Session Septembre 2023: Augmentation, saturation et arbres de décision pour la détection des catégories grammaticales en langues africaines

Elvis MBONING [1]

[1]NTeALan Research and Developpement
Makepe, Parcours vita / Douala - Cameroon

Sponsorisé par l'association NTeALan

Sommaire

Legends

Exigences du challenge

Il nous a été demandé de challenger les travaux du collectifs Masakhane sur la tâche de POS.

⇒ **Résultats de Masahkane**:

Model	bam	bbj	ewe	fon	hau	ibo	kin	lug	luo	mos	nya	pcm	sna	swa	tsn	twi	wol	xho	yor	zul	AVG
CRF	89.1	78.9	88.0	88.1	89.8	75.2	95.3	88.3	84.6	86.0	77.7	85.6	85.9	89.3	81.4	81.5	91.0	81.8	92.0	84.2	85.7
Massively-multilingual PLMs																					
mBERT (172M)	89.9	75.2	86.0	87.6	90.7	76.5	96.9	89.6	87.0	86.5	79.9	90.4	87.5	92.0	81.9	83.9	92.5	85.9	93.4	86.8	87.0
XLM-R-base (270M)	90.1	83.6	88.5	90.1	92.5	77.2	96.7	89.1	87.2	90.7	79.9	90.5	87.9	92.9	81.3	84.1	92.4	87.4	93.7	88.0	88.2
XLM-R-large (550M)	90.2	**85.4**	88.8	90.2	92.8	78.1	97.3	90.0	88.0	91.1	80.5	90.8	88.1	**93.2**	82.2	84.9	**92.9**	88.1	94.2	89.4	88.8
RemBERT (575M)	90.6	82.6	**88.9**	**90.8**	**93.0**	**79.3**	98.0	90.3	87.5	90.4	82.4	90.9	89.1	93.1	83.6	**86.0**	92.1	**89.3**	94.7	**90.2**	89.1
Africa-centric PLMs																					
AfroLM (270M)	89.2	77.8	87.5	82.4	92.7	77.8	97.4	90.8	86.8	89.6	81.1	89.5	88.7	92.8	**83.8**	83.9	92.1	87.5	91.1	88.8	87.6
AfriBERTa-large (126M)	89.4	79.6	87.4	88.4	**93.0**	**79.3**	97.8	89.8	86.5	89.9	79.7	89.8	87.8	93.0	82.5	83.7	91.7	86.1	94.5	86.9	87.8
AfroXLMR-base (270M)	90.2	83.5	88.5	90.1	**93.0**	79.1	98.2	90.9	86.9	90.9	82.7	90.8	89.2	92.9	82.7	84.3	92.4	88.5	94.5	89.4	88.9
AfroXLMR-large (550M)	**90.5**	85.3	88.7	90.4	**93.0**	78.9	**98.4**	**91.6**	**88.1**	**91.2**	**83.2**	**91.2**	**89.5**	**93.2**	83.0	84.9	**92.9**	88.7	**95.0**	90.1	**89.4**

Table 2: **Accuracy of baseline models on MasakhaPOS dataset** . We compare several multilingual PLMs including the ones trained on African languages. Average is over 5 runs.

Exigences du challenge

Il nous a été demandé de challenger les travaux du collectifs Masakhane sur la tâche de POS.

⇒ Résultats de Masahkane:

