# Case - Natural Language Processing

INTRODUCTION  
Assume that your organization is developing a product with a chat bot functionality, i.e. the end user talks to a device, the speech gets translated into text and now it is your task to map the given text to an assumed intent of the end user.

The data comes partitioned into training, validation and test, but whether you need to train a model or approach the task with pre-trained models is up to you. But the solution should no matter be evaluated on the test data set, that is, evaluated on whether for each text snippet in the test data set, your algorithm maps that text into the right intent.

DATA SET  
In this folder you will a json file named json\_data.json, which contains text-based questions/commands and the intent of these questions/commands. In the json file the first layer of key values will be separated into training, validation and test. Ensure you have a good understanding of the data set.

PERFORMANCE METRICS  
Your solution does not need to perform the task perfectly, but the performance needs to be reported in some statistics of your choosing, and the task will be used as an offset for further discussion.

PERSPECTIVE  
Consider the following situations and questions for discussion in the presentation

* How should we handle text input from a user which has an intent not currently present in our data?
* How do we handle text inputs which could have multiple intents?
* How will dialect or slang impact the solution?
* What do you expect your solutions weaker sides to be?

LIBRARIES  
You are free to choose any solution approach. Some libraries which may or may not be helpful are:

* Spacy: <https://github.com/explosion/spaCy>
* Haystack: <https://github.com/deepset-ai/haystack>
* NeuralCoref: <https://github.com/huggingface/neuralcoref>
* BERT: <https://huggingface.co/docs/transformers/model_doc/bert>
* USE: <https://spacy.io/universe/project/spacy-universal-sentence-encoder>
* Faiss: <https://github.com/facebookresearch/faiss>

# My Notes

(n: literature research: “A survey of joint intent detection and slot-filling models in natural language understanding” by H.Weld et al. 2021)

Typically set up as a sentence classification problem. That is, a feature or features are constructed from the sentence and these are passed through a classification algorithm to predict a class for the sentence from a predefined set of classes.

Issues in intent detection:

**Ambiguity in interpretation**

The problem of identifying the decision boundary between samples close together in feature space, yet belonging to different classes. This issue may be more prevalent with short texts, since they may include insufficient information and not follow correct grammar.

[Ren and Xue 2020] proposed training triples of samples - an anchor sample, a positive sample in the same class and a negative sample from a different class. Combining convolutional and BERT encodings of each one and mapping them to Euclidean space with Siamese shared weights, an intermediate loss of the anchor-positive distance minus the anchor-negative distance is minimised.

\*i.e. from Contrastive learning

**Small training sets**

Insert pre-trained models in the pipeline, for e.g. token embeddings. Or use unsupervised learning.

Text pre-processing can optimize the usage of small datasets: lowercase, replacing numerical digits with a unique token (e.g. §).

**Imbalanced classes**

Training a model on imbalanced data can cause poor performance on minority classes.   
It can be countered by oversampling: either replicating the training samples for rare classes, or adding augmented sentences (e.g. replacing 1 or more words with one of their synonyms).

**Multi-lingual and multi-domain generalization capability**

Ensemble models, one for each language. Relying on models with multi-lingual capability, such as mBert.

Language-specific representations may be converted into task-specific representations. The first level (the creation of the language representations) can be initialized via pre-training e.g. language modeling in the target language

**Emerging and unseen intents**

Keep an archive of previously seen intents, either as text or vector representation. It could also be possible to evaluate the vector representation of the query and try to understand whether it can be associated with one of the existing intents via clusters, or be part of a new intent entirely.

**Words Out-Of-Vocabulary**

Replace words with frequency < k (e.g. k=2) with an UNKNOWN token, but this has a considerable impact when the input utterances are short

Use sub-word encoding (like the BERT WordPiece encoding)

{Wang et al 2019b, Character-CNN-BGRU}

**Multi-intent prediction**

One straightforward solution is to use the top couple of predictions from existing single label classifiers. Another way is having binary classifiers for every label in single classifiers. Or treating double labels as separate atomic labels.

