NLU in Other Languages

November 2020 - ML Conference

Burak IŞIKLI - Software Architect, Akbank

Burak IŞIKLI

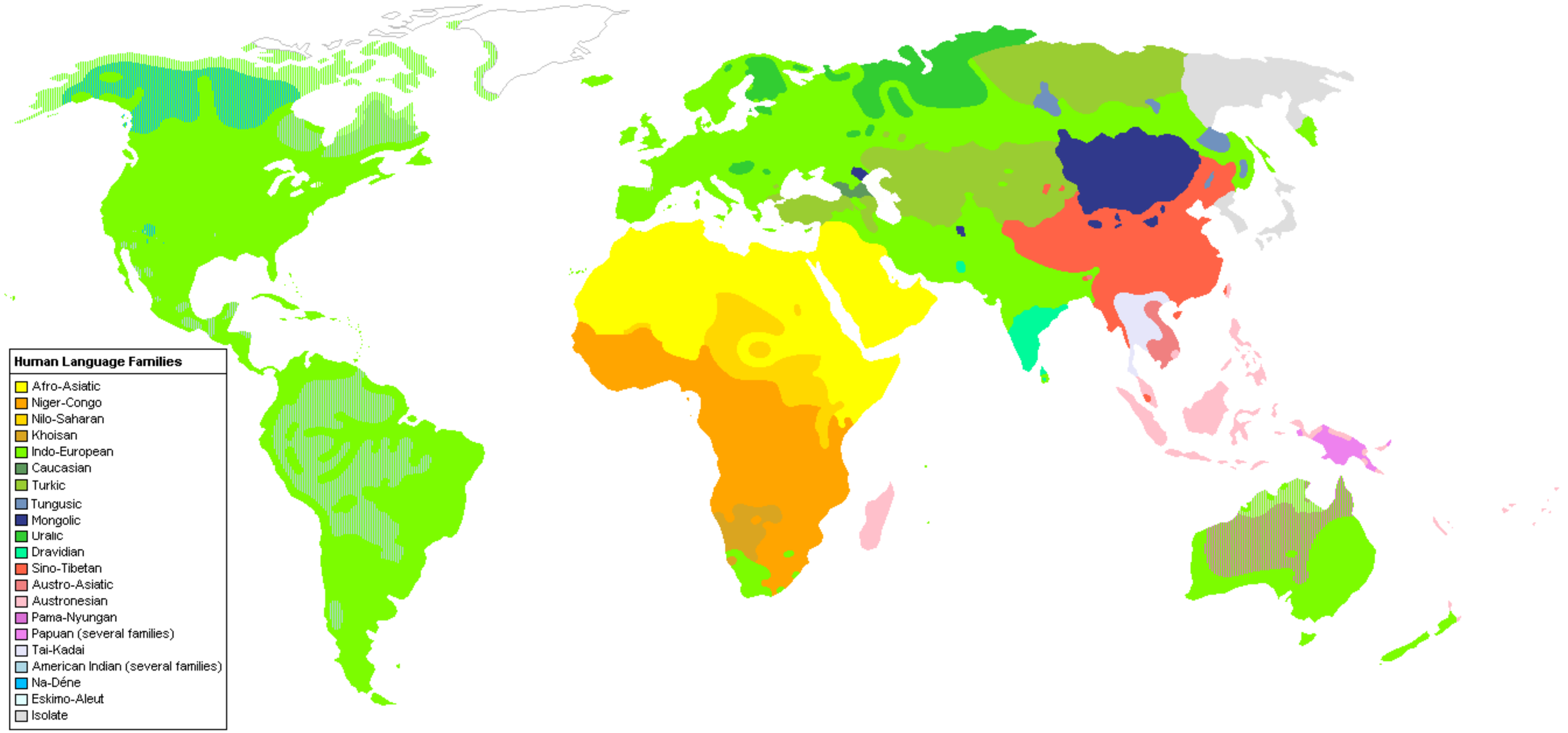
PhD Candidate (Computer Science) Özyeğin University
Software Architect, Akbank

burak.isikli at gmail dot com, akbank dot com dot tr
http://tr.linkedin.com/in/burakkk
@burakisikli
www.burakisikli.com

~7000 languages

Languages

Source: Wikipedia



Agglutinative Languages

Sources: Wikipedia&Duolingo

An agglutinative language is a type of synthetic language with morphology that primarily uses agglutination.

