CMPUT 497 Notes:

* Regular expressions are things used to extract specific strings from text
  + Algebraic notation for a characterizing a set of strings
* In NLPs, we can use math, logic-based, probabilistic and machine learning models.
* Text Normalization: Converting text into a standard form
  + Tokenization is the aim of separating word from text we aim to analyze
    - White space not always sufficient
    - Emoticons/hashtags might need to be included
  + Lemmatization: Determining I two words have same root despite differences (sang, sung)
    - Stemming is when we strip suffixes from end of word
  + We use sentence segmentation here (splitting texts into single sentences)
* Edit distance measures how close two strings are two each other based on number of edits to change one to other
* We use regular expressions to find patterns and lots of work to search through
  + Will return texts which match patterns
  + Think grep
* Patterns:
  + Concatenation is denoted as /x/ where x is the word/character/phrases you are looking for. Note this is case sensitive.
    - If we use square brackets inside it, we indicate a disjunction in which it will find matched for either option in the []. Such as /[Ww]ood/ Will return wood or Wood if found in text
  + TO use a range of letters and numbers, we use the – inside the [] such as /[1-9]/ will return matches which contain any of the numbers from 1-9
    - ^ has 2 uses
      * If we use ^ after the first [ in [], we search for matches which do not contain values in square bracket
      * Otherwise it mean the character
    - If we have a ?, it indicates that the character before may or may not be included
  + \* will allow us to search for zero or more occurrences of previous thing. To make it one or more, include a copy of the thing you are searching for before the asterix e.g. x\* 🡪 xx\* nut as this can get long, we use the + to do this
    - /./ will return any single character except carriage return. For example /.age/ can return page, rage wage etc. Often used with \* to mean any string of chars
      * To see if a word appears twice in a sentence we can do the following /word.\*word/
  + Anchors: Chars the hold regular expressions to particular places in a string
    - ^ and $ are common ones
    - ^ I start of a line while $ is end of a line
    - So we can combo the two in many ways such as /^Start and end\.$/ to return Start and end.
    - \b matches word boundary and \b matches non boundary e. /b\cat\b/ will only return cat and not cathartic but if we use/ b\cat\B/ it will return cathartic.
      * Very important
  + Note: To not use wildcards, we need to use a \
  + Disjunction is when we use the | operator to match options on either side of the |.
  + Precedence uses () in which we may need to find a disjunction in the middle of specific words (think plurals) so we can do the following to get fish or fishes /fis(h|hes)/
    - Can be useful for counters such as counting rows/columns, so we can do the following to get column counters /(Column [0-9]+ \*)\*/
  + There is an operator precedence
    - Table

      Description automatically generated with low confidence
  + Most regular expressions match largest possible sequence(greedy) but there are ways to enforce otherwise using the ? operator in conjunction with other operators such as \*? Or +? In which the former is a \* operator which matches as little text as possible
  + False positives and negatives can occur as a side effect of trying to increase precision or recall
  + We can use {n} to look for n occurs of previous char/expression
  + Aliases
    - Graphical user interface, text

      Description automatically generated

Graphical user interface, text, chat or text message

Description automatically generated

* + - Text

      Description automatically generated
* For python, we need to import the re module and use the groups method to get results of regex
* Using groups is important as they will capture different parts of the target sentence

Finite State Automata

* Recall: Language is a set of strings
* Automata is an abstract model in which it takes an input strings and returns a Boolean to see if it will be accepted. The model itself would define what strings are accepted
* Chart, sunburst chart

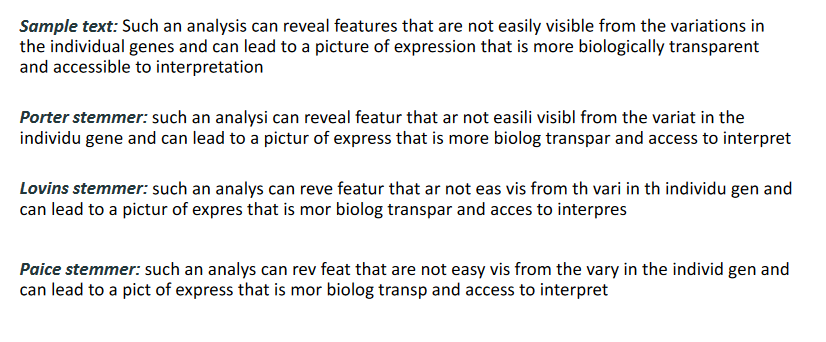
  Description automatically generated
* Focus on regular and context free automata
  + Regular deals with regex, finite-state, shallow parsing & named entity recognition
  + Context free on the other hand deals with push-down automata and usually handles most NLP stuff
* Natural languages in theory aren’t regular but can be treated as such however this varies from language to language (some have white space, some don’t)
* We can convert most regex’s to finite states
  + Sheep example:
  + Diagram

