[Natural Language Processing (NLP)] (CheatSheet)

Text Preprocessing

- Lowercase text: `text = text.lower()`
- Remove punctuation: `import string text = text.translate(str.maketrans("", "", string.punctuation))`
- Remove digits: `text = ''.join(c for c in text if not c.isdigit())`
- Remove stopwords: `from nltk.corpus import stopwords stopwords_list = stopwords.words('english') text = ' '.join(word for word in text.split() if word not in stopwords_list)`
- Remove custom stopwords: `custom_stopwords = ['the', 'and', 'is'] text = ' '.join(word for word in text.split() if word not in custom_stopwords)`
- Remove short words: `text = ' '.join(word for word in text.split() if len(word) > 2)`
- Remove long words: `text = ' '.join(word for word in text.split() if len(word) < 15)`
- Replace specific words: `text = text.replace('old_word', 'new_word')`
- Remove HTML tags: `from bs4 import BeautifulSoup text = BeautifulSoup(text, 'html.parser').get_text()`
- Remove URLs: `import re text = re.sub(r'http\S+', '', text)`
- Remove email addresses: `import re text = re.sub(r'\S+@\S+', '', text)`
- Expand contractions: `import contractions text = contractions.fix(text)`
- Normalize Unicode characters: `import unicodedata text = unicodedata.normalize('NFKD', text)`
- Remove accented characters: `import unidecode text = unidecode.unidecode(text)`
- Remove extra whitespaces: `text = ' '.join(text.split())`

Tokenization

- Tokenize text into words (NLTK): `from nltk.tokenize import word_tokenize tokens = word_tokenize(text)`
- Tokenize text into words (spaCy): `import spacy nlp = spacy.load('en_core_web_sm') doc = nlp(text) tokens = [token.text for token in doc]`
- Tokenize text into sentences (NLTK): `from nltk.tokenize import sent_tokenize sentences = sent_tokenize(text)`

By: Waleed Mousa

- Tokenize text into sentences (spaCy): `import spacy nlp = spacy.load('en_core_web_sm') doc = nlp(text) sentences = [sent.text for sent in doc.sents]`
- Tokenize text into n-grams (NLTK): `from nltk.util import ngrams n = 2 # Change n to the desired n-gram size ngrams_list = list(ngrams(text.split(), n))`

Part-of-Speech Tagging

- POS tagging (NLTK): `from nltk import pos_tag tagged_tokens = pos_tag(tokens)`
- POS tagging (spaCy): `import spacy nlp = spacy.load('en_core_web_sm') doc = nlp(text) tagged_tokens = [(token.text, token.pos_) for token in doc]`
- Fine-grained POS tagging (spaCy): `import spacy nlp = spacy.load('en_core_web_sm') doc = nlp(text) tagged_tokens = [(token.text, token.tag_) for token in doc]`

Named Entity Recognition (NER)

- NER (NLTK): `from nltk import ne_chunk ne_chunks = ne_chunk(tagged_tokens)`
- NER (spaCy): `import spacy nlp = spacy.load('en_core_web_sm') doc = nlp(text) entities = [(ent.text, ent.label_) for ent in doc.ents]`
- NER with custom labels (spaCy): `import spacy from spacy.tokens import Span nlp = spacy.load('en_core_web_sm') doc = nlp(text) custom_entities = [Span(doc, start, end, label='CUSTOM') for start, end in custom_entity_offsets] doc.ents = list(doc.ents) + custom_entities`

Text Normalization

- Stemming (NLTK): `from nltk.stem import PorterStemmer stemmer = PorterStemmer() stemmed_tokens = [stemmer.stem(token) for token in tokens]`
- Lemmatization (NLTK): `from nltk.stem import WordNetLemmatizer lemmatizer = WordNetLemmatizer() lemmatized_tokens = [lemmatizer.lemmatize(token) for token in tokens]`
- Lemmatization with POS (NLTK): `from nltk.stem import WordNetLemmatizer lemmatizer = WordNetLemmatizer() lemmatized_tokens = [lemmatizer.lemmatize(token, pos='v') for token, pos in tagged_tokens]`
- Lemmatization (spaCy): `import spacy nlp = spacy.load('en_core_web_sm') doc = nlp(text) lemmatized_tokens = [token.lemma_ for token in doc]`

