# Natural Language Processing Introduction

Unit 2: Statistical approaches

February 2020

## Statistical approaches

### Statistical approaches for NLP

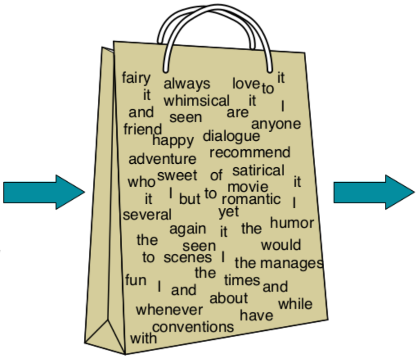
Statistical approaches have been the dominant paradigm for NLP before deep learning. They use the appeareance of words to correlate the meaning and form language models and probability distributions using the raw text data.

## Why are important?

- Explores statistical relationships between words
- Quick bechmarks for NLP systems
- Powerful techniques when building hybrid systems
- Useful techniques to deal with NLP tasks where there is not enough information to work with deep learning

## Bag of words

I love this movie! It's sweet, but with satirical humor. The dialogue is great and the adventure scenes are fun... It manages to be whimsical and romantic while laughing at the conventions of the fairy tale genre. I would recommend it to just about anyone. I've seen it several times, and I'm always happy to see it again whenever I have a friend who hasn't seen it yet!



| | |
|---|---|
| it | 6 |
| I | 5 |
| the | 4 |
| to | 3 |
| and | 3 |
| seen | 2 |
| yet | 1 |
| would | 1 |
| whimsical | 1 |
| times | 1 |
| sweet | 1 |
| satirical | 1 |
| adventure | 1 |
| genre | 1 |
| fairy | 1 |
| humor | 1 |
| have | 1 |
| great | 1 |
| ... | ... |

## Bag of words

We represent a text document as if it were a bag-of-words, that is, an unordered set of words with their position ignored, keeping only their frequency in the document. In the example in the figure, instead of representing the word order in all the phrases like “I love this movie” and “I would recommend it”, we simply note that the word I occurred 5 times in the entire excerpt, the word it 6 times, the words love, recommend, and movie once, and so on.

## Naive Bayes

### Bayesian inference

Bayesian inference has been known since the work of Bayes (1763), and was first applied to text classification by Mosteller and Wallace (1964). The intuition of Bayesian classification is to use Bayes' rule to transform probabilities that have some useful properties. Bayes' rule gives us a way to break down any conditional probability $P(x|y)$ into three other:

$$P(x|y) = \frac{P(y|x)P(x)}{P(y)}$$

## Naive bayes classifier

Naive Bayes is a probabilistic classifier, meaning that for a document d, out of all classes $c \in C$ the classifier returns the class $\hat{c}$ which has the maximum posterior probability given the document:

$$\hat{c} = \underset{c \in C}{argmax}\ P(c|d)$$

$$\hat{c} = \underset{c \in C}{argmax}\ P(c|d) = \underset{c \in C}{argmax}\ \frac{P(d|c)P(c)}{P(d)}$$

$$\hat{c} = \underset{c \in C}{argmax}\ P(c|d) = \underset{c \in C}{argmax}\ P(d|c)P(c)$$

$$\hat{c} = \underset{c \in C}{argmax}\ P(f_1, f_2, f_3, ..., f_n)P(c)$$

## Naive bayes assumption

The conditional independence assumption that the probabilities $P(f_i|c)$ are independent given the class c and hence can be 'naively'multiplied as follows:

$$P(f_1, f_2, f_3, .., f_n|c) = P(f_1|c)P(f_2|c)P(f_3|c)...P(f_n|c)$$

$$CNB = \underset{c \in C}{argmax}\ P(c) \prod_{f \in F} P(f|c)$$

To apply the naive Bayes classifier to text, we need to consider word positions, by simply walking an index through every word position in the document:

$$pos \leftarrow \text{positions in the document}$$

$$CNB = \underset{c \in C}{argmax}\ P(c) \prod_{i \in pos} P(w_i|c)$$

## Naive bayes assumption

The conditional independence assumption that the probabilities $P(f_i|c)$ are independent given the class c and hence can be ‘naively’multiplied as follows:

