|  |  |  |
| --- | --- | --- |
|  |  | Natural Language processing  Author classification |

December 11

**Comparing a generative probabilistic & discriminative neural network**

PONER AQUI LOS TEXTOS QUE VAMOS A COMPARAR AUSTEN



Bárbara Flores

Daniela Jiménez

Shaila Güereca

Table of contents

[Objective 3](#_Toc152598133)

[Generative Probabilistic Model 3](#_Toc152598134)

[Model description 3](#_Toc152598135)

[Qualitatively and quantitatively results 3](#_Toc152598136)

[Discriminative Neural Network 4](#_Toc152598137)

[Model description 4](#_Toc152598138)

[Parameters of the model 5](#_Toc152598139)

[Cross – entropy loss function 6](#_Toc152598140)

[Qualitatively and quantitatively results 8](#_Toc152598141)

[Training real data 8](#_Toc152598142)

[Results of the Generative Probabilistic Model 8](#_Toc152598143)

[Results of the Discriminative Neural Network 8](#_Toc152598144)

[Training synthetic data 8](#_Toc152598145)

[Results of the Generative Probabilistic Model 8](#_Toc152598146)

[Results of the Discriminative Neural Network 8](#_Toc152598147)

[Pros and cons 8](#_Toc152598148)

[References 9](#_Toc152598149)

# Objective

The most important difference between Bayes and logistic regression is that logistic regression is **discriminative** classifier while naïve Bayes is a **generative** classifier. A generative model would have the goal of understanding what each text of the authors looks like and a discriminative model, by contrast, is only trying to distinguish between each author (perhaps with-out learning much about them).

# Generative Probabilistic Model

## Model description

## Qualitatively and quantitatively results

# Discriminative Neural Network

## Model description

For this discriminative neural network, we will be using logistic regression, this type of algorithm is the baseline supervised machine learning algorithm for classification. Logistic regression can be used to classify an observation into one of two classes (in this case two authors), or into many classes.

Logistic regression is a probabilistic classifier that makes use of supervised machine learning. Machine learning classifiers require a training corpus ofinput/output pairs . A machine learning system for classification has four components:

* A feature representation of the input. For each input observation , this will be a vector of features . We will refer to feature for input as .
* A classification function that computes , the estimated class, via .
* An objective function for learning, usually involving minimizing error on training examples. In this case the cross-entropy loss function.
* An algorithm for optimizing the objective function: the stochastic gradient descent.

Also, logistic regression has two phases:

*Training:* We will train the system (the weights and ) using stochastic gradient descent and

the cross-entropy loss.

*Test:* Given a test example we compute and return the higher probability label or

The goal of our logistic regression is to train a classifier that can make a binary decision about the class (author) of a new input observation and the sigmoid classifier will help us make that decision.

Consider a single input observation , which we will represent by a vector a of a features . The classifier output can be 1 (meaning the observation is a member of the class: Austen) or 0 (the observation is not a member or the class). We want to know the probability that this observation is a member of the class). So, the features represent counts of words in a document, is the probability that the document is from Austen and, is the probability that the document is from Caroll.

Logistic regression solves this by learning, from a training set, a vector of weights and a bias term. Each weight is a real number and is associated with one of the input features . The weight represents how important that input feature is to the classification decision, and can be positive (in this case, providing evidence that the instance being classified belongs to Austen) or negative (providing evidence that the instance being classified belongs to Carroll).

Regarding the bias term, we can also call it intercept, and is another real number that’s added to the weighted inputs.

To make a decision on a test instance – after we have learned the weights in training – the classifier first multiplies each by its weight , sums up the weighted features, and adds the bias term . The resulting single number expresses the weighted sum of the evidence for the class:

These sums will be represented by the dot product notation from linear algebra. The dot product of two vectors and , written as , is the sum of the products of the corresponding elements of each vector. Thus, the following is an equivalent equation from the previous formula:

But nothing in the previous equation forces to be a real probability, to lie between 0 and 1. In fact, since weights are real valued, the output might even be negative; ranges from - to