## Instruments that may be useful from the libraries & co:

* Spacy: <https://github.com/explosion/spaCy>

|  |  |
| --- | --- |
| **Tokenization** | Segmenting text into words, punctuations marks etc . |
| **Part-of-speech** (POS) **Tagging** | Assigning word types to tokens, like verb or noun. |
| **Dependency Parsing** | Assigning syntactic dependency labels, describing the relations between individual tokens, like subject or object. |
| **Lemmatization** | Assigning the base forms of words. For example, the lemma of “was” is “be”, and the lemma of “rats” is “rat”. |
| **Sentence Boundary Detection** (SBD) | Finding and segmenting individual sentences. |
| **Named Entity Recognition** (NER) | Labelling named “real-world” objects, like persons, companies or locations. |
| **Entity Linking** (EL) | Disambiguating textual entities to unique identifiers in a knowledge base. |
| **Similarity** | Comparing words, text spans and documents and how similar they are to each other. |
| **Text Classification** | Assigning categories or labels to a whole document, or parts of a document. |
| **Rule-based Matching** | Finding sequences of tokens based on their texts and linguistic annotations, similar to regular expressions. |
| **Training** | Updating and improving a statistical model’s predictions. |
| **Serialization** | Saving objects to files or byte strings. |

* Haystack: <https://github.com/deepset-ai/haystack>

## Overview

Haystack is an **open-source framework** for building **search systems** that work intelligently over large document collections. Recent advances in NLP have enabled the application of question answering, retrieval and summarization to real world settings and Haystack is designed to be the bridge between research and industry.

* **NLP for Search**: Pick components that perform [retrieval](https://haystack.deepset.ai/components/retriever), [question answering](https://haystack.deepset.ai/components/reader), [reranking](https://haystack.deepset.ai/components/ranker) and much more
* **Latest models**: Utilize all transformer based models (BERT, RoBERTa, MiniLM, DPR) and smoothly switch when new ones get published
* **Flexible databases**: Load data into and query from a range of [databases](https://haystack.deepset.ai/components/document-store) such as Elasticsearch, Milvus, FAISS, SQL and more
* **Scalability**: [Scale your system](https://haystack.deepset.ai/guides/optimization) to handle millions of documents and deploy them via [REST API](https://haystack.deepset.ai/guides/rest-api)
* **Domain adaptation**: All tooling you need to [annotate](https://haystack.deepset.ai/guides/annotation) examples, collect [user-feedback](https://haystack.deepset.ai/guides/domain-adaptation#user-feedback), [evaluate](https://haystack.deepset.ai/guides/evaluation) components and [finetune](https://haystack.deepset.ai/guides/domain-adaptation) models.
* NeuralCoref: <https://github.com/huggingface/neuralcoref>

NeuralCoref is a pipeline extension for spaCy 2.1+ which annotates and resolves coreference clusters using a neural network. NeuralCoref is production-ready, integrated in spaCy's NLP pipeline and extensible to new training datasets.

* BERT: <https://huggingface.co/docs/transformers/model_doc/bert>
* USE: <https://spacy.io/universe/project/spacy-universal-sentence-encoder>

Make use of Google's Universal Sentence Encoder directly within spaCy

import spacy\_universal\_sentence\_encoder  
*# load one of the models: ['en\_use\_md', 'en\_use\_lg', 'xx\_use\_md', 'xx\_use\_lg']*  
nlp = spacy\_universal\_sentence\_encoder.load\_model('en\_use\_lg')  
*# get two documents*  
doc\_1 = nlp('Hi there, how are you?')  
doc\_2 = nlp('Hello there, how are you doing today?')  
*# use the similarity method that is based on the vectors, on Doc, Span or Token*  
print(doc\_1.similarity(doc\_2[0:7]))

* Faiss: <https://github.com/facebookresearch/faiss>

# Faiss

Faiss is a library for efficient similarity search and clustering of dense vectors. It contains algorithms that search in sets of vectors of any size, up to ones that possibly do not fit in RAM. It also contains supporting code for evaluation and parameter tuning. Faiss is written in C++ with complete wrappers for Python/numpy. Some of the most useful algorithms are implemented on the GPU. It is developed primarily at [Facebook AI Research](https://ai.facebook.com/).