$$P(f_1, f_2, f_3, .., f_n|c) = P(f_1|c)P(f_2|c)P(f_3|c)...P(f_n|c)$$

$$CNB = \underset{c \in C}{argmax}\ P(c) \prod_{f \in F} P(f|c)$$

## Naive bayes for text classification

To apply the naive Bayes classifier to text, we need to consider word positions, by simply walking an index through every word position in the document:

$$pos \leftarrow \text{positions in the document}$$

$$CNB = \underset{c \in C}{argmax}\ P(c) \prod_{i \in pos} P(w_i|c)$$

## Naive classifier optimization

Naive Bayes calculations, like calculations for language modeling, are done in log space, to avoid underflow and increase speed:

$$pos \leftarrow \text{positions in the document}$$

$$CNB = \underset{c \in C}{argmax}\ log(P(c)) + \sum_{i \in pos} log(P(w_i|c))$$

## Training the naive bayes classifier

Here the vocabulary V consists of the union of all the word types in all classes, not just the words in one class c.

$$\hat{P}(w_i|c) = \frac{count(w_i,c)}{\sum\limits_{w \in V} count(w,c)}$$

In order to prevent the collapse of the probability we can apply a smoothing technique:

$$\hat{P}(w_i|c) = \frac{count(w_i,c)+1}{\sum\limits_{w \in V} count(w,c)+|V|}$$

## Naive bayes training I

**Input:** $D$ documents to train, $C$, set of classes.
**Output:** $logprior, loglikelihood, V$.

```
1: for all $c \in C$ do
2:     $N_{doc} \leftarrow$ number of documents in $D$
3:     $N_c \leftarrow$ number of documents from $D$ in class $C$
4:     $logprior[c] \leftarrow log(\frac{N_c}{N_{doc}})$
5:     $V \leftarrow$ vocabulary of $D$
```

## Naive bayes training II

```
6:      $bigdoc[c] \leftarrow append(d)$ **for** $d \in D$ with class $c$
7:      **for all** word in $w$ in $V$ **do**
8:          $count(w,c) \leftarrow$ number of ocurrences of w in $bigdoc[c]$
9:          $loglikelihood[w,c] \leftarrow \log \frac{count(w,c)+1}{\sum_{wp \in V}(count(wp,c)+1)}$
10:     **end for**
11: **end for**
12: **return** $logprior, loglikelihood, V$
```

## Naive bayes testing I

**Input:** *testdoc*,*logprior*,*loglikelihood*,C,V
**Output:** *sum* array containing the probabilities for each class

```
1: for all class $c \in C$ do
2:     $sum[c] \leftarrow logprior[c]$
3:     for all position $i$ in $testdoc$ do
4:         $word \leftarrow testdoc[i]$
5:         if word $\in V$ then
6:             $sum[c] \leftarrow sum[c] + loglikelihood[word, c]$
7:         end if
8:     end for
9: end for
```

## Predict using naive bayes classification

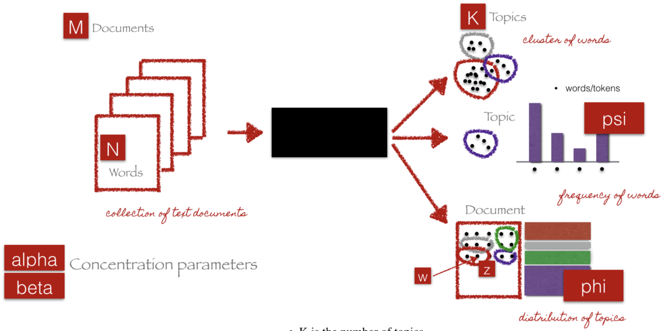
We can perform the prediction using the sum of the likelihoods probabilities for each class, we just need choose c which:

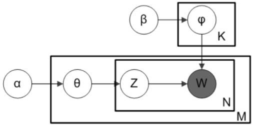
$$\underset{c \in C}{argmax}\ sum[c]$$

## Latent Drichlet allocation

LDA is a generative probabilistic model that assumes each topic is a mixture over an underlying set of words, and each document is a mixture of over a set of topic probabilities.

