DATA IN MOTION

Multilabel Suggester

28.08.2023

Multilabel Suggester

Part Contents

Section Contents

Introduction

- So far we have only been able to determine whether a document was part of one category (`IDENTITY`) or not (`NO-RELATED`);
- Now we would like to be able to suggest also membership to different categories;
- In this way, we could identify documents potentially relevant for the medical domain, the strictly personal domain, etc.

Introduction

- We tried to adjust our current algorithm and see how we can make it work with more than one category;
- After investigating a bit, I think our use-case could be well covered by a multilabel classification algorithm;
- Following that tutorial, we will try to implement something for our use case.

Section Contents

The Data Set

The data set is now represented by a data frame with:

- First Col: the actual document;
- Second Col: a value to indicate whether that document belongs (1) or not (0) to the category `PERSONAL`
- Third Col: a value to indicate whether that document belongs (1) or not (0) to the category `MEDICAL`.

	text	PERSONAL	MEDICAL
0	medical record	0	1
1	name	1	1
2	first name	1	1
3	last name	1	1
4	social security number	1	1
...	...	...	...
131	identifier	1	0
132	car plate	1	1
133	driving license	1	0
134	picture	1	1
135	photo	1	1

136 rows × 3 columns

Preprocessing and Vectorization

Preprocessing and Vectorization

For this part, we keep the whole chain we developed before. So we perform some cleaning to our input documents, we tokenize them and we vectorize them according to the GloVe pretrained model with 50d vectors.

Section Contents

Building the Model

- We divide our data into train and test set (with a 4:1 proportion);
- We then start training our *Multilabel Classifier*. In `scikit-learn`, we would use the `MultiOutputClassifier` object to train the *Multilabel Classifier* model;
- The strategy behind this model is to train one classifier per label (we will use a `LogisticRegression`). Basically, each label has its own classifier;
- The `MultiOutputClassifier` would extend the output into all labels.

Making Predictions

- The `predict` function of the model would gives us just an array of 0 or 1, with one value for each category, to represent whether the test document belongs or not to that category;
- Since we want instead to be able to provide a certain level of suggestion, such as `RELEVANT`, `POTENTIALLY_RELEVANT` and `NOT_RELEVANT`, what we would really need to know are the probabilities with which the algorithm sees the document as belonging to each category;
- For doing that we would use the function `predict_proba`.

Making Predictions

The predict-proba Function

The output of this function is a bit tricky to understand. It gives you a list of arrays. The size of this list is equal to the number of categories you have. So, in our case, since we have only PERSONAL and MEDICAL, we would have a list with 2 arrays inside. Now, every one of these arrays contains a list of arrays whose size is equal to the number of documents you tested. Every element of this internal list is a 2D array itself. At position 0 it has the probability for the document of NOT belonging to the category, while at position 1 it has the probability for the document of belonging to the category.

Computing Model Accuracy

- Usually one can study the accuracy of his model with the `accuracy_score` funtion;
- However, in case of multilabel classification this measure might gives you unrealisticly bad results;
- That is why, we can use the Hamming Loss evaluation metric.;
- Hamming Loss is calculated by taking a fraction of the wrong prediction with the total number of labels. Because Hamming Loss is a loss function, the lower the score is, the better (0 indicates no wrong prediction and 1 indicates all the prediction is wrong).

Computing Model Accuracy

Model Performances

With our setup, we got an accuracy score of 0.68 and a Hamming loss of 0.2.

Saving the Model

Here is an interesting article on two methods to save and load a python model. The author illustrates two possible ways one can achieve this:

- `pickle` (comes automatically with python installation)
- `joblib` (has to be install via `pip install joblib`)

It compares file size of the saved models, time to save and load them and also makes some considerations about security. The overall conclusions is that `pickle` seems to be much faster and performant, but it might be unsecure when your ML models are used online. In the following, for now, we are going to use `pickle` but we should keep the security warning in mind, once we are in production and the user can directly build his own models.