- Turkish
- Finnish
- Japanese
- Swahili
- Hungarian
- Some well known constructed languages such as Esperanto, Klingon, Quenya and Black Speech

Agglutinative Languages

Sources: Wikipedia&Duolingo

evlerden (ev-ler-den) (Turkish) = from the houses (English)

ni-na-soma(Swahili) = I am reading (English)

talo (Finnish) = a house (English)
talossa (Finnish) = in a house (English)
talossani (Finnish) = in my house (English)
taloissani (Finnish) = in my houses (English)
taloissanikin (Finnish) = in my houses, too (English)

https://bit.ly/36EPiqS

Agglutinative Languages

Turkish	English
kork(-mak)	(to) fear
korku	fear
korkusuz	fearless
korkusuzlaş (-mak)	(to) become fearless
korkusuzlaşmış	One who has become fearless
korkusuzlaştır(-mak)	(to) make one fearless
korkusuzlaştırıl(-mak)	(to) be made fearless
korkusuzlaştırılmış	One who has been made fearless
korkusuzlaştırılabil(-mek)	(to) be able to be made fearless
korkusuzlaştırılabilecek	One who will be able to be made fearless
korkusuzlaştırabileceklerimiz	Ones who we can make fearless
korkusuzlaştırabileceklerimizden	From the ones who we can make fearless
korkusuzlaştırabileceklerimizdenmiş	I gather that one is one of those we can make fearless
korkusuzlaştırabileceklerimizdenmişçesine	As if that one is one of those we can make fearless
korkusuzlaştırabileceklerimizdenmişçesineyken	when it seems like that one is one of those we can make fearless

Agglutinative Languages

Statistical Modeling of Agglutinative Languages, Dilek Hakkani-Tür, 2000

Language	Corpus size	Vocabulary size
Turkish	1M words	106,547
	10M words	417,775
English	1M words	33,398
	10M words	97,734

Table 3.2: Vocabulary sizes for two Turkish and English corpora.

Wikipedia Stats

Source: https://bit.ly/32KVBYU

Rank	*	**	Language	Articles	New articles (year)	New articles (month)	Growth (year)	Growth (month)	Share	New share (year)	New share (month)	Diff (year)	Diff (month)
1.	-	-	English	2,567,509	537,464	29,535	+26%	+1%	22.543%	19.450%	16.635%	-0.971	-0.094
2.	-	-	German	808,044	161,820	11,031	+25%	+1%	7.095%	5.856%	6.213%	-0.390	-0.014
3.	-	-	French	709,312	145,219	9,179	+26%	+1%	6.228%	5.255%	5.170%	-0.306	-0.017
4.	-	-	Polish	539,688	110,678	7,521	+26%	+1%	4.739%	4.005%	4.236%	-0.231	-0.008
5.	-	-	Japanese	523,639	104,732	7,159	+25%	+1%	4.598%	3.790%	4.032%	-0.255	-0.009
6.	+1	-	Italian	499,234	144,876	11,312	+41%	+2%	4.383%	5.243%	6.371%	+0.279	+0.031
7.	-1	-	Dutch	481,064	114,335	6,525	+31%	+1%	4.224%	4.138%	3.675%	-0.024	-0.009
8.	-	-	Portuguese	429,730	141,428	4,720	+49%	+1%	3.773%	5.118%	2.658%	+0.434	-0.018
9.	-	-	Spanish	402,430	119,165	8,319	+42%	+2%	3.533%	4.312%	4.685%	+0.252	+0.018
10.	+1	-	Russian	318,850	113,368	6,654	+55%	+2%	2.800%	4.103%	3.748%	+0.419	+0.015