## Latent Drichlet allocation





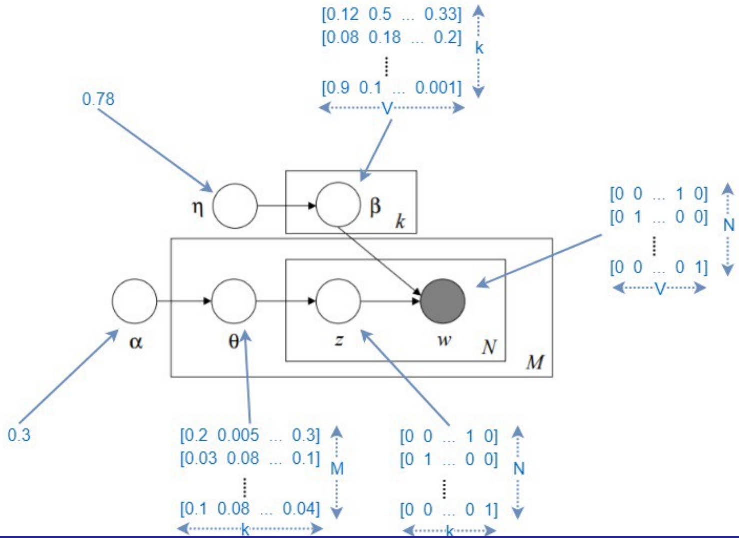
- K is the number of topics
- N is the number of words in the document
- M is the number of documents to analyse
- α is the Dirichlet-prior concentration parameter of the per-document topic distribution
- β is the same parameter of the per-topic word distribution
- φ(k) is the word distribution for topic k
- θ(i) is the topic distribution for document i
- z(i,j) is the topic assignment for w(i,j)
- w(i,j) is the j-th word in the i-th document
- φ and θ are Dirichlet distributions, z and w are multinomials.

## Latent Drichlet allocation

Latent Drichlet allocation (LDA) model elements:

- k: Number of topics a document belongs to (a fixed number)
- V: Size of the vocabulary
- M: Number of documents
- N: Number of words in each document
- w: A word in a document. This is represented as a one hot encoded vector of size V
- **w**: represents a document (i.e. vector of "w"s) of N words
- D: Corpus, a collection of M documents
- z: A topic from a set of k topics. A topic is a distribution words. For example it might be, Animal = (0.3 Cats, 0.4 Dogs, 0 AI, 0.2 Loyal, 0.1 Evil)

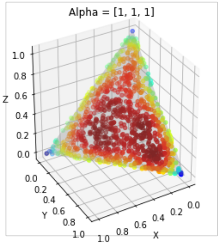
## Latent Drichlet allocation

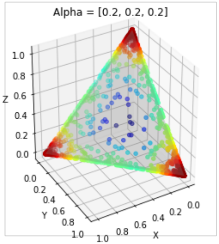


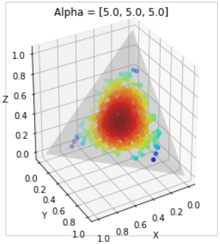
## LDA Model

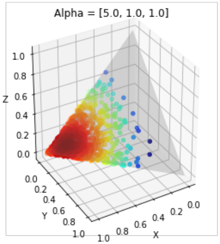
$\theta$ is a random matrix, where $\theta(i, j)$ represents the probability of the i th document to containing words belonging to the jth topic. $\theta$ is modeled as a Drichlet distribution (multivariate generalization of the beta distribution, parametrized by $\alpha \in \mathbb{R}^n$ ).
Similarly $\beta(i, j)$ represents the probability of the i th topic containing the j th word. $\beta$ is also a drichlet distribution.