    Description automatically generated
* For a more formal definition of Finite State automata (FSA’s)
  + It is a tuple with 5 elements:
    - Q is a set of states (q0🡪qx) where qx is end state an q0 is start state
    - ∑ is the alphabets being used as symbols
    - Start state
    - F is the set of final states as there can be more than one
    - δ(q,i) is the transition between one state to another
* Aim with FSA is to see if we can see if string is accepted by machine, check if it is in the correct language and if a regex matches the intended string
* We follow the state automata and see if our string can also do the same. We accept the string if after we process the characters in the string, they are in an accepting state such that 
  + Regular language: Set of strings accepted by finite automata
* Note that most language is not deterministic
* A non deterministic FSA is one that presents different choices for the next state given our current state and input. An espsilon transition usually indicates a NFA
  + Every non deterministic FSA Can be converted to an FSA and vice versa.
* There are a few issues to non-deterministic FSAs
  + We could follow the wrong arc
    - To resolve this we can either make a back up of states depending on choice. We can also look ahead in our input. There is also the possibility of testing all possible paths in parallel or we can even make our state deterministic
  + Not possible to represent Natural language phenomena
* Given S1 and S2 are strings accepted FSAs F1 and F2, the following are also allowed
  + S1 U S2 = {x|x∈ S1 or x∈S2} 🡪 Union
  + S1 ᵒ S2 = {xy|x∈ S1 and y∈S2}🡪 Concatenation
  + F\*={x1🡪xk|k>=0 and each xi∈A}🡪Star
* Pumping Lemma is something regular languages MUST satisfy to be considered regular
  + If we have a finite num of strings, we get a regular language as we can make a non finite state for them and union them all.
  + A pump in this case is repeated things used by the loop multiple times
    - To understand this, assume a deterministic FSA has p states, if you input a string of length p, you are bound to repeat a state, hence the pump
  + Text, letter

    Description automatically generated

Text normalization

* As documents aren’t always straightforward, we need to normalize the data in such a way to make them straightforward
  + Different encodings and languages need to be handled
  + We need to pre-process text
* Co-reference resolution is when we aim to see if two words refer to the same things in the text
  + Using the above, we can re-order the clauses by extracting them and moving them to make parsing the sentence easier
* Use multiple regexs for an entire document
* Cant resolve all co-references in document if you split it up into sentences and look at them separately
* Therefore before we use NLP tools, we need to normalize the text. Most common three are the following
  + Tokenizing words
    - Terms used:
      * Word: A string of finite characters in the text
      * Term: A normalized word
      * Token is an instance of a word or term occurring in text.
    - Python allows for this with .split() which one can specify delimiter
    - You can also import NLTK and using that one can separate from whitespace and word boundaries
      * You can also use regexs as a parameter
    - An issue that can arise is that the text might transform (abbreviations such as we’re becoming we are) and compound words become a lot more difficult
      * We usually can specify multi-words in the tokenizer
    - Multiple ways to tokenize different words, numbers. Some languages are more difficult as they do not have whitespace and in some languages a word is a sequence of two words. Some words are compounds or phonetically challenging.
    - Trying to figure out where to split as sentence is hard as not all sentences end on a new line or period.
      * nltk has a built in sentence tokenizer to avoid this.
  + Normalization:
    - Done after we get all tokens from text
      * Can deal with accents and diacritics here
    - Case folding is when we make all the letters lower case
      * Do note that semantically this is wrong
    - Abbreviations can cause a few issues but there exists lexicons with common abbreviations (also called gazetteers) that can help expand abbreviations consistently.
    - Lemmatization is when we reduce words to their base form/dictionary head word (the lemma)
    - Stemming is when we remove the end of words to do something similar to lemmatization but is very language dependant.
    - Table

      Description automatically generated
    - Porter Alg is the most common to step words
      * Delete the stems if what remains is longer than one character
    - Comparison between 3 different stemmers 
  + String similarities + spelling
    - To check spelling, compare the tokens to the word found in dictionary often using the Levenshtein distance between the words to see what the min num of edits to change one word to the correct one
    - We need spell correction as we might need to correct user input or correct processed documents
    - To spell correct we can isolate the word and compare it to spelling checks (Context here is very important) or we can contextualize spelling checks to look at the surrounding words to correct context errors
      * + For isolation, we get a list of potential correct words and compare the edit distance using the Levenshtien distance or use the Jaccard coefficient between k-grams
        + K gram index is a mapping between k-gram and the words that contain them. We use these indexes to see which words has the most k-grams
        + Jaccard of A and B is defined as:



* + - * + Min edit distances is the min num of single character edits to change one word to another:

Operations can cost one but sometimes substitutions cost 2 if using Levenshtein

* + - * + Difficult to tell words being wrong in context

Can try the following:

One idea: hit-based spelling correction

Retrieve correct terms close to each of terms in text

Count hits of all possible correct phrases in a large text

Keep phrase with largest hit count

Try all possible pgrases and fix a single term at a time but can take too long and thus machine learning can help by using a model to see if the word works in the context

* Language specific definitions:
  + Graphical user interface, text, application, email

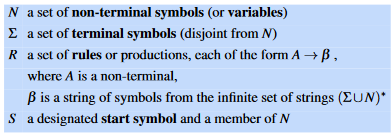
    Description automatically generated

Parts of Speech tagging

* 8 Parts of speech:
  + Noun
  + Verb
  + Adverb
  + Pronoun
  + Preposition
  + Conjunction
  + Participle
  + Article
* Proper name is often an entire multiword phrase
* Entity: Anything that can be referred to with a proper name
* Parts of Speech(PoS) and entities help us with sentence structure and meaning
  + For example knowing if a word is a noun/verb allows us to guess likely neighboring verbs & syntactic structure
  + Important when parsing
* PoS tagging is when we take a sequence of words and assign them a PoS
* Named Entity Recognition (NER) assigns words/phrases like person, location, organization etc.
* Sequence labelling is when we assign a words x\_i a label y\_i such that output Y has the same length of input X
  + Hidden Markov models and Conditional Random Field (generative and discriminative respectively)
* PoS definitions
  + Graphical user interface, text