Source: Grave et al

Learning Word Vectors for 157 Languages

Language	# tokens	# words
German	1,384,170,636	3,005,294
French	1,107,636,871	1,668,310
Japanese	998,774,138	916,262
Russian	823,849,081	2,230,231
Spanish	797,362,600	1,337,109
Italian	702,638,442	1,169,177
Polish	386,874,622	1,298,250
Portuguese	386,107,589	815,284
Chinese	374,650,371	1,486,735
Czech	178,516,890	784,896
Finnish	127,176,620	880,713
Hindi	39,733,591	183,211

Table 1: Comparison of the size of the Wikipedia corpora for selected languages. The second column indicates the number of words which appear at least five times in the corpus.

Language	# tokens	# words
Russian	102,825,040,945	14,679,750
Japanese	92,827,457,545	9,073,245
Spanish	72,493,240,645	10,614,696
French	68,358,270,953	12,488,607
German	65,648,657,780	19,767,924
Italian	36,237,951,419	10,404,913
Portuguese	35,841,247,814	8,370,569
Chinese	30,176,342,544	17,599,492
Polish	21,859,939,298	10,209,556
Czech	13,070,585,221	8,694,576
Finnish	6,059,887,126	9,782,381
Hindi	1,885,189,625	1,876,665

Table 3: Comparison accross languages of the size of the datasets obtained using the Common Crawl. The second column indicates the vocabulary size of the models trained on this data.

Tokenization

Learning Word Vectors for 157 Languages

- Chinese -> Stanford word segmenter
- Japanese -> Mecab
- Vietnamese -> UETsegmenter.
- Latin, Cyrillic, Hebrew or Greek -> tokenizer from Europarl preprocessing tools
- Remaining languages -> ICU tokenizer

- Byte pair encoding?

Byte Pair Encoding

Language-Independent Tokenisation Rivals Language-Specific Tokenisation for Word Similarity Prediction, Bollegala et al., LREC 2020

Composition	model	N	de	en	es	fa	it	ja	th	tr
weighted + PC removal	LST	50K	38.90	55.00	59.73	54.52	53.52	19.16	63.58	27.09
		100K	51.82	61.84	67.43	59.23	65.34	23.29	**64.69**	28.73
		1M	63.12	71.39	**75.41**	60.92	70.53	**30.98**	63.81	29.31
		10M	65.61	**71.49**	74.81	60.01	**70.85**	30.87	64.85	28.95
	LM	20K	53.79	57.29	57.83	59.84	54.30	25.68	62.49	37.18
		50K	62.45	66.69	65.45	62.72	63.05	28.97	61.92	38.74
		100K	64.56	67.58	71.38	64.20	65.63	29.39	59.33	36.68
		1M	**68.14**	68.23	74.35	64.26	70.16	22.29	38.47	32.68
	BPE	20K	53.03	52.56	56.36	56.86	55.66	23.89	52.04	37.18
		50K	60.17	64.28	64.24	63.19	62.76	21.93	55.92	**41.22**
		100K	62.60	65.17	68.75	**65.40**	65.76	21.80	53.66	28.51

Language-Specific tokenization (LST)
Language Modelling (LM)
Byte-Pair Encoding (BPE)