Model	bam	bbj	ewe	fon	hau	ibo	kin	lug	luo	mos	nya	pcm	sna	swa	tsn	twi	wol	xho	yor	zul	AVG
CRF	89.1	78.9	88.0	88.1	89.8	75.2	95.3	88.3	84.6	86.0	77.7	85.6	85.9	89.3	81.4	81.5	91.0	81.8	92.0	84.2	85.7
Massively-multilingual PLMs																					
mBERT (172M)	89.9	75.2	86.0	87.6	90.7	76.5	96.9	89.6	87.0	86.5	79.9	90.4	87.5	92.0	81.9	83.9	92.5	85.9	93.4	86.8	87.0
XLM-R-base (270M)	90.1	83.6	88.5	90.1	92.5	77.2	96.7	89.1	87.2	90.7	79.9	90.5	87.9	92.9	81.3	84.1	92.4	87.4	93.7	88.0	88.2
XLM-R-large (550M)	90.2	**85.4**	88.8	90.2	92.8	78.1	97.3	90.0	88.0	91.1	80.5	90.8	88.1	**93.2**	82.2	84.9	**92.9**	88.1	94.2	89.4	88.8
RemBERT (575M)	90.6	82.6	**88.9**	**90.8**	**93.0**	**79.3**	98.0	90.3	87.5	90.4	82.4	90.9	89.1	93.1	83.6	**86.0**	92.1	**89.3**	94.7	**90.2**	89.1
Africa-centric PLMs																					
AfroLM (270M)	89.2	77.8	87.5	82.4	92.7	77.8	97.4	90.8	86.8	89.6	81.1	89.5	88.7	92.8	**83.8**	83.9	92.1	87.5	91.1	88.8	87.6
AfriBERTa-large (126M)	89.4	79.6	87.4	88.4	**93.0**	**79.3**	97.8	89.8	86.5	89.9	79.7	89.8	87.8	93.0	82.5	83.7	91.7	86.1	94.5	86.9	87.8
AfroXLMR-base (270M)	90.2	83.5	88.5	90.1	**93.0**	79.1	98.2	90.9	86.9	90.9	82.7	90.8	89.2	92.9	82.7	84.3	92.4	88.5	94.5	89.4	88.9
AfroXLMR-large (550M)	**90.5**	85.3	88.7	90.4	**93.0**	78.9	**98.4**	**91.6**	**88.1**	**91.2**	**83.2**	**91.2**	**89.5**	**93.2**	83.0	84.9	**92.9**	88.7	**95.0**	90.1	**89.4**

Table 2: **Accuracy of baseline models on MasakhaPOS dataset** . We compare several multilingual PLMs including the ones trained on African languages. Average is over 5 runs.

⇒ Exigences du challenge

- notre proposition devrait respecter les contraintes écologiques
- liberté sur le choix d'approches et d'algorithmes

Legends

Analyses des fichiers train, dev et test (1)

Le corpus de données est composé de **18 langues**. Chaque langue est un dossier contenant **4 fichiers (train.txt, dev.txt et test.txt)**. Annoté avec le système d'UD.


```
Mwɔ̌’ NOUN
pfʉ́tə́ VERB
nə́ ADP
mwâsi NOUN
máp DET
yə́ DET
cwə NOUN
Cyəpɔ PROPN
Sǐ NOUN
kù’ VERB
```

Analyses des fichiers train, dev et test (2)

Je me suis rendu compte de plusieurs erreurs d'annotations (ex. train ggb):

- Mauvais marquage des ponctuations de fin
- Tags ambigues répétitives et orthographe non standard

Duplicates: 468

No	Variable	Stats / Values	Freqs / (% of Valid)
1	**sentence_id** [int64]	Mean (sd) : 364.4 (216.0) min < med < max: 1.0 < 363.0 < 751.0 IQR (CV) : 374.0 (1.7)	751 distinct values
2	**word** [object]	1. . 2. ná 3. á 4. bá 5. lə 6. pú 7. a 8. gaá 9. wá 10. é 11. other	699 (5.7%) 417 (3.4%) 307 (2.5%) 216 (1.7%) 188 (1.5%) 184 (1.5%) 174 (1.4%) 164 (1.3%) 156 (1.3%) 155 (1.3%) 9,687 (78.5%)
3	**tags** [object]	1. NOUN 2. VERB 3. PRON 4. DET 5. PUNCT 6. PART 7. PROPN 8. AUX 9. ADP	2,502 (20.3%) 2,045 (16.6%) 1,308 (10.6%) 956 (7.7%) 870 (7.0%) 811 (6.6%) 772 (6.3%) 694 (5.6%) 663 (5.4%)

Algorithme: augmentation des données d'entrée

J'avais constaté lors du précèdent challenge de la plus value de l'augmentation par position sur ce type de tâche de classification.

