Topic 1: Processing wals data

Wikipedia defines The World Atlas of Language Structures (WALS) as a database of structural properties of languages gathered from descriptive materials. For each language in the WALS database, there are around 192 different features. But for many languages, there are many missing values.

Task 1.a:

Task description:

We need to create a system, where feeding in the WALS code of a particular language generates a list of the most similar languages.

Solution:

For the system, right at the beginning, two separate Python lists from the WALS dataset were extracted. One of the lists (L1) contained the WALS codes of different languages. It was also realized that “useful” features for the task were contained in all columns starting from Column 4. And thus, the second list (L2) was populated by the “useful” features. But there were many missing values for L2. So, a ‘NaN’ value was inserted into all the missing places.

For calculation of the similarity, the Jaccard similarity was used. In the Jaccard similarity, members of two sets are compared to see which members are shared and which are distinct. This gives an indication of how similar two sets are. Jaccard similarity was used for the calculation of similarity of languages based on the feature data from the WALS data.

A Python script was written (Task_1a.py) to accomplish this task.

Sample Results:

Source	Similar Langauges
hin	[('guj', 0.03125), ('lmn', 0.03125), ('pan', 0.03125), ('mhi', 0.02857142857142857), ('ngt', 0.02857142857142857)]
ben	[('lmn', 0.033942558746736295), ('mhi', 0.033942558746736295), ('chp', 0.03125), ('miz', 0.03125), ('sla', 0.03125)]
eng	[('dut', 0.03125), ('ger', 0.03125), ('kse', 0.03125), ('pol', 0.03125), ('spa', 0.03125)]
cze	[('cyv', 0.03125), ('kew', 0.03125), ('lat', 0.03125), ('prs', 0.03125), ('apl', 0.02857142857142857)]
miz	[('baw', 0.03664921465968586), ('lai', 0.03664921465968586), ('dni', 0.033942558746736295), ('sla', 0.033942558746736295), ('ao', 0.03125)]

Results:

It is apparent that the WALS features are very useful in determining language similarity. But a case-wise analysis of the languages shown in the sample results show that the missing values in the WALS dataset do cause problems with accuracy. But nevertheless, the results are nevertheless interesting.

For instance, for the language Bengali with the WALS code ben, the most similar languages are listed as Lamani (lmn), Marathi (mhi), Chipewyan (chp), Mizo (miz) and Slave (sla). Similarities between Bengali and Marathi is a well explored and well established fact. Also the geographical positioning of Mizo speakers and their proximity to Bengali speakers may justify the language closeness as shown in the results. But Lamani is a language that is mostly spoken by the 'Banjara' people (nomadic tribes). In terms of geographical distribution of its speakers, it is very far away from Benglai speakers. Thus the inclusion of Lamani in the results is strange. Finally, the inclusion of Slave and Chipewyan in the results, which are languages from Canada is also strange. It can be believed that the lack of entries in the feature matrix obtained from the WALS dataset might be the reason for such observations.

Task 1.b:

Task description:

We need to create a system, where feeding in the genus generates the centroid language of that genus, i.e. a language most similar to other languages of the genus.

Solution:

We started by extracting the list of all genera with corresponding languages and their WALS codes as mentioned in the WALS datasets using the script “Database_maker.py”. This resulted in a csv file which contained the extracted information in the following format:
Genus, Language, WALS
A separate script “Genus_Extractor.py” was used to create a list of all genera in the dataset, resulting in the ‘Genus’ file. This file was used to select the genus for which the lookup had to be done.

We use the extracted csv file and the script (‘Integrate_b.py’) from the previous task to iterate over languages in a genus and determine the language most similar to others in that genus and print it. All of this is contained in the script ‘Task1_b.py’. The methodology of selecting the most similar language was to generate the ‘most’ similar language on the basis of Jaccard similarity for every language in the genus and storing it in a list. The language with the maximum frequency in the list is classified as the most similar language.