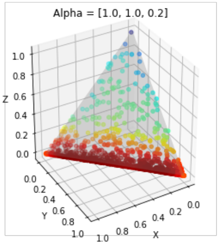
## Latent Drichlet allocation

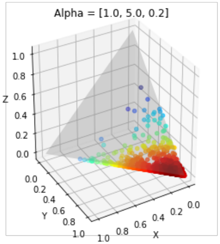












## Latent Drichlet allocation

We can describe the generative process of LDA as, given the M number of documents, N number of words, and prior K number of topics, the model trains to output:

- $\alpha$: Distribution related parameter that governs what the distribution of topics is for all the documents in the corpus looks like.
- $\theta$: Random matrix where $theta(i, j)$ represents the probability of the i th document to containing the j th topic
- $\eta$:Distribution related parameter that governs what the distribution of words in each topic looks like
- $\beta$: A random matrix where $beta(i, j)$ represents the probability of i th topic containing the j th word.

## LDA

We need to learn:

$$P(\theta_{1:M}, z_{1:M}, \beta_{1:k} | D; \alpha_{1:M}, \eta_{1:k})$$

I have a set of M documents, each document having N words, where each word is generated by a single topic from a set of K topics. I'm looking for the joint posterior probability of:

## LDA

- $\theta$ A distribution of topics, one for each document
- $z$ N topics for each document
- $\beta$ A distribution of words, one for each topic given
- $D$ All the data we have (i.e. the corpus, and using paramter
- $\alpha$ A parameter vector for each document (document — Topic distribution)
- $\eta$ A parameter vector for each topic (topic — word distribution)

## Logistic regression

Logistic regression can be used to classify an observation into one of two classes (like ‘positive sentiment’ and ‘negative sentiment’), or into one of many classes. The most important difference between naive Bayes and logistic regression is that logistic regression is a discriminative classifier while naive Bayes is a generative classifier.

## Generative vs. Discriminative classifiers

A generative model would have the goal of understanding how the classes look like. You might literally ask such a model to 'generate', an example of the class. A discriminative model, by contrast, is only trying to learn to distinguish the classes (perhaps without learning much about them).

## Generative vs. Discriminative models

A generative model like naive Bayes makes use of this likelihood term, which expresses how to generate the features of a document if we knew it was of class c. By contrast a discriminative model in this text categorization scenario attempts to directly compute $P(c|d)$. Perhaps it will learn to assign high weight to document features that directly improve its ability to discriminate between possible classes.

## Probabilistic classifiers

Like naive Bayes, logistic regression is a probabilistic classifier that makes use of supervised machine learning. Machine learning classifiers require a training corpus of M observations input output pairs $(x^{(i)}, y^{(i)})$.

## Probabilistic classifiers

A machine learning system for classification then has four components:

- A feature representation of the input. For each input observation $x^{(i)}$ , this will be a vector of features $[x_1, x_2, ..., x_n]$. We will generally refer to feature $(i)$ for input $x^{(j)}$ as $x_i^{(j)}$.
- A classification function that computes $\hat{y}$, the estimated class, via $p(y|x)$.
- An objective function for learning, usually involving minimizing error on training examples.
- An algorithm for optimizing the objective function.

## Logistic regression phases

**training**: we train the system (specifically the weights w and b) using stochastic gradient descent and the cross-entropy loss. **test**: Given a test example x we compute $p(y|x)$ and return the higher probability label y $=$ 1 or y $=$ 0.

## Logistic regression

Logistic regression perform is learning task, from a training set, using a vector of weights and a bias term. Each weight $w_i$ is a real number, and is associated with one of the input features $x_i$ . The weight $w_i$ represents how important that input feature is to the classification decision, and can be positive (meaning the feature is associated with the class) or negative (meaning the feature is not associated with the class). The bias term, also called the intercept.

$$z = (\sum_{i=1}^{n} w_i x_i) + b$$

## Logistic regression

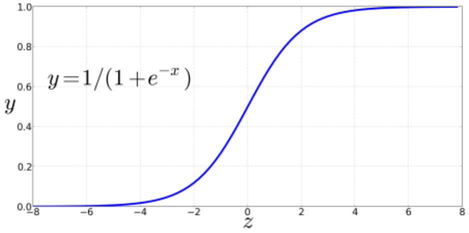
Logistic regression function can be represented as the dot ptoduct of the features and the weights adding the bias term:

$$z = w \cdot x + b$$

## Sigmoid function

To create a probability, we'll pass z through the sigmoid function, $\sigma(z)$. The sigmoid function (named because it looks like an s) is also called the logistic function, and gives logistic regression its name.

