NATURAL LANGUAGE PROCESSING

Textual Crossroads: Classifying Documents between the Bible and Hamlet

December 11

Comparing a generative probabilistic & discriminative neural network

Bárbara Flores
Daniela Jiménez
Shaila Güereca

Table of contents

Objective

The objective of this study is to classify documents, distinguishing between the Bible and Hamlet, two emblematic literary works with markedly different styles and themes. To achieve this, we will employ a generative method like Naive Bayes and a discriminative method such as a Neural Network. Our aim is to identify the distinctive patterns that characterize these works, ultimately developing a robust classifier capable of discerning the subtle differences defining the essence of the Bible and the poetic expression of Hamlet. In exploring this to text, we aspire to not only contribute to a deeper understanding of the complexities that set apart these two timeless literary influences but also to comprehend the differences between these two methods of generative and discriminative classification.

Introduction

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).

The generative approach, exemplified by Naive Bayes, seeks to model the joint distribution of words in each text, providing insights into the unique linguistic features of the Bible and Hamlet. On the other hand, the discriminative approach, embodied by a Neural Network, focuses on learning the decision boundary that separates the documents based on their authorship.

Generative Probabilistic Model: Naive Bayes

Model description

The Naive Bayes model is a probabilistic classification algorithm based on Bayes' theorem and stands out for its simplicity and effectiveness in classification tasks, particularly when dealing with situations involving limited training data. The term "Naive" originates from the simplifying assumption of conditional independence among features, implying that each feature contributes independently to the probability of belonging to a specific class.

In the field of Natural Language Processing, the Naive Bayes model finds significant applications in document classification. This task especially benefits from the integration of the Bag of Words (BoW) representation, where each feature corresponds to a word. The BoW technique treats each document as an unordered "bag" of words, disregarding order and focusing on word frequency. This approach, though simple, is crucial for the effectiveness of the Naive Bayes model in document classification, enabling the efficient processing of large volumes of text and capturing relevant information through keywords.

Bayes' theorem

The Bayes' theorem is a mathematical proposition that describes the relationship between the probability of an event given the occurrence of another event, and the probability of that other event given the occurrence of the first one.

The formula for Bayes' Theorem in English is as follows:

$$P(A|B) = \frac{P(B|A) * P(A)}{P(B)}$$

Where:

- $P(A|B)$ is the conditional probability that event A occurs given that event B has occurred.
- $P(B|A)$ is the conditional probability that event B occurs given that A has occurred.
- $P(A)$ is the initial probability of event (A) occurring independently of (B).
- $P(B)$ is the initial probability of event (B) occurring independently of (A).

This formula expresses how our belief in the occurrence of event A is updated based on the occurrence of event B, considering the prior probabilities of A and B independently.

Naïve Bayes model

In the Naive Bayes model, we apply the principles of Bayes' theorem to the context of classification. Here, A typically represents the class to which an object or document belongs, while B represents the observed features of the object or document. To streamline the calculation, the Naive Bayes model assumes conditional independence between features given the classes, leading to the following simplified form of the formula:

$$p(x_1, x_2, \dots\dots x_T|C_i) \approx p(n_1, n_2, \dots . n_v|c_i)$$

$$\approx \prod_{k=0}^{V-1} p(n_k|c_i)$$

When it comes to classification, inferring $\boldsymbol{c_i}$ given $\boldsymbol{X}$, the probability is expressed as:

$$p(c_i|x_1, x_2, \dots\dots x_T) = p(c_i|n_1, n_2, \dots . n_v)$$

$$\propto p(c_i)\, p(n_1, n_2, \dots . n_v|c_i)$$

$$\approx p(c_i) \prod_{k=0}^{V-1} p(n_k|c_i)$$

Where:

- $p(c_i|x_1, x_2, \dots x_T)$ is the conditional probability of class $\boldsymbol{c_i}$ given the observed features $x_1, x_2, \dots x_T$
- $p(c_i|n_1, n_2, \dots n_v)$ is the conditional probability of class $\boldsymbol{c_i}$ given the counts of features $n_1, n_2, \dots . n_v$
- $p(c_i) \prod_{k=0}^{V-1} p(n_k|c_i)$ represents the product of the conditional probabilities of each feature count $n_k$given class $c_i$for all features in the dataset.

