UNIVERSIDAD DE BURGOS
ESCUELA POLITÉCNICA SUPERIOR
Grado en Ingeniería Informática

TFG del Grado en Ingeniería Informática

NoisyNER - Named entity recognition in social media Documentación Técnica

Presentado por Malte Janssen
en Universidad de Burgos — July 1, 2019
Tutor: Bruno Baruque Zanón

Contents

List of Figures

List of Tables

Apéndice A

Software plan

A.1 Introduction

In the following annex, the organisational aspects of the study of different NER classifiers and the development of the software are documented. More precisely, the software development process and tools that were used to manage the process are described, followed by a examination of the course of the project. The second part of the annex examines the project's viability, including the calculation of involved costs and profit possibilities.

A.2 Project Management

Scrum The project's management is inspired by the Scrum model used in agile software development. The model is based on the assumption that projects are too big to be planned in it's entirety at the start. Therefore only a rough outline is made at the start. The project is divided into several milestones that provide an agile approach.

Scrum is a team based approach to project management. Due to the fact that this bachelor thesis is only written by one person the majority of the concepts can't be applied exactly as intended by Scrum. Consequently the project management approach is only loosely based on Scrum.

One concept that is applied are Sprints. In this case most sprints cycles had a duration of approximately a month. Some are bigger and some are smaller due to the complexity of tasks at hand and the time available. Sprint meetings between the author and the project's coordinator were held every two weeks, usually around the middle and end of each sprint. In the meeting in the middle the tasks progress was discussed, while in the meeting at the end the results of the sprint was discussed and the next sprint was vaguely planned. The second meeting can therefore be seen as the Sprint Planning and Sprint Review. The project's coordinator can be seen as the Project Owner of the Scrum model, prioritising tasks and guiding the project's direction.

Github and ZenHub The project is hosted on github. At the beginning to organise tasks waffle was used. After waffle shut down project organisation was done with the help of ZenHub. Zenhub provides a board to visualise tasks, as well as it provides an overview of the remaining workload. It also offers several tools to inspect work velocity. Each Github issue is assigned a amount of story points, estimating the tasks size. In this case each point is the equivalent of the workload of <points hours.

A.3 Time plan

The Kick-Off Meeting took place in the second week of December 2018. The elemental ideas of the project were discussed. Due to exams and other private responsibilities the project wasn't directly started after the meeting. Instead the 9. of January marked the beginning of the project.

Some milestones were smaller than others in a similar time frame. This is due to responsibilities of other classes and exam periods which reduced time availability during the semester.
The next paragraphs give an overview over the the phases of development.

Milestone 1 (9.01-14.01.2019): Initial project setup/ Research regarding NLP

In the first relatively small Milestone the projects infrastructure was set up. Also some research regarding the theoretical concepts of NLP and NER were done and documented.

Milestone 2 (8.02-31.02.2019): NLTK-Experiments/ Preprosessing Data

The second Milestone consisted of researching and experimenting with one of the libraries (NLTK) used later on in the project. An initial NER chunker was introduced and functions prepossessing the Data were written. Ideally this milestone would have been bigger and the project would have advanced much more, but my laptop broke and had to be send in. Due too the bad infrastructure of the university not much could have been worked on.

Milestone 3 (01.03-08.04.2019): NLTK

Milestone 3 was the biggest Milstone until that point. Things got serious as different NLTK chunkers got introduced and a training script was written. Initially the due date was the end of march. But some complications in having to figure out how to train NER chunkers without any use cases in the internet combined by the midterm exam period it had to be postponed by a week.

Milestone 4 (09.04-13.05.2019): ~~Other Classifiers~~ Deep Learning Classifier

In the initial plan it was planned to create a few other classifiers with other libraries such as Stanford Core, Sklearn and OpenNLP. But that plan changed a bit. Together with the projects coordinator the decision to develop a different classifier using more advanced machine learning techniques was made. Do to the postponement of the last milestone, exams and working on a bigger project in another class most of April couldn't be used for this milestone as the commit history suggests. The project of the other class was related to this theme as it covered sentiment analysis with the sckikit-learn library. The concepts learned and applied there were adapted towards named entity recognition and incorporated into this project.

Milestone 4 (01.05-16.06.2019): ~~Evaluation~~ Evaluation/ Improve Deep Learning Classifier

At the start of the project this milestone was called Evaluation in which all prior classifiers were supposed to be evaluated and documented. But due to the complexity of the task at hand the classifiers didn't perform at their best. Therefore this phase was only partially used to do experiments and the bigger focus was improving the performance of the deep learning classifier. The validation performance was increased from a validation F1 score of 7 to a F1 score of around 35. All work was done by the 16.06 but the commit didn't go through and probably shown error message was overseen. Therefore it appears like the milestone dragged on for longer than it was worked on.

Milestone 5 (17.06-23.06.2019): Web App

The fifth, short milestone was used to create a web app with the help of flask and html.

Milestone 6 (14.06-02.07.2019)

The last remaining time was mainly used to document the steps that were not documented in the prior milestones yet. Also a comparison of different deep learning architectures was carried out. Furthermore the code was organised differently in some cases.

A.4 Feasibility study

Economic viability

Legal Feasibility

Apéndice B

Requirements Specification

B.1 Introduction

This chapter contains requirement specifications for the developed software. It provides an overview over the software's required functionality. Since the main goal was not to develop an application the Requirements will be split up into two parts. The first one defining the requirements of the developed software up to the application and the second part will just define the requirements specific to the application.

B.2 General Objectives

The general objective of the software is to find named entities in noisy texts. To do this different libraries were used to develop and train classifiers in that context. The following steps and objectives are related to the software's requirements:

- Utilise the scikit-learn classifiers and the NLTK library: The read Data is converted into feature-sets and then fed into the classifiers which are then trained. At prediction/evaluation time classifiers have to produce output.
- Development of a deep Learning Classifier: Another approach to do named entity recognition was to develop a deep learning classifier.
- Develop a simple web app: A web app running the deep learning classifier is developed as a demo.

B.3 Requirements Catalogue

This section lists the software's functional requirements.

Actors The only actor of the software is it's user. There are no different roles an actor can have, as the only actor he has the whole feature-set of the program available. His goal is to get as accurate named entity predictions as possible.

B.4 Requirements specification

Table B.1: Requirement «Receive Input»

ID:	**Name: Receive Input**
Description	The user can provide input text
Pocess Description	The user has the ability to introduce text into the applications text box. The text is then passed into the application by pressing the "classify" button.