$$y = \sigma(z) = \frac{1}{1 + e^{-z}}$$



## Sigmoid function

The sigmoid function take a real-valued number and maps it into the range [0, 1], which is just what we want for a probability. Because it is nearly linear around 0 but has a sharp slope toward the ends, it tends to squash outlier values toward 0 or 1. And it's differentiable, which will be handy for learning. If we apply the sigmoid to the sum of the weighted features, we get a number between 0 and 1. To make it a probability, we just need to make sure that the two cases, $p(y = 1)$ and $p(y = 0)$, sum to 1.

$$P(y = 1) = \sigma(w \cdot x + b)$$

$$P(y = 0) = 1 - \sigma(w \cdot x + b)$$

## Sigmoid function

Now we have an algorithm that given an instance x computes the probability $P(y = 1|x)$. To make a decission for a test instance x, we say yes if the probability $P(y = 1|x)$ is more than .5, and no otherwise:

$$\hat{y} = \begin{cases} 1 \; if \;\; P(y = 1|x) > 0.5 \\ 0 \; otherwise \end{cases}$$

## Cost function

Rather than measure similarity, we usually talk about the opposite of this: the distance between the system output and the gold output, and we call this distance the loss function or the cost function. A natural way to think about the cost function is the mean square error loss:

$$L_{MSE} = \frac{1}{2}(\hat{y} - y)^2$$

## Cross entropy loss

It turns out that this MSE loss, which is very useful for some algorithms like linear regression, becomes harder to optimize (non-convex), when it's applied to probabilistic classification. Instead, we use a loss function that prefers the correct class labels of the training example to be more likely. This is called conditional maximum likelihood estimation: we choose the parameters w, b that maximize the log probability of the true y labels in the training data given the observations x. The resulting loss function is the negative log likelihood loss, generally called the cross entropy loss.

## Cross entropy loss

A perfect classifier would assign probability 1 to the correct outcome (y=1 or y=0) and probability 0 to the incorrect outcome. That means the higher $\hat{y}$ (the closer it is to 1), the better the classifier; the lower $\hat{y}$ is (the closer it is to 0), the worse the classifier. Thevnegative log of this probability is a convenient loss metric since it goes from 0 to infinity. This loss function also insures that as probability of the correct answer is maximized, the probability of the incorrect answer is minimized; since the two sum to one, any increase in the probability of the correct answer is coming at the expense of the incorrect answer.

$$L_{CE} = -[y\ log(\sigma(w \cdot x + b)) + (1 - y)\ log(1 - \sigma(w \cdot x + b))]$$

## Cross entropy loss

We'll define the cost function for the whole dataset as the average loss for each example:

$$Cost(w, b) = \frac{1}{m}\sum_{i=1}^{m} L_{CE}(\hat{y}^{(i)}, y)$$

$$= -\frac{1}{m}\sum_{i=1}^{m} y^{(i)}\ log(\sigma(w \cdot x^{(i)} + b) + (1 - y^{(i)})\ log(1 - \sigma(w \cdot x^{(i)} + b))$$

## Gradient descent

Our goal with gradient descent is to find the optimal weights: minimize the loss function we've defined for the model. We'll explicitly represent the fact that the loss function L is parameterized by the weights, which we'll refer to in machine learning in general as $\theta$ (in the case of logistic regression $\theta = w, b$):

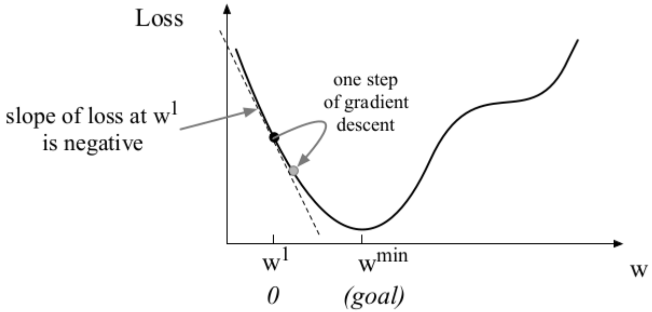
$$\hat{\theta} = \underset{\theta}{argmin} \frac{1}{m} \sum_{i=1}^{m} L_{CE}(\hat{y}^{(i)}, y; \theta)$$