By: Waleed Mousa

- Lowercase and lemmatize (spaCy): `import spacy nlp = spacy.load('en_core_web_sm') doc = nlp(text.lower()) lemmatized_tokens = [token.lemma_ for token in doc]`
- Lowercase, lemmatize, and remove stopwords (spaCy): `import spacy nlp = spacy.load('en_core_web_sm') doc = nlp(text.lower()) lemmatized_tokens = [token.lemma_ for token in doc if not token.is_stop]`

Dependency Parsing

- Dependency parsing (spaCy): `import spacy nlp = spacy.load('en_core_web_sm') doc = nlp(text) for token in doc: print(token.text, token.dep_, token.head.text)`
- Extract subject-verb-object triples (spaCy): `import spacy nlp = spacy.load('en_core_web_sm') doc = nlp(text) for token in doc: if token.dep_ in ['nsubj', 'dobj']: print(token.text, token.dep_, token.head.text)`
- Visualize dependency tree (spaCy): `import spacy from spacy import displacy nlp = spacy.load('en_core_web_sm') doc = nlp(text) displacy.render(doc, style='dep', jupyter=True)`

Chunking

- Noun phrase chunking (NLTK): `from nltk import RegexpParser grammar = r'NP: {<DT>?<JJ>*<NN>+}' chunk_parser = RegexpParser(grammar) chunks = chunk_parser.parse(tagged_tokens)`
- Verb phrase chunking (NLTK): `from nltk import RegexpParser grammar = r'VP: {<VB.*><NP|PP|CLAUSE>+$}' chunk_parser = RegexpParser(grammar) chunks = chunk_parser.parse(tagged_tokens)`
- Custom chunking (NLTK): `from nltk import RegexpParser grammar = r'CUSTOM: {<JJ>+<NN>}' chunk_parser = RegexpParser(grammar) chunks = chunk_parser.parse(tagged_tokens)`

Word Embeddings

- Load pre-trained word embeddings (Word2Vec): `from gensim.models import KeyedVectors model = KeyedVectors.load_word2vec_format('path/to/embeddings.bin', binary=True)`
- Get word vector: `vector = model['word']`
- Find similar words: `similar_words = model.most_similar('word')`
- Find analogies: `analogies = model.most_similar(positive=['king', 'woman'], negative=['man'])`

By: Waleed Mousa

- Compute word similarity: `similarity = model.similarity('word1', 'word2')`
- Compute sentence similarity: `from scipy.spatial.distance import cosine sentence1_vector = np.mean([model[word] for word in sentence1.split()], axis=0) sentence2_vector = np.mean([model[word] for word in sentence2.split()], axis=0) similarity = 1 - cosine(sentence1_vector, sentence2_vector)`
- Train custom Word2Vec embeddings: `from gensim.models import Word2Vec sentences = [['word1', 'word2', 'word3'], ['word4', 'word5', 'word6']] model = Word2Vec(sentences, size=100, window=5, min_count=1, workers=4)`
- Load pre-trained GloVe embeddings: `from gensim.scripts.glove2word2vec import glove2word2vec glove_input_file = 'path/to/glove.txt' word2vec_output_file = 'path/to/glove.word2vec' glove2word2vec(glove_input_file, word2vec_output_file) model = KeyedVectors.load_word2vec_format(word2vec_output_file, binary=False)`
- Load pre-trained FastText embeddings: `from gensim.models.fasttext import load_facebook_vectors model = load_facebook_vectors('path/to/fasttext.bin')`
- Use spaCy's pre-trained word embeddings: `import spacy nlp = spacy.load('en_core_web_lg') doc = nlp(text) word_vectors = [token.vector for token in doc]`

