[Text Preprocessing and Feature Extraction] [cheatsheet]

1. Text Cleaning

- Remove punctuation: `text = re.sub(r'[^\w\s]', '', text)`
- Remove digits: `text = re.sub(r'\d', '', text)`
- Remove whitespace: `text = text.strip()`
- Remove multiple spaces: `text = re.sub(r'\s+', ' ', text)`
- Remove newlines: `text = text.replace('\n', '')`
- Remove tabs: `text = text.replace('\t', '')`
- Remove HTML tags: `text = re.sub('<.*?>', '', text)`
- Remove URLs: `text = re.sub(r'http\S+', '', text)`
- Remove email addresses: `text = re.sub(r'\S+@\S+', '', text)`
- Remove special characters: `text = re.sub(r'[^a-zA-Z0-9\s]', '', text)`

2. Text Normalization

- Convert to lowercase: `text = text.lower()`
- Convert to uppercase: `text = text.upper()`
- Convert to titlecase: `text = text.title()`
- Remove accented characters: `text = unicodedata.normalize('NFKD', text).encode('ASCII', 'ignore').decode('utf-8')`
- Convert to ASCII: `text = text.encode('ascii', 'ignore').decode('utf-8')`
- Convert to Unicode: `text = text.encode('utf-8').decode('utf-8')`
- Expand contractions: `text = contractions.fix(text)`
- Normalize whitespace: `text = ' '.join(text.split())`
- Normalize quotes: `text = re.sub(r'[''"""']', '"', text)`
- Normalize hyphens and dashes: `text = re.sub(r'[-----]', '-', text)`

3. Tokenization

- Split into words: `words = text.split()`
- Split into sentences: `sentences = nltk.sent_tokenize(text)`
- Tokenize using regular expressions: `tokens = re.findall(r'\w+', text)`
- Tokenize using NLTK word tokenizer: `tokens = nltk.word_tokenize(text)`
- Tokenize using spaCy: `doc = nlp(text); tokens = [token.text for token in doc]`

By: Waleed Mousa

- Tokenize using Keras: `tokenizer = Tokenizer(); tokenizer.fit_on_texts([text]); tokens = tokenizer.word_index`
- Tokenize using Gensim: `tokens = gensim.utils.simple_preprocess(text)`
- Tokenize using TextBlob: `blob = TextBlob(text); tokens = blob.words`
- Tokenize using Stanford CoreNLP: `props = {'annotators': 'tokenize,ssplit', 'pipelineLanguage': 'en', 'outputFormat': 'json'}; client = CoreNLPClient(properties=props, timeout=30000); ann = client.annotate(text); tokens = [token.word for sent in ann.sentence for token in sent.token]`

4. Stopword Removal

- Remove stopwords using NLTK: `stopwords = nltk.corpus.stopwords.words('english'); tokens = [word for word in tokens if word.lower() not in stopwords]`
- Remove stopwords using spaCy: `stopwords = spacy.lang.en.stop_words.STOP_WORDS; tokens = [token for token in tokens if token.lower() not in stopwords]`
- Remove stopwords using Gensim: `stopwords = gensim.parsing.preprocessing.STOPWORDS; tokens = [token for token in tokens if token.lower() not in stopwords]`
- Remove stopwords using a custom list: `stopwords = ['the', 'and', 'is']; tokens = [token for token in tokens if token.lower() not in stopwords]`

5. Stemming and Lemmatization

- Porter stemmer: `stemmer = PorterStemmer(); stems = [stemmer.stem(token) for token in tokens]`
- Snowball stemmer: `stemmer = SnowballStemmer('english'); stems = [stemmer.stem(token) for token in tokens]`
- Lancaster stemmer: `stemmer = LancasterStemmer(); stems = [stemmer.stem(token) for token in tokens]`
- WordNet lemmatizer: `lemmatizer = WordNetLemmatizer(); lemmas = [lemmatizer.lemmatize(token) for token in tokens]`
- spaCy lemmatizer: `lemmas = [token.lemma_ for token in doc]`
- TextBlob lemmatizer: `lemmas = [word.lemmatize() for word in blob.words]`
- Lemmatize using Stanford CoreNLP: `props = {'annotators': 'tokenize,ssplit,pos,lemma', 'pipelineLanguage': 'en', 'outputFormat': 'json'}; client = CoreNLPClient(properties=props, timeout=30000); ann = client.annotate(text); lemmas = [token.lemma for sent in ann.sentence for token in sent.token]`

