#_ important NLTK Operations [+100]

Setting Up and Accessing Data:

- `import nltk`: Import the NLTK library.
- `nltk.download('all')`: Download all NLTK collections, including all packages, corpora, models, etc.
- `nltk.download('punkt')`: Download only the 'punkt' tokenizer models.

Reading and Tokenizing Text:

- `nltk.word_tokenize(text)`: Tokenize a string to split off punctuation other than periods.
- `nltk.sent_tokenize(text)`: Tokenize a text into a list of sentences.
- `nltk.tokenize.WhitespaceTokenizer()`: Tokenizer that splits text at whitespace.
- `nltk.tokenize.TreebankWordTokenizer()`: Tokenizer using regular expressions from the Penn Treebank.

Filtering Stop Words:

- `nltk.corpus.stopwords.words('english')`: Get a list of English stop words.
- `[word for word in word_list if word not in stopwords]`: Filter stop words from a list.

Stemming and Lemmatization:

- `nltk.PorterStemmer()`: Create a new Porter stemmer.
- `nltk.LancasterStemmer()`: Create a new Lancaster stemmer.
- `nltk.stem.SnowballStemmer('english')`: Create a stemmer for English language.
- `nltk.WordNetLemmatizer()`: Create a new WordNet Lemmatizer.
- `lemmatizer.lemmatize(word)`: Lemmatize a word using WordNet's built-in morphy function.

By: Waleed Mousa

POS Tagging:

- `nltk.pos_tag(tokens)`: Tag a list of tokens with part-of-speech markers.
- `nltk.corpus.brown.tagged_words()`: Access tagged words from the Brown corpus.
- `nltk.CorpusReader.tagged_words()`: Use any corpus reader to access tagged words.

Parsing and Chunking:

- `nltk.RegexpParser(grammar)`: Chunk parser based on regular expressions.
- `nltk.ne_chunk(tagged)`: Recognize named entities using a chunker.
- `nltk.Tree`: Class for representing a tree structure.

Named Entity Recognition:

- `nltk.chunk.ne_chunk(tagged)`: Chunk sentences into named entities.

Working with Corpora:

- `nltk.corpus.gutenberg.fileids()`: Get file IDs for the Gutenberg corpus.
- `nltk.corpus.gutenberg.words('fileid')`: Access words from a specific file in the Gutenberg corpus.
- `nltk.Text(nltk.corpus.gutenberg.words('fileid'))`: Create an NLTK text from a corpus.

Frequency Distributions:

- `nltk.FreqDist(tokens)`: Create a frequency distribution from a list of tokens.
- `freq_dist.most_common(n)`: List the n most common tokens.
- `freq_dist['word']`: Count the frequency of a given sample.

Concordance and Similarity:

By: Waleed Mousa

- `nltk.Text(tokens).concordance("word")`: Find all occurrences of the word.
- `nltk.Text(tokens).similar("word")`: Find words used in similar contexts.

Collocations and Bigrams:

- `nltk.bigrams(tokens)`: Create bigrams from tokens.
- `nltk.Text(tokens).collocations()`: Find collocations in the text.

Classification and Tagging:

- `nltk.NaiveBayesClassifier.train(training_set)`: Train a naive Bayes classifier.
- `classifier.classify(unseen_data)`: Classify unseen data after training.
- `nltk.DecisionTreeClassifier.train(training_set)`: Train a decision tree classifier.

WordNet Interface:

- `nltk.corpus.wordnet.synsets('word')`: Get synsets for a word.
- `synset.definition()`: Get the definition of the synset.
- `synset.examples()`: Get examples of the word in use.
- `synset.lemmas()`: Get the lemmas of the synset.

Custom Corpora:

- `nltk.corpus.PlaintextCorpusReader(corpus_root, '.*')`: Read a plain text corpus.
- `nltk.corpus.TaggedCorpusReader()`: Read a tagged text corpus.
- `nltk.corpus.XMLCorpusReader()`: Read an XML corpus.

Chunking and Parsing:

- `nltk.RegexpParser(chunk_pattern)`: Define a chunk parser using regular expressions.

By: Waleed Mousa

- `nltk.PCFG.fromstring(grammar)`: Define a probabilistic context-free grammar.
- `nltk.ChartParser(grammar)`: Create a chart parser with a given grammar.

