CS584: Natural Language Processing Notes

Steven DeFalco

Spring 2024

Contents

1 Regular Expressions, Text Normalization, Edti Distance

Regular expressions can be used to specify strings we might want to extract from a document. ***Normalizing*** text means converting it to a more convenient, standard form. For example, most of what we are going to do with language relies on first separating out or ***tokenizing*** words from running text. Another part of text normalization is ***lemmatization***, the task of determining that two words have the same root, despite their surface differences. For example, the words *sang*, *sung*, and *sings* are forms of the verb *sing*. The word *sing* is the common lemma of these words, and a ***lemmatizer*** maps from all of these to *sing*. ***Stemming*** refers to a simpler version of lemmatization in which we mainly just strip suffixes from the end of the word. Text normalization also includes ***sentence segmentation***: breaking up a text into individual setnences, using cues like periods or exclamation points. Finally, we'll need to compare words and other strings. We'll introduce a metrix called ***edit distance*** that measures how similar two strings are based on the number of edits (insertions, deletions, substitutions) it takes to change one string into the other.

1.1 Regular Expressions

Formally, a regular expression is an algebraic notation for characterizing a set of strings. Regular expressions are particularly useful for searching in texts, when we have a pattern to search for a ***corpus*** of texts to search through. A regular expression search function will search through the corpus, returning all texts that match the pattern.

1.1.1 Basic Regular Expression Patterns

The simplest kind of regular expression is a sequence of simple characters; putting characters in sequence is called ***concatenation***. Regular expressions are **case sensitive**. We can solve this with the use of square braces [and]. The string of characters inside the braces specifies **disjunction** of characters to match. For example, the pattern /[wW]/ matches patterns containing either w or W.

The square braces can also be used to specify what a single character *cannot* be, by use of the caret ^. If the caret is the first symbol after the open square brace, the resulting pattern is negated. For example, the pattern /[^a]/ matches any single character (including special characters) except a. This is only true when the caret is the first symbol after the open square brace.

We use the question mark /?/, which means *the preceding character or nothing.* For example, /woodchucks?/ will match with either *woodchuck* or *woodchucks.* The ***Kleene star*** means *zero or more occurrences of the immediately previous*

character or regular expression. So /a^*/ means *any string of zero or more as.*/ *Kleene+* means *one or more occurrences of the immediaately preceding character or regular expression.* The period is used as a ***wildcard*** expression that mat ches any single character.

1.1.2 Disjunction, Grouping, and Precedence

The ***disjunction*** operator, also called the **pipe** symbol | mathes either the preceding or following strings. Typically, regular expressions always match the *largest* string they can: we say that patterns are **greedy**, expanding to cover as much of a string as they can. There are, however, ways to enforce **non-greedy** matching, using another meaning of the ? qualifiers. The operator $*$? is a Kleene star that matches as little text as possible. The operator +? is a Kleene plus that matches as little text as possible.

1.1.3 More Operators

Below are some aliases for common ranges, which can be used mainly to save typing.

- / d: any digit
- / D: any non-digit
- / w: any alphanumeric/underscore
- / W: a non-alphanumeric
- / s: whitespace (space, tab)
- / S: non-whitespace

A range of numbers can also be specified. So /$\{n, m\}$/ specifies from n to m occurrences of the previous char or expression. REs for counting are summarized below.

- $*$: zero or more occurrences of the previous char or expression
- +: one or more occurrences of the previous char or expression
- ?: zero or one occurrences of the previous char or expression
- $\{n\}$: exactly n occurrences of the previous char or expression
- $\{n, m\}$: from n to m occurrrences of the previous char or expression
- $\{n, \}$: at least n occurrences of the previous char or expression
- $\{, m\}$: up to m occurrences of the previous char or expression

Some certain special characters are referred to by special notation based on the backslash. The most common of these are the **newline** character / n and the **tab** character /t

- / *: an asterisk
- / .: a period
- / ?: a question mark
- / n: a newline
- / t: a tab

1.2 Words

A ***lemma*** is a set of lexical forms having the same stem, the same major part-of-speech, adn the same word sense. The ***word form*** is the full inflected or derived form of the word. ***Types*** are the number of distinct words in a corpus; if the set of words in the vocabular is V, the number of types is the vocabulary size $|V|$. ***Tokens*** are the total number N of running words.

The larger the corpora we look at, the more word types we find, and in fact this relationship bettween the number of types $|V|$ and number of tokens N is called ***Herdan's Law*** or ***Heaps' Law***. It is shown below where k and β are positive constants, and $0 < \beta < 1$.

$$|V| = kN^{\beta}$$

The value of β depends on the corpus size and the genre. Roughly then we can say that the vocabulary size for a text goes up significantly faster than the square root of its length in words.