By: Waleed Mousa

6. Part-of-Speech Tagging

- POS tagging using NLTK: `pos_tags = nltk.pos_tag(tokens)`
- POS tagging using spaCy: `pos_tags = [(token.text, token.pos_) for token in doc]`
- POS tagging using TextBlob: `pos_tags = blob.tags`
- POS tagging using Stanford CoreNLP: `props = {'annotators': 'tokenize,ssplit,pos', 'pipelineLanguage': 'en', 'outputFormat': 'json'}; client = CoreNLPClient(properties=props, timeout=30000); ann = client.annotate(text); pos_tags = [(token.word, token.pos) for sent in ann.sentence for token in sent.token]`

7. Named Entity Recognition

- NER using NLTK: `entities = nltk.chunk.ne_chunk(pos_tags)`
- NER using spaCy: `entities = [(ent.text, ent.label_) for ent in doc.ents]`
- NER using Stanford CoreNLP: `props = {'annotators': 'tokenize,ssplit,pos,ner', 'pipelineLanguage': 'en', 'outputFormat': 'json'}; client = CoreNLPClient(properties=props, timeout=30000); ann = client.annotate(text); entities = [(ent.entityMentionText, ent.entityType) for sent in ann.sentence for ent in sent.mentions]`

8. Dependency Parsing

- Dependency parsing using spaCy: `deps = [(token.text, token.dep_, token.head.text) for token in doc]`
- Dependency parsing using Stanford CoreNLP: `props = {'annotators': 'tokenize,ssplit,pos,depparse', 'pipelineLanguage': 'en', 'outputFormat': 'json'}; client = CoreNLPClient(properties=props, timeout=30000); ann = client.annotate(text); deps = [(token.word, token.dep, token.governor) for sent in ann.sentence for token in sent.token]`

9. Chunking

- Chunking using NLTK: `chunks = nltk.chunk.regexp.RegexpParser('NP: {<DT>?<JJ>*<NN>}').parse(pos_tags)`
- Chunking using spaCy: `chunks = [(chunk.text, chunk.label_) for chunk in doc.noun_chunks]`

10. Sentence Boundary Detection

By: Waleed Mousa

- Sentence boundary detection using NLTK: `sentences = nltk.sent_tokenize(text)`
- Sentence boundary detection using spaCy: `sentences = [sent.text for sent in doc.sents]`
- Sentence boundary detection using Stanford CoreNLP: `props = {'annotators': 'tokenize,ssplit', 'pipelineLanguage': 'en', 'outputFormat': 'json'}; client = CoreNLPClient(properties=props, timeout=30000); ann = client.annotate(text); sentences = [' '.join([token.word for token in sent.token]) for sent in ann.sentence]`

11. Coreference Resolution

- Coreference resolution using spaCy: `coref_clusters = [[(mention.start, mention.end) for mention in cluster] for cluster in doc._.coref_clusters]`
- Coreference resolution using Stanford CoreNLP: `props = {'annotators': 'tokenize,ssplit,pos,lemma,ner,parse,coref', 'pipelineLanguage': 'en', 'outputFormat': 'json'}; client = CoreNLPClient(properties=props, timeout=30000); ann = client.annotate(text); coref_chains = [[(mention.sentenceIndex, mention.headIndex, mention.startIndex, mention.endIndex) for mention in chain.mention] for chain in ann.corefChain]`

12. Semantic Role Labeling

- Semantic role labeling using spaCy: `srl = [(token.text, token._.srl) for token in doc]`
- Semantic role labeling using Stanford CoreNLP: `props = {'annotators': 'tokenize,ssplit,pos,lemma,ner,parse,depparse,coref,natlog,openie', 'pipelineLanguage': 'en', 'outputFormat': 'json'}; client = CoreNLPClient(properties=props, timeout=30000); ann = client.annotate(text); srl = [(srl.subject, srl.relation, srl.object) for sent in ann.sentence for srl in sent.openieTriple]`