    Description automatically generated
* PoS have 2 classes: Open and Closed.
  + Closed classes have fixed memberships (i.e. prepositions) whereas open classes allow for new ones to be added relatively easily
* Closed classes tend to include function words(of it, and you etc.) . Short and frequent structuring uses in grammar.
* Nouns
  + Proper nouns: Specific named entities or person
  + Common nouns(concrete terms) are split up into count nouns(singular and plural) and mass nouns(homogeneous group)
* Verbs: Action/process words
* Adjectives describe qualities and properties nouns. Not all languages have this
* Adverbs are words which modify something
  + Directions/locative adverbs (home, downhill ,here etc.) specify location/direction of action
  + Degree adverbs: Specify extent of verbs or property
  + Manner adverbs describe manner of actions/processes
  + Temporal adverbs describe time the actions took place
* Interjections: Includes greetings or question responses
* Prepositions indicate special/temporal/other relations
* A particle is often combed with verbs. Resemble prepositions or adverbs. When paired with a verb, called phrasal verbs and their meaning is non-compositional
* Determiners mark start of noun phrase (this and that)
  + Articles are a type of determiner which mark discourse properties
* Conjunctions join 2 phrases, clauses and sentences.
  + Co-ordinating ones(and or but) join elements of equal status while subordinating ones are used when one of the elements have some embedded status (sometimes called complementizers)
* Pronouns refer to a another thing in a shorthanded way. Personal ones refer to people/entities while possessive ones indicate possession. Wh ones are used in questions
* Auxiliary verbs mark a semantic feature of a main verb.
  + Includes copula verb (be, do have) and modal verbs which mark mood associated with event depicted by main verb
* Tags:
  + Table

    Description automatically generated
* In PoS tagging, we take a sequence of words and tagset then output a sequence of tags which have the same length of input sequence. Each tag corresponds to a single word
* Tagging is disambiguation task as words themselves are ambiguous as many words have more than one POS depending on context so we aim to resolve this with tagging
* Most tagging algs have high accuracy
* Given an ambiguous word, choose the tag which is more frequent in the corpus training
* Named entity: Anything we can refer to with a proper name
  + Includes dates, times and other temporal or numerical expressions
* Aim of NER is to scan texts made of proper names and tag entity type
  + Common tags are PER for person, LOC for location, ORG for organization or GPE for geopolitical,
* BIO tagging is when we treat NER like a word-by-word sequencing labelling task. Has variants called IO and BIOES labelling
* Syntactic constituency: Groups of words can behave as a single unit
* Noun phrase: Sequence of words surrounding at least one noun
* Preposed and postposed constructions help proved constituency
  + These are phrases that can be placed in different areas including the beginning and end (pre and post respectively)
* Context Free grammars (CFGs) are widely used. Formalism is equivalent to Backus-Naur Form (BNF)
* These are made of a set of rules/productions which express how symbols of language can be grouped together and ordered together. They also include a lexicon of words and symbols.
* NP = Noun Phrase and VP= verb phrase
* Verb phrase is a verb followed by other things
* CFG symbols are split up into 2 classes:
  + Terminal symbols: Symbols which correspond to words in language
  + Non Terminal: Symbols which express abstractions.
* In a Context free (CF) rule, items to the right of the arrow (🡪) is an ordered list of one or more terminals and non-terminals while the left is a single non-terminal symbol expressing some cluster/generalization
* One can look at CFGs as a device for generating sentences and assigning a specified structure to a given sentence
* A derivation of the string of words is when we expand given rules in a sequence
* Parse tree shows this sequence of rule expansions
* We can say the parent of a node in a tree dominates its children and their children and immediately dominates its children,
* A start symbol is the symbol in which we can derive a set of strings from. We usually denote it as S and each grammar must have one.
* A prepositional phrase is a preposition followed by a noun phrase
* Most parse trees can be represented in bracketed notation with an example below
  + A picture containing diagram

    Description automatically generated
* Grammatical sentences are ones which can be derived from a grammar which is in the formal language defined by said grammar
  + Opposite is ungrammatical
* Using formal languages to model a natural language is called generative grammar
* CFGs have 4 parameters:
  + 
* Text, letter

  Description automatically generated
* Text

  Description automatically generated
* Syntactic parsing: Mapping string of words to parse tree
* Declarative structure: Subject noun phrase followed by verb phrase
* Imperative structure: Sentences beginning with verb phrase and often have no subject
* Yes-No Question: Start with auxiliary verb followed by NP and VP
* Wh-words are who, whose, when, where, what, which, how, why and this make up a wh-phrase
  + We can make the wh-phrase not the subject of the sentence leading to a wh-non-subject question. This is an example of long distance dependencies is the wh-non subject is far away from predicate its related to.
* A clause is a complete thought. We can use this definition to say that S is a node of a parse tree in which the main verb o S has all arguments
* Frequent types of noun phrases is: pronouns, proper nouns and Det Nominal construction.
* We can make a rule recursive by starting a sentence with a Det 🡪 NP
* Nominals usually follow determiner and can contain pre/post-head noun modifiers (Nominal🡪Noun)
* We can include cardinal, ordinal numbers or quantifiers before head noun.
* Can group adjectives to make an Adjective Phrase AP
* Head nouns can be modified by postmodifiers which are
  + Prepositional phrases, nonfinite clauses and relative clauses.
* We can add the rule Nominal 🡪Nominal PP to account for more than one postnominal prepositions
* Gerundive postmodifiers consist of a verb phrase in which the verb is in the ing form (Nominal🡪Nominal GerundVP)
  + Text