Sample Results:

Enter Genus: Slavic

Languages under genus Slavic :
Belorussian
Bosnian
Bulgarian
Czech
Kashubian
Macedonian
Polabian
Polish
Russian
Serbian-Croatian
Slovak
Slovene
Slovincian
Sorbian
Sorbian (Lower)
Sorbian (Upper)
Ukrainian

Most related:
Bulgarian

Enter Genus: Indic

Languages under genus Indic :
Assamese
Awadhi
Bagri
Bengali
Bengali (Chittagong)
Bhili
Bhojpuri
Brokskat
Darai
Dhivehi
Dogri

Domaaki
Domari
Gojri
Gujarati
Halbi
Hindi
Kalami
Kalasha
Kashmiri
Khowar
Kokni
Konkani
Kumauni
Lamani
Magahi
Maithili
Marathi
Mawchi
Nepali
Oriya
Oriya (Kotia)
Panjabi
Romani (Ajia Varvara)
Romani (Bugurdzi)
Romani (Burgenland)
Romani (Kalderash)
Romani (Lovari)
Romani (North Russian)
Romani (Sepecides)
Romani (Welsh)
Savi
Shekhawati
Shina
Sindhi
Sinhala
Torwali
Urdu
Vasavi
Vedda

Most related:
Assamese

Task 1.c:

Task description:

We need to find the weirdest language (the most dissimilar language) in the WALS dataset.

Solution:

Taking help from the previous tasks, we repurpose the script used in Task 1a to generate the most dissimilar language corresponding to a particular language by sorting the Jaccard similarity scores in an ascending manner and then extracting the key of the first element from the sorted dictionary (“Integrate_c.py’). We then looped through all the languages in the WALS dataset and saved the most dissimilar language in a list. The language that occurred most frequently in that list was ruled as the weirdest language (“Task_1c.py”).

Result:

‘Xiriana’ (WALS code:bah) seems to be the weirdest language as per the script (‘Calc.py’).

Topic 2: Tokenization task

Task 0: Comparing the accuracy of different tokenization models with a particular test data.

For this particular experiment, an Italian test data was used (it_isdt-ud-test.conllu) that was obtained from the GitHub page of the Universal Dependencies project via the link: (https://github.com/UniversalDependencies/UD_Italian-ISDT/blob/master/it_isdt-ud-test.conllu)

Different Italian tokenizer models were run on the same data to extract the relevant statistics. The details of the model, gold data details and then the model performance details are given below.

Models:

italian-isdt-ud-2.5-191206.udpipe
italian-partut-ud-2.5-191206.udpipe
italian-postwita-ud-2.5-191206.udpipe
italian-twittiro-ud-2.5-191206.udpipe
italian-vit-ud-2.5-191206.udpipe

Gold Data details:

Number of SpaceAfter=No features in gold data: 1425
Tokenizer tokens - gold: 9680
Tokenizer multiword tokens - gold: 736
Tokenizer words - gold: 10417
Tokenizer sentences - gold: 482

Comparison of model performances:

Model	**Kind of tokens**	**No of Tokens**	**Precision**	**Recall**	**F1**
italian-isdt-ud-2.5-191206.udpipe	Tokenizer tokens	9684	99.91%	99.95%	99.93%
italian-partut-ud-2.5-191206.udpipe	Tokenizer tokens	9683	99.41%	99.44%	99.43%
italian-postwita-ud-2.5-191206.udpipe	Tokenizer tokens	9521	98.42%	96.81%	97.61%
italian-twittiro-ud-2.5-191206.udpipe	Tokenizer tokens	9676	99.53%	99.49%	99.51%
italian-vit-ud-2.5-191206.udpipe	Tokenizer tokens	9662	99.42%	99.24%	99.33%

Model	Kind of tokens	No of Tokens	Precision	Recall	F1
italian-isdt-ud-2.5-191206.udpipe	Tokenizer multiword tokens	737	99.46%	99.59%	99.52%
italian-partut-ud-2.5-191206.udpipe	Tokenizer multiword tokens	731	99.32%	98.64%	98.98%
italian-postwita-ud-2.5-191206.udpipe	Tokenizer multiword tokens	729	98.90%	97.96%	98.43%

italian-twittiro-ud-2.5-191206.udpipe	Tokenizer multiword tokens	719	98.47%	96.20%	97.32%
italian-vit-ud-2.5-191206.udpipe	Tokenizer multiword tokens	731	99.59%	98.91%	99.25%

