VNLP

Release 0.1

Meliksah Turker

Jun 30, 2022

GET STARTED

VNLP is a Python library for developers working on Turkish NLP.

It implements the *state of the art* research papers in an efficient manner with an *intuitive* API to allow inference.

- Try the Demo.
- See the Official Website.
- See the Github page.
- See the Pypi page.

CHAPTER

ONE

CONTENTS

1.1 Quickstart

1.1.1 Installation

Installation is possible via pip.

```
$ pip install vngrs-nlp
```

1.1.2 Usage

VNLP provides 2 very intuitive and simple API options.

1.1.3 Python API

You can simply import, initialize and predict!

For example:

```
>>> from vnlp import NamedEntityRecognizer
>>> ner = NamedEntityRecognizer()
>>> ner.predict("Ben Melikşah, 29 yaşındayım, İstanbul'da ikamet ediyorum ve VNGRS AI␣
↪Takımı'nda çalışıyorum.")
[('Ben', 'O'),
('Melikşah', 'PER'),
(',', 'O'),
('29', 'O'),
('yaşındayım', 'O'),
(',', 'O'),
('İstanbul', 'LOC'),
("'", 'O'),
('da', 'O'),
('ikamet', 'O'),
('ediyorum', 'O'),
('ve', 'O'),
('VNGRS', 'ORG'),
('AI', 'ORG'),
('Takımı', 'ORG'),
("'", 'O'),
```

(continues on next page)

(continued from previous page)

```
('nda', 'O'),
('çalışıyorum', 'O'),
('.'), 'O']
```

- See Main Classes in *Contents* for the rest of the functionality.

1.1.4 Command Line API

Command Line API is even simpler than Python API!

The format is

```
$ vnlp --task TASK_NAME --text INPUT_TEXT
```

Example:

```
$ vnlp --task sentiment_analysis --text "Sipariş geldiğinde biz karnımızı␣
↪atıştırmalıklarla doyurmuştuk."
0
```

To list available tasks/functionality:

```
$ vnlp --list_tasks

# List of available tasks:
stemming_morph_analysis
named_entity_recognition
dependency_parsing
part_of_speech_tagging
sentiment_analysis
split_sentences
correct_typos
convert_numbers_to_words
deasciify
lower_case
remove_punctuations
remove_accent_marks
drop_stop_words
```

1.2 Dependency Parser

class vnlp.dependency_parser.dependency_parser.**DependencyParser**(*model='SPUContextDP'*, *evaluate=False*)

Main API class for Dependency Parser implementations.

Available models: [‘SPUContextDP’, ‘TreeStackDP’]

In order to evaluate, initialize the class with “evaluate = True” argument. This will load the model weights that are not trained on test sets.

predict(*sentence: str*, *displacy_format: bool = False*, *pos_result: Optional[List[Tuple[str, str]]] = None*) → List[Tuple[int, str, int, str]]

High level user API for Dependency Parsing.

Parameters

- **sentence** – Input sentence.
- **displacy_format** – When set True, returns the result in spacy.displacy format to allow visualization.
- **pos_result** – Part of Speech tags. To be used when displacy_format = True.

Returns

List of (token_index, token, arc, label).

Raises

ValueError – Sentence is too long. Try again by splitting it into smaller pieces.

Example:

```
from vnlp import DependencyParser
dependency_parser = DependencyParser()
dependency_parser.predict("Onun için yol arkadaşlarımızı titizlikle seçer,␣
↪kendilerini iyice sınarız.")

[(1, 'Onun', 6, 'obl'),
 (2, 'için', 1, 'case'),
 (3, 'yol', 4, 'nmod'),
 (4, 'arkadaşlarımızı', 6, 'obj'),
 (5, 'titizlikle', 6, 'obl'),
 (6, 'seçer', 10, 'parataxis'),
 (7, ',', 6, 'punct'),
 (8, 'kendilerini', 10, 'obj'),
 (9, 'iyice', 10, 'advmod'),
 (10, 'sınarız', 0, 'root'),
 (11, '.', 10, 'punct')]

# Visualization with Spacy:
import spacy
from vnlp import DependencyParser
dependency_parser = DependencyParser()
result = dependency_parser.predict(Oğuz'un kırmızı bir Astra'sı vardı.",␣
↪displacy_format = True)
spacy.displacy.render(result, style="dep", manual = True)
```

1.2.1 SentencePiece Unigram Context Dependency Parser

class vnlp.dependency_parser.spu_context_dp.**SPUContextDP**(*evaluate*)

SentencePiece Unigram Context Dependency Parser class.