1.3 Text Normalization

Before almost any natural language processing of a text, the text has to be normalized. At least three tasks are commonly applied as part of any normalization process:

1. Tokenizing (segmenting) words
2. Normalizing word formats
3. Segmenting sentences

One commonly used tokenization standard is known as the ***Penn Treebank tokenization*** standard. This standard separates out clitics (*doesn't* becomes

does plus *n't*), keeps hyphenated words together and separates out all punctuation.

Most tokenization schemes have two parts: a token learner and a token segmenter. The ***token learner*** takes a raw training corpus and introduces a vocabulary, a set of token. The ***token segmenter*** takes a raw test sentence and segments it into the tokens in the vocabulary. The most widely used algorithm is byte-pair encoding. The BPE token learner begins with a vocabulary that is just the set of all indivdual characters. It then examines the training corpus, chooses the two symbols that are most frequently adjacent, adds a new merged symbol to the vocabulary, and replaces every adjacent in teh corpus with the new symbol. It continues to count and merge, creating new longer and longer character strings, until k merges haev been done creating k novel tokens; k is thus a parameter of the algorithm. The resuling vocabulary consists of the original set of characters plus k new symbols. The algorithm is usually run inside words, so the input corpus is first white-space-separated to give a set of strings, each corresponiding to the characters of a word, plus a special end-of-word symbols, and its counts.

2 Lecture 2: Machine Learning Basics

The ***training set*** is the sample of data used to fit the model. The ***validation set*** is the sample of data used to provide unbiased evluation of a model fit on the training dataset while tuning model hyperparameters. The ***test dataset*** is the sample of data used to provide an unbiased evaluation of a final model fit on the training dataset; this cannot be used for training.

Cross-validation allows us to estimate the generalization error based on training examples alone. In ***k-fold cross validation*** the original sample is randomly partitioned in k equal sized subsamples. Of the k subsamples, a single subsample is retained as the validation data for testing the model, and the remaining $k-1$ subsamples are used as training data.

In ***supervised learning***, we have a training dataset containing samples. In each samples, x_i are inputs and y_i are albels. We try to predict the labels using the inputs, and this is the basis of training. With ***unsupervised learning***, we have no labels and instead expect the model to learn some patterns from the dataset.
For **document classifications**, build a feature vector for each document

$$\{x_i, y_i\}_{i=1}^{N}$$

where x_i can be word counts, TF-IDF, topic distributions, etc...

A ***loss function*** (Loss(x, y, w)) qualifies how wrong you would be if you used w to make a prediction in x when the correct output is y. It is the object we want

to minimize. Loss is a function of the parameters w and we can try to minimize it directly. We reduce the estimation problem to a **minimization** problem.

We have a cost function $J(\theta)$ we want to minimize. ***Gradient descent*** is an algorithm to minimize $J(\theta)$: for the current value θ, calculate the gradient of $J(\theta)$, then take small step in direction of negative gradient, and repeat. The ***gradient*** is the direction that increases the loss the most; we want to take the *negative* gradient in gradient descent.

Gradient descent is slow. An ***epoch*** is one pass through the data. Each iteration requires going over all training examples; this is expensive and slow when we have lots of data. In practice, we can use ***stochastic gradient descent***. At each epoch, we can randomly shuffle the data to ensure that each data point creates in *independent* change on the model, without being biased by the same points before them. In this implementation, it may never reach the local minima and could oscilate around it due to the fluctuations in each step. Instead of using the whole data for calculating gradient, in ***mini-batch gradient descent*** we use only a mini-batch of it instead of the whole dataset.

Learning rate affects the update of gradients; it is the amount that we update the weights. A too large learning rate may lead to diverging values. A too small learning rate my cost a lot of time to convert to the minimum points. ***Adaptive learning rate*** involves reducing the learning rate by some factor every few epochs. Divide the learning rate of each parameter by the root mean square of its previous derivatives.

2.1 Neural Networks

Logistic regression gives only linear decision boundaries which are limited and unhelpful when a problem is complex. Neural networks can learn much more complex functions and **nonlinear** decision boundaries. Some of the *pros* of neural networks are the following:

- used on a variety of domains
- predictions are very quick

Backpropagation provides an efficient procedure to compute derivatives. Without nonlinearities, deep neural networks can't do anything more than a linear transform. Extra layers could just be compiled into a single linear transform.

When initializing...

- initialize weights to small random values
- initialize hidden layers to 0 and output biases to optimal value if weights were 0

Regularization prevents overfitting when we have a lot of features. A loss function in practice includes regularization over all parameters. ***Early stopping*** is a popular regularization technique in which we monitor the performance for every epoch on the validation set during the training set and terminate the training as soon as the validation error reaches a minimum.

Dropout provides a computationally inexpensive but powerful method of regularizing a broad family of models. Each neuron has some probability of being dropped out during each iteration.