Sentiment Analysis

- TextBlob sentiment analysis: `from textblob import TextBlob blob = TextBlob(text) sentiment = blob.sentiment.polarity`
- VADER sentiment analysis: `from nltk.sentiment.vader import SentimentIntensityAnalyzer analyzer = SentimentIntensityAnalyzer() sentiment = analyzer.polarity_scores(text)`
- Flair sentiment analysis: `from flair.models import TextClassifier from flair.data import Sentence classifier = TextClassifier.load('en-sentiment') sentence = Sentence(text) classifier.predict(sentence) sentiment = sentence.labels[0].value`
- Transformers sentiment analysis (BERT): `from transformers import pipeline classifier = pipeline('sentiment-analysis') sentiment = classifier(text)[0]['label']`

Text Classification

- Naive Bayes classifier (NLTK): `from nltk.classify import NaiveBayesClassifier train_data = [(text1, 'class1'), (text2, 'class2'),`

...] classifier = NaiveBayesClassifier.train(train_data) predicted_class = classifier.classify(text)

- Naive Bayes classifier (scikit-learn): from sklearn.feature_extraction.text import CountVectorizer from sklearn.naive_bayes import MultinomialNB vectorizer = CountVectorizer() X = vectorizer.fit_transform(texts) y = labels classifier = MultinomialNB() classifier.fit(X, y) predicted_class = classifier.predict(vectorizer.transform([text]))[0]
- Support Vector Machine (SVM) classifier (scikit-learn): from sklearn.feature_extraction.text import TfidfVectorizer from sklearn.svm import LinearSVC vectorizer = TfidfVectorizer() X = vectorizer.fit_transform(texts) y = labels classifier = LinearSVC() classifier.fit(X, y) predicted_class = classifier.predict(vectorizer.transform([text]))[0]
- Logistic Regression classifier (scikit-learn): from sklearn.feature_extraction.text import CountVectorizer from sklearn.linear_model import LogisticRegression vectorizer = CountVectorizer() X = vectorizer.fit_transform(texts) y = labels classifier = LogisticRegression() classifier.fit(X, y) predicted_class = classifier.predict(vectorizer.transform([text]))[0]
- Random Forest classifier (scikit-learn): from sklearn.feature_extraction.text import TfidfVectorizer from sklearn.ensemble import RandomForestClassifier vectorizer = TfidfVectorizer() X = vectorizer.fit_transform(texts) y = labels classifier = RandomForestClassifier() classifier.fit(X, y) predicted_class = classifier.predict(vectorizer.transform([text]))[0]
- FastText classifier: from fasttext import train_supervised train_data = "train.txt" model = train_supervised(input=train_data, lr=1.0, epoch=25, wordNgrams=2) predicted_class = model.predict(text)[0][0]
- BERT classifier (Transformers): from transformers import pipeline classifier = pipeline('text-classification', model='bert-base-uncased') predicted_class = classifier(text)[0]['label']

Topic Modeling

- Latent Dirichlet Allocation (LDA) (gensim): from gensim import corpora, models dictionary = corpora.Dictionary(texts) corpus = [dictionary.doc2bow(text) for text in texts] lda_model = models.LdaMulticore(corpus, num_topics=10, id2word=dictionary, passes=10)

By: Waleed Mousa

- Non-Negative Matrix Factorization (NMF) (scikit-learn): `from sklearn.feature_extraction.text import TfidfVectorizer from sklearn.decomposition import NMF vectorizer = TfidfVectorizer() X = vectorizer.fit_transform(texts) nmf_model = NMF(n_components=10, random_state=1) nmf_model.fit(X) topic_words = [vectorizer.get_feature_names()[i] for i in nmf_model.components_.argsort()[:, :10]]`
- Hierarchical Dirichlet Process (HDP) (gensim): `from gensim import corpora, models dictionary = corpora.Dictionary(texts) corpus = [dictionary.doc2bow(text) for text in texts] hdp_model = models.HdpModel(corpus, dictionary)`