B.5 Use Cases

The in figureB.1 shown use case diagram contains the two actions a user can make. The use case input text enables the user to input text that he wished to be classified. The second use case, get Classification, returns the classifications to the user.

Table B.2: Requirement «Process input»

ID:	**Name: Process input**
Description	After text is received it is processed it into the data formats processesable by the deep learning classifier
Process description	Whenever new Text is entered into the application it is read and converted into a format readable by the system in order to work with that data.
Data conversion	1. 1. The input string data is word tokenised 2. 2. Each word that was returned by the tokenisation is replaced by it's index in the vocabulary if possible. If not the index of the unknown word is used. 3. 3. For each word all characters are replaced by it's index in the character vocabulary if possible. If not the index of the unknown word is used. This results in a list of list filled by a representation of indices for each word. 4. 4. Each element of the in the previous step created list is padded with zeros until each word list has the same length as the longest word. This is done because the neural network expects all inputs to have the same lengths. 5. 5. The in step 2 and 4 created lists are converted into Tensors.

Table B.3: Requirement «Classification»

ID:	**Name: Classification**
Description	The application feeds the in B.2 created tensors into the model and gets the predictions returned.

Table B.4: Requirement «Output results»

ID:	**Name: Output results**
Description	The results received from the model (B.3 the system outputs the results.
Process description	For better visualisation the results are outputted as an image. The image contains the original text with found entities being highlighted in specific colours
Colours	1. Person: green (/1f5a07) 2. Corporation: red (/9f0120) 3. Creative-work: pink (/ff2770) 4. Group: blue (/4e4e94) 5. Location: yellow (/eae11a) 6. Product: purple (/941aea)

Table B.5: Requirement «Selection of Classifier»

ID:	**Name: Selection of Classifier**
Description	A Classifier can be selected out of a predefined selection of different classifiers.
Options	1. option1 2. option2 3. optionN

Table B.6: Requirement «Fetch random tweet»

ID:	**Name: Fetch random tweet**
Description	A random tweet is fetched to highlight classification
Process Description	x

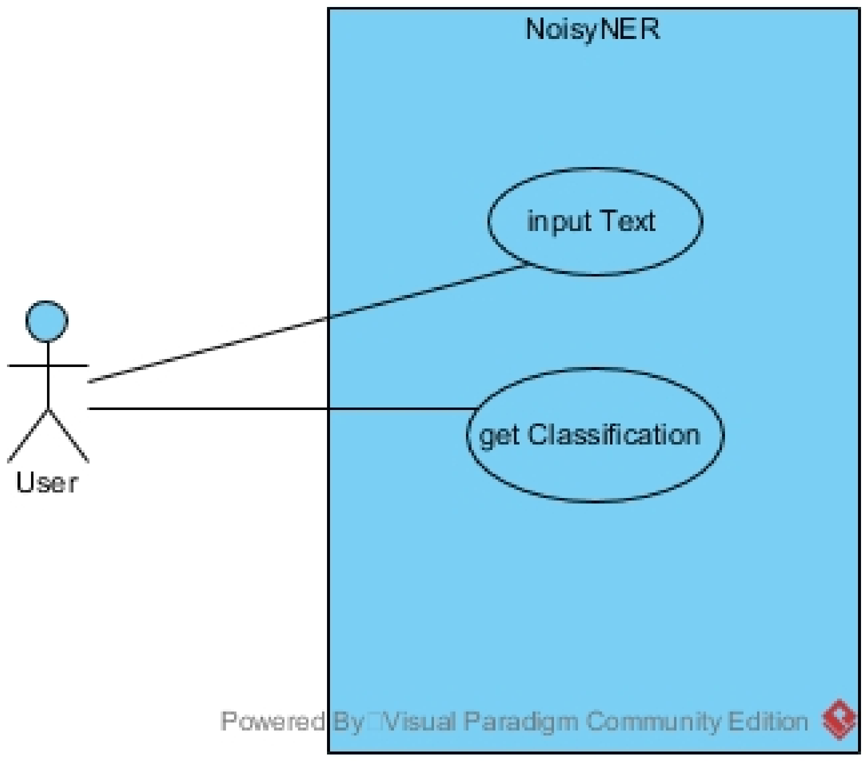

Figure B.1: System's use case diagram

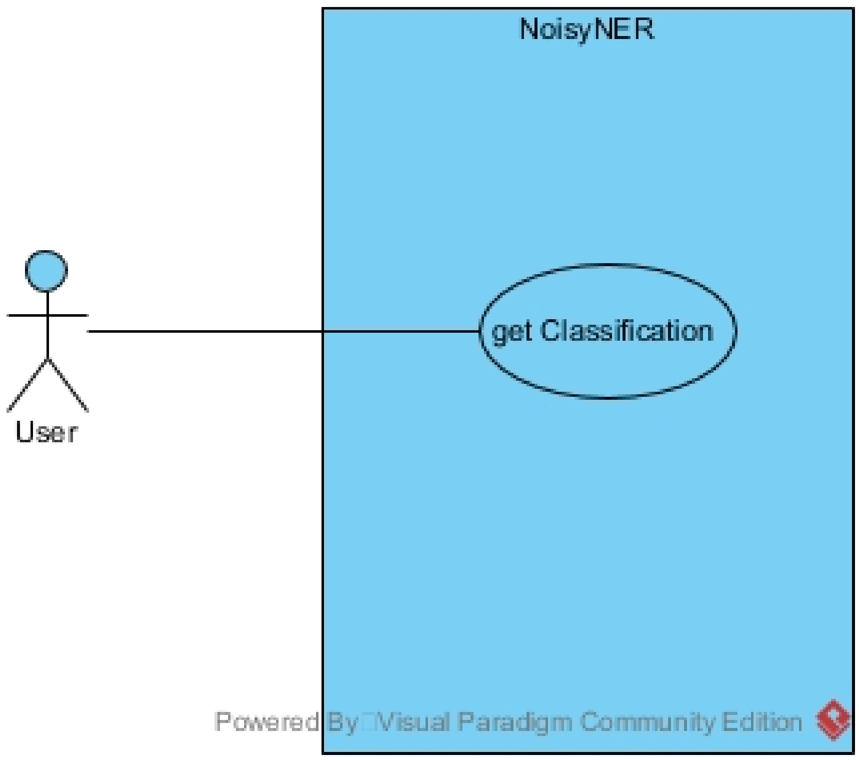

Figure B.2: Caption

Use Case	get Classification		
Description	The user enter requests the classification of named entities within a text.		
Preconditions	User has written text into the text field		
Trigger	User presses the "classify" Button		
Flow of events	Step	Action	
	1.	Data gets pre-processed	
	2.	pre-processed data is handed to the model	
	3.	classification by the model	
	3.	model returns classification	
	3.	output classification	

Apéndice C

Design Specification

C.1 Introduction

This annex serves as a detailed description of the software's implementation, structure and way of functioning

C.2 Data Model

Package Structure

The package structure is inspired by the PyTorch Template Project[1] which attempts to provide a clear folder structure which is suitable for many deep learning projects.