Model	Kind of tokens	No of Tokens	Precision	Recall	F1
italian-isdt-ud-2.5-191206.udpipe	Tokenizer words	10422	99.82%	99.87%	99.84%
italian-partut-ud-2.5-191206.udpipe	Tokenizer words	10414	99.29%	99.26%	99.28%
italian-postwita-ud-2.5-191206.udpipe	Tokenizer words	10250	98.28%	96.71%	97.49%
italian-twittiro-ud-2.5-191206.udpipe	Tokenizer words	10395	99.09%	98.88%	98.98%
italian-vit-ud-2.5-191206.udpipe	Tokenizer words	10393	99.36%,	99.13%	99.24%

Model	Kind of tokens	No of Tokens	Precision	Recall	F1
italian-isdt-ud-2.5-191206.udpipe	Tokenizer sentences	482	98.76%	98.76%	98.76%
italian-partut-ud-2.5-191206.udpipe	Tokenizer sentences	495	93.54%,	96.06%	94.78%

italian-postwita-ud-2.5-191206.udpipe	Tokenizer sentences	284	42.25%,	24.90%	31.33%
italian-twittiro-ud-2.5-191206.udpipe	Tokenizer sentences	185	23.24%	8.92%	12.89%
italian-vit-ud-2.5-191206.udpipe	Tokenizer sentences	505	91.09%	95.44%	93.21%

Conclusion:

The results suggest that the model “italian-isdt-ud-2.5-191206.udpipe” was the best performing model on average for tokenization task involving the dataset mentioned above.

Task 1 a: Determining best tokenization model for Cantonese

For this particular experiment, the initial version of the WALS task processing system was used to extract the top 5 languages that were the “most similar” to Cantonese.

The results for Cantonese are:

Language	Sim Lang 1	Sim Lang 2	Sim Lang 3	Sim Lang 4	Sim Lang 5
Cantonese (wals=cnt)	Mandarin	Bai	Hmong Njua	Ao	Apalaí

Unfortunately, there are no trained models for languages like Bai, Hmong Njua, Ao or Apalaí. So for the purpose of experiment, I use tokenization models from Chinese and accounting for the geographic proximity, we use the models for Vietnamese, Japanese and Korean to see which suits Cantonese better.

Models:

chinese-gsdsimp-ud-2.5-191206.udpipe
chinese-gsd-ud-2.5-191206.udpipe
classical_chinese-kyoto-ud-2.5-191206.udpipe
vietnamese-vtb-ud-2.5-191206.udpipe
japanese-gsd-ud-2.5-191206.udpipe
korean-gsd-ud-2.5-191206.udpipe
korean-kaist-ud-2.5-191206.udpipe

Gold Data details:

Number of SpaceAfter=No features in gold data: 13917
Tokenizer words - gold: 13918
Tokenizer sentences - gold: 1004

Comparison of model performances:

Model	Kind of tokens	No of Tokens	Precision	Recall	F1
chinese-gsdsimp-ud-2.5-191206.udpipe	Tokenizer words	15853	64.19%	73.11%	68.36%
chinese-gsd-ud-2.5-191206.udpipe	Tokenizer words	13901	77.05%	76.96%	77.00%
classical_chinese-kyoto-ud-2.5-191206.udpipe	Tokenizer words	18591	53.07%	70.89%	60.70%
vietnamese-vtb-ud-2.5-191206.udpipe	Tokenizer words	4290	44.99%	13.87%	21.20%
japanese-gsd-ud-2.5-191206.udpipe	Tokenizer words	12724	62.11%	56.78%	59.33%
korean-gsd-ud-2.5-191206.udpipe	Tokenizer words	3091	38.14%	8.47%	13.86%
korean-kaist-ud-2.5-191206.udpipe	Tokenizer words	3585	50.29%	12.95%	20.60%