Probability and Estimation:

- `nltk.probability.FreqDist(samples)`: Create a frequency distribution of given samples.
- `nltk.probability.LidstoneProbDist(freqdist, gamma)`: Create a Lidstone probability distribution estimator.

Custom Tokenizers:

- `nltk.tokenize.RegexpTokenizer(pattern)`: Tokenize a text into a sequence of alphabetic and non-alphabetic characters.
- `nltk.tokenize.PunktSentenceTokenizer(train_text)`: Train a tokenizer for splitting text into sentences.

n-grams:

- `nltk.ngrams(sequence, n)`: Return a sequence of n-grams from a sequence of items.

Metrics:

- `nltk.metrics.distance.edit_distance(s1, s2)`: Calculate the Levenshtein edit-distance between two strings.
- `nltk.metrics.scores.accuracy(reference, test)`: Compute the accuracy of the test results.

Text Categorization:

- `nltk.text.TextCollection(corpus)`: Create a collection of text documents.

Linguistic Fieldwork:

By: Waleed Mousa

- `nltk.field.fieldwork.Wordlist(filename)`: Access a wordlist for linguistic fieldwork.

Experimental Tasks:

- `nltk.align.Alignment(alignment)`: Deal with word alignments between two tokenized sentences.
- `nltk.align.bleu_score(reference, hypothesis)`: Compute the BLEU score for machine translation.

Graphical Representations:

- `nltk.draw.tree.TreeView(tree)`: Visualize a syntax tree.
- `nltk.draw.dispersion.dispersion_plot(text, words)`: Plot the dispersion of words in a text.

Semantic Interpretation:

- `nltk.sem.evaluate_logic(expression, bindings)`: Evaluate a logical expression against variable bindings.
- `nltk.sem.relextract.extract_rels(subjcls, objcls, doc, corpus='ace', pattern=None)`: Extract relationships between specified types of named entities.

Machine Learning:

- `nltk.classify.util.accuracy(classifier, gold)`: Determine the accuracy of a classifier against the gold standard.
- `nltk.cluster.KMeansClusterer(num_means, distance)`: Implement k-means clustering.
- `nltk.inference.resolution.ResolutionProver()`: A prover that uses resolution.

Text Processing Pipelines:

- `nltk.tokenize.punkt.PunktSentenceTokenizer()`: Unsupervised machine learning sentence tokenizer.
- `nltk.tokenize.punkt.PunktTrainer()`: Train a Punkt tokenizer.

By: Waleed Mousa

Lexical Analysis:

- `nltk.collocations.BigramAssocMeasures()`: Find bigram collocation measures.
- `nltk.collocations.BigramCollocationFinder.from_words(words)`: Find bigram collocations in a list of words.

Graphical Tools:

- `nltk.app.wordnet_app.app()`: Start the WordNet browser application.
- `nltk.app.chunkparser_app.app()`: Start the Chunk Parser application.

Data Collection:

- `nltk.corpus.reader.wordlist.WordListCorpusReader()`: A corpus reader for wordlists.
- `nltk.corpus.reader.plaintext.PlaintextCorpusReader()`: A corpus reader for plain text documents.

Using Models:

- `nltk.model.ngram.NgramModel(n, train, estimator=None)`: Build an n-gram language model.
- `nltk.model.ngram.lidstone(gamma)`: Apply Lidstone's Law of Succession to probability distribution.

Sentiment Analysis:

- `nltk.sentiment.util.demo_liu_hu_lexicon(sentence)`: Sentiment analysis using Liu and Hu opinion lexicon.
- `nltk.sentiment.vader.SentimentIntensityAnalyzer()`: Give a sentiment intensity score to sentences.

Speech Tagging and Classification:

- `nltk.tag.DefaultTagger(tag)`: Create a tagger that assigns the same tag to every token.

By: Waleed Mousa

- `nltk.tag.UnigramTagger(train)`: Train a unigram part-of-speech tagger.
- `nltk.tag.BigramTagger(train)`: Train a bigram part-of-speech tagger.

Utilities and String Operations:

- `nltk.util.string_span_tokenize(s, sep)`: Tokenize a string, given a separator.
- 80. `nltk.util.ngrams(sequence, n, pad_left=False, pad_right=False)`: Generate n-grams.
- `nltk.util.everygrams(sequence, min_len=1, max_len=-1)`: Generate all possible n-grams within a range of lengths from a sequence.
- `nltk.util.skipgrams(sequence, n, k)`: Generate skip-grams from a sequence.
- `nltk.util.bigrams(sequence)`: Generate bigrams from a sequence.
- `nltk.util.trigrams(sequence)`: Generate trigrams from a sequence.
- `nltk.util.pad_sequence(sequence, n, pad_left=False, pad_right=False, left_pad_symbol=None, right_pad_symbol=None)`: Pad a sequence for n-gram generation.