- This is a context aware Deep GRU based Dependency Parser that uses SentencePiece Unigram tokenizer and pre-trained Word2Vec embeddings.
- It achieves 0.7117 LAS (Labeled Attachment Score) and 0.8370 UAS (Unlabeled Attachment Score) on all of test sets of Universal Dependencies 2.9.

- For more details about the training procedure, dataset and evaluation metrics, see ReadMe.

predict(*sentence: str*, *displacy_format: bool = False*, *pos_result: Optional[List[Tuple[str, str]]] = None*) → List[Tuple[int, str, int, str]]

Parameters

- **sentence** – Input sentence.
- **displacy_format** – When set True, returns the result in spacy.displacy format to allow visualization.
- **pos_result** – Part of Speech tags. To be used when displacy_format = True.

Returns
List of (token_index, token, arc, label).

Raises
ValueError – Sentence is too long. Try again by splitting it into smaller pieces.

1.2.2 Tree-stack Dependency Parser

class vnlp.dependency_parser.treestack_dp.**TreeStackDP**(*evaluate*)

Tree-stack Dependency Parser class.

- This dependency parser is *inspired* by Tree-stack LSTM in Transition Based Dependency Parsing.
- "Inspire" is emphasized because this implementation uses the approach of using Morphological Tags, Pre-trained word embeddings and POS tags as input for the model, rather than implementing the exact network proposed in the paper.
- It achieves 0.6914 LAS (Labeled Attachment Score) and 0.8048 UAS (Unlabeled Attachment Score) on all of test sets of Universal Dependencies 2.9.
- Input data is processed by NLTK.tokenize.TreebankWordTokenizer.
- For more details about the training procedure, dataset and evaluation metrics, see ReadMe.

predict(*sentence: str*, *displacy_format: bool = False*, **args*) → List[Tuple[int, str, int, str]]

Parameters

- **sentence** – Input sentence.
- **displacy_format** – When set True, returns the result in spacy.displacy format to allow visualization.

Returns
List of (token_index, token, arc, label).

Raises
ValueError – Sentence is too long. Try again by splitting it into smaller pieces.

1.3 Named Entity Recognizer

class vnlp.named_entity_recognizer.named_entity_recognizer.**NamedEntityRecognizer**(*model='SPUContextNER'*, *evaluate=False*)

Main API class for Named Entity Recognizer implementations.

Available models: [‘SPUContextNER’, ‘CharNER’]

In order to evaluate, initialize the class with “evaluate = True” argument. This will load the model weights that are not trained on test sets.

predict(*sentence: str*, *displacy_format: bool = False*) → List[Tuple[str, str]]

High level user API for Named Entity Recognition.

Parameters

- **sentence** – Input sentence/text.
- **displacy_format** – When set True, returns the result in spacy.displacy format to allow visualization.

Returns

NER result as pairs of (token, entity).

Example:

```
from vnlp import NamedEntityRecognizer
ner = NamedEntityRecognizer()
ner.predict("Benim adım Melikşah, 29 yaşındayım, İstanbul'da ikamet ediyorum ve␣
↪VNGRS AI Takımı'nda çalışıyorum.")

[('Benim', 'O'),
('adım', 'O'),
('Melikşah', 'PER'),
(',', 'O'),
('29', 'O'),
('yaşındayım', 'O'),
(',', 'O'),
("İstanbul'da", 'LOC'),
('ikamet', 'O'),
('ediyorum', 'O'),
('ve', 'O'),
('VNGRS', 'ORG'),
('AI', 'ORG'),
("Takımı'nda", 'ORG'),
('çalışıyorum', 'O'),
('.', 'O')]

# Visualization with Spacy:
import spacy
from vnlp import NamedEntityRecognizer
ner = NamedEntityRecognizer()
result = ner.predict("İstanbul'dan Foça'ya giderken Zeynep ile Bursa'ya uğradık.
↪", displacy_format = True)
spacy.displacy.render(result, style="ent", manual = True)
```