Note: The structuring of the pytorch package is not ideal. Basically there is the main pytorch package, which includes a heroku-app package which contains a lot of duplicate classes which are already contained within the pytorch package. Ideally the flask app would have been inside the main pytorch package, but due to the limitations of size on heroku this was not possible. Therefore the for the demo app necessary files have simply been copied into the heroku-app package to be hosted on heroku. This part of the structure will not be further examined as it is also contained within the pytorch package.

The general code structure and it's dependencies are visualised in diagram C.1. The repositories code is divided into two main packages. The

[1] **packagestruc**.

NLTK_SKLEARN package and the pytorch package. Both packages have a very similar structure. The only big difference is that the pytorch package also contains a logger package to log the training process. The next design sections will concentrate on the bigger and more interesting part of the project, the pytorch part.
Most of the portrayed classes are in reality not classes but python modules and are included to get a better overview about the project. They are marked as abstract classes for differentiation.
A description of the packages containing the application's classes is provided in the following sections. For reasons of clarity and comprehensibility, the classes are not shown in a single diagram but are split into several class diagrams based on their respective package.

Main package

In the main package several python scripts can be found. It includes the scripts to load the data (buildVocab and buildData), as well as the training, evaluation and prediction scripts. These are the executable parts of the program and will be explained in further detail in the manual.

dataLoader

The dataLoader package shown in C.2 contains the DataLoader class. It is responsible for creating the word, character and tag mappings. It also includes functionality to generate batches of data and translates mapped data into Data processable by the neural network.

logger

The logger is utilised by the training and evaluation script to document the training's process and results.

utils

The utils package contains various utility function used by nearly all other scripts. For instance there are functions to load and save the current state of the model and to translate the mappings of labels form their indice representation back into the natural one. It also contains two utility classes. The first one is the RunningAverage class which computes a average of the loss functions and is used in the training and evaluation script. The second

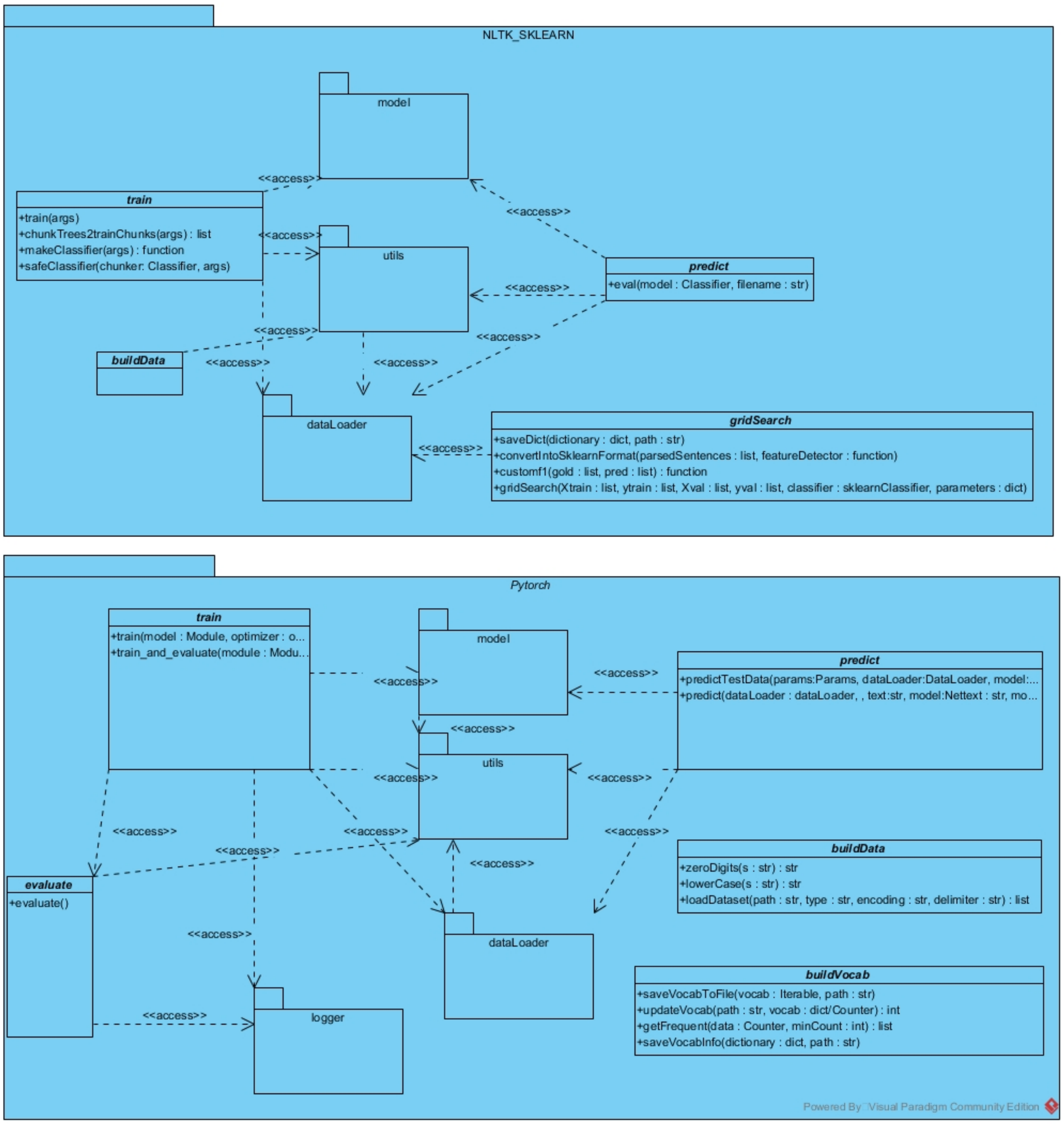
NLTK_SKLEARN
model
train
+train(args)
+chunkTrees2trainChunks(args) : list
+makeClassifier(args) : function
+safeClassifier(chunker: Classifier, args)
utils
predict
+eval(model : Classifier, filename : str)
buildData
dataLoader
gridSearch
+saveDict(dictionary : dict, path : str)
+convertIntoSklearnFormat(parsedSentences : list, featureDetector : function)
+customf1(gold : list, pred : list) : function
+gridSearch(Xtrain : list, ytrain : list, Xval : list, yval : list, classifier : sklearnClassifier, parameters : dict)
<<access>>
Pytorch
train
+train(model : Module, optimizer : o...
+train_and_evaluate(module : Modu...
model
predict
+predictTestData(params:Params, dataLoader:DataLoader, model:...
+predict(dataLoader : dataLoader, , text:str, model:Nettext : str, mo...
utils
buildData
+zeroDigits(s : str) : str
+lowerCase(s : str) : str
+loadDataset(path : str, type : str, encoding : str, delimiter : str) : list
evaluate
+evaluate()
dataLoader
buildVocab
+saveVocabToFile(vocab : Iterable, path : str)
+updateVocab(path : str, vocab : dict/Counter) : int
+getFrequent(data : Counter, minCount : int) : list
+saveVocabInfo(dictionary : dict, path : str)
logger
<<access>>
Powered By Visual Paradigm Community Edition