Text Transformation:

- `nltk.textwrap.shorten(text, width, **kwargs)`: Shorten a string of text.
- `nltk.tokenize.util.regexp_span_tokenize(text, pattern)`: Tokenize a string of text into spans.

Corpus Readers:

- `nltk.corpus.reader.chasen.ChasenCorpusReader()`: Read Chasen corpus file format.
- `nltk.corpus.reader.conll.ConllCorpusReader()`: Read CoNLL corpus file format.
- `nltk.corpus.reader.tagged.TaggedCorpusReader()`: Read and handle tagged corpora.

By: Waleed Mousa

Corpus Sampling and Shuffling:

- nltk.probability.probability_sampling(sample, size): Randomly sample from a probability distribution.
- nltk.probability.MLEProbDist(freqdist): Probability distribution with Maximum Likelihood Estimate.
- nltk.util.shuffle(list): Shuffle a list randomly.

Dialog and Chat:

- nltk.chat.util.Chat(pairs, reflections={}): Create a simple chatbot.
- nltk.chat.util.reflections: Default reflections for chatbots which can map first-person statements to second-person.

Corpus Statistics:

- nltk.probability.LaplaceProbDist(freqdist): Create a Laplace probability distribution, adding one smoothing.
- nltk.probability.ELEProbDist(freqdist): Expected Likelihood Estimate probability distribution.
- nltk.tag.brill.brill24(): Return the 24 template Brill tagger.

Morphological Analysis:

- nltk.stem.Cistem(): Stemmer for German and Austrian German.
- nltk.stem.isri.ISRIStemmer(): Stemmer for Arabic.
- nltk.stem.rslp.RSLPStemmer(): Stemmer for Portuguese.
- nltk.stem.snowball.SnowballStemmer(language): Stemmer for multiple languages using the Snowball algorithm.

Language Modeling:

- nltk.lm.preprocessing.padded_everygram_pipeline(order, text): Create a pipeline that pads a text and produces everygrams.
- nltk.lm.Vocabulary(text): Create a vocabulary from a given text.
- nltk.lm.MLE(order): A maximum likelihood estimate n-gram language model.

By: Waleed Mousa

- `nltk.lm.Laplace(order)`: A Laplace smoothed n-gram language model.

Textual Entailment:

- `nltk.inference.discourse.reading_discourse(discourses)`: Process textual entailment data.
- `nltk.inference.discourse.process_raw_discourse(raw_discourse)`: Process a raw discourse for textual entailment.

Semantic Similarity:

- `nltk.cluster.util.cosine_distance(u, v)`: Calculate the cosine distance between two vectors.
- `nltk.cluster.util.jaccard_distance(label1, label2)`: Calculate the Jaccard distance between two sets.
- `nltk.cluster.api.ClusterI`: Interface for clustering algorithms.

Experimental Learning:

- `nltk.induce.demo()`: Demonstrate induction logic.
- `nltk.align.ibm1.IBMModel1`: Class for training and using IBM Model 1 for machine translation.

Corpus Annotation:

- `nltk.metrics.agreement.AnnotationTask(data)`: Determine the level of agreement between different annotators.
- `nltk.metrics.agreement.binary_distance(label1, label2)`: Compute a simple binary distance between two labels.

Advanced Parsing:

- `nltk.parse.generate.generate_from_grammar(grammar, start_symbol=None)`: Generate sample sentences from a given grammar.
- `nltk.parse.pchart.ProbabilisticChartParser(grammar, **kwargs)`: Parse text with a probabilistic chart parser.

By: Waleed Mousa

- `nltk.parse.transitionparser.TransitionParser(model)`: Train or use a transition-based parser for parsing sentences.

Extending NLTK:

- `nltk.internals.config_java(options)`: Configure NLTK's Java interface.
- `nltk.misc.chomsky.chomsky_normal_form(grammar, factor="right", horzMarkov=None, vertMarkov=0, childChar="|", parentChar="^")`: Convert a CFG into Chomsky Normal Form.