Project Guidelines

Machine Learning, Master in Data Science, UPC

This document contains the guidelines for the course project. Please read with care!

General Information

ML Course Project Competition Page

This project is meant to give you the opportunity to apply the techniques seen during the course to a real-world dataset. The project should cover all aspects of the modelling methodology seen in class from preprocessing to generating a final predictive model together with an assessment of its prediction quality.

The project is to be done in teams of **two** persons; singles are not allowed. This year we introduce as a novelty the use of Kaggle ML platform to develop and carry out your project. The main advantage of this is that you can monitor the evolution of your team (or competing teams) using this page's automatic prediction scoring.

In the course project's competition page you will find a labelled *training set*, and an unlabelled *test set*. Over the course of the project, you may submit label predictions for the test set, and Kaggle automatically tells you the test score obtained (based on a subset of the test examples). Final scores based on the full test set are released after the competition has finished.

Apart from building your predictions, you are expected to hand in a written report. This document should describe the work carried out, the problems encountered and the solutions found together with final results and conclusions of your study.

To carry out your analysis you should use the language `python`. Remember that there are many useful packages that extend its basic functionality. Certainly you can find inspiration in the notebooks from our weekly laboratory sessions. If you use code or ideas or any kind of resource from elsewhere you should cite it appropriately. Plagiarism will be prosecuted.

The project

The problem you are going to tackle is that of predicting the root node in a free tree[1]. In this problem, an example is a free tree (in the form of a list of undirected edges) and the

[1] A *free tree* is an unrooted tree.

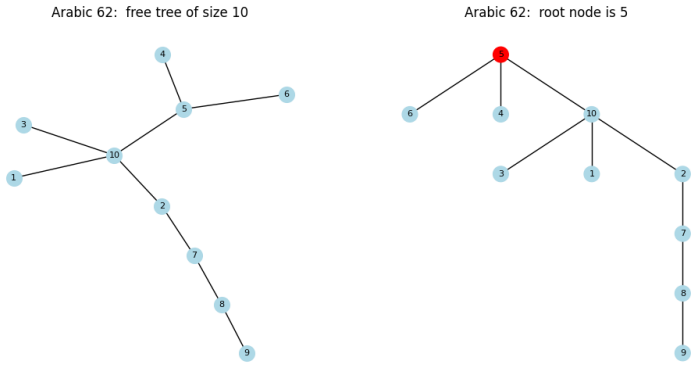
problem is to predict what node is likely to be the root of this tree.

Our data originally comes from a parallel corpus of 21 languages, with 995 sentences in each language[2].

The trees in the data provided are syntactic dependency trees. This means that each sentence in the corpus is represented by a tree whose nodes are the sentence words and edges correspond to syntactic dependencies among those words. The running hypothesis is that the most important word in the sentence (the *root word*) corresponds to a central node within the tree, so we are going to use *centrality scores* as features to represent nodes.

We have randomly divided the 995 sentences into a training partition with 500 x 21 sentences and a test partition with the remaining 495 x 21 sentences. The partition is done *respecting sentence ids*, that is, the same sentence is included with its 21 language versions either into the train partition or into the test partition. This is to avoid possible leakage from train to test that translated sentences may have.

As a reference, the following picture corresponds to sentence #62 in Arabic language, where on the left you can see a representation of the *free tree* (input) and on the right you can see a representation of the *rooted tree* where the root node has been placed at the top.



In the training set labels are the node ids of the root of the trees; in the test it is your task to predict what node is likely to be the root for each tree.

Using binary classification

The underlying problem that you have to solve is the following: for each vertex in a tree, predict whether it is the root or not. So, our root prediction problem can be reduced to a binary classification problem at a node level. We need information on how to characterize nodes within trees in order to find features that will correlate with the *rootness quality* of a node. Good candidates for this are *centrality scores* which essentially capture importance

[2] A *parallel corpus* is a corpus that contains translations of the same sentence into several languages. This is why we have the same number of sentences in each of the 21 languages.

of vertex in the context of the graph/tree where it lies. In this project, we propose that you use such centrality scores as features for the root/no_root predictions.

An initial good set of candidates for centrality scores are: *degree*, *closeness*, *betweenness*, and *PageRank* centrality. Feel free to include others that you think may help, available ones can be found in the networkx centrality documentation page. Given the edgelist of a tree, you can compute these measures using the `networkx` package with the following code:

```
! pip install networkx
import networkx as nx

def centralities(edgelist):
        """
         - edgelist is a list of node pairs e.g. [(7,2),(1,7),(1,9),...]
         - returns a dictionary of vertex -> (centrality values)
        """
        T = nx.from_edgelist(edgelist)
        dc = nx.degree_centrality(T)
        cc = nx.harmonic_centrality(T)
        bc = nx.betweenness_centrality(T)
        pc = nx.pagerank(T)

        return {v: (dc[v], cc[v], bc[v], pc[v]) for v in T}
```

Dataset preparation

To generate a training set suitable for binary classification models using centralities as vertex features, we proceed as follows; in pseudocode:

- **for** each sentence in the `train.csv` provided:
  - build tree T from edgelist for sentence
  - compute *centralities* of vertices in T
  - **for** each vertex v in tree T:
    - generate features from centralities
    - set target = 1 if vertex is the root, otherwise it is 0
    - add row (features, target) to binary classification training dataset

You can obtain a binary classification dataset by applying this procedure to all training sentences; let us call the resulting binary classification dataset the *expanded, node-level* training dataset. Once you have built this expanded dataset, you may proceed modelling classifiers as usual (well, beware of some peculiarities of this dataset, see below).