## Gradient descent

Gradient descent is a method that finds a minimum of a function by figuring out in which direction (in the space of the parameters $\theta$) the function's slope is rising the most steeply, and moving in the opposite direction. For logistic regression, this loss function is conveniently convex. A convex function has just one minimum; there are no local minima to get stuck in, so gradient descent starting from any point is guaranteed to find the minimum.

## Gradient descent

Gradien descent algorithm:


Loss
slope of loss at w^1 is negative
one step of gradient descent
w^1
0
w^min
(goal)
w

## Gradient descent

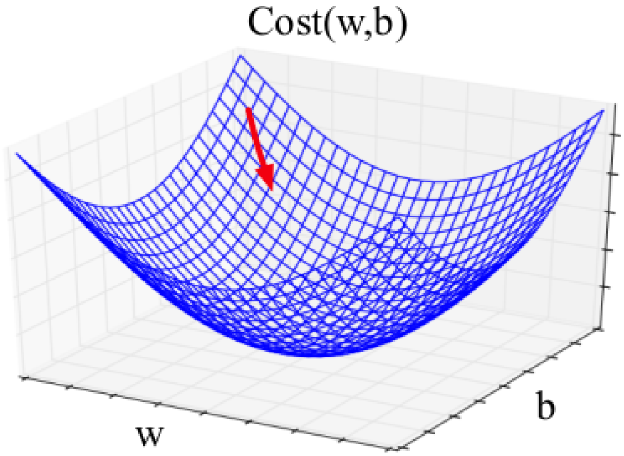
The gradient descent algorithm finds the gradient of the loss function at the current point and moves in the opposite direction. The gradient of a function of many variables is a vector pointing in the direction the greatest increase in a function. The gradient is a multi-variable generalization of the slope.

## Gradient descent

The gradient is just such a vector; it expresses the directional components of the sharpest slope along each of those N dimensions. If we're just imagining two weight dimension (say for one weight w and one bias b), the gradient might be a vector with two orthogonal components, each of which tells us how much the ground slopes in the w dimension and in the b dimension.

## Gradient descent

Gradien descent for 2 dimensional cost function:



## Gradient descent

The gradient of the loss function parametrized by $\theta$ is the following equation:

$$\nabla_\theta L(f(x;\theta),y)) = \begin{bmatrix} \frac{\partial}{\partial w_1} L(f(x;\theta),y)) \\ \frac{\partial}{\partial w_2} L(f(x;\theta),y)) \\ \vdots \\ \frac{\partial}{\partial w_n} L(f(x;\theta),y)) \end{bmatrix}$$

The rule to update the parameters is the following:

$$\theta_{t+1} = \theta_t - \eta \nabla_\theta L(f(x;\theta),y)$$

## Gradient for logistic regression

In order to update $\theta$, we need a definition for the gradient $\nabla_\theta L(f(x;\theta), y))$. Recall that for logistic regression, the cross entropy loss function is:

$$L_{CE}(w,b) = -[y\ log(\sigma(w \cdot x + b)) + (1-y)\ log(1-\sigma(w \cdot x + b))]$$

It turns out that the derivative of this function for one observation vector x is:

$$\frac{\partial L_{CE}(w,b)}{\partial w_j} = [\sigma(w \cdot x + b) - y]x_j$$

## Gradient for logistic regression

The loss for a batch of data or an entire dataset is just the average loss over the $m$ examples:

$$s^{(i)} = \sigma(w \cdot x^{(i)} + b)$$

$$Cost(w, b) = -\frac{1}{m}\sum_{i=1}^{m} y^{(i)}\ log(s^{(i)}) + (1 - y^{(i)})\ log(1 - s^{(i)})$$

And the gradient for multiple data points is the sum of the individual gradients:

$$\frac{\partial Cost(w, b)}{\partial w_j} = \sum_{i=1}^{m} [\sigma(w \cdot x^{(i)} + b) - y^{(i)}]x_j^{(i)}$$