Discriminative Neural Network: Logistic Regression

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 of m input/output pairs $(x^{(i)}, y^{(i)})$. A machine learning system for classification has four components:

- A feature representation of the input. For each input observation $x^{(i)}$, this will be a vector of features $[x_{1,} x_2, \dots, x_n]$. We will refer to feature i for input $x^{(j)}$ as $x_i^{(j)}$.
- A classification function that computes $\hat{y}$, the estimated class, via $\rho(y|x)$.
- 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 w and b) using stochastic gradient descent and the cross-entropy loss.
>
> Test: Given a test example x we compute $\rho(y|x)$ and return the higher probability label $y = 1$ or $y = 0$

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 x, which we will represent by a vector a of a features $[x_{1,} x_2, \dots, x_n]$. The classifier output y can be 1 (meaning the observation is a member of the class: Hamlet) or 0 (the observation is not a member or the class). We want to know the probability $P(y = 1|x)$ that this observation is a member of the class). So, the features represent counts of words in a document, $P(y = 1|x)$ is the probability that the document is from Hamlet and, $P(y = 0|x)$ is the probability that the document is from the bible.
Logistic regression solves this by learning, from a training set, 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 (in this case, providing evidence that the instance being classified belongs to Hamlet) or negative (providing evidence that the instance being classified belongs to the bible).
Regarding the bias term, we can also call it intercept, and is another real number that's added to the weighted inputs.
To decide on a test instance – after we have learned the weights in training – the classifier first multiplies each x_I by its weight w_I, sums up the weighted features, and adds the bias term b. The resulting single number z expresses the weighted sum of the evidence for the class:

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

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

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

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

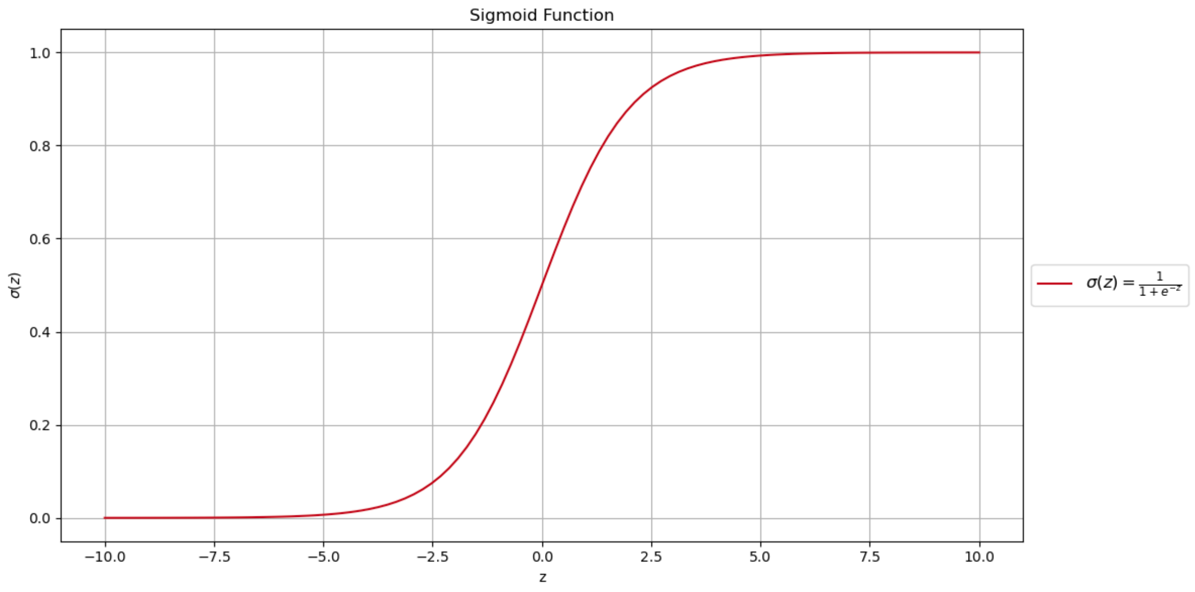


Figure 1: Range (0,1)