13. Sentiment Analysis

- Sentiment analysis using NLTK: `sentiment = nltk.sentiment.vader.SentimentIntensityAnalyzer().polarity_scores(text)`
- Sentiment analysis using spaCy: `sentiment = doc._.sentiment`
- Sentiment analysis using TextBlob: `sentiment = blob.sentiment`
- Sentiment analysis using Stanford CoreNLP: `props = {'annotators': 'tokenize,ssplit,pos,parse,sentiment', 'pipelineLanguage': 'en', 'outputFormat': 'json'}; client = CoreNLPClient(properties=props,`

```
timeout=30000); ann = client.annotate(text); sentiment =
[(sent.sentiment, sent.sentimentValue) for sent in ann.sentence]
```

14. Topic Modeling

- LDA using Gensim: `lda_model = gensim.models.LdaMulticore(corpus, num_topics=10); topics = lda_model.print_topics()`
- NMF using scikit-learn: `nmf_model = NMF(n_components=10); topics = nmf_model.fit_transform(tfidf_matrix)`
- LSA using Gensim: `lsa_model = gensim.models.LsiModel(corpus, num_topics=10); topics = lsa_model.print_topics()`
- HDP using Gensim: `hdp_model = gensim.models.HdpModel(corpus); topics = hdp_model.print_topics()`

15. Text Similarity

- Cosine similarity using scikit-learn: `cosine_similarity(tfidf_matrix[0], tfidf_matrix[1])`
- Jaccard similarity using NLTK: `nltk.jaccard_distance(set(doc1), set(doc2))`
- Euclidean distance using scikit-learn: `euclidean_distances(tfidf_matrix[0], tfidf_matrix[1])`
- Manhattan distance using scikit-learn: `manhattan_distances(tfidf_matrix[0], tfidf_matrix[1])`
- Word Mover's Distance using Gensim: `model.wmdistance(doc1, doc2)`

16. Keyword Extraction

- TF-IDF using scikit-learn: `tfidf_vectorizer = TfidfVectorizer(); tfidf_matrix = tfidf_vectorizer.fit_transform(docs); keywords = tfidf_vectorizer.get_feature_names()`
- TextRank using Gensim: `keywords = gensim.summarization.keywords(text)`
- RAKE using NLTK: `rake_nltk_var.extract_keywords_from_text(text); keywords = rake_nltk_var.get_ranked_phrases()`
- YAKE using yake: `keywords = yake.KeywordExtractor().extract_keywords(text)`

17. Text Summarization

- TextRank using Gensim: `summary = gensim.summarization.summarize(text)`

By: Waleed Mousa

- LexRank using Sumy: `summarizer = LexRankSummarizer(); summary = summarizer(text, sentences_count=3)`
- LSA using Sumy: `summarizer = LsaSummarizer(); summary = summarizer(text, sentences_count=3)`
- KL-Sum using Sumy: `summarizer = KLSummarizer(); summary = summarizer(text, sentences_count=3)`

18. Readability Metrics

- Flesch Reading Ease using textstat: `textstat.flesch_reading_ease(text)`
- Flesch-Kincaid Grade Level using textstat: `textstat.flesch_kincaid_grade(text)`
- Gunning Fog Index using textstat: `textstat.gunning_fog(text)`
- SMOG Index using textstat: `textstat.smog_index(text)`
- Automated Readability Index using textstat: `textstat.automated_readability_index(text)`
- Coleman-Liau Index using textstat: `textstat.coleman_liau_index(text)`
- Linsear Write Formula using textstat: `textstat.linsear_write_formula(text)`
- Dale-Chall Readability Score using textstat: `textstat.dale_chall_readability_score(text)`

19. Text Vectorization

- Bag-of-Words using scikit-learn: `vectorizer = CountVectorizer(); bow_matrix = vectorizer.fit_transform(docs)`
- TF-IDF using scikit-learn: `tfidf_vectorizer = TfidfVectorizer(); tfidf_matrix = tfidf_vectorizer.fit_transform(docs)`
- Word2Vec using Gensim: `model = gensim.models.Word2Vec(sentences, vector_size=100, window=5, min_count=1)`
- GloVe using Gensim: `model = gensim.models.KeyedVectors.load_word2vec_format('glove.6B.100d.txt', binary=False)`
- FastText using Gensim: `model = gensim.models.FastText(sentences, vector_size=100, window=5, min_count=1)`
- Doc2Vec using Gensim: `model = gensim.models.doc2vec.Doc2Vec(documents, vector_size=100, window=5, min_count=1)`
- Sentence-BERT using sentence-transformers: `model = SentenceTransformer('bert-base-nli-mean-tokens'); embeddings = model.encode(sentences)`

