4.2 Classifier Chaining

Classifier Chaining is a technique also aims convert the multi-label classification problem to a series of binary classification problems. Unlike divide-and-conquer from the `Binary Relevance` approach, Classifier Chaining can be seen as a linked list of binary classifiers with each prior classifier's output being fed in to the proceeding classifier's input that forms a chain of classifiers. According to scikit-learn's documentation, the each model makes a prediction based on all available features plus the prediction made by the models earlier in the chain [10].

4.3 Label Powerset

Label Powerset takes a different approach than Binary Relevance and Classifier Chaining. Contrary to the approach of transforming a multi-label classification into multiple binary classification problems, Label Powerset made the transformation from multi-label to multi-class by making combinations of all available labels into a power set and apply a multi-class classifier to every single combination of labels. For example, suppose there exists 3 different labels A, B, and C for a multi-label problem. Thus, there will be $2^3 = 8$ possible combinations of all labels, namely, $[0, 0, 0]$, $[0, 0, 1]$, $[1, 0, 0]$, ..., $[1, 1, 1]$. For each combination, a single multi-class classifier will be assigned to be trained, where in each combination a digit of 1 indicates that a label has been selected by the classifier and 0 vice-versa [11].

4.4 Classifiers

Below we will introduce 6 different classifiers we used to perform the multi-label classification task on the political election manifesto dataset. Those classifiers have been chosen because of their capabilities in performing multi-label classification according to scikit-learn's documentation [12].

4.4.1 MultinomialNB

`MultinomialNB` implements a Naive-Bayes algorithm as a classifier which is ideal for tasks with discrete features, such as text classification [13]. The term "*Multinomial*" originates from the concept of Multinomial Distribution, which is considered as a generalization of the Binomial Distribution. Since the dataset we are using for this project exhibit some discrete features, the `MultinomialNB` classifier is a suitable choice.

4.4.2 DecisionTreeClassifier

`Decision Tree` is a more commonly used method in supervised learning for both classification and regression tasks. The structure of the decision trees are typically in the form of inter-dependent conditionals that only proceeds further when the preceding conditions are met. For this project, we decided to use scikit-learn's `DecisionTreeClassifier` because it is able to handle multi-output problems with categorical data and use them to learn decision rules to make predictions [14].

4.4.3 ExtraTreesClassifier and RandomForestClassifier

The `ExtraTreesClassifier` and `RandomForestClassifier` are a developed as twins in scikit-learn Ensemble Learning library that both utilize multiple randomized decision trees to better average the prediction results yielded from individual trees [15]. Those two methods take advantage of the randomness introduced into the original decision tree model that allows them to automatically select random subsets of features from the feature list and draw the line to split one node in the tree to several children nodes with the best generated thresholds. Noticeably, these two method perform very similarly due to their common origin, but with the primary difference of the two being that the `ExtraTreesClassifier` does not replace the data after the data being sampled, which does not allow repeated results to be observed, whereas in the `RandomForestClassifier` this could be the case. Another major difference is that in `RandomForestClassifier` the decision of splitting a non-homogeneous parent node into child nodes is only made when the best split threshold is observed, but in the `ExtraTreesClassifier`, the decision of split is made in random [16].

4.4.4 KNeighborsClassifier

The `KNeighborsClassifier` is an implementation of the k-nearest-neighbor (kNN) supervised learning in scikit-learn. The main idea of using kNN in multi-label text classification is to train a model that can recognize several closely located data points and then use the majority voting mechanism to predict a new data point's class belonging. [17].

4.4.5 MLPClassifier

The `MLPClassifier` stands for Multi-layer Perceptron Classifier in scikit-learn's `neural_network` library. In supervised learning, a perceptron is an algorithm for binary classification, which assigns an input vector with one of the two candidate labels. A multi-layer perceptron generally consists of 3 or more layers of perceptrons that maps an m-dimensional input vector space to a n-dimensional output vector space ($m \geq 3, n \geq 1$) using back-propagation and gradient descent. scikit-learn's implementation of the MLPClassifier also allows for multi-label classification by passing the original output into a tailored selection function that rounds the result up at 0.5 or above and decide whether an instance belongs to a class [18].

5 Results

5.1 Metrics

To evaluate the performance of the classifiers, we will be using common Natural Language Processing metrics; precision, recall and F1. Additionally, we will be using the `hamming_loss` function from scikit-learn which can be used to compare two bit strings. It will return a fraction of labels that are incorrectly predicted. The code and results are available on the project's GitHub repository [1].