Model	Kind of tokens	No of Tokens	Precision	Recall	F1
chinese-gsdsimp-ud-2.5-191206.udpipe	Tokenizer sentences	962	77.23%	74.00%	75.58%
chinese-gsd-ud-2.5-191206.udpipe	Tokenizer sentences	992	74.29%	73.41%	73.85%
classical_chinese-kyoto-ud-2.5-191206.udpipe	Tokenizer sentences	2260	0.22%	0.50%%	0.31
vietnamese-vtb-ud-2.5-191206.udpipe	Tokenizer sentences	799	61.45%	48.90%	54.46%
japanese-gsd-ud-2.5-191206.udpipe	Tokenizer sentences	1032	82.17%	84.46%	83.30%
korean-gsd-ud-2.5-191206.udpipe	Tokenizer sentences	329	44.98%	14.74%	22.21%
korean-kaist-ud-2.5-191206.udpipe	Tokenizer sentences	621	45.09%	27.89%	34.46%

Conclusion:

The results suggest that the Japanese model was the best performing model on average for tokenization task involving the dataset mentioned above.

Task 1 b: Determining best tokenization model for Bhojpuri

The five most similar languages to Bhojlpuri (on the basis of the methodology followed in Task 1a) are:

Language	Sim Lang 1	Sim Lang 2	Sim Lang 3	Sim Lang 4	Sim Lang 5
Bhojpuri (wals=bho)	Assamese	Gujarati	Hindi	Punjabi	Maithili

The only trained model available among the five languages given above is for Hindi. But experiments are done with Urdu, Telugu, Marathi and Tamil.

Models:

hindi-hdtb-ud-2.5-191206.udpipe
urdu-udtb-ud-2.5-191206.udpipe
telugu-mtg-ud-2.5-191206.udpipe
marathi-ufal-ud-2.5-191206.udpipe
tamil-ttb-ud-2.5-191206.udpipe

Gold Data details:

Number of SpaceAfter=No features in gold data: 5
Tokenizer words - gold: 6665
Tokenizer sentences - gold: 357

Tokenizer words - system:, gold: 6665, precision: Tokenizer sentences - system:, gold: 357, precision:

Comparison of model performances:

Model	Kind of tokens	No of Tokens	Precision	Recall	F1
hindi-hdtb-ud-2.5-191206.udpipe	Tokenizer words	6663	99.97%	99.94%	99.95%
urdu-udtb-ud-2.5-191206.udpipe	Tokenizer words	6681	99.52%	99.76%	99.64%
telugu-mtg-ud-2.5-191206.udpipe	Tokenizer words	6694	99.13%	99.56%	99.35%
marathi-ufal-ud-2.5-191206.udpipe	Tokenizer words	6703	98.91%	99.47%	99.19%
tamil-ttb-ud-2.5-191206.udpipe	Tokenizer words	6694	99.13%	99.56%	99.35%

Model	Kind of tokens	No of Tokens	Precision	Recall	F1
hindi-hdtb-ud-2.5-191206.udpipe	Tokenizer sentences	350	89.43%	87.68%	88.54%
urdu-udtb-ud-2.5-191206.udpipe	Tokenizer sentences	340	34.71%	33.05%	33.86%
telugu-mtg-ud-2.5-191206.udpipe	Tokenizer sentences	379	86.54%	91.88%	89.13%
marathi-ufal-ud-2.5-191206.udpipe	Tokenizer sentences	381	86.61%	92.44%	89.43%
tamil-ttb-ud-2.5-191206.udpipe	Tokenizer sentences	51	9.80%	1.40%	2.45%

Conclusion:

The results suggest that the Hindi, Marathi and Urdu models were the best performing models on average for tokenization task involving the dataset mentioned above.