Figure C.1: Package Structure

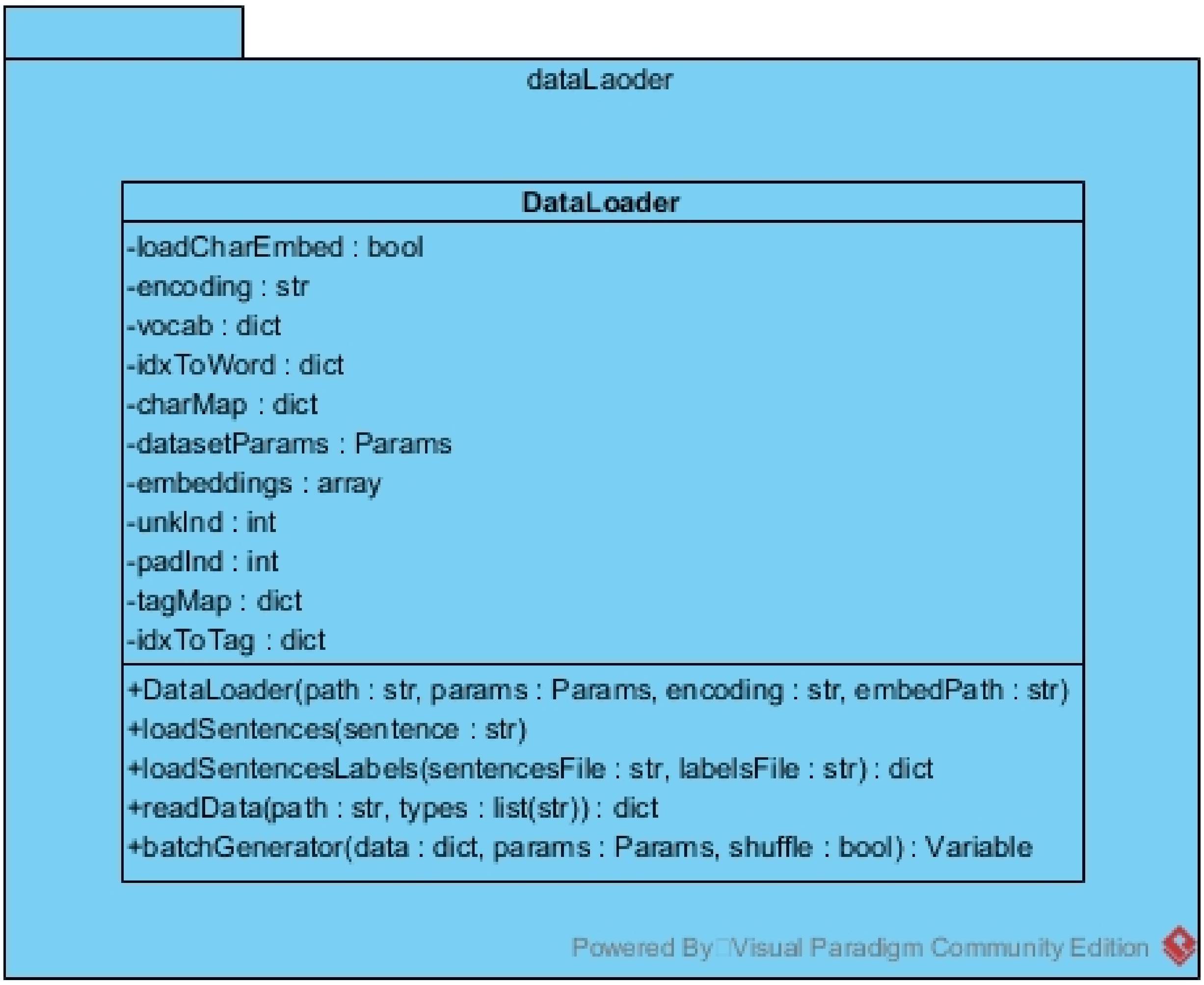
dataLaoder
DataLoader
-loadCharEmbed : bool
-encoding : str
-vocab : dict
-idxToWord : dict
-charMap : dict
-datasetParams : Params
-embeddings : array
-unkInd : int
-padInd : int
-tagMap : dict
-idxToTag : dict
+DataLoader(path : str, params : Params, encoding : str, embedPath : str)
+loadSentences(sentence : str)
+loadSentencesLabels(sentencesFile : str, labelsFile : str) : dict
+readData(path : str, types : list(str)) : dict
+batchGenerator(data : dict, params : Params, shuffle : bool) : Variable
Powered By Visual Paradigm Community Edition

Figure C.2: dataLaoder Package

class Params is used to store the models parameters. This includes dataset parameters and also model architecture parameters.

model

This package contains classes related to the inner workings of the deep learning model. The class Net represents the model and contains, depending on the models architecture, the layers inside the layers package. It also includes the modules eval and loss defining the models loss and evaluation functions.