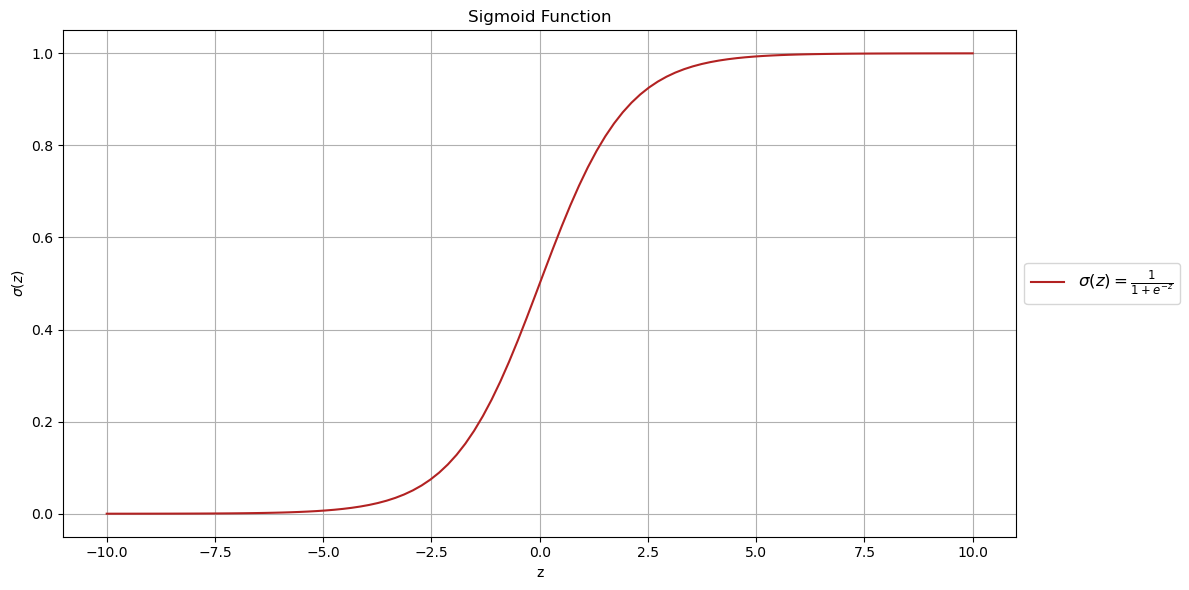


Figure 1: Range (0,1

To create a probability, we will pass throught the sigmoid function, . The sigmoid function is also called the logistic function and gives logistic regression its name. The sigmoid has the following equation:

This sigmoid has a number of advantages; it takes a real valued number and maps it into the range (0,1), which is just a probability. Because it is nearly linear around 0 but flattens towards, the ends, it tends to squash outlier values toward 0 or 1.

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 need to make sure that the two cases, and , sum to 1. We do this as follows:

The sigmoid function has the property:

So, we could have expressed:

The sigmoid function gives us an instance and a way to compute the probability but we also need a threshold to decide about which class to apply to a test instance , this is called the decision boundary.

## Parameters of the model

Logistic regression is an instance of supervised classification in which we know the correct label (either 0 or 1) for each observation . The systems produce a , the system’s estimate of the true *.* We want to learn parameters ( that make that make for each training observation as close as possible to the true y.

This requires two components, the first is a metric for how close the current label ( is to the true gold label (. For this, we usually use the distance between the system output and the gold output, and we call this distance the loss function or the cost function. In this case we will be using cross – entropy loss.

The second thing we need is an optimization algorithm for iteratively updating the weights so as to minimize this loss function. The standard algorithm for this is gradient descent.

## Cross – entropy loss function

We need a loss function that expresses, for an observation , how close the classifier output ( is to the correct output (, which is 0 or 1):

We do this via a loss function that prefers the correct class labels of the training examples to be more likely. This is called conditional maximum likelihood estimation: we choose the parameters that maximize the log probability of the true labels in the training data given the observation . The resulting loss function is the negative log likelihood loss, generally called the cross-entropy loss.

Let’s derive this loss function, applied to a single observation x. We’d like to learn weights that maximize the probability of the correct label p(y|x). Since there are only two discrete outcomes (1 or 0), this is a Bernoulli distribution, and we can express the probability p(y|x) that our classifier produces for one observation as the following:

If y = 1, the equation simplifies to ; if y = 0, simplifies to 1 − .

Now we take the log of both sides. Whatever values maximize a probability will also maximize the log of the probability:

log *p*(*y|x*) = log [

= *y* log + (1 *−y*) log()

The equation above describes a log likelihood that should be maximized. In order to turn this into a loss function (something that we need to minimize), we’ll just flip the sign on the equation. The result is the cross-entropy loss *L*CE:

*L*CE(*, y*) = *−* log *p*(*y|x*) = *−* [*y* log + (1 *−y*) log(1 *−* )]

Finally, we can plug in the definition of = *σ* (w *·* x + *b*):

*L*CE(*, y*) = *−* [ *y* log *σ* (w *·* x + *b*) + (1 *−y*) log(1 *−σ* (w *·* x + *b*))]

With this formula the loss function will be smaller if the model’s estimate is close to correct, and bigger if the model is confused.