By: Waleed Mousa

20. Language Detection

- Language detection using langdetect: `lang = langdetect.detect(text)`
- Language detection using spaCy: `lang = spacy.load('en_core_web_sm').vocab.lang`
- Language detection using Polyglot: `lang = Text(text).language.name`
- Language detection using fastText: `model = fasttext.load_model('lid.176.bin'); lang = model.predict(text)[0][0].split('__')[-1]`

21. Text Translation

- Text translation using googletrans: `translator = googletrans.Translator(); translated_text = translator.translate(text, dest='es').text`
- Text translation using TextBlob: `blob = TextBlob(text); translated_text = blob.translate(to='es')`
- Text translation using Google Cloud Translation API: `from google.cloud import translate_v2; client = translate_v2.Client(); translated_text = client.translate(text, target_language='es')['translatedText']`

22. Text Generation

- Text generation using GPT-2 with transformers: `from transformers import GPT2LMHeadModel, GPT2Tokenizer; model = GPT2LMHeadModel.from_pretrained('gpt2'); tokenizer = GPT2Tokenizer.from_pretrained('gpt2'); input_ids = tokenizer.encode(text, return_tensors='pt'); output = model.generate(input_ids, max_length=100, num_return_sequences=1); generated_text = tokenizer.decode(output[0], skip_special_tokens=True)`
- Text generation using LSTM with Keras: `model = Sequential(); model.add(LSTM(256, input_shape=(max_len, len(chars)))); model.add(Dense(len(chars), activation='softmax')); model.compile(loss='categorical_crossentropy', optimizer='adam'); model.fit(X, y, batch_size=128, epochs=10); generated_text = generate_text(model, tokenizer, 'The quick brown fox', max_len=100)`

23. Spelling Correction

- Spelling correction using TextBlob: `blob = TextBlob(text); corrected_text = blob.correct()`

By: Waleed Mousa

- Spelling correction using PySpellChecker: `from spellchecker import SpellChecker; spell = SpellChecker(); corrected_text = ' '.join([spell.correction(word) for word in text.split()])`
- Spelling correction using autocorrect: `from autocorrect import Speller; spell = Speller(lang='en'); corrected_text = ' '.join([spell(word) for word in text.split()])`

24. Text Preprocessing Pipelines

- NLTK preprocessing pipeline: `def preprocess(text): tokens = nltk.word_tokenize(text); tokens = [token.lower() for token in tokens]; tokens = [token for token in tokens if token not in stopwords.words('english')]; tokens = [stemmer.stem(token) for token in tokens]; return tokens`
- spaCy preprocessing pipeline: `def preprocess(text): doc = nlp(text); tokens = [token.lemma_.lower() for token in doc if not token.is_stop and not token.is_punct]; return tokens`
- Gensim preprocessing pipeline: `def preprocess(text): tokens = gensim.utils.simple_preprocess(text); tokens = [token for token in tokens if token not in gensim.parsing.preprocessing.STOPWORDS]; return tokens`

25. Text Data Augmentation

- Synonym replacement using NLTK: `def synonym_replacement(text): words = text.split(); for i, word in enumerate(words): synonyms = set([synset.lemma_names()[0] for synset in nltk.corpus.wordnet.synsets(word)]); if len(synonyms) > 0: words[i] = random.choice(list(synonyms)); return ' '.join(words)`
- Random insertion using NLTK: `def random_insertion(text): words = text.split(); for i in range(len(words)): if random.random() < 0.1: synonyms = set([synset.lemma_names()[0] for synset in nltk.corpus.wordnet.synsets(words[i])]); if len(synonyms) > 0: words.insert(i+1, random.choice(list(synonyms))); return ' '.join(words)`
- Random swap using NLTK: `def random_swap(text): words = text.split(); for i in range(len(words)-1): if random.random() < 0.1: words[i], words[i+1] = words[i+1], words[i]; return ' '.join(words)`
- Random deletion using NLTK: `def random_deletion(text): words = text.split(); for i, word in enumerate(words): if random.random() < 0.1: del words[i]; return ' '.join(words)`