## Introduction

Faiss contains several methods for similarity search. It assumes that the instances are represented as vectors and are identified by an integer, and that the vectors can be compared with L2 (Euclidean) distances or dot products. Vectors that are similar to a query vector are those that have the lowest L2 distance or the highest dot product with the query vector. It also supports cosine similarity, since this is a dot product on normalized vectors.

## Dataset Exploration

**Instances in splits**: training 15000, validation 3000, test 4500

**How many instances for each class/intent?**   
150 intents in the training set (['translate', 'order\_status', 'goodbye', 'account\_blocked', 'what\_song', 'international\_fees', 'last\_maintenance', 'meeting\_schedule', 'ingredients\_list', 'report\_fraud,…],   
each with frequency exactly 100

**How many words in the utterances?**Using the NLTK tokenizer. General average over the 150 intents: 8.5 words. Maximum: ‘book\_flight’=15.7  
Minimum: ‘yes’=4.1

The **vocabulary**?

INFO : The 15000 utterances of the training dataset contain in total 128,344 words, with an average of 8.5 word per utterance  
INFO : The vocabulary has 5195 different words  
INFO : There are 2361 words with frequency=1  
INFO : There are 625 words with frequency=2  
INFO : There are 338 words with frequency=3

**Preprocessing** steps:  
The dataset is already lowercased

## Models

Possible baselines, with the caveat that we do not have slot labels, but only sentence classification labels, as in:  
["what is the scheduled arrival time for my flight", "flight\_status"]

Simple baseline:

Use Google’s Universal Sentence Encoder to turn the sentences into vectors.

Use a simple classification architecture (e.g. 2-layers FF-NN) to choose the correct intent.

[It’s possible to visualize the vectors in a 2-dimensional space via PCA.]

First step: overfit on fragment.  
INFO : class\_names = ['definition', 'meaning\_of\_life', 'timer', 'transfer']

INFO : \*\*\*\*\*\* Current epoch: 1 \*\*\*\*\*\*   
INFO : Training sample: 12/ 60 ...  
INFO : Training sample: 24/ 60 ...  
INFO : Training sample: 36/ 60 ...  
INFO : Training sample: 48/ 60 ...  
INFO : Training sample: 60/ 60 ...

INFO : loss=1.38 ; accuracy=0.333  
INFO : precision=[0.0, 0.36, 0.0, 0.2]  
Recall=[0.0, 0.9, 0.0, 0.1]  
INFO : F1\_score=[0.0, 0.51, 0.0, 0.13]

INFO : \*\*\*\*\*\* Current epoch: 40 \*\*\*\*\*\*

INFO : Training sample: 12/ 60 ...  
INFO : Training sample: 24/ 60 ...  
INFO : Training sample: 36/ 60 ...  
INFO : Training sample: 48/ 60 ...  
INFO : Training sample: 60/ 60 ...

INFO : loss=0.52 ; accuracy=1.0  
INFO : precision=[1.0, 1.0, 1.0, 1.0]  
Recall=[1.0, 1.0, 1.0, 1.0]  
INFO : F1\_score=[1.0, 1.0, 1.0, 1.0]

Second step: add early-stop based on the validation set

INFO : class\_names = ['accept\_reservations', 'account\_blocked', 'alarm', 'application\_status', 'apr', 'are\_you\_a\_bot', 'balance', … 'what\_is\_your\_name', 'what\_song', 'where\_are\_you\_from', 'whisper\_mode', 'who\_do\_you\_work\_for', 'who\_made\_you', 'yes']