5.2 Binary relevance

The results obtained by testing the classifiers on the 1986 dataset using the Classifier Chaining method can be seen in table 2. The best-performing classifier overall is the `DecisionTreeClassifier`, although this model is outperformed by all other models on the precision score.
The `RandomForestClassifier` has the highest precision, but has a pretty low recall, F1 and a higher Hamming loss than the `DecisionTreeClassifier`. The worst-performing classifier is the `KNeighborsClassifier`, with the poorest score in recall, F1 and Hamming loss.

	Metrics			
	Precision	Recall	F1	Hamming Loss
`MultinomialNB`	0.810	0.059	0.092	0.043
`DecisionTreeClassifier`	0.593	**0.548**	**0.569**	**0.037**
`ExtraTreesClassifier`	0.855	0.207	0.333	**0.037**
`RandomForestClassifier`	**0.901**	0.167	0.282	0.038
`KNeighborsClassifier`	0.600	0.002	0.004	0.045
`MLPClassifier`	0.734	0.254	0.377	0.038

Table 2: Binary Relevance results on 1986 Manifesto

[1] https://github.com/ernestvmo/ElectionManifestoProject

5.3 Classifier Chaining

The results obtained by testing the classifiers on the 1986 dataset using the Classifier Chaining method can be seen in table 3. A similar observation to the Binary Relevance results can be analyzed from the Classified Chaining results. The best performing classifier is again the `DecisionTreeClassifier`, and the worst classifier is again the `KNeighborsClassifier`.

	Metrics			
	Precision	Recall	F1	Hamming Loss
`MultinomialNB`	0.875	0.055	0.104	0.042
`DecisionTreeClassifier`	0.577	**0.556**	**0.566**	**0.038**
`ExtraTreesClassifier`	0.861	0.161	0.271	0.039
`RandomForestClassifier`	**0.893**	0.138	0.239	0.039
`KNeighborsClassifier`	0.296	0.037	0.065	0.047
`MLPClassifier`	0.732	0.172	0.278	0.040

Table 3: Classifier Chaining results on 1986 Manifesto

5.4 Label Powerset

The results obtained by testing the classifiers on the 1986 dataset using the Label Powerset method can be seen in table 4. Contrary to the Binary Relevance or Classifier Chaining results, the Label Powerset methods performed quite poorly on the 1986 Election Manifesto. The highest-performing classifier was the `RandomForestClassifier`, with a F1 score averaged to **0.365**. Additionally, the **MLPClassifier** model did not converge, which invalidates its performance (though it is not the best-performing classifier regardless).

	Metrics			
	Precision	Recall	F1	Hamming Loss
`MultinomialNB`	0.030	0.029	0.053	0.047
`DecisionTreeClassifier`	0.248	0.185	0.212	0.062
`ExtraTreesClassifier`	0.374	0.216	0.273	0.051
`RandomForestClassifier`	**0.409**	**0.331**	**0.365**	0.051
`KNeighborsClassifier`	0.153	0.015	0.027	**0.048**
`MLPClassifier (not converged)`	0.355	0.077	0.127	**0.048**

Table 4: Label Powerset results on 1986 Manifesto

5.5 Testing the other datasets

From the results collected by testing on the 1986 manifesto, we could gather that the best performing dataset was clearly the `DecisionTreeClassifier`, since it had the highest F1 score. We decided to use this classifier to test out the performance on the other two manifestos. As the BinaryRelevance and ClassifierChaining performed about the same, we decided to test both methods as well. Those

results can be viewed in table 5. The BinaryRelevance outperforms the Classifier Chaining, but the difference is not significant.

	1994				1998			
	Prec.	Recall	F1	Hamming	Prec.	Recall	F1	Hamming
Binary Relevance	**0.594**	0.545	**0.569**	**0.023**	**0.605**	0.547	**0.575**	**0.022**
Classifier Chaining	0.565	**0.558**	0.561	0.025	0.565	**0.557**	0.561	0.023

Table 5: Results on the 1994 and 1998 Election Manifestos

6 Discussion

From our results on the 1986 Election Manifesto in tables 2, 3, 4 and 5, one can conclude that the approaches of Binary Relevance and Classifier Chaining perform approximately the same, with the Binary Relevance method having a slight advantage. Label Powerset, on the other hand, didn't deliver a satisfying performance, with the highest precision score barely exceeding 0.4. Though in general the results were not impressive, two classifiers of `RandomForestClassifier` and `DecisionTreeClassifier` still stood out with decently high metrics.