    Description automatically generated
* A postnomial relative clause starts with a relative pronoun(i.e. that and who) in which they can act as the subject of embedded verbs
  + A picture containing text

    Description automatically generated
* Predeterminers are words which appear/modify before NPs
* Current rules for VPs
  + Text

    Description automatically generated
* We can get more complex verb phrases as we can use embedded sentences to follow a verb(sentential complements) (VP🡪Verb S)
* We can follow a VP with another VP
* While we usually subcategorize verbs into transitive or intransitive, we an have 100 different subclasses. We can have constitutes which complement the verb.
* Text

  Description automatically generated
* VPs and NP can be connected with conjunction.
* Coordinate noun phrases can made of two NPs connected with a conjunction
  + NP🡪 NP and NP. VP🡪VP and VP. S🡪S and S
* Treebank: Corpus in which every sentence in collection is paired with a corresponding parse tree. Most famous is the Penn Treebank.
* Penn treebank uses traces (NONE nodes) to mark syntactic movements (long distance dependencies).
* Diagram

  Description automatically generated
* We have many long and flat rules when making the Penn treebank into grammars
* Text

  Description automatically generated
* Text, letter

  Description automatically generated
* We usually use a lexical head for most phrases (Noun for NP and Verb for VP)
* Most CF rules have a head
* The head is the most grammatically important in a sentences
  + Passed up parse tree
* 2 grammars are strongly equivalent if they generate the same set of strings and if they assign same phrase structure to each sentence while 2 grammars are weakly equivalent if they generate the same set of strings but don’t assign same phrase structure to each stance
* Most CFG is in Chomsky normal form (CNF). This is when a CFG is epsilon free and each production is in either of the form A🡪 B C or A 🡪a. These is usually binary branching in the sense that they have binary trees.
* Chomsky adjunction is when we use a symbol A with a potentially infinite sequence of Symbols B with rule A🡪 A B.

N -Gram Language Models

* We assign a probability to each possible next word in models
  + Essential for speech recognition
  + With spelling/grammatical error correction, we need to see which combos of words are more probable
  + Essential for machine translation
* Probabilities important for augmentative and alternative communication systems (AAC) which are used if person cant speak/sign but can eye gaze/other eye movements
* Language models (LMs): Models that assign probabilities to sequences of words
* N-gram: sequence of n words (2-words, 3 words etc.)
* P(w|h) where given history h, probability of word h
  + Can use relative frequency counts(take large corpus and count number of times we see w after h).
    - Text, letter

      Description automatically generated
    - Not big to give good estimates
    - Language is creative
  + Given a word X\_i and sequence of words w\_1:n and join probability of each word in a sequence having value P(X=w\_1,Y=w\_2,Z=w\_3,W=W\_n)
    - To get P(w\_1:n)=P(w1)P(w2|w1)P(w3|w1w2)…P(wn|w1:n-1)
    - 
    - Tough as we don’t know exact prob of a word given long preceding sentence
  + Use n-grams to get the probability by only the last few words
  + Bigrams( 2-gram model) uses conditional probability of preceding word ( P(w\_n|w\_n-1))
    - 
  + Generally : Text

    Description automatically generated with medium confidence
* Markov assumption: Assumption that probability of word depends only on previous word
* Maximum Likelihood estimation helps with n-gram probabilities
  + Normalize counts to be between 0 and 1
  + Text, letter

    Description automatically generated
  + Can be simplified to A picture containing text

    Description automatically generated
  + Resulting parameter sets max likelihood estimation of training set T given model M.
* Augment start of sentence with <s> and end of sentence with </s>
  + Worked example:
    - A picture containing text

      Description automatically generated
    - Text, letter

      Description automatically generated
    - Relative freq ratio
    - Get prob of a sentence, multiply all conditional probabilities with each other
* Bigrams can encode facts which are syntactic in nature i.e. verb🡪noun/adjective
* More common to use trigrams which condition on previous 2 words rather than the one or sometimes even 4-gram/5-gram given decent training data.
* We use log probabilities for most language model probabilities
  + We don’t get really small numbers anymore
  + Add probabilities instead of multiplying
    - Take exponential at end then to return to normal
* Extrinsic evaluation: Embed it in application and measure how much it improves
  + Good way to know if improvement in component helps current task
  + Expensive
* Intrinsic evaluation: Measure quality of model independent of application
  + Need test set(unseen data)-used to measure performance. Can be held out of training
  + Train on training set(training corpus)
* Fit dataset means assigning a higher probability to test set
  + Tighter fit to data set
* If we train on value which is both in test and training, we artificially assign a high value when it happens in training
* Training on test set introduces bias and inaccuracies in perplexities
* Sometimes we use an unseen test set (devset) .
* Need large test. We divide data into 80% training 10% development and 10% test\
* Perplexity (PP) is defines as the following