Text Summarization

- TextRank summarization (gensim): `from gensim.summarization import summarize summary = summarize(text, ratio=0.2)`
- LexRank summarization (sumy): `from sumy.parsers.plaintext import PlaintextParser from sumy.nlp.tokenizers import Tokenizer from sumy.summarizers.lex_rank import LexRankSummarizer parser = PlaintextParser.from_string(text, Tokenizer('english')) summarizer = LexRankSummarizer() summary = summarizer(parser.document, sentences_count=3)`
- Luhn summarization (sumy): `from sumy.parsers.plaintext import PlaintextParser from sumy.nlp.tokenizers import Tokenizer from sumy.summarizers.luhn import LuhnSummarizer parser = PlaintextParser.from_string(text, Tokenizer('english')) summarizer = LuhnSummarizer() summary = summarizer(parser.document, sentences_count=3)`
- LSA summarization (sumy): `from sumy.parsers.plaintext import PlaintextParser from sumy.nlp.tokenizers import Tokenizer from sumy.summarizers.lsa import LsaSummarizer parser = PlaintextParser.from_string(text, Tokenizer('english')) summarizer = LsaSummarizer() summary = summarizer(parser.document, sentences_count=3)`
- BART summarization (Transformers): `from transformers import pipeline summarizer = pipeline("summarization", model="facebook/bart-large-cnn") summary = summarizer(text, max_length=100, min_length=30, do_sample=False)`
- T5 summarization (Transformers): `from transformers import pipeline summarizer = pipeline("summarization", model="t5-base", tokenizer="t5-base", framework="tf") summary = summarizer(text, max_length=100, min_length=30, do_sample=False)`

By: Waleed Mousa

Language Translation

- Google Translate API: `from googletrans import Translator translator = Translator() translated_text = translator.translate(text, dest='fr').text`
- Transformers translation (MarianMT): `from transformers import pipeline translator = pipeline("translation_en_to_fr", model="Helsinki-NLP/opus-mt-en-fr") translated_text = translator(text)[0]['translation_text']`
- Transformers translation (T5): `from transformers import pipeline translator = pipeline("translation_en_to_de", model="t5-base", tokenizer="t5-base", framework="tf") translated_text = translator(text)[0]['translation_text']`

Text Generation

- GPT-2 text generation: `from transformers import pipeline generator = pipeline('text-generation', model='gpt2') generated_text = generator(text, max_length=100, num_return_sequences=1)[0]['generated_text']`
- XLNet text generation: `from transformers import pipeline generator = pipeline('text-generation', model='xlnet-base-cased') generated_text = generator(text, max_length=100, num_return_sequences=1)[0]['generated_text']`
- CTRL text generation: `from transformers import pipeline generator = pipeline('text-generation', model='ctrl') generated_text = generator(text, max_length=100, num_return_sequences=1)[0]['generated_text']`

Coreference Resolution

- Neural coreference resolution (spaCy): `import spacy nlp = spacy.load('en_core_web_sm') doc = nlp(text) for cluster in doc._.coref_clusters: print(cluster.main, cluster.mentions)`
- Rule-based coreference resolution (neuralcoref): `import spacy import neuralcoref nlp = spacy.load('en_core_web_sm') neuralcoref.add_to_pipe(nlp) doc = nlp(text) for cluster in doc._.coref_clusters: print(cluster.main, cluster.mentions)`

Keyword Extraction

- TF-IDF keyword extraction (scikit-learn): `from sklearn.feature_extraction.text import TfidfVectorizer vectorizer = TfidfVectorizer() X = vectorizer.fit_transform(texts) feature_names = vectorizer.get_feature_names() keywords = [feature_names[i] for i in X.toarray()[0].argsort()[::-1][:5]]`
- RAKE keyword extraction (RAKE-NLTK): `from rake_nltk import Rake rake = Rake() rake.extract_keywords_from_text(text) keywords = rake.get_ranked_phrases()[:5]`
- TextRank keyword extraction (gensim): `from gensim.summarization import keywords keywords = keywords(text).split('\n')[:5]`
- YAKE keyword extraction (yake): `import yake kw_extractor = yake.KeywordExtractor() keywords = kw_extractor.extract_keywords(text)`