Spell Checker

Source: https://norvig.com/spell-correct.html

For English, simple solution: Peter Norvig's solution:

```
import re
from collections import Counter

def words(text): return re.findall(r'\w+', text.lower())

WORDS = Counter(words(open('big.txt').read()))

def P(word, N=sum(WORDS.values())):
    "Probability of `word`."
    return WORDS[word] / N

def correction(word):
    "Most probable spelling correction for word."
    return max(candidates(word), key=P)

def candidates(word):
    "Generate possible spelling corrections for word."
    return (known([word]) or known(edits1(word)) or known(edits2(word)) or [word])

def known(words):
    "The subset of `words` that appear in the dictionary of WORDS."
    return set(w for w in words if w in WORDS)

def edits1(word):
    "All edits that are one edit away from `word`."
    letters    = 'abcdefghijklmnopqrstuvwxyz'
    splits     = [(word[:i], word[i:])    for i in range(len(word) + 1)]
    deletes    = [L + R[1:]               for L, R in splits if R]
    transposes = [L + R[1] + R[0] + R[2:] for L, R in splits if len(R)>1]
    replaces   = [L + c + R[1:]           for L, R in splits if R for c in letters]
    inserts    = [L + c + R               for L, R in splits for c in letters]
    return set(deletes + transposes + replaces + inserts)

def edits2(word):
    "All edits that are two edits away from `word`."
    return (e2 for e1 in edits1(word) for e2 in edits1(e1))
```

Spell Checker

What about agglutinative languages?

- Impractical to collect this corpus in order to use edit distance
- Solutions?

Recall

Language	Corpus size	Vocabulary size
Turkish	1M words	106,547
	10M words	417,775
English	1M words	33,398
	10M words	97,734

Table 3.2: Vocabulary sizes for two Turkish and English corpora.

Spell Checker

Solution-1: Using morphological analysis

Algorithm:
Find the each word of the sentence
If the word has a stem, then it'll be most likely correct one
If it hasn't, then try to fix it using one of the algorithm(graph algorithm, transformer etc.)

Problems:
Morphological analysis/disambiguation component required
Too much depends on it
Code switching

Spell Checker

Solution-2: Neural Spell Checker

Approaches:
Using transformers (bert, albert, roberta etc.)
Sequence to sequence learning from scratch

Problems:
Wrong corrections
Longer inference time

Understanding

Text Classification

Sample Case: Intent Classification

Do you have enough labelled data?

Data Augmentation

Back-translation

Leverage machine translation to paraphrase a text

1. Take a sentence (e.g. Turkish)
2. Translate to another language (e.g. German)
3. Translate German sentence back into Turkish sentence.
4. Check whether the new sentence is different from our original one

Possible Problems:

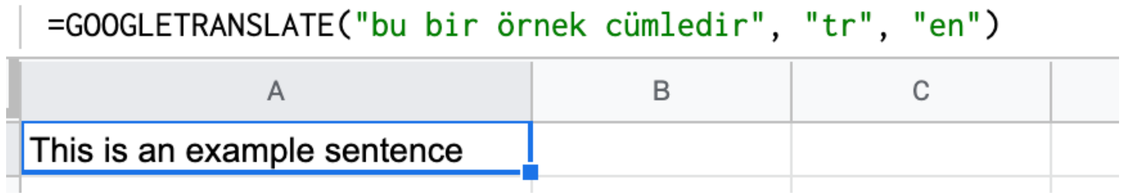
- Machine translation system required

Data Augmentation

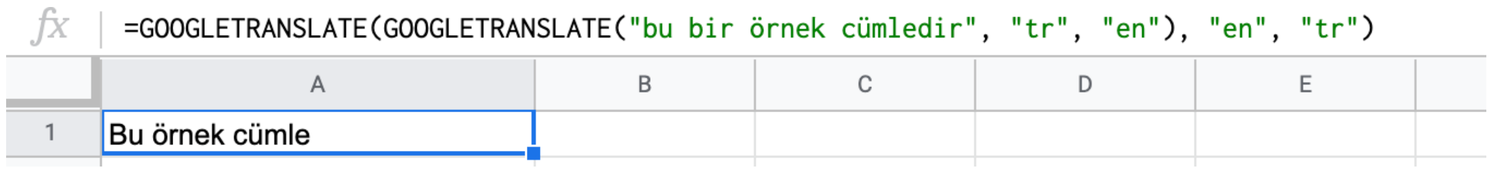
Back-translation

Google Spreadsheet for translation (GOOGLETRANSLATE())