1.3.1 SentencePiece Unigram Context Named Entity Recognizer

class vnlp.named_entity_recognizer.spu_context_ner.**SPUContextNER**(*evaluate*)

SentencePiece Unigram Context Named Entity Recognizer class.

- This is a context aware Deep GRU based Named Entity Recognizer that uses SentencePiece Unigram tokenizer and pre-trained Word2Vec embeddings.
- It achieves 0.9928 Accuracy and 0.9833 F1 score on test sets of "wikiann", "gungor.ner" and "teghub-TurkishNER-BERT" datasets.
- It achieves 0.9766 F1 score for "ORG", 0.9852 F1 score for "PER" and 0.9742 F1 score for "LOC" entities. (Treating entity of interest as positive, all others as negative class.)
- For more details about the training procedure, dataset and evaluation metrics, see ReadMe.

predict(*sentence: str*, *displacy_format: bool = False*) → List[Tuple[int, str, int, str]]

Parameters

- **sentence** – Input sentence/text.
- **displacy_format** – When set True, returns the result in spacy.displacy format to allow visualization.

Returns

NER result as pairs of (token, entity).

1.3.2 CharNER

class vnlp.named_entity_recognizer.charner.**CharNER**(*evaluate*)

CharNER Named Entity Recognizer.

- This is an implementation of CharNER: Character-Level Named Entity Recognition.
- There are slight modifications to the original paper:
- This version is trained for Turkish language only.
- This version uses simple Mode operation among the character predictions of each token, instead of Viterbi Decoder
- It achieves 0.9589 Accuracy and 0.9200 F1_macro_score.
- Input data is processed by NLTK.tokenize.WordPunctTokenizer so that each punctuation becomes a new token.
- Entity labels are: ['O', 'PER', 'LOC', 'ORG']
- For more details about the training procedure, dataset and evaluation metrics, see ReadMe.

predict(*text: str*, *displacy_format: bool = False*) → List[Tuple[str, str]]

Parameters

- **text** – Input text.
- **displacy_format** – When set True, returns the result in spacy.displacy format to allow visualization.

Returns

NER result as pairs of (token, entity).

1.4 Normalizer

class vnlp.normalizer.normalizer.**Normalizer**

Normalizer class

- It contains the following functions to process and normalize text:
  - Spelling/Typo correction
  - Deasciification
  - Convert numbers to word form
  - Lower case
  - Punctuation Remover
  - Remove accent marks
- For more details about the algorithms and datasets, see Readme.

convert_numbers_to_words(*tokens: List[str]*, *num_dec_digits: int = 6*, *decimal_seperator: str = ','*) → List[str]

Converts numbers to word form.

Parameters

- **tokens** – List of input tokens.
- **num_dec_digits** – Number of precision (decimal points) for floats.
- **decimal_seperator** – Decimal seperator character. Can be either "." or ",".

Returns

List of converted tokens

Raises

ValueError – Given 'decimal seperator' is not a valid decimal seperator value. Use either "." or ",".

Example:

```
from vnlp import Normalizer
normalizer = Normalizer()
normalizer.convert_numbers_to_words("sabah 3 yumurta yedim ve tartıldığımda 1,
↪15 kilogram aldığımı gördüm".split())

['sabah',
'üç',
'yumurta',
'yedim',
've',
'tartıldığımda',
'bir',
'virgül',
'on',
'beş',
'kilogram',
'aldığımı',
'gördüm']
```

correct_typos(*tokens: List[str]*) → List[str]

Detects and corrects spelling mistakes and typos.