* Diagram

  Description automatically generated with medium confidence
* Text

  Description automatically generated with medium confidence
* Think of perplexity as the weighted average branching factor of a language. This is the number of possible next words of a language
* Sampling: Choose random points according to likelihood
  + sampling from a language model—which represents a distribution over sentences—means to generate some sentences, choosing each sentence according to its likelihood as defined by the model
* n-gram dependant on training data
  + higher n value, better job modelling data
  + longer context on which we trin model, more coherent sentences form
  + bigram has local word to word coherence whereas high n values lead to similar data to our training data
  + Make sure our training corpa have similar genre, appropriate dialect and/or variety
  + Models may be subject to sparsity. Some perfectly acceptable English word sequences are bound to be missing from it
  + If we get a probability of zero from a model, occurring test data but not in training set, we have a problem as we underestimate probability of word which might occur and if any probability of any word is zero in test set then the entire test set probability is zero
* Closed vocabulary system: We know all words which can occur
* Out of vocabulary(OOV) words/unknown words : Words we haven’t seen when training but in test
* OOV rate: % of OOV words in test set
  + Avoid this by making problem back into closed set by :
    - Text

      Description automatically generated

Smoothing:

* Smoothing: Preventing model from assigning 0 probability to unseen events by shaving a bit of probability mass from some more frequent events.
  + Think of it as a way to discounting non-zero counts in order to get probability mass which will be assigned to 0 counts.
    - Dicount ratio is discounted/non-discounted
  + Laplace smoothing:
    - Add one to n-gram counts
    - Not that useful but acts as a baseline for other smoothing algs
    - Useful for text classification
    - Text

      Description automatically generated with low confidence
    - Text

      Description automatically generated
    - For add one bigrams we get the following
      * Text, letter

        Description automatically generated
  + Add k smoothing
    - Add a fractional count k to each count
    - A picture containing text

      Description automatically generated
    - Need a method to choose k for example optimizing on devset
    - Not good for language modelling
  + Backoff and interpolation
    - We can use less context to generalize more for contexts model hasn’t learned yet
    - Backoff: If evidence is sufficient we use trigram, otherwise the bigram, otherwise the unigram. Think along the lines we go down one step in n-gram if we have 0 evidence for higher n gram.
      * Continue to back off until we get non zero count
      * Must discount higher order n gram model to save probability mass
      * We use katz backoff to rely on a discounted probability P\* if we seen n-gram before
    - Interpolation: We mix probability estimates from all the n-gram estimators weighing and combining tri/bi/unigram.
      * Add a lambda which is the weight

A picture containing chart

Description automatically generated

Text, letter

Description automatically generated

* + - * We can condition on the context
        + Text, letter

          Description automatically generated
      * Lambdas are learned from held out corpus
      * Held out corpus can be used to set hyperparameters
      * Can use EM algs to learn lambda
      * We need alpha function to distribute probability mass to lower order n-gram
      * Text

        Description automatically generated
      * Combined with Good-Turing
  + Kneser-Ney smoothing
    - Uses absolute discounting(subtracts fixed amount d from training set)
    - Mainly deals with smaller counts
    - Text

      Description automatically generated
    - Kneser-Ney discounting uses absolute discounting
    - Text

      Description automatically generated
    - Text, letter

      Description automatically generated
      * First term is normalized discount while he second term is the number of word types we discounted (no of times we applied normalized descent)
    - Graphical user interface, text, application, Word

      Description automatically generatedCKN depends on whether we are counting highest order n gram being interpolated or one of the lower order n grams
      * Text

        Description automatically generated
    - At termination of recursion, we get the following uniform distribution
    - Text

      Description automatically generated with medium confidence
    - Better model uses multiple discount values
  + Huge Language Models and Stupid backoff
    - N gram models can be shrunk by pruning by storing n grams with counts greater than a threshold or using entropy to remove less important n grams
    - Can use bloom filters like techniques
    - Stupid backoff: give up the idea of trying to make language model a true probability distribution. No discounting for higher level n-grams and if higher order n gram has 0 prob, we backoff to lower
    - Text

      Description automatically generated
  + Perplexity and Entropy
    - Entropy is a measure of info. Given random variable X ranging over whatever we aim to predict (Chi), entropy of X is
      * A picture containing text, watch

        Description automatically generated
      * If we use log 2 we measure entropy in bits
      * E.g.
      * Table

        Description automatically generated
    - We mainly use entropy for sequences
      * A screenshot of a computer

        Description automatically generated with low confidence
    - Entropy rate is entropy divided by number of words but we need to consider sequences of infinite length for true entropy
      * Text, letter

        Description automatically generated
    - Stationary process is stationary if probabilities it assigns to a sequence are invariant with respect to shifts in time index. Probability distribution for words at time t is the dame as probability dist at time t+1. This Markov models and n grams are stationary. But natural language is not stationary
    - When we don’t know actual probability dist, that generated data, we use cross entropy. Based of model m which is a model of p A picture containing chart

      Description automatically generated
    - Text

      Description automatically generated
    - A picture containing chart

      Description automatically generated
    - Graphical user interface

      Description automatically generated

Hidden Markov Models (HMMs)