Nous avons repris et amélioré notre méthode d'augmentation, le principe est le suivant:

- Pour chaque corpus de train, de test et dev, constituer un dictionnaire de référence de chaque mot et leur position dans chaque phrase
- Pour chaque phrase, bouclez sur chaque mot
- récupérer la position du mot, aller chercher dans le dictionnaire la référence d'un autre mot ayant la même position et la même catégorie
- limiter l'augmentation à 5 phrases par mot

Exemple: Augmentation des données d'entrée

Voici un exemple:

```
Number of sentences to augment:  599
[('Bréndá', 'PROPN'), ('Biya', 'PROPN'), ('mú', 'NOUN'), ('yə', 'DET'), ('mjwĭ', 'NOUN'), ('Foguŋ', '
 [('Claude', 'PROPN'), ('Biya', 'PROPN'), ('mú', 'NOUN'), ('yə', 'DET'), ('mjwĭ', 'NOUN'), ('Foguŋ',
    --> generated sentences for word  : Bréndá
    --> number of sentences generated : 1
[('Bréndá', 'PROPN'), ('Biya', 'PROPN'), ('mú', 'NOUN'), ('yə', 'DET'), ('mjwĭ', 'NOUN'), ('Foguŋ', '
 [('Bréndá', 'PROPN'), ('Djamen', 'PROPN'), ('mú', 'NOUN'), ('yə', 'DET'), ('mjwĭ', 'NOUN'), ('Foguŋ'
    --> generated sentences for word  : Biya
    --> number of sentences generated : 2
[('Bréndá', 'PROPN'), ('Biya', 'PROPN'), ('mú', 'NOUN'), ('yə', 'DET'), ('mjwĭ', 'NOUN'), ('Foguŋ', '
 [('Bréndá', 'PROPN'), ('Biya', 'PROPN'), ('Ntwɔ̀ shɛ', 'NOUN'), ('yə', 'DET'), ('mjwĭ', 'NOUN'), ('Fo
    --> generated sentences for word  : mú
    --> number of sentences generated : 3
[('Bréndá', 'PROPN'), ('Biya', 'PROPN'), ('mú', 'NOUN'), ('yə', 'DET'), ('mjwĭ', 'NOUN'), ('Foguŋ', '
 [('Bréndá', 'PROPN'), ('Biya', 'PROPN'), ('mú', 'NOUN'), ('yɔ', 'DET'), ('mjwĭ', 'NOUN'), ('Foguŋ',
    --> generated sentences for word  : yə
    --> number of sentences generated : 4
[('Bréndá', 'PROPN'), ('Biya', 'PROPN'), ('mú', 'NOUN'), ('yə', 'DET'), ('mjwĭ', 'NOUN'), ('Foguŋ', '
 [('Bréndá', 'PROPN'), ('Biya', 'PROPN'), ('mú', 'NOUN'), ('yə', 'DET'), ('sɔnyə', 'NOUN'), ('Foguŋ',
    --> generated sentences for word  : mjwĭ
    --> number of sentences generated : 5
[('Bréndá', 'PROPN'), ('Biya', 'PROPN'), ('mú', 'NOUN'), ('yə', 'DET'), ('mjwĭ', 'NOUN'), ('Foguŋ', '
 [('Bréndá', 'PROPN'), ('Biya', 'PROPN'), ('mú', 'NOUN'), ('yə', 'DET'), ('mjwĭ', 'NOUN'), ('tátámcuŋ
    --> generated sentences for word  : Foguŋ
    --> number of sentences generated : 6
[('Bréndá', 'PROPN'), ('Biya', 'PROPN'), ('mú', 'NOUN'), ('yə', 'DET'), ('mjwĭ', 'NOUN'), ('Foguŋ', '
 [('Bréndá', 'PROPN'), ('Biya', 'PROPN'), ('mú', 'NOUN'), ('yə', 'DET'), ('mjwĭ', 'NOUN'), ('Foguŋ',
```