Task 2: Training a new tokenization model for Sanskit

Training and Testing data was obtained from the Universal Dependencies Github page (https://github.com/UniversalDependencies/UD_Sanskrit-Vedic) before being processed using UDPipe. The model training statistics are given below:

Gold Data details:

Number of SpaceAfter=No features in gold data: 0
Tokenizer words - gold: 9672
Tokenizer sentences - gold: 1473

Comparison of model performances:

Model	Kind of tokens	No of Tokens	Precision	Recall	F1
Sanskrit	Tokenizer words	9672	100.00%	100.00%	100.00%
Hindi	Tokenizer words	9672	100.00%	100.00%	100.00%
german-gsd-ud-2.5-191206.udpipe	Tokenizer words	10476	84.65%	91.69%	88.03%
german-hdt-ud-2.5-191206.udpipe	Tokenizer words	9672	100.00%	100.00%	100.00%

Model	Kind of tokens	No of Tokens	Precision	Recall	F1
Sanskrit	Tokenizer sentences	1473	29.96%	20.71%	24.49%
Hindi	Tokenizer sentences	20	0.00%	0.00%	0.00%
german-gsd-ud-2.5-191206.udpipe	Tokenizer words	21	0.00%	0.00%	0.00%
german-hdt-ud-2.5-191206.udpipe	Tokenizer words	22	0.00%	0.00%	0.00%

Topic 3: POS harmonization

A number of changes were done to the harmomize.py successively to improve the performance of the system. The changes and the results are shown in the table below.

English tagset results:

	UAS	LAS
Original	15.21%	4.37%
Replacing all tags starting with ‘V’ with ‘VERB’	25.20%	10.77%
Replacing all tags starting with ‘J’ with ‘ADJ’	26.73%	12.08%
Replacing all tags starting with ‘N’ with ‘NOUN’	43.84%	25.99%
Replacing all tags starting with ‘R’ with ‘ADV’	44.11%	27.60%
Replacing all tags starting with ‘J’ with ‘ADJ’	44.11%	27.60%
Replacing all tags starting with ‘UH’ with ‘INTJ’	44.11%	27.87%
Replacing all tags starting with ‘IN’ with ‘ADP’	46.36%	32.40%
Replacing all tags starting with ‘DT’ with ‘DET’	51.86%	41.94%
Replacing all tags starting with ‘CD’ with ‘NUM’	52.65%	43.17%
Replacing all tags starting with ‘CC’ with ‘CCONJ’	52.96%	43.58%

Topic 5: Delex Parsing

Czech Solvak experiments

Treebank data: Slovak (sk_snk-ud-test.conllu)

Parser	UAS	LAS
sk.delex.parser.udpipe	85.72%	83.33%
cs.delex.parser.udpipe	71.55%	65.17%
cs.sup.parser.udpipe	74.72%	67.65%
sk.sup.parser.udpipe	86.81%	84.48%

Treebank data: Czech (cs_pud-ud-test.conllu)

Parser	UAS	LAS
cs.delex.parser.udpipe	84.15%	79.36%
sk.delex.parser.udpipe	73.49%	64.97%
cs.sup.parser.udpipe	85.48%	80.80%
sk.sup.parser.udpipe	72.46%	64.89%

Portugese Spanish experiments

I trained my own delex parser using the data from UD_Portuguese-GSD/pt_gsd-ud-train.conllu and used Portugese and Spanish test data to observe UAS and LAS scores.

Treebank data: Portugese (pt_gsd-ud-test.conllu)

Parser	UAS	LAS
pt.sup.parser.udpipe	71.59%	58.25%
pt_mine.delex.parser.udpipe	83.88%	80.67%

Treebank data: Spanish (es_gsd-ud-test.conllu)

Parser	UAS	LAS
pt.sup.parser.udpipe	75.38%	67.99%
pt_mine.delex.parser.udpipe	73.44%	62.18%

Hindi Urdu Bhojpuri experiments

I trained my own delex parser using the data from ud-treebanks-v2.5/UD_Hindi-HDTB/hi_hdtb-ud-train.conllu and used Hindi, Urdu and Bhojpuri test data to observe UAS and LAS scores. The results are surprising as both languages are supposed to be related.