This implementation uses StemmerAnalyzer and Hunspell to detect typos. Detected typos are corrected by Hunspell algorithm using "tdd-hunspell-tr-1.1.0" dict.

Parameters
tokens – List of input tokens.

Returns
List of corrected tokens.

Example:

```
from vnlp import Normalizer
normalizer = Normalizer()
normalizer.correct_typos("Kasıtlı yazişm hatasıı ekliyoruum".split())

["Kasıtlı", "yazım", "hatası", "ekliyorum"]
```

static deasciify(*tokens: List[str]*) → List[str]

Deasciifies the given text for Turkish.

This function uses Emre Sevinç's implementation.

Parameters
tokens – List of input tokens.

Returns
List of deasciified tokens.

Example:

```
from vnlp import Normalizer
Normalizer.deasciify("dusunuyorum da boyle sey gormedim duymadim".split())

["düşünüyorum", "da", "böyle", "şey", "görmedim", "duymadım"]
```

static lower_case(*text: str*) → str

Converts a string of text to lowercase for Turkish language.

This is needed because Python does not properly handle all Turkish characters, e.g., "İ" -> "i".

Parameters
text – Input text.

Returns
Text in lowercase form.

Example:

```
from vnlp import Normalizer
Normalizer.lower_case("Test karakterleri: İIĞÜÖŞÇ")

'test karakterleri: ıığüöşç'
```

static remove_accent_marks(*text: str*) → str

Removes accent marks from the given string.

Parameters
text – Input text.

Returns
Text stripped from accent marks.

Example:

```
from vnlp import Normalizer
Normalizer.remove_accent_marks("merhâbâ")

'merhaba'
```

static remove_punctuations(*text: str*) → str

Removes punctuations from the given string.

Parameters
text – Input text.

Returns
Text stripped from punctuations.

Example:

```
from vnlp import Normalizer
Normalizer.remove_punctuations("merhaba,.!")

'merhaba'
```

1.5 Part of Speech Tagger

class vnlp.part_of_speech_tagger.part_of_speech_tagger.**PoSTagger**(*model='SPUContextPoS', evaluate=False, *args*)

Main API class for Part of Speech Tagger implementations.

Available models: [‘SPUContextPoS’, ‘TreeStackPoS’]

In order to evaluate, initialize the class with “evaluate = True” argument. This will load the model weights that are not trained on test sets.

predict(*sentence: str*) → List[Tuple[str, str]]

High level user API for Part of Speech Tagging.

Parameters
sentence – Input text(sentence).

Returns
List of (token, pos_label).

Example:

```
from vnlp import PoSTagger
pos = PoSTagger()
pos.predict("Vapurla Beşiktaş'a geçip yürüyerek Maçka Parkı'na ulaştım.")

[("Oğuz'un", 'PROPN'),
```

(continues on next page)

(continued from previous page)

```
('kırmızı', 'ADJ'),
('bir', 'DET'),
("Astra'sı", 'PROPN'),
('vardı', 'VERB'),
('.', 'PUNCT')]
```

1.5.1 SentencePiece Unigram Context Part of Speech Tagger

class vnlp.part_of_speech_tagger.spu_context_pos.**SPUContextPoS**(*evaluate*)

SentencePiece Unigram Context Part of Speech Tagger class.

- This is a context aware Deep GRU based Part of Speech Tagger that uses SentencePiece Unigram tokenizer and pre-trained Word2Vec embeddings.
- It achieves 0.9010 Accuracy and 0.7623 F1 macro score on all of test sets of Universal Dependencies 2.9.
- For more details about the training procedure, dataset and evaluation metrics, see ReadMe.

predict(*sentence: str*) → List[Tuple[str, str]]

Parameters
sentence – Input text(sentence).

Returns
List of (token, pos_label).

1.5.2 Tree-stack Part of Speech Tagger

class vnlp.part_of_speech_tagger.treestack_pos.**TreeStackPoS**(*evaluate*, *stemmer_analyzer=None*)

Tree-stack Part of Speech Tagger class.