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

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

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, $P(y = 1)$ and $P(y = 0)$, sum to 1. We do this as follows:

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

$$P(y = 1) = \frac{1}{1 + \exp(-(w \cdot x + b))}$$

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

The sigmoid function has the property:

$$1 - \sigma(x) = \sigma(-x)$$

So, we could have expressed:

$$P(y = 0) = \text{as } \sigma(-(w \cdot x + b))$$

The sigmoid function gives us an instance x and a way to compute the probability $\mathrm{P}(y = 1|x)$ but we also need a threshold to decide about which class to apply to a test instance x, 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 y (either 0 or 1) for each observation x. The systems produce a $\hat{y}$, the system's estimate of the true y. We want to learn parameters (w and b) that make that make $\hat{y}$ 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 ($\hat{y}$) is to the true gold label (y). 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 x, how close the classifier output ($\hat{y} = \sigma(w \cdot x + b)$) is to the correct output (y, which is 0 or 1):

$$\mathrm{L}(\hat{y}, y) = \text{How much } \hat{y} \text{ differs from the true } y$$

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 w, b that maximize the log probability of the true y labels in the training data given the observation x. 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:

$$\mathrm{p(y|x)} = \hat{y}^{y}\,(1 - \hat{y})^{1-y}$$

If $y = 1$, the equation simplifies to $\hat{y}$; if $y = 0$, simplifies to $1 - \hat{y}$.
Now we take the log of both sides. Whatever values maximize a probability will also maximize the log of the probability:

$$\begin{aligned}\log p(y|x) &= \log\ \hat{y}^{y}\,[(1-\hat{y})^{1-y}] \\ &= y \log \hat{y} + (1-y)\log(1-\hat{y})\end{aligned}$$

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}:

$$L_{CE}(\hat{y}, y) = -\log p(y|x) = -[y \log \hat{y} + (1-y)\log(1-\hat{y})]$$

Finally, we can plug in the definition of $\hat{y} = \sigma(w \cdot x + b)$:

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

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 $\hat{y}$ is (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. If y equals 0, instead, the higher $1-\hat{y}$ is (closer to 1), the better the classifier. The negative log of $\hat{y}$ (if the true y equals 1) or $1-\hat{y}$ (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 sums 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 $\hat{y}$.

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:

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

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) is 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:

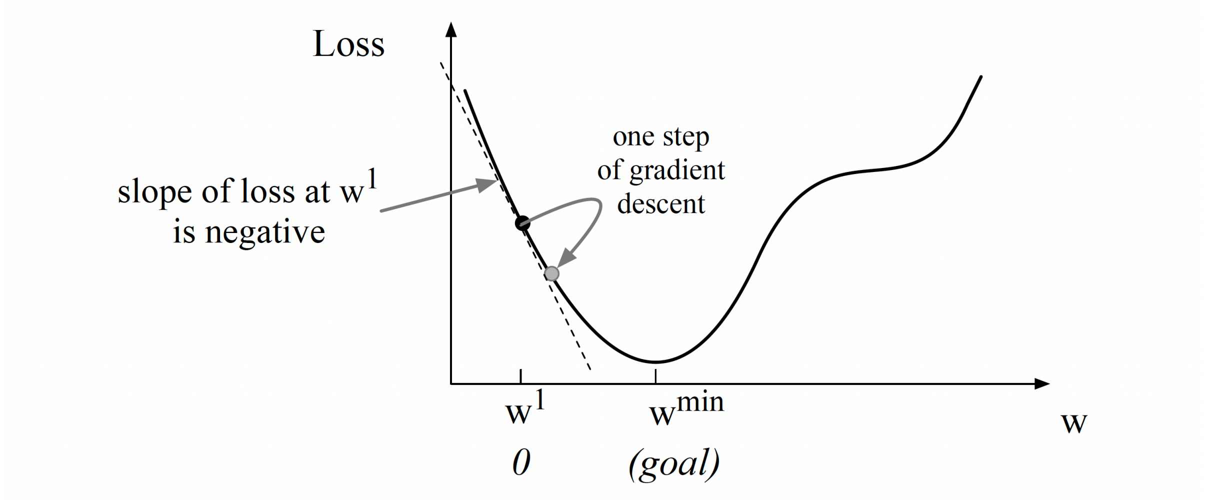

Loss
slope of loss at w^1 is negative
one step of gradient descent
w^1
0
w^{min}
(goal)
w

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 $\frac{d}{dw} L(f(x; w), y)$ 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):

$$w^{t+1} = w^t - \eta \frac{d}{dw} L(f(x; w), y)$$

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 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 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.

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 w_i for each x_i. For each dimension/variable w_i 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 w_i, we express the slope as a partial derivative $\frac{\partial}{\partial wi}$ of the loss function. Essentially, we're asking: "How much would a small change in that variable w_i influence the total loss function L?"

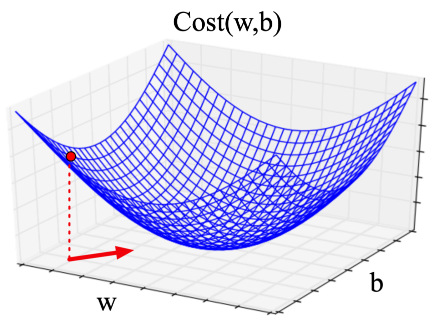
Formally, then, the gradient of a multi-variable function f is a vector in which each component expresses the partial derivative of f with respect to one of the variables. We'll use the inverted Greek delta symbol ∇ to refer to the gradient, and represent $\hat{y}$ as $f(x; \theta)$ to make the dependence on θ more obvious:

$$\nabla L(f(x;\theta),y) = \begin{bmatrix} \frac{\partial}{\partial w_1} L(f(x;\theta),y) \\ \frac{\partial}{\partial w_2} L(f(x;\theta),y) \\ \vdots \\ \frac{\partial}{\partial w_n} L(f(x;\theta),y) \\ \frac{\partial}{\partial b} L(f(x;\theta),y) \end{bmatrix}$$ (Jurafsky, 5)

Thus, the final equation for updating θ based on the gradient is: $\theta^{t+1} = \theta^t - \eta \nabla L(f(x; \theta), y)$

The gradient for logistic regression

To update θ we need a definition for the gradient $\nabla L(f(x;\ \theta),y)$. Recall that the cross-entropy loss function for logistic regression is: $L_{CE}(\hat{y},y) = -[y \log \sigma(w \cdot x + b) + (1-y)\log(1-\sigma(w \cdot x + b))]$



(Jurafsky,5)

The derivative of this function for one observation vector x is:

$$\frac{\delta L_{CE}(\hat{y},y)}{\delta w_j} = [\sigma(\mathbf{w} \cdot \boldsymbol{x} + b) - y]\boldsymbol{x_j} \quad = \quad (\hat{y} - y)\boldsymbol{x_j} \quad or \quad -(y - \hat{y})\boldsymbol{x_j}$$

In these equations, the gradient with respect to a single weight $\boldsymbol{w_j}$ represents an intuitive value: the difference between the true y and our estimated $\hat{y} = \sigma(\mathbf{w} \cdot \boldsymbol{x} + b)$ for that observation, multiplied by the corresponding input value $\boldsymbol{x_j}$

Training real data

Results of the Generative Probabilistic Model

In the generative probabilistic model, specifically Naive Bayes, we applied a method that models the joint distribution of words in each text. The goal was to understand the distinctive linguistic patterns of Hamlet and The Bible independently. The model operates under the naive assumption that the features (words in this context) are conditionally independent given the class.