=GOOGLETRANSLATE("bu bir örnek cümledir", "tr", "en")



=GOOGLETRANSLATE(GOOGLETRANSLATE("bu bir örnek cümledir", "tr", "en"), "en", "tr")



One suggestion

Data Augmentation

Back-translation

Open Source Translation Libraries

Thanks to Huggingface & University of Helsinki, there are lots of open source models:

https://huggingface.co/models?search=Helsinki-NLP

Source: https://bit.ly/2K7pnRb

Data Augmentation

Back-translation

MarianMT

```
def translate(texts, model, tokenizer, language="de"):
template = lambda text: f"{text}" if language == "en" else f">>{language}<< {text}"
src_texts = [template(text) for text in texts]

encoded = tokenizer.prepare_seq2seq_batch(src_texts)
translated = model.generate(**encoded)

translated_texts = tokenizer.batch_decode(translated, skip_special_tokens=True)
return translated_texts

def back_translate(texts, target_lang="de", source_lang="en"):
# Translate to target language
de_texts = translate(texts, target_model, target_tokenizer,
language=target_lang)

# Translate from target language back to source language
back_translated_texts = translate(de_texts, source_model, source_tokenizer,
language=source_lang)
return back_translated_texts
```

Data Augmentation

Source: https://bit.ly/2K7pnRb

Back-translation

MarianMT

```
en_texts = ["What's up", "Hi", "That's not bad"]

aug_texts = back_translate(en_texts, source_lang="en", target_lang="de")
print(aug_texts)
```

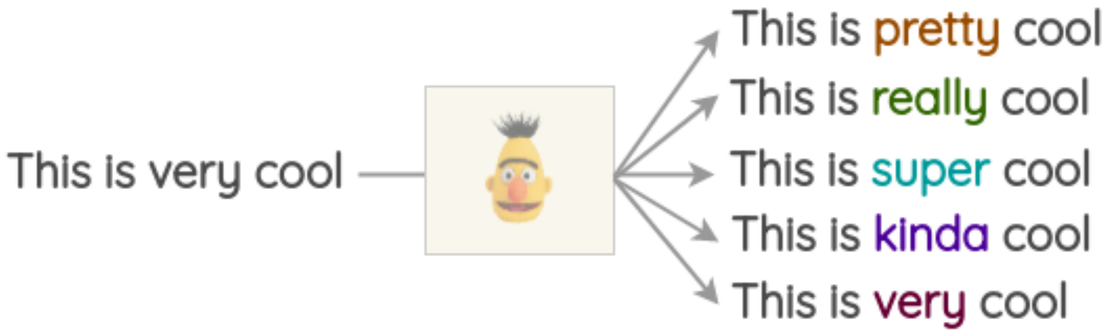
```
["What's going on?", 'Hello.', "That's not bad."]
```

Data Augmentation

Source: https://bit.ly/2K7pnRb

Masked Language Model

```
from transformers import pipeline
nlp = pipeline('fill-mask')
nlp('This is <mask> cool')
```




```
[{'score': 0.515411913394928, 'sequence': '<s> This is pretty cool</s>', 'token': 1256}, {'score': 0.1166248694062233, 'sequence': '<s> This is really cool</s>', 'token': 269}, {'score': 0.07387523353099823, 'sequence': '<s> This is super cool</s>', 'token': 2422}, {'score': 0.04272908344864845, 'sequence': '<s> This is kinda cool</s>', 'token': 24282}, {'score': 0.03471591323614120 5, 'sequence': '<s> This is very cool</s>', 'token': 182}]
```

Data Augmentation

Transformer

If you have small labelled data, and large amount of unlabeled data
Then fine tune model using this small data
Predict unlabeled data using fine tuned model

Confidence -> Low or High?