- This Part of Speech Tagger is *inspired* by Tree-stack LSTM in Transition Based Dependency Parsing.
- "Inspire" is emphasized because this implementation uses the approach of using Morphological Tags, Pre-trained word embeddings and POS tags as input for the model, rather than implementing the exact network proposed in the paper.
- It achieves 0.89 Accuracy and 0.71 F1_macro_score on test sets of Universal Dependencies 2.9.
- Input data is processed by NLTK.tokenize.TreebankWordTokenizer.
- For more details about the training procedure, dataset and evaluation metrics, see ReadMe.

predict(*sentence: str*) → List[Tuple[str, str]]

Parameters
sentence – Input text(sentence).

Returns
List of (token, pos_label).

1.6 Sentence Splitter

class vnlp.sentence_splitter.sentence_splitter.**SentenceSplitter**

This is a rule based sentence splitter adapted from Philipp Koehn and Josh Schroeder's project.

The code is reduced and simplifed for Turkish language.

Abbreviations lexicon is expanded.

split_sentences(*text: str*) → List[str]

Given a string of sentences, returns list of strings, where each string in the list is a sentence.

Parameters
text – Input sentences.

Returns
List of splitted sentences.

Example:

```
from vnlp import SentenceSplitter
sentence_splitter = SentenceSplitter()
sentence_splitter.split_sentences('Av. Meryem Beşer, 3.5 yıldır süren dava ile␣
↪ilgili dedi ki, "Duruşma bitti, dava lehimize sonuçlandı." Bu harika bir␣
↪haber.')

['Av. Meryem beşer, 3.5 yıldır süren dava ile ilgili dedi ki, "Duruşma bitti,␣
↪dava lehimize sonuçlandı."',
'Bu harika bir haber.']
```

1.7 Sentiment Analyzer

class vnlp.sentiment_analyzer.sentiment_analyzer.**SentimentAnalyzer**(*model='SPUCBiGRUSentimentAnalyzer', evaluate=False*)

Main API class for Sentiment Analyzer implementations.

Available models: ['SPUCBiGRUSentimentAnalyzer']

In order to evaluate, initialize the class with "evaluate = True" argument. This will load the model weights that are not trained on test sets.

predict(*text: str*) → int

High level user API for discrete Sentiment Analysis prediction.

1: Positive sentiment.

0: Negative sentiment.

Parameters
text – Input text.

Returns
Sentiment label of input text.

Example:

```
from vnlp import SentimentAnalyzer
sentiment_analyzer = SentimentAnalyzer()
sentiment_analyzer.predict("Sipariş geldiğinde biz karnımızı çoktan␣
↪atıştırmalıklarla doyurmuştuk.")

0
```

predict_proba(*text: str*) → float

High level user API for probability estimation of Sentiment Analysis.

Parameters
text – Input text.

Returns
Probability that the input text has positive sentiment.

Example:

```
from vnlp import SentimentAnalyzer
sentiment_analyzer = SentimentAnalyzer()
sentiment_analyzer.predict_proba("Sipariş geldiğinde biz karnımızı çoktan␣
↪atıştırmalıklarla doyurmuştuk.")

0.08
```

1.7.1 SentencePiece Unigram Context BiGRU Sentiment Analyzer

class vnlp.sentiment_analyzer.spu_context_bigru_sentiment.**SPUCBiGRUSentimentAnalyzer**(*evaluate*)

SentencePiece Unigram Context Bidirectional GRU Sentiment Analyzer class.

- This is a Bidirectional GRU based Sentiment Analyzer that uses SentencePiece Unigram tokenizer and pre-trained Word2Vec embeddings.
- It achieves 0.9469 Accuracy, 0.9380 F1 macro score and 0.9147 F1 score (treating class 0 as minority).
- For more details about the training procedure, dataset and evaluation metrics, see ReadMe.

predict(*text: str*) → List[Tuple[str, str]]

Parameters
text – Input text.

Returns
Sentiment label of input text.

predict_proba(*text: str*) → float

Parameters
text – Input text.