We want to minimize this negative log probability because 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 if y equals 1, the higher is (the closer it is to 1), the better the classifier; the lower is (the closer it is to 0), the worse the classifier. If *y* equals 0, instead, the higher 1 *−*  is (closer to 1), the better the classifier. The negative log of (if the true *y* equals 1) or 1 *−*  (if the true *y* equals 0) is a convenient loss metric since it goes from 0 (negative log of 1, no

loss) to infinity (negative log of 0, infinite loss). This loss function also ensures that as the 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. It’s called the cross-entropy loss, because is also the formula for the **cross-entropy** between the true probability distribution *y* and our estimated distribution .

## Finding the minimum of the Cross – entropy loss function

Gradient descent will help us to find the optimal weights: minimize the loss function we’ve defined for the model. In the equation below, we will represent the fact that the loss function L is parameterized by the weights, which we will refer to in machine learning in general as θ (in the case of logistic regression θ = w, b). So, the goal is to find the set of weights which minimizes the loss function, averaged over all examples:

Gradient descent is a method that finds a minimum of a function by figuring out in which direction (in the space of the parameters θ) the function’s slope is rising the most steeply, and moving in the opposite direction. The intuition is that finds the direction where the ground is sloping the steepest, and ,move downhill in that direction.

For logistic regression, this loss function is conveniently convex. A convex function has at most one minimum; there are no local minima to get stuck in, so gradient descent starting from any point is guaranteed to find the minimum. (By contrast, the loss for multi-layer neural networks is non-convex, and gradient descent may get stuck in local minima for neural network training and never find the global optimum.)

Although the algorithm (and the concept of gradient) are designed for direction *vectors*, let’s first consider a visualization of the case where the parameter of our system is just a single scalar *w*:

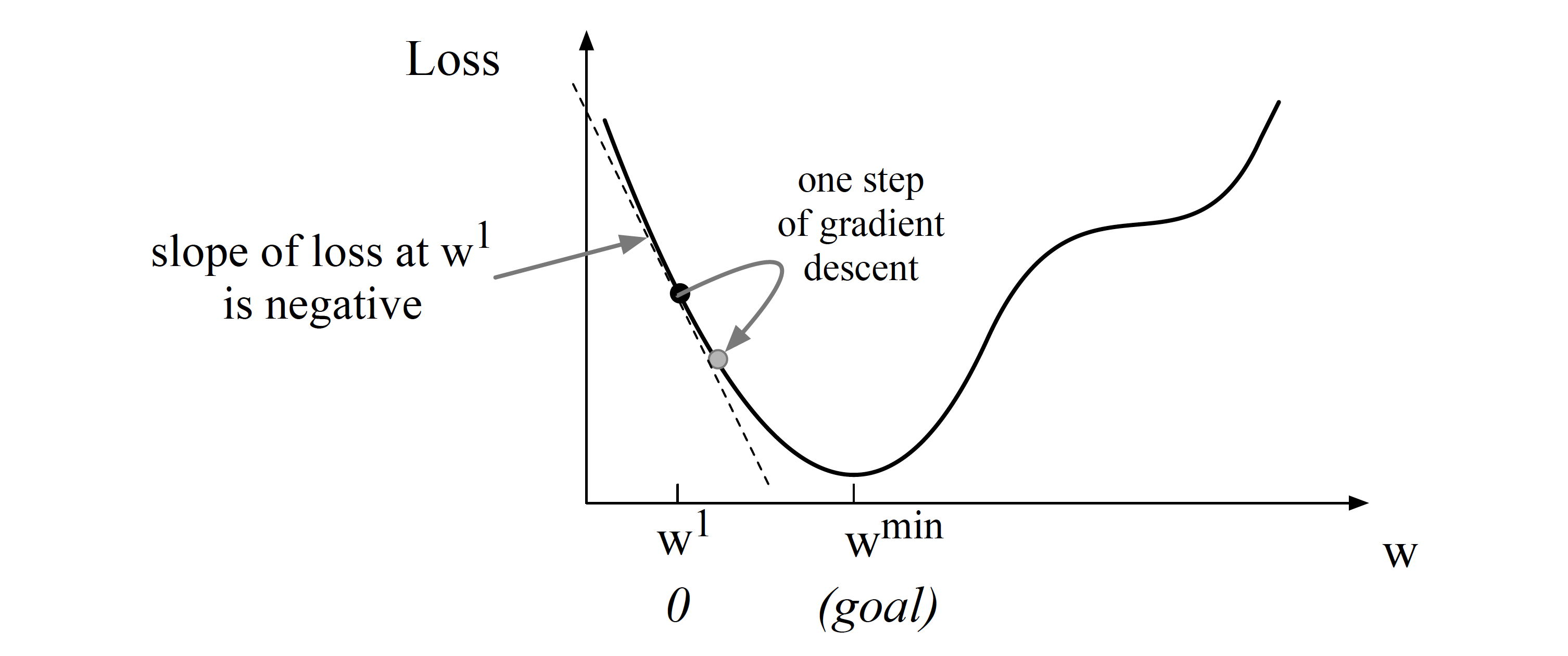


Figure 2: Finding the minimum in a loss function, by moving w.