* Recall a proper name can often be a multiword phrase
* Named Entity: Anything referred to with a proper name
* POS (Parts of Speech) and Named Entities help define sentence structure.
  + Can use to tell us about neighboring words and syntactic structure
* Part of speech tagging: Taking a sequence of words and assigning each word a part of speech
* Named Entity Recognition: Assigning phrases/words tags such as location or organization
* Sequence Labeling: Taking a word x and assigning it a tag y such that the sequence of tags is same length of sequence of words
* Graphical user interface, text

  Description automatically generated
* Text, table

  Description automatically generated
* Tagging is a disambiguation task as words themselves are ambiguous (have more than 1 possible PoS) and goal of PoS is to resolve those ambiguities
* Baseline: Given an ambiguous word, choose the tag most frequent in training data
* Dates and times are also named entities
* Hidden Markov Models are an augmentation of Markov Chains which tell us about the probabilities of sequences of random variables(known as states) which can take on values from some set
  + Words/tags/symbols representing anything are all sets
* Markov chains make strong assumptions if we we want to predict sequence futures, current state is only one that matters
  + Prev states do not matter
*  where q(1🡪i-1 are state variables)
* Start distribution of π is important (tells us the probability of where we start)
* Markov Chains are made from:
  + Text

    Description automatically generated
* Most events we are interested in are hidden(don’t observe directly)
* HMMs allow us to deal with both observed and hidden events which we think are causal factors in model
  + Text

    Description automatically generated
  + Its Markov assumption is and has output independence Text, letter

    Description automatically generated
* Tag transition probability P(t\_t|t\_i-1) is prob of a tag occurring after previous tag 
* Text, letter

  Description automatically generated
* Decoding: Given an HMM and observation sequence, find most probable sequence of states
* Text

  Description automatically generated
* Text, letter

  Description automatically generated
* Probability of word appearing depends only on own tag and independent of neighboring words and tags A picture containing text

  Description automatically generated
* the probability of a tag is only dependant on previous tag onlyA picture containing text

  Description automatically generated
* Text, letter

  Description automatically generated
* Mainly use Viterbi Alg for HMM (type of dynamic programming)
  + Graphical user interface, text

    Description automatically generated
  + First sets up prob matrix/lattice with a column for each observation and row for each state
  + Each cell of lattice represents probability that the HMM is in state j after the first t observations and passing through most probable sequence/
  + Value of cell calculated as follows 
    - Taken as max over all possible state sequences
  + Fills each cell recursively
  + For given state q\_j at time, value of cell is Logo, company name

    Description automatically generated
  + 3 factors in above form are:
    - Text

      Description automatically generated with medium confidence

Context Free Grammars

* Probabilistic parsing to is used to solve disambiguation
* CKY allows us to represent the ambiguities but cant resolve them
* PCFGs are a probabilistic augmentation of context free grammars in which the rule is associated with a probability. Has the following rules
  + Text

    Description automatically generated
  + PCDFGs augment each rule in r with a conditional probability which is the P( a non terminal A expands into Sequence β)=P(A🡪 β)= P(A🡪 β|A)=P(RHS|LHS)
  + Sum of possible expansions must be 1
* Consistent: Sum of all probabilities of all sentences in language is 1
  + Recursive rules can cause a grammar to be inconsistent
* Disambiguation: Assigning a probability to each parse tree of a sentence

Text

Description automatically generated

Text, letter

Description automatically generated

* Yield: String of words
* Disambiguation alg picks parse tree most probable S s.t.
  + Text

    Description automatically generated = Text

    Description automatically generated
  + As we max over all parse trees, P(s) is constant and as P(S,T) =P(T) we get the following: Text

    Description automatically generated
* Important in language modelling
* Probability of an ambiguous sentence is sum of probabilities of all parse trees for sentence
  + 
* N-gram model often ignores prediction cues
* PCFGS allows us to condition on entire previous context
* Most probabilistic parsers use the probabilistic CFK alg in which the PCFG must be in Chomsky Normal form
  + Each sentence has an index between words
  + We use said index to allows us encode a 2-D matrix from the CKY parse tee
  + Given sentence length n and grammar we V non terminals, we use upper triangular position of an n+1 x n+1 matrix
  + Each cell table [I,j] contains a list of constituents which could span sequence of words i🡪j
  + There is a 3rd dimension here which is the probability for that non-terminal/constituent and has a max length of V
  + Therefore each cell [I,j,A] in the )n+1)x(nx1)xV matrix is probability of a constituent of type A which spans through i🡪j of input
* CKY alg:
  + Text, letter

    Description automatically generated
* Stochastic Context-Free Grammar is a augmentation of this
* Given a treebank, probability of each expansion is to count number of times expansion occurs and then normalizing A picture containing text