Upon **training** the Naive Bayes model, we achieved high accuracy, with a correct classification rate of 99.62% on the training set. This can be observed in the following confusion matrix:

	Predicted: Hamlet	Predicted: The Bible
Actual: Hamlet	2,807	8
Actual: The Bible	13	2,763

Confusion Matrix with Naïve Bayes of the training data (90% of the total dataset)

Now let's analyze the outcomes from our **testing dataset**, representing 10% of the total sentences.

	Predicted: Hamlet	Predicted: The Bible
Actual: Hamlet	290	1
Actual: The Bible	2	328

Confusion Matrix with Naïve Bayes of the testing data (10% of the total dataset)

In the **testing** phase, we achieved an accuracy of 99.52%, a slightly lower but still commendable result compared to the training data. Now let's examine specific instances of misclassification.

It's interesting to note instances where our Naive Bayes model misclassified statements:

“wot ye not that such a man as I can certainly divine?“

is indeed from the Bible but was incorrectly predicted as Hamlet.

“By and by easily is said.”

is originally from Hamlet but was mistakenly classified as the Bible.

“The King James Bible”.

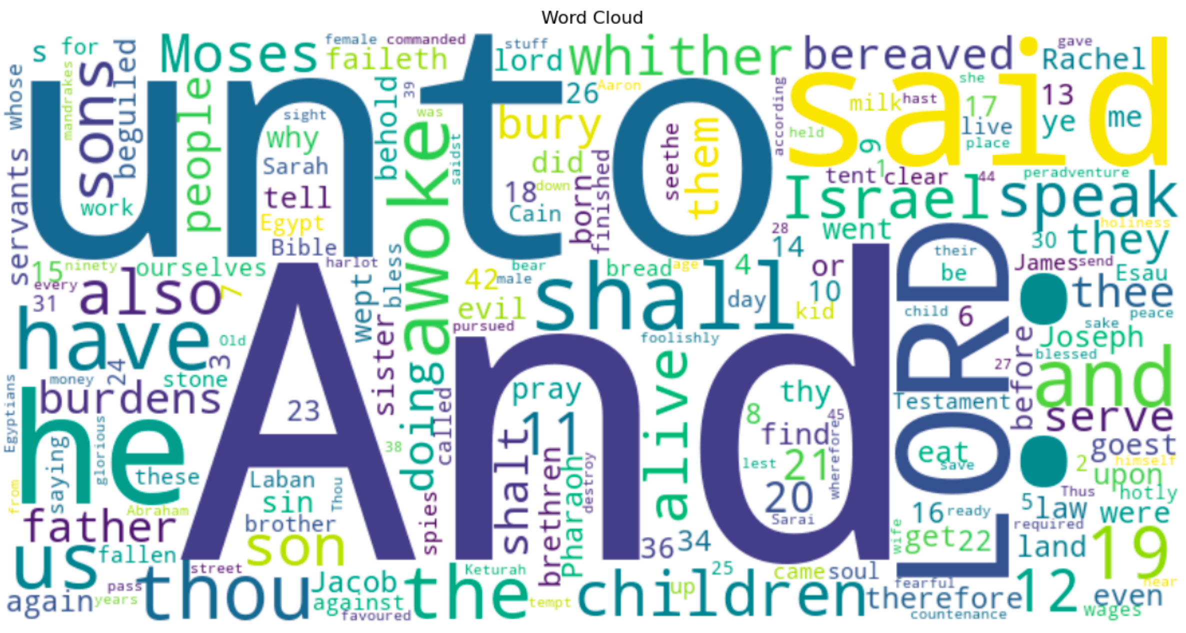
Is from the Bible, yet it was incorrectly classified as Hamlet.