## Learning rate

The magnitude of the amount to move in gradient descent is the value of the slope weighted by a learning rate $\eta$:

$$\frac{\partial}{\partial w} f(x; w)$$

A higher (faster) learning rate means that we should move $w$ more on each step. The change we make in our parameter is the learning rate times the gradient:

$$w_{t+1} = w_t - \eta \frac{\partial}{\partial w} f(x; w)$$

## Stochastic gradient descent algorithm

**Input:** $L$, Loss function, $f$ estimate function, $x$ inputs, $y$ labels
**Output:** $\theta$ parameters for f function, $LV$ vector of losses

```
1: $\theta \leftarrow 0$
2: **repeat** T times
3: **for all** $(x^{(i)}, y^{(i)})$ **do**
4:     $\hat{y}^{(i)} = f(x^{(i)}; \theta)$
5:     $LV_i = L(\hat{y}^{(i)}, y^{(i)})$
6:     $g \leftarrow \nabla_\theta L(f(x^{(i)}; \theta), y^{(i)})$
7:     $\theta \leftarrow \theta - \eta g$
8: **end for**
9: **return** $\theta$, $LV$
```

## Stochastic gradient descent

```
**function** STOCHASTIC GRADIENT DESCENT($L()$, $f()$, $x$, $y$) **returns** $\theta$
    # where: L is the loss function
    #       f is a function parameterized by $\theta$
    #       x is the set of training inputs $x^{(1)}, x^{(2)}, \ldots, x^{(n)}$
    #       y is the set of training outputs (labels) $y^{(1)}, y^{(2)}, \ldots, y^{(n)}$

$\theta \leftarrow 0$
**repeat** T times
  For each training tuple $(x^{(i)}, y^{(i)})$ (in random order)
  Compute $\hat{y}^{(i)} = f(x^{(i)};\theta)$    # What is our estimated output $\hat{y}$?
  Compute the loss $L(\hat{y}^{(i)}, y^{(i)})$   # How far off is $\hat{y}^{(i)}$) from the true output $y^{(i)}$?
  $g \leftarrow \nabla_\theta L(f(x^{(i)};\theta), y^{(i)})$   # How should we move $\theta$ to maximize loss ?
  $\theta \leftarrow \theta - \eta\, g$    # go the other way instead
return $\theta$
```

## Stochastic gradient descent example

Let's assume the following paramters: $x^{(1)} = [x_1, x_2] = [3, 2]$
$w_1 = w_2 = b = 0$
$\eta = 0.1$ The single update step requires that we compute the gradient multiplied by the learning rate:

$$\theta_{t+1} = \theta_t - \eta \nabla_\theta L(f(x^{(i)}; \theta), y^{(i)})$$

## Stochastic gradient descent example

In our mini example there are three parameters, so the gradient vector has 3 dimensions, for $w_1, w_2$, and $b$. We can compute the first gradient as follows:

$$\nabla_{w,b} = \begin{bmatrix} \frac{\partial L_{CE}(w,b)}{\partial w_1} \\ \frac{\partial L_{CE}(w,b)}{\partial w_2} \\ \frac{\partial L_{CE}(w,b)}{\partial b} \end{bmatrix} = \begin{bmatrix} (\sigma(w \cdot x + b) - y)x_1 \\ (\sigma(w \cdot x + b) - y)x_2 \\ \sigma(w \cdot x + b) - y \end{bmatrix}$$

$$= \begin{bmatrix} (\sigma(0) - 1)x_1 \\ (\sigma(0) - 1)x_2 \\ \sigma(0) - 1 \end{bmatrix} = \begin{bmatrix} -0.5x_1 \\ -0.5x_2 \\ -0.5 \end{bmatrix} = \begin{bmatrix} -1.5 \\ -1.0 \\ -0.5 \end{bmatrix}$$

