



Model Suggester First Implementation

04.08.2023



Model Suggester First Implementation



Part Contents

1 Model Suggester First Implementation



Section Contents

1 Model Suggester First Implementation

Introduction

The Model Evaluator Service

The Evaluation Model

The UI

The Suggester Retrainer Service

Next Steps



Where we stand

- ▶ From our previous studies about text classification, we decided to go with a model based on the cosine distance and the gloVe vectorization;
- ▶ Input text is vectorized according to gloVe 50D pre-trained model;
- ▶ The text category is determined by computing the minimal cosine distance from a set of trained words and determining whether it is below a certain threshold or not.



What we need

- ▶ A service which takes as input as EPackage and evaluates all the features' names and possible descriptions according to our model;
- ▶ A model to display the result of the analysis;
- ▶ A simple UI to display the results and trigger the whole process;
- ▶ A second service that is responsible for retraining our suggester when the user has made changes on the suggested results.



Section Contents

1 Model Suggester First Implementation

Introduction

The Model Evaluator Service

The Evaluation Model

The UI

The Suggester Retrainer Service

Next Steps



The Model Evaluator Service

- ▶ I built an API `ModelEvaluatorService` which provides just one method:

```
EvaluationSummary evaluateModel(EPackage ePackage);
```

- ▶ The `EvaluationSummary` is an `EObject` of a new model I created;
- ▶ The first implementation of the `ModelEvaluatorService` evaluates the model based on GDPR privacy related terms;
- ▶ Additional implementations can then be added for other sets of regulations (OpenData, etc.).



Section Contents

1 Model Suggester First Implementation

Introduction

The Model Evaluator Service

The Evaluation Model

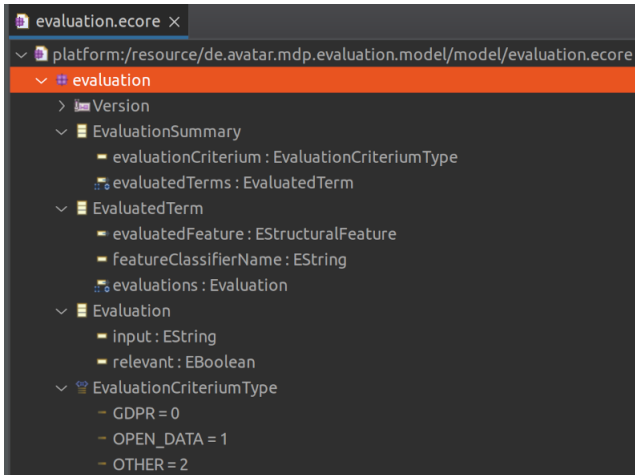
The UI

The Suggester Retrainer Service

Next Steps



The Evaluation Model





Section Contents

1 Model Suggester First Implementation

Introduction

The Model Evaluator Service

The Evaluation Model

The UI

The Suggester Retrainer Service

Next Steps



The UI

- ▶ I built a simple Vaadin view, in which you can search for a model in the system (which are registered through the `DynamicPackageLoader`) and trigger an evaluation;
- ▶ The `EvaluationSummary` is then displayed in a table, in which terms that are found relevant are marked in green, while terms that are found non relevant are shown in red;
- ▶ The user has then the possibility to change the result for each term.



The UI

DATA IN MOTION

Avatar Model Repo

Search

Upload

Evaluate

Evaluate

Enter search

Name	URI	Evaluate
evaluation	http://avatar.de/mdp/evaluation/1.0.0	<div></div> Evaluate
person	http://avatar.de/mr/person/1.0	GDPR <div></div> Evaluate
test	http://avatar.de/mr/test/1.0	<div></div> Evaluate

Feature Name	Evaluated Docs
Address:city	<div>Document</div> Is Relevant?
	city <div></div>
Address:context	<div>Document</div> Is Relevant?
	context <div></div>

Save Model



The UI

- ▶ When the user saves the changes, the creation of the coupled model with the information on the evaluation should be triggered;
- ▶ This part **has not been implemented yet**;
- ▶ When the user saves the changes, he/she can also trigger a retraining of the suggestion model, based on such changes;
- ▶ This will pass the list of evaluated documents to a dedicated service which will then add them to the suggestion model and recomputes the threshold for the determination of a document category.



The UI

The screenshot displays the 'Evaluate' section of the 'DATA IN MOTION' interface. On the left, a sidebar contains links for 'Avatar Model Repo', 'Search', 'Upload', and 'Evaluate' (which is currently selected). The main area features a search bar and a table with the following data:

Name	No URI	Evaluate
evaluation	http://avatar.de/mdp/evaluation/1.0.0	<input type="button" value="Evaluate"/>
person	http://avatar.de/mr/person/1.0	GDPR <input type="button" value="Evaluate"/>
test	http://avatar.de/mr/test/1.0	<input type="button" value="Evaluate"/>

A modal dialog box titled 'Retrain Suggestion Model' is centered on the screen, asking: 'Do you want to retrain the suggestion model with the annotated features?'. It has 'No' and 'Yes' buttons. Below the dialog, a table shows feature annotations:

Feature Name	Is Relevant?
Address: city	<input checked="" type="checkbox"/>
Address: context	<input type="checkbox"/>

A 'Save Model' button is located at the bottom left of the main content area.



Section Contents

1 Model Suggester First Implementation

Introduction

The Model Evaluator Service

The Evaluation Model

The UI

The Suggester Retrainer Service

Next Steps



The Suggester Retrainer Service

- ▶ A second API, `ModelSuggesterRetrainerService` has been created;
- ▶ It contains a method to trigger the retraining of the suggester:

```
void retrainModelSuggester(List<String> pertinentDocs, List<String>  
    unrelevantDocs);
```

- ▶ I have implemented such API for the GDPR standards, triggering the retraining of the model;
- ▶ Additional implementations can then be added for other sets of regulations (OpenData, etc.).



The Retraining Mechanism

When the suggester model is retrained:

- ▶ The relevant documents are added to both the test and train set (if duplicated docs are present, the old ones are removed);
- ▶ The non-relevant documents are added to the test set;
- ▶ The documents of both train and sets are then cleaned, tokenized and vectorized again;
- ▶ The minimal cosine distance between each test document and the train set is computed;
- ▶ A new optimal threshold is determined based on the accuracy of the selection given a certain value of threshold;
- ▶ The new parameters and the new sets are saved in a file for further use.



Section Contents

1 Model Suggester First Implementation

Introduction

The Model Evaluator Service

The Evaluation Model

The UI

The Suggester Retrainer Service

Next Steps



Next Steps

- ▶ We could in principle also add the One-Class SVM model as possibility for the suggester, and see over time which one seems to perform better;
- ▶ This is something we have to monitor when someone actually starts using it;
- ▶ We have to think how we want to build the coupled model when a model has been evaluated.



Conclusion



Useful Links

OSGi Working Group

Working Group: www.osgi.org

WG Blog: www.osgi.org/blog

Twitter: [@osgiwg](https://twitter.com/osgiwg)

Bndtools: bndtools.org

Data In Motion

Web: www.datainmotion.com

Blog: datainmotion.com/blog

Twitter: [@motion_data](https://twitter.com/motion_data)

Jürgen Albert

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

Mark Hoffmann

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