The misclassification of such short sentences, given the simplistic nature of Naive Bayes, is understandable. The inherent limitations associated with the brevity and complexity of the sentences pose notable challenges for the model. However, this exercise provides valuable insights into the nuances of natural language processing and the complexities that arise when attempting to differentiate between distinctive literary works such as Hamlet and the Bible.

Results of the Discriminative Neural Network

We will be using logistic regression model to distinguish between two distinct classes: Hamlet and The Bible. The model estimates the probability that an input belongs to a particular class by fitting a logistic function to the input features. To enhance our analysis, we'll employ the TF-IDF (Term Frequency-Inverse Document Frequency) technique. TF-IDF assigns significance to words based on their frequency in a document relative to their occurrence across the entire corpus. This transformation facilitates a numerical representation of textual data, emphasizing the importance of specific words while diminishing the impact of common terms.

We performed a qualitative evaluation of our model by creating a word cloud image which visualizes the TF-IDF weights. This word cloud portrays the relative importance or weights of different words in the dataset based on their TF-IDF scores, offering a visual representation of the text features' significance within the specified class. The following graph provides a graphical insight into the importance of words as determined by their TF-IDF values in the training dataset:



If we review the accuracy we obtain from our **training data,** we can observe that it is very high, correctly classifying 99.98% of the cases. Upon closer inspection, we can see the following confusion matrix:

	Predicted: Hamlet	Predicted: The Bible
Actual: Hamlet	2,815	0
Actual: The Bible	1	2,775

Confusion Matrix with Logistic Regression of the training data (90% of the real dataset)

It's very interesting to see that the only incorrect classification we have, where the statement is truly from the Bible but is being predicted as Hamlet, is the following:

"tell me them, I pray you."

But why is it predicting it incorrectly? This is because in Hamlet, there is a sentence that contains almost the same words:

"speak the speech, I pray you,"

The incorrect classification is completely understandable because even for a human, it would be difficult to distinguish which author is correct for the statement.

Now let's review the results of our **testing data**; for this dataset, we used 10% of the total sentences.

For the test data, we obtained an accuracy of 98.87%, slightly lower compared to that of the training data, but still quite good for an initial accuracy.

If we look at the confusion matrix for the test data, we can see that we got a total of 7 misclassifications, 3 of which incorrectly classified a text from Hamlet and 4 of which incorrectly classified a text from the Bible:

	Predicted: Hamlet	Predicted: The Bible
Actual: Hamlet	288	3
Actual: The Bible	4	326

Confusion Matrix with Logistic Regression of the test data (10% of the real dataset)

Let's analyze an example of each of the incorrect classifications. We'll start with the following statement, which was classified as if it were part of Hamlet but is actually from the Bible:

“The King James Bible”

To analyze this statement, let's recap what Hamlet is about. The story commences with the appearance of the ghost of Hamlet's father, King Hamlet, who discloses that he was murdered by his brother, now King Claudius. Claudius, who has wedded Hamlet's mother, Queen Gertrude, urges Hamlet to seek revenge for his father's death. With little further explanation, the word "King" is extremely prevalent in Hamlet, which could be the reason it was categorized as belonging to that text.

Now let's look at a phrase that was incorrectly classified as belonging to the Bible but is actually from Hamlet.

"So be it"

The phrase is so short and lacks context, making it very difficult to determine which author it belongs to, even for a human.

Training synthetic data

We generated synthetic data reminiscent of Shakespearean writing and biblical text using our generative Naive Bayes model. Each set contains 1,000 sentences with an average length of 15 words by sentence. We used class-specific probability distributions to choose words for each sentence, ensuring that the generated sentences reflect the word usage patterns of each class. It's worth mentioning that since we are using probabilities for the creation of synthetic data, different executions of the code will yield different results.