Caractérisation des données

Après plusieurs tests, nous avons choisi **36 features** de nos modèles à évaluer. Les features construits sur les mots+contextes étant les plus pertinents:

```
'word': word,
#'bias': 1.0,
'word.tones': tones if tones else "",
'word.normalized': unicodedata.normalize('NFKD', word),
'word.position': i,
'word.has_hyphen': int('-' in word),
'word.lower()': word.lower(),
'word.start_with_capital': int(word[0].isupper()) if i > 0 else -1,
'word.have_tone': 1 if len_tone>0 else 0,
'word.prefix': word[:2] if len(word)>2 else "",
'word.root': word[3:] if len(word)>2 else "",
'word.ispunctuation': int(word in string.punctuation),
'word.isdigit()': int(word.isdigit()),
'word.EOS': 1 if word in ['.','?','!'] else 0,
'word.BOS': 1 if i == 0 else 0,
'-1:word': sent[i-1][0] if i > 0 else "",
'-1:word.position': i-1 if i > 0 else -1,
'-1:word.tag': sent[i-1][1] if i > 0 else "",
#'-1:word.letters': word_decomposition(sent[i-1][0]) if i > 0 else -1,
'-1:word.normalized': unicodedata.normalize('NFKD', sent[i-1][0]) if i > 0 else "",
'-1:word.start_with_capital': int(sent[i-1][0][0].isupper()) if i > 0 else -1,
'-1:len(word-1)': len(sent[i-1][0]) if i > 0 else -1,
'-1:word.lower()': sent[i-1][0].lower() if i > 0 else "",
'-1:word.isdigit()': int(sent[i-1][0].isdigit()) if i > 0 else -1,
'-1:word.ispunctuation': int((sent[i-1][0] in string.punctuation)) if i > 0 else 0,
'-1:word.BOS': 1 if (i-1) == 0 else 0,
'-1:word.EOS': 1 if i > 0 and sent[i-1][0] in ['.','?','!'] else 0,
'+1:word': sent[i+1][0] if i < len(sent)-1 else "",
'+1:word.tag': sent[i+1][1] if i < len(sent)-1 else "",
'+1:word.position': i+1,
#'+1:word.letters': word_decomposition(sent[i+1][0]) if i < len(sent)-1 else -1,
'+1:word.normalized': unicodedata.normalize('NFKD', sent[i+1][0]) if i < len(sent)-1 else "",
'+1:word.start_with_capital': int(sent[i+1][0][0].isupper()) if i < len(sent)-1 else -1,
'+1:len(word+1)': len(sent[i+1][0]) if i < len(sent)-1 else -1,
'+1:word.lower()': sent[i+1][0].lower() if i < len(sent)-1 else "",
'+1:word.isdigit()': int(sent[i+1][0].isdigit()) if i < len(sent)-1 else -1,
'+1:word.ispunctuation': int((sent[i+1][0] in string.punctuation)) if i < len(sent)-1 else -1,
'+1:word.BOS': 1 if i < 0 else 0,
'+1:word.EOS': 1 if i < len(sent)-1 and sent[i+1][0] in ['.','?','!'] else 0
```

Legends

Par où commencer ?

Masahkane a utilisé dans son article des CRFs et les Transformers pour l'entraînement. Nous avons d'abord voulu vérifier, en comparant tous les classifiers classiques de sklearn, les algorithmes les plus performants pour cette tâche. Nous avons utilisé **lazypredict**.

Nous constatons que **les arbres de décisions** sont les plus performantes pour cette tâche (données non augmentées).