  Description automatically generated
* If no treebank but have parser, we can generate counts by parsing corpus of sentences with parser
  + Problem is most sentences have multiple parses so we don’t know which one to count rules in. Instead, we need to keep a separate count for each parse of a sentence and weight each of these partial counts by the probability of the parse it appears in. But to get these parse probabilities to weight the rules, we need to already have a probabilistic parser.
    - To solve this, we incrementally improve our estimates by beginning with a parser with equal rule probabilities, then parse the sentence, compute a probability for each parse, use these probabilities to weight the counts, re-estimate the rule probabilities, and so on, until our probabilities converge. The standard algorithm for computing this solution is called the inside-outside algorithm. This algorithm includes an expectation step and maximization step
* Coordination ambiguity and attachment ambiguity can allow sentences to become syntactically ambiguous
* PCFGs (probabilistic context free grammars) have a few issues as probability estimators:
  + Poor independence assumptions as CFG rules tend to force an independence assumption on probabilities which lead to a poor modelling of structural dependencies across parse trees
    - Probability of a group of independent events is their product of probabilities but for PCFGs, leads to poor estimates as specifics nodes expand differently in different portions in a parse tree
    - PCFGs don’t allow us to condition on surrounding context leading to difficulties
    - Parent annotation can help deal with this
  + Lack of lexical conditioning as they don’t deal nor model syntactic facts about certain word
    - Recall words play an important part in PCFGs since the parse probability of words includes P( word|PoS)
    - Lexical info is important in other places in the grammar. One importance is clearing prepositional phrase attachment ambiguities
    - Vp attachment and NP attachment are crucial here
    - PCFGs will always prefer either of the 2.
      * NP attachment is more common
    - Identities of verbs, nouns and prepositions allows us to determine if we need NP or VP attachment
    - Lexical dependencies is important
* We can modify PCFGs to deal with lexicalized rules which includes the Collins and Charniak parsers
* Syntactic constituents could be associated with a lexicalized Head and lexicalized grammar is one in which each non terminal in tree is annotated with lexical head
* Head tag: PoS tags of headwords
* We can associate said head tags with non terminal symbols as well
* To generate this trees,we must augment each rule to identify right hand constituent to be head daughter and the headword for a node is then set to headword of its head daughter leading to the head tag to the PoS tag of headword.
* Lexical rules: Expansion of a pre-terminal to word
  + Deterministic and have a probability of 1
* Internal rules: Express other rule expansions
  + Must estimate probability by using the product of smaller independent probability estimates for which we could acquire reasonable counts
* Both rules express different probabilities
* The Collins parser thinks of the RHS of every rules consists of a head non-terminal together with non-terminals to the left of the head and the non-terminals to the right of the head
  + Think of it like this 
  + L\_n is actually a complex symbol representing the category and its hear & head tag.
  + We break this down using a generative story leading to Collins Model 1. Give the LHS, we generate the head of each rule then generate dependants of head one-by-one from the inside out. Each have own probability
* We add the STOP non terminal to signal when we should stop generating dependencies on a given side.
* Generate from left side till we reach stop then generate from the right
* Text

  Description automatically generated
* Text categorization: Assigning label/category to entire text/document
* Sentiment: +ve or -ve view writer expresses
* Sentimental analysis: Extracting an authors sentiment
  + Binary classification task
  + Words tend to provide clues
* Another example of binary classification is spam detection: Assigning an email to either spam inbox or non-spam
* Language id is important in language processing
* Authorship attribution: Determine texts author
* Text classification is also used when we assign a library subject category/topic label to txt
* Goal of classification is to take single observation and classify it based of features
  + Mostly done through supervised learning ( We have data input and with some associated output. Aim is to guide input to output)
  + Aka, for each input x and set of classes Y= y\_1🡪y\_m, return a predicted class from Y. Class can also be denoted as c . We use N documents which have been labeled with a class and aim is to learn if it can map a document to its correct class.
* Probabilistic classifier: Tells us probability of being in the class of observation
* Generative classifiers: Model how a class can generate input data and output which class is likely to have generated said observation
* Discriminative classifiers: Learn from features on how to differentiate between possible classes
* Multinominal naïve Bayes classifier: Bayesian classifier which makes a naïve assumption about how the features interact.
* Bag-of-words: Unorder set of words with their position ignored and only caring about frequency.
* Estimate of correct class : 
* Recall bayes rule on conditional probability: Text

  Description automatically generated
* Using substitution: Text

  Description automatically generated with low confidence
* Naives Bayes is generative model: First class from P(c), and then words are generated by sampling from P(d|c).
* Prior probability: P(c) and likelihood probability: P(D|c) 🡪 Text

  Description automatically generated
* Text

  Description automatically generated
  + Hard to compute
* 2 assumptions to make:
  + Positions don’t matter when it comes to classification
  + Naïve bayes assumption: Conditional independence assumption that the probabilities P(f\_i|c) are independent given class c and can be multiplied by the following
* Text, letter

  Description automatically generated
* Word positions are important to apply the naïve Bayes Classifiers.
  + Walk an index through every word position in document
  + Logo, company name

    Description automatically generated
  + But to prevent underflow, we use log
  + A picture containing shape

    Description automatically generated
* Linear classifier: Classifiers using a linear combination of inputs
* Prior probability is the number of documents with the class c divided by total number of documents. A picture containing text

  Description automatically generated
* P(f\_i,c), we assume a feature is just a word in the bag-of-words thus aim is to get P(w\_i|c) which is gotten by the fraction of the times the word w\_i appears in all documents of topic c. To do this we first concatenate all documents together and use the frequency of w\_i to give us an MLE s.t. Text

  Description automatically generatedwhere V is vocabulary of all words in all classes.
* Problem with this is if we don’t get the word classified with the classifier we want, when we train, we will always get zero, thus we can add Laplace smoothin s.t. Diagram

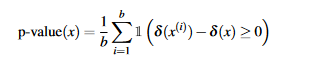
  Description automatically generated with low confidence
* We can remove unknown words from test document and ignore them at all
* Sometimes we remove stop words such as the and a which are very frequent words. We sort all vocabularies in documents and make it such that the top 10/100 words are stop words or us predefined lists. Note it doesn’t improve vocabulary
* Text