Returns
Probability that the input text has positive sentiment.

1.8 Stemmer: Morphological Analyzer & Disambiguator

class vnlp.stemmer_morph_analyzer.stemmer_morph_analyzer.**StemmerAnalyzer**(*evaluate=False*)

StemmerAnalyzer Class.

This is a Morphological Disambiguator.

- This is an implementation of The Role of Context in Neural Morphological Disambiguation.
- There are slight modifications to the original paper:
- This version uses GRU instead of LSTM, which decreases the number of parameters by 25% with no actual performance penalty.
- This version has an extra Dense layer before the output(p) layer.
- During training, the positions of candidates and labels are shuffled in every batch.
- It achieves 0.9596 accuracy on ambigious tokens and 0.9745 accuracy on all tokens on trmorph2006 dataset, compared to 0.910 and 0.964 in the original paper.
- For more details about the implementation, training procedure and evaluation metrics, see ReadMe.

predict(*sentence: str*) → List[str]

High level user API for Morphological Disambiguation.

Parameters
sentence – Input text(sentence).

Returns
List of selected stem and morphological tags for each token.

Example:

```
from vnlp import StemmerAnalyzer
stemmer = StemmerAnalyzer()
stemmer.predict("Üniversite sınavlarına canla başla çalışıyorlardı.")

['üniversite+Noun+A3sg+Pnon+Nom',
'sınav+Noun+A3pl+P3sg+Dat',
'can+Noun+A3sg+Pnon+Ins',
'baş+Noun+A3sg+Pnon+Ins',
'çalış+Verb+Pos+Prog1+A3pl+Past',
'.+Punc']
```

1.9 Stopword Remover

class vnlp.stopword_remover.stopword_remover.**StopwordRemover**

Stopword Remover class.

Consists of Static and Dynamic stopword detection methods.

Static stopwords list is taken from Zemberek and some minor improvements are done.

- Dynamic stopword algorithm is implemented according to two papers.
- On Stopwords, Filtering and Data Sparsity for Sentiment Analysis of Twitter proposes to classify stopwords according to their frequency..

- Finding a "Kneedle" in a Haystack: Detecting Knee Points in System Behavior proposes to determine a cut-point automatically.

add_to_stop_words(*novel_stop_words: List[str]*)

Updates self.stop_words by adding given novel_stop_words to existing dictionary.

Parameters
novel_stop_words – Tokens to be added to existing stop_words dictionary.

Example:

```
from vnlp import StopwordRemover
stopword_remover = StopwordRemover()
stopword_remover.add_to_stop_words(['ama', 'aşı', 'gelip', 'eve'])
```

drop_stop_words(*list_of_tokens: List[str]*) → List[str]

Given list of tokens, drops stop words and returns list of remaining tokens.

Parameters
list_of_tokens – List of input tokens.

Returns
List of tokens stripped of stopwords

Example:

```
from vnlp import StopwordRemover
stopword_remover = StopwordRemover()
stopword_remover.drop_stop_words("acaba bugün kahvaltıda kahve yerine çay mı␣
↪içsem ya da neyse süt içeyim".split())

['bugün', 'kahvaltıda', 'kahve', 'çay', 'içsem', 'süt', 'içeyim']
```

dynamically_detect_stop_words(*list_of_tokens: List[str]*, *rare_words_freq: int = 0*) → List[str]

Dynamically detects stop words and returns them as list of tokens.

Use a large corpus with at least hundreds of unique tokens for a reasonable result.

Parameters

- **list_of_tokens** – List of input tokens
- **rare_words_freq** – Maximum frequency of words when deciding rarity. Default value is 0 so it does not detect any rare words by default.

Returns
List of dynamically detected stop words.

Raises
ValueError – Number of unique tokens must be at least 3 for Dynamic Stop Word Detection.