Automation and Retraining

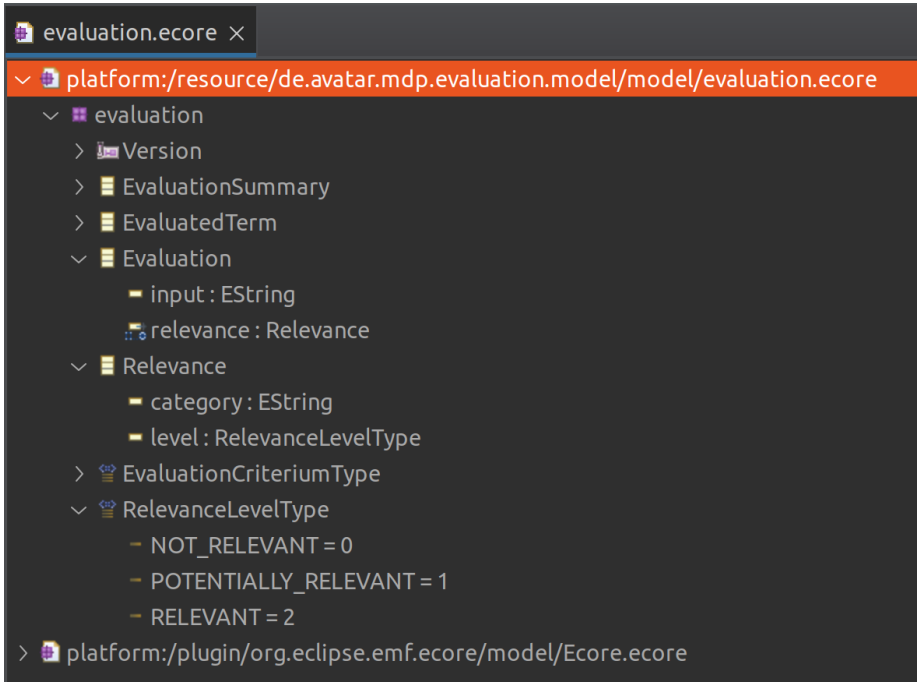
- We built two separate scripts, one for making predictions and one for retraining the model;
- We would also store the model performance every time a new model is built, in such a way to be able to study the development after some retraining (this is useful to check the model);
- The predictions are provided in the form:

```
"doc1": {
        "PERSONAL": "POTENTIALLY_RELEVANT",
        "MEDICAL": "NOT_RELEVANT"
},
"doc2": {
        "PERSONAL": "RELEVANT",
        "MEDICAL": "POTENTIALLY_RELEVANT"
}
```

Section Contents

The Evaluation EMF Model

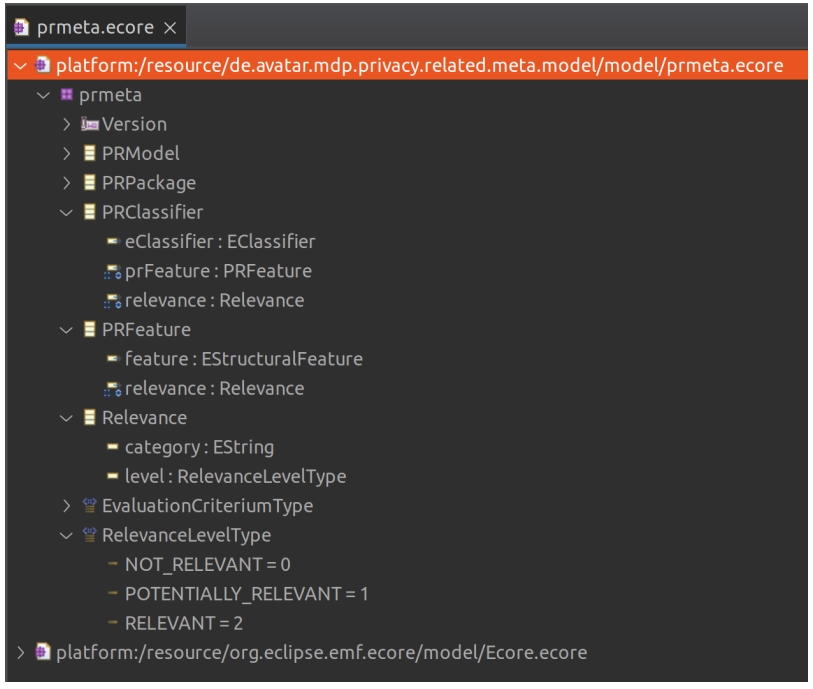
- To account for the introduction of multiple categories, the `Evaluation` EMF model had to be refactored;
- In particular, now we have a `Relevance` object with a `category` and a `level` attribute.