Given a random initialization of *w* at some value *w*1, and assuming the loss function *L* happened to have the shape in the figure above, we need the algorithm to tell us whether at the next iteration we should move left (making *w*2 smaller than *w*1) or right (making *w*2 bigger than *w*1) to reach the minimum.

The gradient descent algorithm answers this question by finding the gradient of the loss function at the current point and moving in the opposite direction. The gradient of a function of many variables is a vector pointing in the direction of the greatest increase in a function. The gradient is a multi-variable generalization of the slope, so for a function of one variable like the one in the figure above, we can informally think of the gradient as the slope. The dotted line shows the slope of this hypothetical loss function at point *w* = *w*1. We can see that the slope of this dotted line is negative. Thus to find the minimum, gradient descent tells us to go in the opposite direction: moving *w* in a positive direction.

The magnitude of the amount to move in gradient descent is the value of the slope  *L*( *f* (*x*; *w*)*, y*) weighted by a learning rate ***η***. 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 (or the slope, in the example):

*dw*

*wt*+1 = *wt −η L*( *f* (*x*; *w*)*, y*)

Now we will see the intuition from a function of one scalar variable *w* to many variables, because we don’t just want to move left or right, we want to know where in the N-dimensional space (of the *N* parameters that make up *θ* ) we should move. The gradientis 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 dimensions (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.

*∂wi*

In an actual logistic regression, the parameter vector *w* is much longer than 1 or 2, since the input feature vector *x* can be quite long, and we need a weight *wi* for each *xi*. For each dimension/variable *wi* in *w* (plus the bias *b*), the gradient will have a component that tells us the slope with respect to that variable. In each dimension *wi*, we express the slope as a partial derivative of the loss function. Essentially, we’re asking: “How much would a small change in that variable *wi* influence the total loss function *L*?”

Formally, then, the gradient of a multi-variable function *f* is a vector in which each component expresses the partial derivative of *f* with respect to one of the vari- ables. We’ll use the inverted Greek delta symbol ∇ to refer to the gradient, and represent *y*ˆ as *f* (*x*; *θ* ) to make the dependence on *θ* more obvious:

**Stochastic gradient descentt***y*ˆ*y* (1 *− y*ˆ)1*−y*

## Qualitatively and quantitatively results

# Training real data

## Results of the Generative Probabilistic Model

## Results of the Discriminative Neural Network

# Training synthetic data

## Results of the Generative Probabilistic Model

## Results of the Discriminative Neural Network

# Pros and cons

Logistic regression has a number of advantages over naive Bayes. Naive Bayes has overly strong conditional independence assumptions. Consider two features which are strongly correlated; in fact, imagine that we just add the same feature f1 twice.

Naive Bayes will treat both copies of f1 as if they were separate, multiplying them both in, overestimating the evidence. By contrast, logistic regression is much more robust to correlated features; if two features f1 and f2 are perfectly correlated, regression will simply assign part of the weight to w1 and part to w2. Thus, when there are many correlated features, logistic regression will assign a more accurate probability than naive Bayes. So logistic regression generally works better on larger documents or datasets and is a common default.

Despite the less accurate probabilities, naive Bayes still often makes the correct classification decision. Furthermore, naive Bayes can work extremely well (sometimes even better than logistic regression) on very small datasets (Ng and Jordan, 2002) or short documents (Wang and Manning, 2012). Furthermore, naive Bayes is easy to implement and very fast to train (there’s no optimization step). So it’s still a reasonable approach to use in some situations.

# References

<https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.LogisticRegression.html>

<https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression>

<https://www.analyticsvidhya.com/blog/2021/09/creating-a-movie-reviews-classifier-using-tf-idf-in-python/#:~:text=Classifying%20documents%20in%20TF%2DIDF,SVM%20to%20predict%20document%20categories>.

<https://arxiv.org/pdf/1806.06407.pdf>

<https://www.kaggle.com/code/kashnitsky/logistic-regression-tf-idf-baseline>

https://stanford.edu/~shervine/teaching/cs-229/cheatsheet-supervised-learning