The following rows result from doing this process for Arabic sentence #62:

language	sentence	n	vertex	degree	closeness	betweenness	pagerank	is_root
Arabic	62	10	1	0.111111	3.950000	0.0	0.058583	0
Arabic	62	10	2	0.222222	5.000000	18.0	0.104807	0
Arabic	62	10	3	0.111111	3.950000	0.0	0.058583	0
Arabic	62	10	4	0.111111	3.616667	0.0	0.060961	0
Arabic	62	10	5	0.333333	5.283333	15.0	0.162218	1
Arabic	62	10	6	0.111111	3.616667	0.0	0.060961	0
Arabic	62	10	7	0.222222	4.500000	14.0	0.108763	0
Arabic	62	10	8	0.222222	3.983333	8.0	0.115810	0
Arabic	62	10	9	0.111111	3.016667	0.0	0.064220	0
Arabic	62	10	10	0.444444	6.083333	27.0	0.205094	0

Particularities of our data

1. There are **dependencies** among examples in our data:

    - At the *sentence* level, same sentence ids of different languages correspond to translations of the same sentence. This is why the train and test partition have been obtained by adding all sentences with same id to exactly one of the partitions.
    - Examples in the *expanded* (node-level) dataset correspond to vertices of sentences in our data. So, examples are clearly not independent: those examples that correspond to vertices in the same sentence are closely related for example in the range of centrality values that they may have. Thus, caution needs to be taken e.g. when doing cross-validations to avoid that vertices corresponding to the same sentence end up in different folds.

2. **Normalization**: different sentences have different lengths so it is probably a good idea to normalize the data. But the whole-dataset column-normalization that we typically apply may not be such a good idea here, it is advisable that we normalize within sentences (see previous point).

- **Class imbalance**: due to the fact that only one node is a root given a tree, the resulting expanded dataset is highly imbalanced. Your classifiers will need to take this into account in some way.

Files available

Two datasets are provided, a `train.csv` file containing the training sentences with their root, and a `test.csv` file containing the test sentences (no root, this is for you to guess). Additionally, we provide a `sample_submission.csv` file that has been generated by guessing the root uniformly at random among all possible nodes.

Columns

- `id` - for the test sentences only, an id for scoring purposes in the submission file
- `language` - language of sentence
- `sentence` - sentence number, note that each number is present for all languages (parallel corpus)
- `n` - length of the sentence (or number of nodes in the tree)
- `root` - vertex id of the root node (target)
- `edgelist` - list of edges representing the syntactic dependence tree

Note that the sentence number is included for resampling strategies, but should not be used for the predictions.

When reading an edgelist from csv files, it is read as a string. The following code shows a simple way to turn such edgelist strings into a python edgelist, which you can then use to create a `networkx` tree for example:

```
import ast
import networkx as nx

edges = ast.literal_eval(edgelist_str)
T = nx.from_edgelist(edges)
```

Teams and participation

Every student enrolled in the ML course can join the competition. However, since the project is to be done in pairs, only one account will be considered in the final submission. **Please include in your final report the email that your team is using to compete**.

Evaluation metric

Submissions are evaluated using 0-1 loss: for each sentence in the test set, you should provide your guess for the root. The evaluation score is the fraction of correct guesses for the whole test set. **Note that scores will tend to have low accuracies; a random guess for an n-node tree is** $1/n$.

Submission File

Your submission file should contain the following information:

```
id,root
1,10
2,12
...
```

Needless to say, the root node id should be consistent with the labels of the tree edges given in the `test.csv` file.

Report and final delivery

The final report should include:

1. A brief description of the work and its goals, data available, and any additional information that you may have used.
2. The data exploration process, including: pre-processing, feature engineering, etc.
3. Modeling methods considered, validation protocol and the reasons why the choices were made.
4. Results obtained with each method used (along with best set of parameters), comparison of results.
5. Final model chosen and an estimation of its generalization performance.
6. Scientific and personal conclusions

Note that the report should not describe explanations seen in class; every table or plot should be appropriately described. The style of the report should resemble what you encounter in a scientific publication. Your code should be **reproducible**; that means using "seeds" if your code is stochastic.

Make sure you include a variety of linear and non-linear methods seen during the course.

All deliveries are to be made through the racó. **Important: only one member of the team should upload the material** Additionally, you should submit your final predictions for the test set.

For the final delivery, make sure you include in a compressed file the following:

1. The written report (pdf document). It should not exceed **5 pages**; if you need more space, consider placing the secondary information in a **separate appendix file**.
2. Any script or code you have used (`python` notebooks, scripts, or any other code)
3. A flat text file with precise instructions on how to execute and reproduce your results.
4. Your final submission file to the Kaggle competition.

Code and report should be ***separate documents*** (namely, a *python notebook* is not a report).

Project evaluation

Your final project will be evaluated on the basis of the clarity of your report as well as on its technical quality. Conditions for a good score are:

1. The appropriate use of techniques and methods seen in class (30 %)
2. Care and rigor of methodology (resampling protocol, quality metrics, etc.) (30 %)
3. Quality of obtained results (generalization error, simplicity, interpretability) (10 %)
4. Quality of written report (conciseness, completeness, clarity, format, etc.) (30%)

Special attention will be given to insights into the results obtained, gaining knowledge on the data analyzed and obtaining useful conclusions. All experimental decisions should be appropriately justified (on the resampling protocol or preprocessing, for example). Merely applying the methods to the data and showing the table of results is not enough, there has to be an interpretation of the results obtained.

You should not obsess over Kaggle's scoreboard. In particular, you should avoid using Kaggle's scores as a guiding mechanism for model selection; make a submission whenever you have a significantly different model. For model selection and hyper-parameter tuning you should use the customary cross-validation or similar resampling procedures over the training data as we have shown in many lab sessions.

Important dates

- **June 3rd**: deadline for submitting your final report and code through the racó
- **June 4th**: Kaggle competition finishes and final scoreboard is released