|  |  |
| --- | --- |
| INFO : \*\*\*\*\*\* Current epoch: 1 \*\*\*\*\*\*  INFO : Training sample: 60/ 300 ...  INFO : Training sample: 120/ 300 ...  INFO : Training sample: 180/ 300 ...  INFO : Training sample: 240/ 300 ...  INFO : Training sample: 300/ 300 ...  INFO : loss=5.01 ; accuracy=0.003  INFO : precision=[0.0, 0.0, 0.0, … 0.0, 0.0, 0.0]  Recall=[0.0, 0.0, 0.0, … 0.0, 0.0, 0.0]  INFO : F1\_score=[0.0, 0.0, 0.0, …, 0.0, 0.0, 0.0] | Evaluation  INFO : Sample: 30/ 150 ...  INFO : Sample: 60/ 150 ...  INFO : Sample: 90/ 150 ...  INFO : Sample: 120/ 150 ...  INFO : Sample: 150/ 150 ...  INFO : loss=5.0 ; accuracy=0.0  INFO : precision=[0.0, 0.0, 0.0, … 0.0, 0.0, 0.0]  INFO : F1\_score=[0.0, 0.0, 0.0, 0… 0.0, 0.0, 0.0, 0.0]  INFO : validation\_loss=5.001 ; best\_validation\_loss=inf |
| INFO : \*\*\*\*\*\* Current epoch: 20 \*\*\*\*\*\*  INFO : Training sample: 60/ 300 ...  INFO : Training sample: 120/ 300 ...  INFO : Training sample: 180/ 300 ...  INFO : Training sample: 240/ 300 ...  INFO : Training sample: 300/ 300 ...  INFO : loss=4.83 ; accuracy=**0.827**  INFO : precision=[1.0, 1.0, 1.0, 0.67, 1.0, 1.0, 0.4, … 0.67, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 0.0, 1.0, 1.0]  Recall=[1.0, 1.0, 1.0, … 0.5, 1.0, 1.0, 1.0, 1.0, 1.0, 0.0, 1.0, 1.0, 1.0, 0.5, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 0.5, 1.0, 1.0, 0.5, 1.0, 0.0, 1.0, 1.0]  INFO : F1\_score=[1.0, 1.0, 1.0, 0.8, 1.0, 1.0, 0.57, …1.0, 1.0, 1.0, 1.0, 1.0, 0.67, 1.0, 1.0, 0.67, 1.0, 0.0, 1.0, 1.0] | Evaluation  INFO : Sample: 30/ 150 ...  INFO : Sample: 60/ 150 ...  INFO : Sample: 90/ 150 ...  INFO : Sample: 120/ 150 ...  INFO : Sample: 150/ 150 ...  INFO : loss=4.88 ; accuracy=**0.533**  INFO : precision=[0.0, 0.0, 0.5, 1.0, 0.5, 0.0, 0.0, … 0.33, 1.0, 1.0, 1.0, 0.0, 0.5, 0.0, 0.0, 1.0, 0.0, 1.0, 0.0, 0.5, 1.0, 0.0, 0.0, 1.0, 0.0]  Recall=[0.0, 0.0, 1.0, 1.0, 1.0, … 0.0, 1.0, 0.0, 1.0, 1.0, 0.0, 0.0, 1.0, 0.0]  INFO : F1\_score=[0.0, 0.0, 0.67, 1.0, 0.67, 0.0, 0.0, 0.0, 0.0, 1.0, 1.0, 0.0, 0.0, 0.4,… 0.67, 1.0, 0.0, 0.0, 1.0, 0.0]  INFO : validation\_loss=**4.875** ; best\_validation\_loss=**4.884** |

At the point of mandatory stop by max epochs, it appears to still have room for improvement.

Third step: put the model on GPU and check that we have the same results, to speed up the training when we will use the whole training and validation datasets

Eventually getting

|  |  |
| --- | --- |
| FO : \*\*\*\*\*\* Current epoch: 30 \*\*\*\*\*\*  INFO : Training sample: 60/ 300 ...  INFO : Training sample: 120/ 300 ...  INFO : Training sample: 180/ 300 ...  INFO : Training sample: 240/ 300 ...  INFO : Training sample: 300/ 300 ...  INFO : loss=3.99 ; accuracy=0.973 | INFO : Evaluation  INFO : Sample: 30/ 150 ...  INFO : Sample: 60/ 150 ...  INFO : Sample: 90/ 150 ...  INFO : Sample: 120/ 150 ...  INFO : Sample: 150/ 150 ...  INFO : loss=4.26 ; accuracy=0.76  INFO : validation\_loss=4.259 ; best\_validation\_loss=4.302 |

Fourth step: train the model on the entirety of the training set (15k instances), checking the entirety of the validation set (3k instances). Remember to leave the saved model where it is.  
(and to write the code to load a pre-defined model)