Model	Accuracy	Balanced Accuracy	ROC AUC	F1 Score	Time Taken
XGBClassifier	0.78	0.70	None	0.78	19.17
BaggingClassifier	0.73	0.68	None	0.73	1.56
RandomForestClassifier	0.75	0.65	None	0.74	3.54
ExtraTreesClassifier	0.74	0.65	None	0.74	1.78
DecisionTreeClassifier	0.67	0.64	None	0.67	0.37
ExtraTreeClassifier	0.53	0.46	None	0.53	0.04
LabelSpreading	0.52	0.44	None	0.52	18.52
LabelPropagation	0.52	0.44	None	0.52	8.56
KNeighborsClassifier	0.55	0.43	None	0.54	0.29
SVC	0.57	0.42	None	0.55	12.81
LogisticRegression	0.51	0.37	None	0.48	1.70
LinearDiscriminantAnalysis	0.50	0.37	None	0.47	0.28
CalibratedClassifierCV	0.50	0.36	None	0.47	57.36
LinearSVC	0.50	0.35	None	0.46	17.37
RidgeClassifier	0.49	0.33	None	0.44	0.07
RidgeClassifierCV	0.49	0.33	None	0.44	0.23
NearestCentroid	0.36	0.33	None	0.37	0.06
BernoulliNB	0.43	0.33	None	0.42	0.05
SGDClassifier	0.43	0.32	None	0.41	1.01
Perceptron	0.39	0.31	None	0.38	0.31
GaussianNB	0.37	0.30	None	0.34	0.04
PassiveAggressiveClassifier	0.35	0.29	None	0.34	0.44
LGBMClassifier	0.32	0.28	None	0.32	3.28
QuadraticDiscriminantAnalysis	0.19	0.21	None	0.21	0.16
AdaBoostClassifier	0.28	0.13	None	0.15	1.28
DummyClassifier	0.20	0.07	None	0.06	0.03

CRF et Xgboost ont été choisis

J'ai finalement utilisé pour mes entraînements, les **CRFs** et **XGboost**.

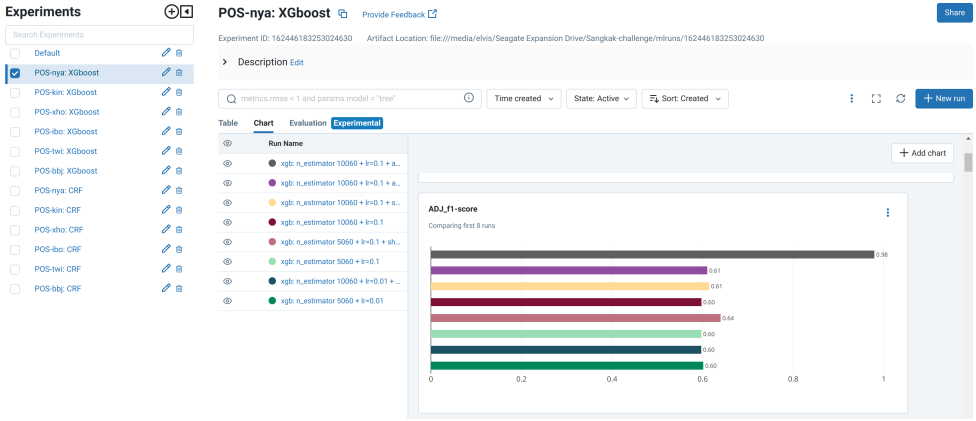
J'ai testé plusieurs types de configurations avec ces deux modèles en comparant à chaque fois, les méthodes avec augmentation et les autres sans augmentations de données

- xgb: n_estimator 10060 + lr=0.1 + augment + shuffle
- xgb: n_estimator 10060 + lr=0.1 + augment
- xgb: n_estimator 10060 + lr=0.1 + shuffle
- xgb: n_estimator 10060 + lr=0.1
- xgb: n_estimator 5060 + lr=0.1 + shuffle
- xgb: n_estimator 5060 + lr=0.1
- xgb: n_estimator 10060 + lr=0.01 + shuffle
- xgb: n_estimator 5060 + lr=0.01

- crf: iter 200 + augment
- crf: re-organised + iter 200 + shuffle + augment
- crf: re-organised + iter 200 + shuffle
- crf: re-organised + iter 200
- crf: iter 200
- crf: iter 100

Mes résultats de mes expérimentations

Mes expérimentations finales ont été réalisées avec l'outil **MLflow**.



Mes résultats de mes expérimentations

Mes résultats montrent qu'en associant **Xgboost** + augmentation + randonisation des données, on arrive quasiment à une score de classification à **0.98** de f1-score sur toutes les langues du corpus.

Legends

Discussions !

THANK YOU FOR YOUR KIND ATTENTION !