Parser: hi_mine.delex.parser.udpipe

Data	UAS	LAS
hi_hdtb-ud-test.conllu	85.65%	79.15%
bho_bhtb-ud-test.conllu	59.72%	42.12%

Tree Projection

Czech-Hindi experiments

Alignment: Intersection

Model	Head	Dependency Labels
Baseline	0.04196567208023837	0.002769734357295732
Assigning dependency relation of source to target	0.04196567208023837	0.1744512988375509
Target head alignment	0.07675521423475597	0.174451298835509

Alignment: Union

Model	Head	Dependency Labels
Baseline	0.04196567208023837	0.002769734357295732
Assigning dependency relation of source to target	0.04196567208023837	0.2487305384195728
Target head alignment	0.11242603550295859	0.2487305384195728

Alignment: Diag

Model	Head	Dependency Labels
Baseline	0.04196567208023837	0.002769734357295732
Assigning dependency relation of source to target	0.04196567208023837	0.22711821729825002
Target head alignment	0.11414662805824835	0.22711821729825002

Alignment: Rev

Model	Head	Dependency Labels
Baseline	0.04196567208023837	0.002769734357295732
Assigning dependency relation of source to target	0.04196567208023837	0.18481681984136977
Target head alignment	0.09584959503126443	0.18481681984136977

Alignment: Fwd

Model	Head	Dependency Labels
Baseline	0.04196567208023837	0.002769734357295732
Assigning dependency relation of source to target	0.04196567208023837	0.2541021444458433
Target head alignment	0.0984514667002392	0.2541021444458433

Thus, the ‘fwd’ alignment gives the best accuracy scores for dependency labels and the ‘diag’ alignment gives the best score for head labels in the Czech-Hindi case.

Embeddings

Source: English, Target: Czech

Results:

Source test data

Data	UAS	LAS
en_esl-ud-test.conllu	87.46%	82.21%
en_ewt-ud-test.conllu	33.79%	27.28%
en_gum-ud-test.conllu	29.05%	23.41%
en_lines-ud-test.conllu	32.88%	25.34%
en_partut-ud-test.conllu	30.75%	23.33%
en_pronouns-ud-test.conllu	77.58%	69.91%
en_pud-ud-test.conllu	29.21%	23.37%

Target test data

Data	UAS	LAS
cs_cac-ud-test.conllu	29.84%	22.21%
cs_cltt-ud-test.conllu	25.01%	19.08%
cs_fictree-ud-train.conllu	36.80%	26.60%
cs_pdt-ud-test.conllu	28.83%	21.09%
cs_pud-ud-test.conllu	28.70%	21.46%

BERT

Monolingual setup

Model: English

Language	Accuracy
English	0.980784432958346
Czech	0.23512668824645608
Hindi	0.23512668824645608
Japanese	0.17073170731707318

Model: Czech

Language	Accuracy
English	0.26938279112192154
Czech	0.952561669829222
Hindi	0.24445910290237466
Japanese	0.1951219512195122

Model: Hindi

Language	Accuracy
English	0.12976588628762542
Czech	0.18746511887487444
Hindi	0.9365875109938434
Japanese	0.0975609756097561

Model: Japanese

Language	Accuracy
English	0.13986013986013987
Czech	0.24986047549949772
Hindi	0.24001759014951626
Japanese	0.9512195121951219

Multilingual setup

Model: English

Language	Accuracy
English	0.9869869261173609
Czech	0.7609666257394798
Hindi	0.6898416886543536
Japanese	0.6097560975609756

Model: Czech

Language	Accuracy
English	0.6981453329279417
Czech	0.9919075789708672
Hindi	0.6447669305189094
Japanese	0.4146341463414634

Model: Hindi

Language	Accuracy
English	0.5652173913043478
Czech	0.6135729434088626
Hindi	0.9811785400175902
Japanese	0.6097560975609756

Model: Japanese

Language	Accuracy
English	0.29759805411979323
Czech	0.3502064962607434
Hindi	0.43078276165347407
Japanese	0.975609756097561