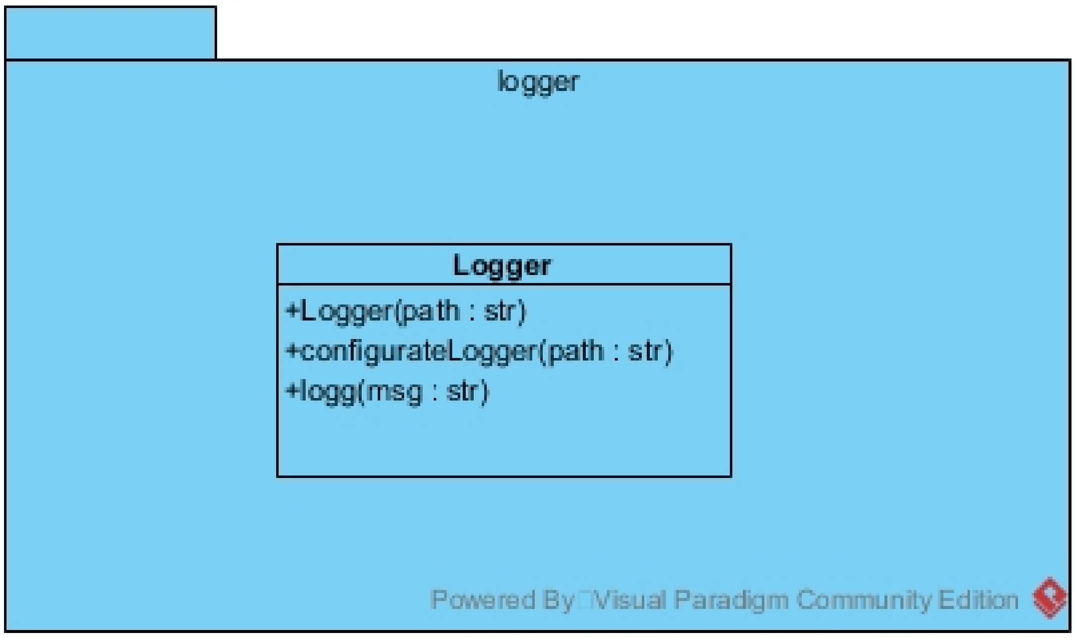
logger
Logger
+Logger(path : str)
+configurateLogger(path : str)
+logg(msg : str)
Powered By Visual Paradigm Community Edition

Figure C.3: logger Package

C.3 Architecture of the web app

Apart from the deep learning aspects of the code the project contains a demo web application. This section provides an overview of the design of said application. As figure C.6 indicates the application itself uses the model-view-controller design pattern. Once a user inputs text to the view it is passed to the controller which contains the app module. This module is used for routing. It updates the model with the text information. The model contains the logic of the application. Here the serve module receives the data and initialises the deep learning model and reloads it's weights with the help of the DataLoader. The deep learning model is accessed through the serve module which's task it is to convert the data into data processable by the model and also acts as a wrapper to the deep learning model. To wrap the model it contains a getModelApi function that returns a lambda function able to process the request. Once the request is processed the controller is notified with the classifications and updates the HTML view.

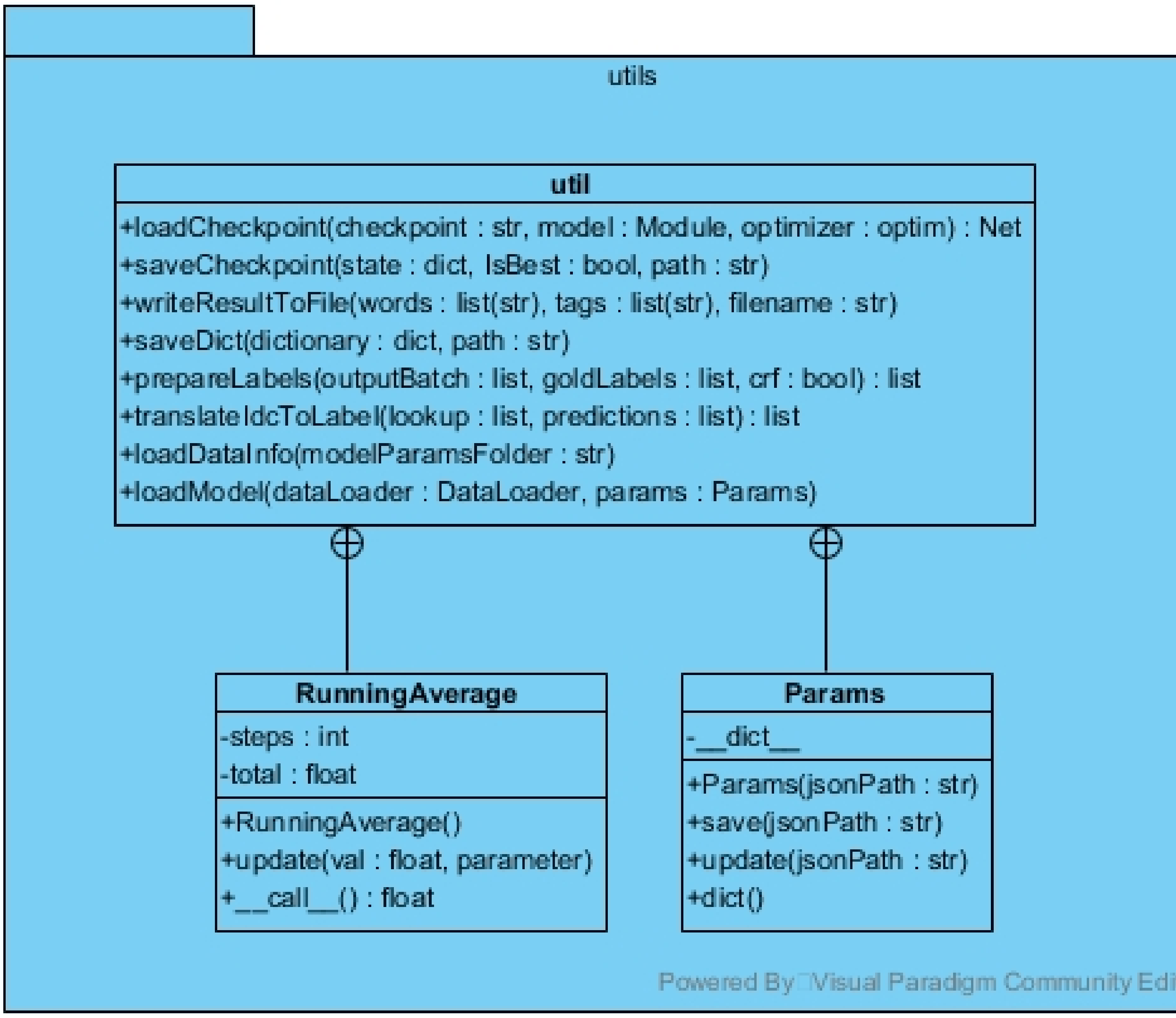
utils
util
+loadCheckpoint(checkpoint : str, model : Module, optimizer : optim) : Net
+saveCheckpoint(state : dict, IsBest : bool, path : str)
+writeResultToFile(words : list(str), tags : list(str), filename : str)
+saveDict(dictionary : dict, path : str)
+prepareLabels(outputBatch : list, goldLabels : list, crf : bool) : list
+translateIdcToLabel(lookup : list, predictions : list) : list
+loadDataInfo(modelParamsFolder : str)
+loadModel(dataLoader : DataLoader, params : Params)
RunningAverage
-steps : int
-total : float
+RunningAverage()
+update(val : float, parameter)
+__call__() : float
Params
-__dict__
+Params(jsonPath : str)
+save(jsonPath : str)
+update(jsonPath : str)
+dict()
Powered By Visual Paradigm Community Edi