Section Contents

The Meta Model

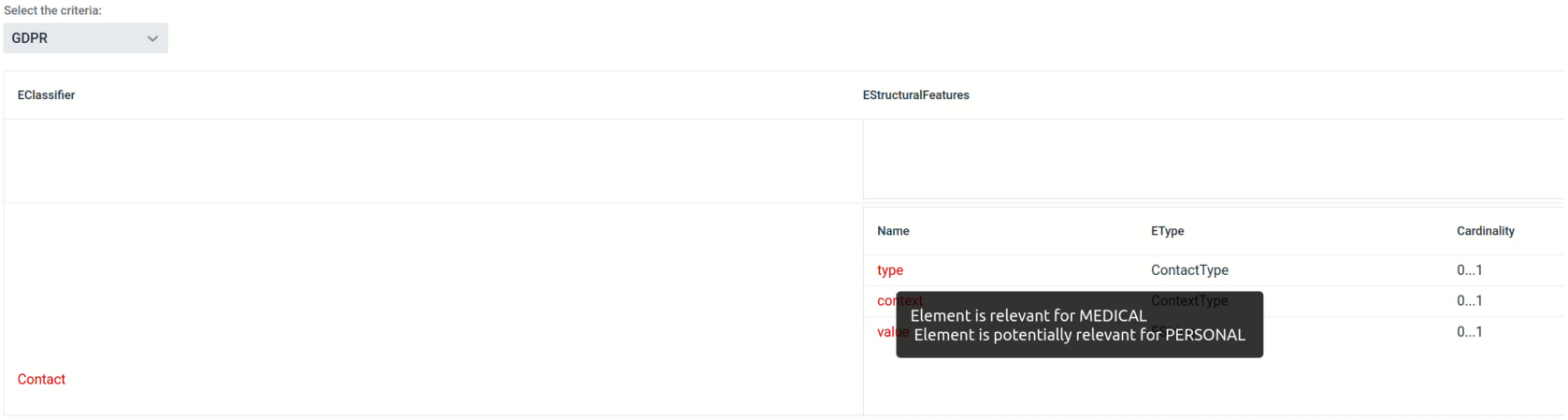
- To account for the introduction of multiple categories, also the `MetaModel` EMF model had to be refactored;
- In particular, now we have a `Relevance` object with a `category` and a `level` attribute.



Section Contents

The UI

- The UI has also been adjusted accordingly;
- Now a tooltip is displayed with the list of categories for which a term is considered relevant and the list of categories for which is considered potentially relevant, if any.



Section Contents

GitHub

To keep track of both this new suggester implementation without removing the old one, we created two branches, both for the `avatar-modelrepo` and the `avatar-mdp` projects:

- `mdp-model-evaluation-cos-distance` and `mdp-model-evaluation-multilabel` for the `avatar-modelrepo`;
- `cosine-dist-suggester` and `multilabel-suggester` for the `avatar-mdp`.

Conclusion

DATA IN MOTION

Useful Links

OSGi Working Group

Working Group: www.osgi.org
WG Blog: www.osgi.org/blog
Twitter: @osgiwg
Bndtools: bndtools.org

Data In Motion

Web: www.datainmotion.com
Blog: datainmotion.com/blog
Twitter: @motion_data

Jürgen Albert

Email: j.albert@data-in-motion.biz

Mark Hoffmann

Email:
m.hoffmann@data-in-motion.biz