Data Augmentation

Source: https://bit.ly/2K7pnRb

Multilingual Transformers

How multilingual is Multilingual BERT?, Pires et.al., 2019

Very good cross lingually without being explicitly trained

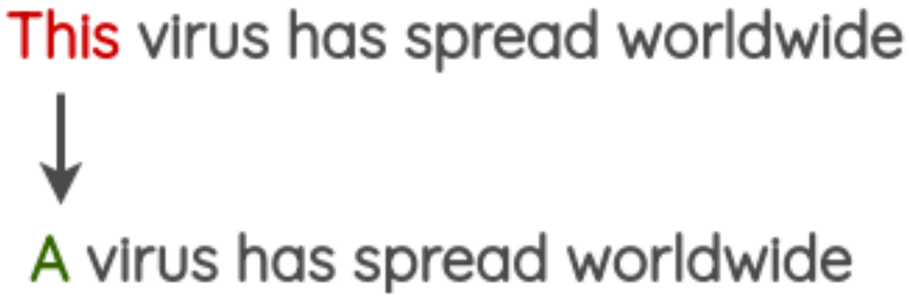
Data Augmentation

Source: https://bit.ly/2K7pnRb

TF-IDF based word replacement

Unsupervised Data Augmentation for Consistency Training, Xie et.al., 2020

Words that have low TF-IDF scores are uninformative
It can be replaced



Data Augmentation

EDA

EDA: Easy Data Augmentation Techniques for Boosting Performance on Text Classification Tasks, Wei et.al., 2020

Synonym Replacement (SR): Randomly replace n words in the sentences with their synonyms.
Random Insertion (RI): Insert random synonyms of words in a sentence, this is done n times.
Random Swap (RS): Two words in the sentences are randomly swapped, this is repeated n-times.
Random Deletion (RD): Random removal for each word in the sentence with a probability p.

Source: https://bit.ly/2K7pnRb

Data Augmentation

Word-Embeddings Substitution

```
import gensim.downloader as api
model = api.load('glove-twitter-25’)
model.most_similar('awesome', topn=5)
```

```
[('amazing', 0.9687871932983398), ('best', 0.9600659608840942), ('fun',
0.9331520795822144), ('fantastic', 0.931392431259153), ('perfect',
0.9243415594100952)]
```

OSCAR corpus: https://oscar-corpus.com/

Data Augmentation

Source: https://bit.ly/2K7pnRb

Thesaurus-based substitution

Data Augmentation

Word Embedding Alignment

Offline bilingual word vectors, orthogonal transformations and the inverted softmax, Smith et.al., 2017

https://github.com/babylonhealth/fastText_multilingual

89 languages

Data Augmentation

Word Embedding Alignment

Offline bilingual word vectors, orthogonal transformations and the inverted softmax, Smith et.al., 2017

```
from fasttext import FastVector
fr_dictionary = FastVector(vector_file='wiki.fr.vec’)
ru_dictionary = FastVector(vector_file='wiki.ru.vec')
fr_vector = fr_dictionary["chat"]
ru_vector = ru_dictionary["кот"]
print(FastVector.cosine_similarity(fr_vector, ru_vector)) # Result should be 0.02
```

Data Augmentation

Word Embedding Alignment

Offline bilingual word vectors, orthogonal transformations and the inverted softmax, Smith et.al., 2017

```
fr_dictionary.apply_transform('alignment_matrices/fr.txt')
ru_dictionary.apply_transform('alignment_matrices/ru.txt’)
print(FastVector.cosine_similarity(fr_dictionary["chat"], ru_dictionary["кот"])) # Result should be 0.43
```