Figure C.4: utils Package

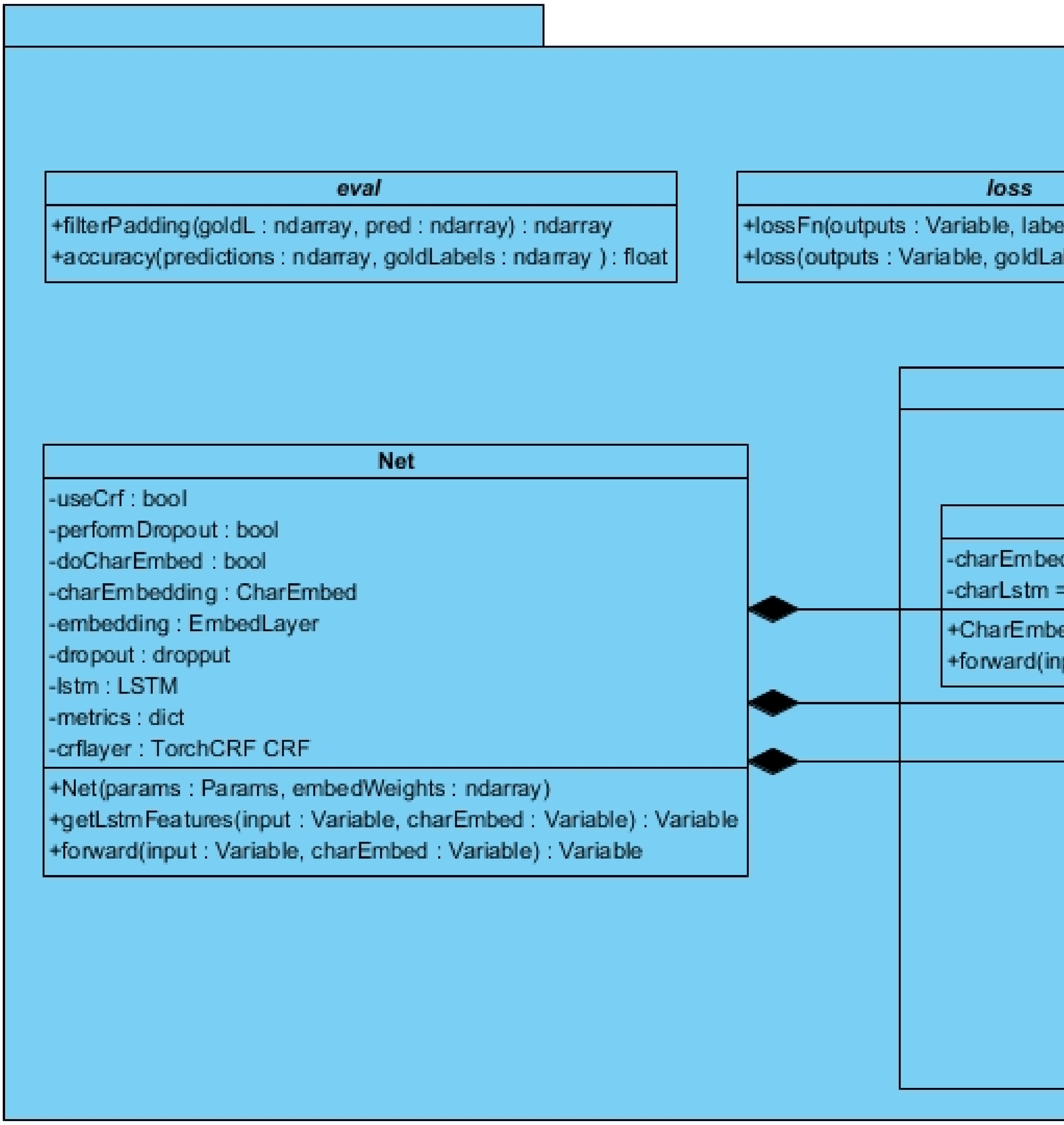
eval
+filterPadding(goldL : ndarray, pred : ndarray) : ndarray
+accuracy(predictions : ndarray, goldLabels : ndarray) : float
loss
+lossFn(outputs : Variable, labe
+loss(outputs : Variable, goldLa
Net
-useCrf : bool
-performDropout : bool
-doCharEmbed : bool
-charEmbedding : CharEmbed
-embedding : EmbedLayer
-dropout : dropput
-lstm : LSTM
-metrics : dict
-crflayer : TorchCRF CRF
+Net(params : Params, embedWeights : ndarray)
+getLstmFeatures(input : Variable, charEmbed : Variable) : Variable
+forward(input : Variable, charEmbed : Variable) : Variable
-charEmbe
-charLstm =
+CharEmbe
+forward(in