## Stochastic gradient descent example

Now that we have a gradient, we compute the new parameter vector $\theta^2$ by moving $\theta^1$ in the opposite direction from the gradient:

$$\theta^2 = \begin{bmatrix} w_1 \\ w_2 \\ b \end{bmatrix} - \eta \begin{bmatrix} -1.5 \\ -1.0 \\ -0.5 \end{bmatrix} = \begin{bmatrix} 0.15 \\ 0.1 \\ .05 \end{bmatrix}$$

So after one step of gradient descent, the weights have shifted to be: $w_1 = .15$, $w_2 = .1$, and $b = .05$

## Regularization

There is a problem with learning weights that make the model perfectly match the training data. If a feature is perfectly predictive of the outcome because it happens to only occur in one class, it will be assigned a very high weight. The weights for features will attempt to perfectly fit details of the training set, in fact too perfectly, modeling noisy factors that just accidentally correlate with the class. This problem is called **overfitting**.

## Regularization

To avoid overfitting, a regularization term is added to the objective function:

$$\hat{w} = \underset{w}{argmax} \sum_{i=1}^{m} log(P(y^{(i)}|x^{(i)})) - \alpha R(w)$$

The new component, $R(w)$ is called a regularization term, and is used to penalize large weights. Thus a setting of the weights that matches the training data perfectly.

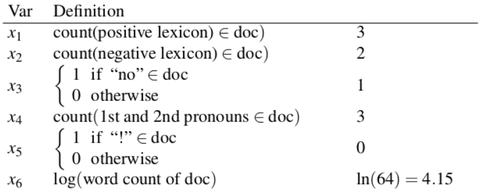
## L2 Regulrazation

L2 regularization is a quadratic function of the weight values, named because it uses the (square of the) L2 norm of weight values. The L2 regularized objective function becomes:

$$\hat{w} = \underset{w}{argmax}[\sum_{i=1}^{m} log(P(y^{(i)}|x^{(i)}))] - \alpha \sum_{j=1}^{n} w_j^2$$

## Logistic regression for sentiment analysis

Suppose we are doing binary sentiment classification on movie review text, and we would like to know whether to assign the sentiment class positive or negative to a review document doc. We'll represent each input observation by the following 6 features $x_1 \ldots x_6$ of the input.



| Var | Definition | |
|---|---|---|
| $x_1$ | count(positive lexicon) $\in$ doc) | 3 |
| $x_2$ | count(negative lexicon) $\in$ doc) | 2 |
| $x_3$ | $\begin{cases} 1 & \text{if “no”} \in \text{doc} \\ 0 & \text{otherwise} \end{cases}$ | 1 |
| $x_4$ | count(1st and 2nd pronouns $\in$ doc) | 3 |
| $x_5$ | $\begin{cases} 1 & \text{if “!”} \in \text{doc} \\ 0 & \text{otherwise} \end{cases}$ | 0 |
| $x_6$ | log(word count of doc) | $\ln(64) = 4.15$ |

## Logistic regression for sentiment analysis

We can build different set of features to represent the input, a common way to do it is using the n-hot encoded version of the example;

- 1.- Extract the number of words in the lexicon. Say k.
- 2.- Assign a unique index for each word of the lesicon
- 3.- For each example construct a vector of k zeros, except for the indexes of the present words set to ones.

n-hot enconded representations become useful where you have enough data.

## Assignments

Assignment 6: Implement a simple naive bayes classification method using the concept seen in class:

- Implement the Naive Bayes training method
- Implement the Naive classification function
- Perform a Naive Bayes classification using the twiter sentiment analysis database
- Write a Jupyter Notebook explaining your code

## Let’s code

## References

[1] Jurafsky D., Martin J.: Speech and Language Processing 2nd. ed. (2009).

[2] Bird S., Klein E., Loper, E.: Natural Language Processing with Python, (2009). ISBN: 978-0-596-51649-9

[3] Indurkhya N., Damerau F. Handbook of Natural Language Processing, Second Edition (2010). ISBN: 978-1-4200-8593-8

[4] Kao, A., Poteet S. Natural Language Processing and Text Mining (2007). ISBN: 78-1-84628-175-4

[5] Sipser, M. Introduction to the theory of computation (2013). ISBN: 978-1-133-18779-0

[6] https://www.sketchengine.eu/corpora-and-languages/corpus-types/