  Description automatically generated
* We need to make changes to Bayes if we wish to perform sentimental analysis which also improves performance
* Binary multinominal naïve Bayes/Binary NB: When we clip word count to document to one as whether a word occurs or not is more important than frequency
* We remove all duplicates words in a document before concatenating them
  + Note a word can be repeated across documents, you only remove all duplicates in the document before concatenating it
* We need to deal with negation
  + During text normalization, prepped the NOT prefix to every word after a token of logical negation
* If we have not enough labelled training data, use derive +ve and -ve from sentiment lexicons(lists of words which are pre-annotated with +ve or -ve sentiment.
* Byte n-grams are important for language id.
* If using only individual word features, bayes classifier acts akin to a language model. Think of it as a class specific unigram set. We can also assign each sentence a probability since it initially assigns to each word P(word|c). To do this, we do: Logo

  Description automatically generated with medium confidence
* Gold labels: Human defined labels
* Confusion matrix: Table to visualize how well an alg works w.r.t gold labels, We need 2 dimensions for system output labels and gold labels.
* Table

  Description automatically generated
* Tp=true positive, fn=false negative tn = true negative and fp= false positive
* Precision measure % of items that system detected which are positive while recall measure % items which were accurately identified by system
* F-measure combines the 2 in such a way that :  where R=recall, P=precision and beta is a parameter which differentially weights the importance o recall and precision . If B>1, we favour recall, if <1, we favour precision ,else we favour equally which becomes 
* F-measure if a harmonic mean of precision and recall. This is a reciprocal of the mean of reciprocals. This is conservative .
* A picture containing chart

  Description automatically generated
* Tune params on dev/valid set and from there decide which is best
* Cross validation allows us to use all our data for training and still use all data for testing
  + Folds: K disjoint subsets of partitioned data
  + We choose a single fold for test set and the rest for training and compute error on test.
  + We ten repeat this by choosing another fold and use the other for training
  + Repeat k times then average.
  + Corpus needs to be blind
* To compare 2 classifiers, we do the following: in which M(A,x) is the score of Model A in test set x. This is called effect size.
* Text, letter

  Description automatically generated with medium confidence Our null and alternate hypotheses
* Statistically significant: If delta we saw has probability below a threshold leading us to reject null hypothesis
  + Use non-parametric tests to get p-value
  + P-value is found by 
* We use either approximate randomization or bootstrap test.
* Bootstrap paired test
  + Can apply to any metric
  + Bootstrapping: Repeatedly drawing large number of smaller samples with replacement from an original larger sample.
  + Only assumption is that sample is a representative of population
* Paired bootstrap example:
  + We use a test set x of 10 documents and write down the results of 2 classifiers A and B here, results are either, both get correct class, one gets correct class or none do
  + Then select a large number b of virtual test sets xi with size 10. To make said virtual test sets, select a cell from row x (first row of table) 10 times with replacement(copy cell value)
  + We can use this table of values to see accidental advantage
    - Assume H0(A<B), we’d expect the effect size to be 0 and it would be surprising if we get a value larger than that as we assumed the null hypothesis is correct. TO see this measure, we do the following: 
    - Not 1(x) means 1 if x is true and 0 otherwise)
    - Since we samples from original test set, ew have a bias to which every model the first test set favours. Thus, our p-value becomes: 
  + Text

    Description automatically generated
* Representational harm: arms caused by a system demeaning a social group
* Toxicity detection: Task of detecting hate speech, abuse, harassment etc.
* Model card contains info such as:
  + Training alg/params
  + Training & evaluation data sources, motivation and preproccessing
  + Intended use and users
  + Model performance across different demographic/groups
* Conversational agents/dialogue systems are programs which communicate with users in a natural language.
  + Task orientated ones use speech to help accomplish a task
  + Chatbots are designed for extended conversation
* Dialogue is a sequence of turns between two conversation parties
* Must know when to stop and start talking is important for a system
* Endpoint detection: Determining when a user has stopped speaking
* Speech acts: Each utterance in a dialogue is a kind of action being performed by speaker
  + Directive: Asking someone/system to do something, asking a question requiring an answer
  + Constative: Stating a constraint
  + Acknowledgement: express speakers attitude regarding hearer wrt to some action
  + Commissive: Committing speaker to some future course of action
* Common ground: Participants establishing what they agree on
* Convo structure
  + Question🡪Answer
  + Proposal🡪Acceptance/rejection
  + Compliments🡪Downplayers
* Adjacency pairs are composed of a first and second pair part
* Side sequence or sub dialogue can occur instead of part pair
* Clarification question Acts as a sub dialogue between request and a response
* Presequences can also occur here
* Initiative is when a conversation is controlled by a single participant
* Mixed initiative is when one asks another question but the other asks to clarify leading to another conversation. Tough for systems
* Implicature: Class of licensed inferences
* Maxims: Heuristics which play a guiding role when interpretating conversations
* Reason dialogue systems are tough to build are because:
  + Turns
  + Speech acts
  + Dialogue structure
  + Initiative
  + Implicature
* Chatbots are a type of dialogue system
* ELIZA worked off pattern/transform rules which is linked to a keyword which might occur in a user sentence
* If no keywords match, it gives sentences such as Please go on or I see
* Turing test used to see determine how lifelike chatbots are