Example:

```
from vnlp import StopwordRemover
stopword_remover = StopwordRemover()
stopword_remover.dynamically_detect_stop_words(""ben bugün gidip aşı olacağım␣
↪sonra da eve gelip telefon açacağım aşı nasıl etkiledi eve gelip anlatırım␣
↪aşı olmak bu dönemde çok ama ama ama ama çok önemli"".split())

['ama', 'aşı', 'gelip', 'eve']
```

1.10 Word Embeddings

- Word2Vec , FastText and SentencePiece Unigram Tokenizer and its associated Word2Vec Turkish word embeddings are trained on a corpora of 32 GBs.
- In terms Tokenization, there are two groups of word embeddings: NLTK.tokenize.TreebankWordTokenizer and SentencePiece Unigram Tokenizer.

1.10.1 SentencePiece Unigram Tokenizer and Word Embeddings

- Sentence Piece Unigram Tokenizer and its associated Word2Vec embeddings come in 2 sizes for each config and can be downloaded from the links below:
- **Medium Tokenizer and its Word2Vec Embeddings:**
  - Medium Tokenizer : vocabulary size: 32_000
  - Large Word2Vec embeddings : vector size: 256
  - Medium Word2Vec embeddings : vector size: 128
- **Small Tokenizer and its Word2Vec Embeddings:**
  - Small Tokenizer : vocabulary size: 16_000
  - Large Word2Vec embeddings : vector size: 256
  - Medium Word2Vec embeddings : vector size: 128
- Word2Vec and FastText embeddings are trained with gensim algorithm.
- Sentence Piece Unigram tokenizer is trained with SentencePiece algorithm.

1.10.2 TreebankWordTokenizer tokenized Word Embeddings

- TreebankWordTokenizer tokenized Word2Vec and FastText embeddings come in 3 sizes and can be downloaded from the links below:
- **Large:**
  - Word2Vec : vocabulary size: 128_000, vector size: 256
  - FastText : vocabulary size: 128_000, vector size: 256
- **Medium:**
  - Word2Vec : vocabulary size: 64_000, vector size: 128
  - FastText : vocabulary size: 64_000, vector size: 128
- **Small:**
  - Word2Vec: : vocabulary size: 32_000, vector size: 64
  - FastText : vocabulary size: 32_000, vector size: 64
- **Usage:**

```
>>> # Word2Vec
>>> from gensim.models import Word2Vec
>>>
```

(continues on next page)

(continued from previous page)

```
>>> model = Word2Vec.load('Word2Vec_large.model')
>>> model.wv.most_similar('gandalf', topn = 10)
[('saruman', 0.7291593551635742),
('thorin', 0.6473978161811829),
('aragorn', 0.6401687264442444),
('isengard', 0.6123237013816833),
('orklar', 0.59786057472229),
('gollum', 0.5905635952949524),
('baggins', 0.5837421417236328),
('frodo', 0.5819021463394165),
('belgarath', 0.5811135172843933),
('sauron', 0.5763844847679138)]
```

```
>>> # FastText
>>> from gensim.models import FastText
>>>
>>> model = FastText.load('FastText_large.model')
>>> model.wv.most_similar('yamaçlardan', topn = 10)
[('kayalardan', 0.8601457476615906),
('kayalıklardan', 0.8567330837249756),
('tepelerden', 0.8423191905021667),
('ormanlardan', 0.8362939357757568),
('dağlardan', 0.8140010833740234),
('amaçlardan', 0.810560405254364),
('bloklardan', 0.803180992603302),
('otlardan', 0.8026642203330994),
('kısımlardan', 0.7993910312652588),
('ağaçlardan', 0.7961613535881042)]
```

```
>>> # SentencePiece Unigram Tokenizer
>>> import sentencepiece as spm
>>> sp = spm.SentencePieceProcessor('SentencePiece_16k_Tokenizer.model')
>>> tokenizer.encode_as_pieces('bilemezlerken')
['bile', 'mez', 'lerken']
>>> tokenizer.encode_as_ids('bilemezlerken')
[180, 1200, 8167]
```

- For more details about corpora, preprocessing and training, see ReadMe.

PYTHON MODULE INDEX

V

INDEX

R

S

T

V