Figure C.5: model Package

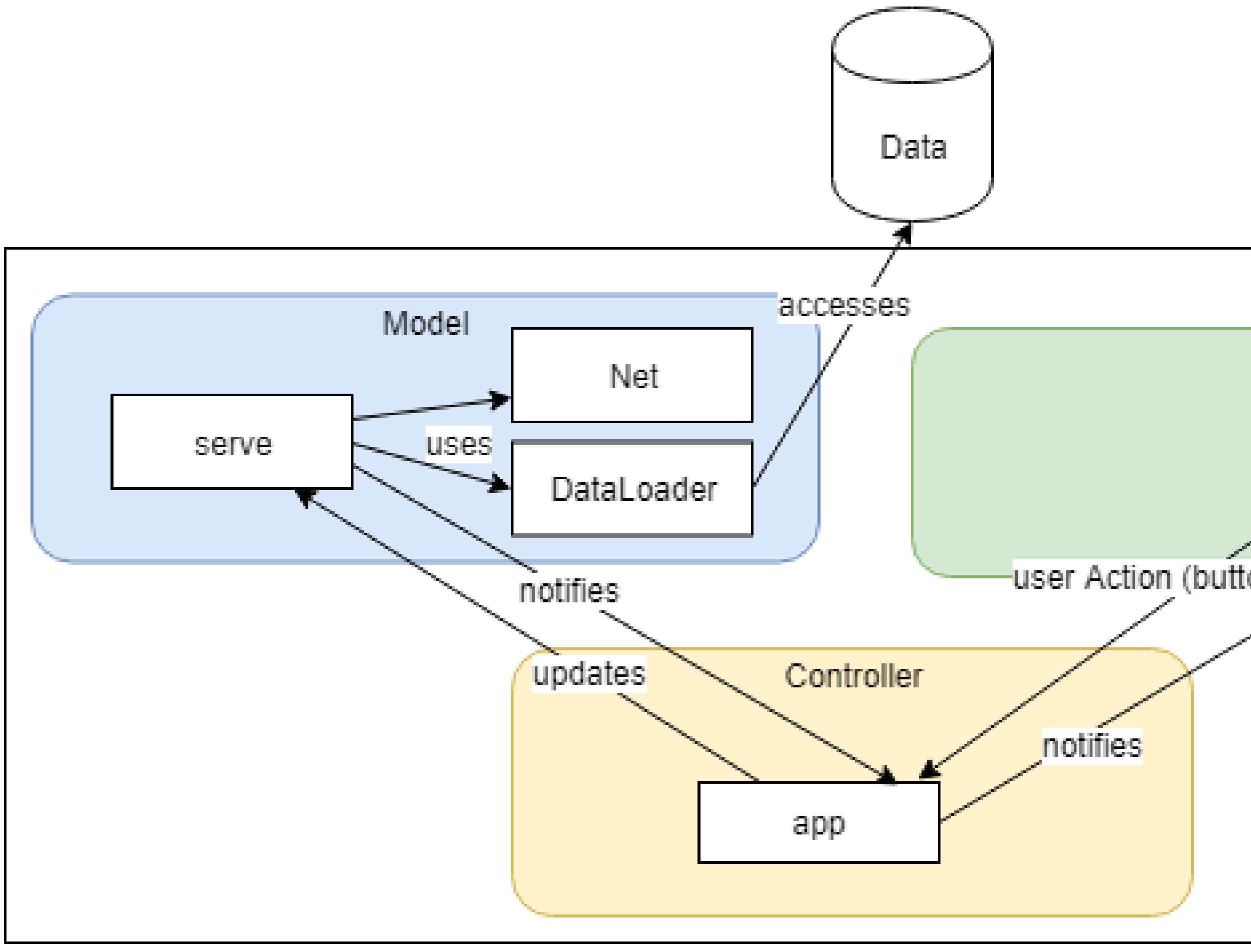

Figure C.6: Caption

C.4 Procedural Design

The sequence diagram shown in figure C.7 shows the sequence of actions happening once a user inputs data and request a classification.

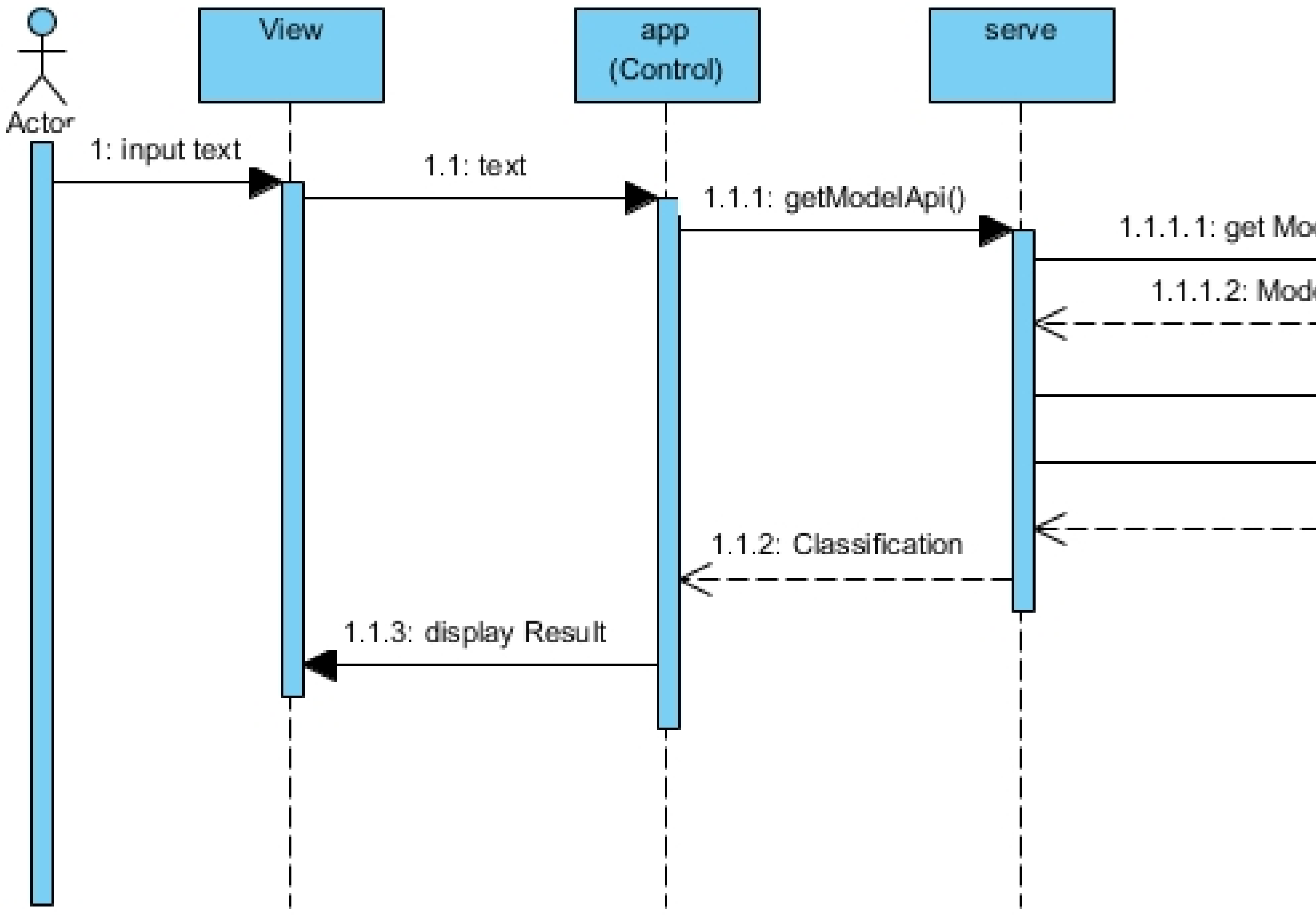

Figure C.7: Caption

Apéndice D

Technical Programming Documentation

D.1 Introduction

This section contains the technical programming documentation, describing the directory structure of the presented DVD and an installation guide.