Source: https://bit.ly/2UrEgjl

Data Augmentation

Zero-shot classification

```
!pip install transformers
from transformers import AutoTokenizer, AutoModel
from torch.nn import functional as F
tokenizer = AutoTokenizer.from_pretrained('deepset/sentence_bert')
model = AutoModel.from_pretrained('deepset/sentence_bert')

sentence = 'Who are you voting for in 2020?'
labels = ['business', 'art & culture', 'politics']

# run inputs through model and mean-pool over the sequence
# dimension to get sequence-level representations
inputs = tokenizer.batch_encode_plus([sentence] + labels,
                                     return_tensors='pt',
                                     pad_to_max_length=True)
input_ids = inputs['input_ids']
attention_mask = inputs['attention_mask']
output = model(input_ids, attention_mask=attention_mask)[0]
sentence_rep = output[:1].mean(dim=1)
label_reps = output[1:].mean(dim=1)

# now find the labels with the highest cosine similarities to
# the sentence
similarities = F.cosine_similarity(sentence_rep, label_reps)
closest = similarities.argsort(descending=True)
for ind in closest:
    print(f'label: {labels[ind]} \t similarity: {similarities[ind]}')
```

```
label: politics          similarity: 0.21561521291732788
label: business          similarity: 0.004524140153080225
label: art & culture     similarity: -0.027396833524107933
```

Data Augmentation

Data Augmentation using Pre-trained Transformer Models, Kumar et.al., 2020

Algorithm 1: Data Augmentation approach

Input : Training Dataset D_{train}
Pretrained model $G \in \{AE, AR, Seq2Seq\}$

1 Fine-tune G using D_{train} to obtain G_{tuned}
2 $D_{synthetic} \leftarrow \{\}$
3 **foreach** $\{x_i, y_i\} \in D_{train}$ **do**
4 Synthesize s examples $\{\hat{x}_i, \hat{y}_i\}_p^1$ using G_{tuned}
5 $D_{synthetic} \leftarrow D_{synthetic} \cup \{\hat{x}_i, \hat{y}_i\}_p^1$
6 **end**

- `prepend` : prepending label y_i to each sequence x_i in the training data without adding y_i to model vocabulary
- `expand` : prepending label y_i to each sequence x_i in the training data and adding y_i to model vocabulary.

Model	SST2 (1%)	SNIPS (1%)	TREC (1%)
No Aug	59.08 (5.59)	57.95 (10.74)	30.65 (8.29)
EDA	59.09 (5.69)	77.46 (7.60)	29.57 (10.55)
CBERT	59.85 (5.72)	80.55 (5.70)	29.96 (9.42)
$\text{BERT}_{\text{expand}}$	61.24 (4.42)	79.75 (5.74)	31.88 (10.03)
$\text{BERT}_{\text{prepend}}$	61.90 (6.78)	81.31 (5.25)	30.28 (8.50)
GPT2	58.62 (5.48)	68.25 (10.67)	26.24 (9.00)
$\text{GPT2}_{\text{context}}$	59.39 (6.61)	77.73 (7.56)	31.54 (10.21)
$\text{BART}_{\text{word}}$	62.35 (6.45)	79.98 (5.64)	**37.48** (10.82)
$\text{BART}_{\text{span}}$	**63.00** (6.64)	**81.68** (4.09)	37.25 (10.90)

Table 4: DA extrinsic evaluation in low-data regime. Results are reported as Mean (STD) accuracy on full test set. Experiments are repeated 15 times.

Other Approaches

1- Random Noise Injection
   a. Spelling error injection
   b. QWERTY Keyboard Error Injection
   c. Unigram Noising
   d. Blank Noising

2- Instance Crossover Augmentation

3- Syntax-tree Manipulation

4- MixUp for Text

5- A few shot Learning

6- Meta Learning

Language Specific Problems

Stemming Problem for Turkish

Telefon almak istiyorum = I want to buy a phone
Telefon almak istemek

Telefon almak istemiyorum = I don't want to buy a phone
Telefon almak istemek

Conclusion

There is no best approach for every language or problem, try and catch

This is not the only problem that requires data augmentation

Thank you!