`RandomForestClassifier` performs better in the precision score, while DecisionTreeClassifier outperforms the other models for recall and F1. For the other classifiers, it has been observed that their precision scores are all looking reasonably plausible, while the other two metrics did not deliver a promising impression. One way to interpret this phenomenon could be the skewed (or biased) distribution of our data, as being mentioned in scikit-learn's documentation on Decision Trees [14], Decision Trees are prone to unbalanced dataset if there exists dominant classes.
`RandomForestClassifier`, on the other hand, is intrinsically immune to the uneven distribution of data as it was composed of multiple decision trees with their individual results being averaged, and it also excels at highly-dimensional classification tasks. That potentially could offer an explanation as to why `RandomForestClassifier` achieves the best precision score overall. However, since `RandomForestClassifier` consists of multiple individual trees, it is nearly impossible to trace the inside of its decision paths, which is the reason why this classifier was generally regarded as a black-box that the most effective way to observe differences in performance among trials is through hyperparameter tuning. [19].

7 Conclusion

From the results, we concluded that the open-source multi-label classifier `DecisionTreeClassifier` does show potential for being an easy alternative to using more complex state of the art models. The other models showed to be slightly inefficient at accurately predicting the themes of a document. It is worth noting that none of the classifiers were tuned to improve their predictive ability. It would be interesting to apply some hyperparameter tuning to the classifiers and improve their accuracy with Bayesian optimization. Additionally, this project did not research the tuning of the

preprocessing and vectorization of the data, and it would be interesting to investigate its impact along with the different multi-label solutions.

8 Group Contribution

Ernest Vanmosuinck:

- Code
- Report: Introduction, Background/Related Works, Data Description, Methods, Results, Conclusion

Pengxu Zheng:

- Report: Introduction, Background/Related Works, Methods, Results, Discussion

References

[1] S. Verberne, E. D'Hondt, A. V. D. Bosch, and M. Marx, "Automatic thematic classification of election manifestos," *Information Processing Management*, vol. 50, no. 4, p. 554–567, 2014. [Online]. Available: https://doi.org/10.1016/j.ipm.2014.02.006

[2] F. Sebastiani, "Machine learning in automated text categorization," *ACM Comput. Surv.*, vol. 34, no. 1, p. 1–47, mar 2002. [Online]. Available: https://doi.org/10.1145/505282.505283

[3] B. Pouliquen, R. Steinberger, and C. Ignat, "Automatic annotation of multilingual text collections with a conceptual thesaurus," *ArXiv*, vol. abs/cs/0609059, 2006.

[4] C. Koster, M. Seutter, and J. Beney, "Multi-classification of patent applications with winnow," vol. 2890, 07 2003, pp. 546–555.

[5] M. Honnibal, I. Montani, S. Van Landeghem, and A. Boyd, "spacy: Industrial-strength natural language processing in python," 2020.

[6] S. Bird, E. Loper, and E. Klein, *Natural Language Processing with Python*. O'Reilly Media Inc., 2009.

[7] S. Kakarla. (2019) Natural language processing: Nltk vs spacy. [Online]. Available: https://www.activestate.com/blog/natural-language-processing-nltk-vs-spacy/

[8] P. Szymański and T. Kajdanowicz, "A scikit-based Python environment for performing multi-label classification," *ArXiv e-prints*, Feb. 2017.

[9] ——. (2017) Binary relevance. [Online]. Available: http://scikit.ml/api/skmultilearn.problem_transform.br.html

[10] scikit-learn developers. (2017) Classifier chain. [Online]. Available: https://scikit-learn.org/stable/modules/generated/sklearn.multioutput.ClassifierChain.html

[11] P. Szymański and T. Kajdanowicz. (2017) Label powerset. [Online]. Available: http://scikit.ml/api/skmultilearn.problem_transform.lp.html

[12] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. (2017) Multiclass and multioutput algorithms. [Online]. Available: https://scikit-learn.org/stable/modules/multiclass.html

[13] ——. (2017) scikit-learn MultinomialNB. [Online]. Available: https://scikit-learn.org/stable/modules/generated/sklearn.naive_bayes.MultinomialNB.html?highlight=multinomialnb#sklearn-naive-bayes-multinomialnb

[14] ——. (2017) scikit-learn Decision Tree. [Online]. Available: https://scikit-learn.org/stable/modules/tree.html#tree

[15] ——. (2017) scikit-learn ExtraTreesClassifier. [Online]. Available: https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.ExtraTreesClassifier.html?highlight=extratreesclassifier#sklearn.ensemble.ExtraTreesClassifier

[16] A. Anand, "Difference between random forest and extremely randomized trees." [Online]. Available: https://stats.stackexchange.com/q/438384

[17] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. (2017) Nearest neighbors. [Online]. Available: https://https://scikit-learn.org/stable/modules/neighbors.html#nearest-neighbors-classification

[18] ——. (2017) Neural network models (supervised). [Online]. Available: https://scikit-learn.org/stable/modules/neural_networks_supervised.html

[19] A. Saabas, "Interpreting random forests." [Online]. Available: https://blog.datadive.net/interpreting-random-forests/