D.2 Directory structure

D.3 Programmer's Manual

The presented DVD contains the following directories (the directory structure is split up into it's main directories since showing them all in one tree would get messy because of new pages):

```
│ Documentation
│ ├─ annex.pdf
│ └─ memoria.pdf
└─ requirements.txt
```

- `NLTK_SKLEARN`
  - `Classifiers` .... contains the trained classifiers
  - `Data` .... contains the Data used for training
    - `wnut` .... contains the wnut challenge data
      - `emerging.dev.conll` .... validation data
      - `emerging.test.annotated` .... test data
      - `wnut17train.conll` .... train data
  - `dataLoader` .... contains data loading modules
    - `reader` .. very slightly adapted ConllCorpusReader from NLTK (original does not support minus in entity name)
      - `api.py`
      - `reader.py`
    - `dataLoader` ..... uses adapted ConllCorpusReader to read data
  - `model` .... classifier related modules
    - `classifier.py` . implements ClassifierChunker used to build all classifiers
    - `features.py` .... contains feature extraction functions
  - `results` .... contains gridtest/prediction results
    - `gridtest` .... contains gridsearch results
    - `prediction` . contains test data predictions of trained classifiers
  - `utils` .... contains util fuctions
    - `readWrite.py` implements functions regarding reading/writing of data
    - `util.py`implements util functions most of them to transform data
  - `buildData.py` .. script that reads, transforms and writes data into correct format
  - `gridSearch.py`... script used for gridsearching with validation data
  - `predict.py` .... script to predict input or test data
  - `train.py` .... scipt to train classifier

```
pytorch
├── Data ........................ contains the Data used for training
│   ├── embed .................. contains pre-trained embedding data
│   │   └── glove.twitter.27B .. pre-trained twitter gloVe embeddings
│   │       └── glove.twitter.27B.200d.txt
│   ├── test ...................... contains test sentences and labels
│   │   ├── labels.txt
│   │   └── sentences.txt
│   ├── train .................... contains train sentences and labels
│   │   ├── labels.txt
│   │   └── sentences.txt
│   ├── validation .......... contains validation sentences and labels
│   │   ├── labels.txt
│   │   └── sentences.txt
│   └── wnut ...................... contains the wnut challenge data
│       ├── tags.txt ............................... contains tagset
│       └── words.txt .................... contains all training words
├── dataLoader ..................... contains data loading modules
│   └── dataLoader.py .. module that loads vocab/ char/ tag mappings
│       and generates batch data
├── experiments .................. contains logs of ran experiments
├── heroku-app ............ contains all modules relevant to web-app
├── logger
│   └── logger.py ......................... implements logger class
├── model ...................... contains all model specific modules
│   ├── layers ........................ contains layers usable by Net
│   │   ├── charEmbed ......... implements character embedding layer
│   │   ├── embedLayer ............ implements word embedding layer
│   │   └── LSTM ........................... implements LSTM layer
│   ├── eval.py ........................ implements metric functions
│   ├── loss.py .......................... implements loss functions
│   └── net.py ................ implements main neural network class
├── results ........................ contains results of prediction
│   └── prediction
├── utils ................................ contains utility modules
│   └── util.py ...................... implements utility functions
├── buildData.py .script used to build the Data (create sentences and
│   labels files
└── buildVocab.py .. script used to build the vocab (create words and
    tags files
```

```
│
├── evaluate.py ..... implements evaluation function used in training
├── hyperParameterSearch.py  script that starts training jobs used for
│   parameter search
├── predict.py .... script used for predicting sample text or test data
└── train.py .............................. script to train neural net
```

D.4 Installation

The project requires Python 3.6.1 or later. To install all dependencies simply install the requirements.txt file.

D.5 System tests

Apéndice E

User Manual

E.1 Introduction

This user manual serves as a guide on how to use the provided scripts.

E.2 User requirements

The project requires Python 3.6.1 or later.

E.3 Installation

To install all dependencies simply install the requirements.txt file.

E.4 User manual

Demo web-app

E.5 Data

This section describes how the Data is saved and pre-processed. The description is divided into two sections because the NLTK classifiers and the pytorch classifier read and process Data in a different way.

Data description

Dataset Formats

WNUT 2017 Format Each line of original Data describes one token (word). Each line contains the word and it's entity.
Example:
@Suzie55 O
whispering O
cause O
I O
may O
have O
had O
1 O
too O
many O
vodka B-product
's O
last O
night O

NLTK

Input Format The input format of the file for the NLTK classifiers contain a token per line. A line is made up by the word, it's POS tag and named entity tag with IOB tagging scheme.
Example:
@Suzie55 JJ O
whispering NN O
cause NN O
I PRP O
may MD O
have VB O
had VBD O
1 CD O
too RB O
many JJ O
vodka NN B-product
's POS O
last JJ O

night NN O

Preprocessing The following steps are necessary to transform the data into the NLTK classifiers input format:

- Remove entity Tags: All entity tags have to be remove in order to tokenize file later on
- Tokenization: Tokenize Datasets into sentences and then words
- POS Tagging: POS tag resulting words from previous step
- Join pos tagged results with previously removed entity tags

Pytorch

Input Format The dataset is divided into two different text files. One containing all it's sentences (sentences.txt), each line containing one sentence, the other containing their associated labels.
Example (Representation of a sentence):
sentences.txt: @Suzie55 whispering cause I may have had 1 too many vodka 's last night and am a lil fragile , hold me ?

labels.txt: O O O O O O O O O O B-product O O O O O O O O O O O O

Output Format

The outout format of all classifiers look the same for easy use of results. Each line describes one token, containing it's word, gold-entity and prdicted entity in IOB tagging scheme. (token gold-label predicted-label). Sentences are devided by an empty line.
Example: @Suzie55 O O
whispering O O
cause O O
I O O
may O O
have O O
had O O

```
1 O O
too O O
many O O
vodka B-product B-product
's O O
last O O
night O O
```