Named Entity Linking

- Wikifier entity linking: `import requests url = "https://www.wikifier.org/annotate-article" params = {"text": text, "lang": "en", "userKey": "YOUR_API_KEY"} response = requests.get(url, params=params) entities = response.json()`
- DBpedia Spotlight entity linking: `import requests url = "https://api.dbpedia-spotlight.org/en/annotate" headers = {"Accept": "application/json"} params = {"text": text} response = requests.get(url, headers=headers, params=params) entities = response.json()`

Text Similarity

- Cosine similarity (scikit-learn): `from sklearn.feature_extraction.text import TfidfVectorizer from sklearn.metrics.pairwise import cosine_similarity vectorizer = TfidfVectorizer() X = vectorizer.fit_transform(texts) similarity_matrix = cosine_similarity(X)`
- Jaccard similarity (scikit-learn): `from sklearn.feature_extraction.text import CountVectorizer from sklearn.metrics import jaccard_score vectorizer = CountVectorizer() X = vectorizer.fit_transform(texts) jaccard_scores = [jaccard_score(X[0], X[i]) for i in range(1, len(texts))]`
- Levenshtein distance (Python): `def levenshtein_distance(s1, s2): return sum(c1 != c2 for c1, c2 in zip(s1, s2)) + abs(len(s1) - len(s2)) distance = levenshtein_distance(text1, text2)`

By: Waleed Mousa

- Semantic similarity (spaCy): `import spacy nlp = spacy.load('en_core_web_lg') doc1 = nlp(text1) doc2 = nlp(text2) similarity = doc1.similarity(doc2)`

Sequence Labeling

- Part-of-Speech (POS) tagging (spaCy): `import spacy nlp = spacy.load('en_core_web_sm') doc = nlp(text) pos_tags = [(token.text, token.pos_) for token in doc]`
- Named Entity Recognition (NER) (spaCy): `import spacy nlp = spacy.load('en_core_web_sm') doc = nlp(text) entities = [(ent.text, ent.label_) for ent in doc.ents]`
- Chunking (spaCy): `import spacy nlp = spacy.load('en_core_web_sm') doc = nlp(text) chunks = [(chunk.text, chunk.label_) for chunk in doc.noun_chunks]`
- Semantic Role Labeling (SRL) (AllenNLP): `from allennlp.predictors.predictor import Predictor predictor = Predictor.from_path("https://storage.googleapis.com/allennlp-public-models/bert-base-srl-2020.03.24.tar.gz") srl = predictor.predict(sentence=text)`

Language Identification

- langdetect: `from langdetect import detect language = detect(text)`
- langid: `import langid language, confidence = langid.classify(text)`
- fastText language identification: `import fasttext model = fasttext.load_model('lid.176.bin') language = model.predict(text)[0][0][-2:]`

Text Preprocessing (Advanced)

- Spell correction (pyspellchecker): `from spellchecker import SpellChecker spell = SpellChecker() corrected_text = ' '.join([spell.correction(word) for word in text.split()])`
- Text normalization (unidecode): `from unidecode import unidecode normalized_text = unidecode(text)`
- Text standardization (ftfy): `from ftfy import fix_text standardized_text = fix_text(text)`
- Emoji handling (emoji): `import emoji text_without_emoji = emoji.get_emoji_regexp().sub(r'', text)`

By: Waleed Mousa

- Hashtag handling (regex): `import re text_without_hashtags = re.sub(r'#\w+', '', text)`
- Mention handling (regex): `import re text_without_mentions = re.sub(r'@\w+', '', text)`
- URL handling (urllib): `from urllib.parse import urlparse def is_url(text): try: result = urlparse(text) return all([result.scheme, result.netloc]) except: return False text_without_urls = ' '.join([word for word in text.split() if not is_url(word)])`