Let's look at the first 4 sentences we have from this synthetic data:

N°	Sentence	Label
1	Arm What . danger their Is Whore No the Ham Then in Ajah prenominate	Hamlet
2	Ah be down pittie , : the and he . , : shalt and house	Bible
3	men burnt is eat the , made spake laide the him loins ' of drink	Bible
4	the say ranck him behoue weepes . insolence , - their Carpenter me loues shall	Hamlet

As we can see, the sentences have no logical order because Naive Bayes does not care about the word order. After generating our synthetic data for both types of text, we proceeded to train and apply it to our two models. The results can be observed in the following section.

Results of the Generative Probabilistic Model

Upon evaluating the performance on our training data, it's notable that we achieve a remarkably high accuracy of 99.78%, signifying the model's ability to correctly classify most instances. For a more detailed breakdown, let's delve into the corresponding confusion matrix.

	Predicted: Hamlet	Predicted: The Bible
Actual: Hamlet	906	4
Actual: The Bible	0	890

Confusion Matrix with Naïve Bayes of the training data (90% of the synthetic dataset)

In the evaluation of our Naive Bayes model on the synthetic testing data, we achieve an accuracy of 99.5%. This is substantiated by the following confusion matrix, providing a detailed breakdown of the model's predictions:

	Predicted: Hamlet	Predicted: The Bible
Actual: Hamlet	89	1
Actual: The Bible	0	110

Confusion Matrix with Naïve Bayes of the testing data (10% of the synthetic dataset)

An interesting observation arises when examining specific instances of misclassification. For example, the next sentence:

“, . : And Rood inhabitants inobled is it keepeth paint re the there turneth”

originally from Hamlet, was misclassified as belonging to The Bible. The misclassification of the sentence could be attributed to several factors. One possible reason is the presence of common words or phrases that may occur in both Hamlet and The Bible, leading to ambiguity in the classification. Additionally, the model might encounter challenges when dealing with less common or archaic words that are unique to the literary style of Shakespearean works, making it susceptible to misclassifications in certain instances.

Additionally, it's worth noting that achieving a higher accuracy on the synthetic data compared to real data is somewhat expected. Training on synthetic data, which closely aligns with the generative model from Step 2a, may lead to a level of overfitting. Overfitting occurs when the model becomes too tailored to the training data, capturing noise or specific characteristics that might not generalize well to real-world scenarios. While the high accuracy on synthetic data is encouraging, it's essential to consider the model's performance on real-world data to assess its robustness and generalization capabilities.

Results of the Discriminative Neural Network

When we observe the performance of Logistic Regression within the synthetic data, we achieve a 100% accuracy in the training data, despite employing L2 as a penalty term. However, this high accuracy might indicate potential overfitting issues, wherein the model overly depends on specific features or patterns exclusive to the training data, which could stem from insufficient **training data**. Insufficient training data might hinder the model's ability to acquire comprehensive knowledge and form strong generalizations.

	Predicted: Hamlet	Predicted: The Bible
Actual: Hamlet	910	0
Actual: The Bible	0	890

Confusion Matrix with Logistic Regression of the testing data (90% of the synthetic dataset)

Despite the model exhibiting overfitting, its performance against the **testing** data is notably good, correctly classifying 98% of the instances. This high accuracy can be attributed to a clear distinction between the styles of the two authors, as captured by the probability distribution. This distinction likely furnishes the model with consistent information for classification. Additionally, the selected TF-IDF features might effectively encapsulate the stylistic disparities between the authors' writing, empowering the model to discern and differentiate between them more effectively.

	Predicted: Hamlet	Predicted: The Bible
Actual: Hamlet	88	2
Actual: The Bible	2	108

Confusion Matrix with Logistic Regression (10% of the synthetic dataset)

These were the 4 sentences that were classified incorrectly:

N°	Synthetic Sentences	Based on
1	*“ . And For I Most and at confine cubits made may men that to you”*	Bible
2	*“ I Philistines because doore fifteenth it lest pillars the then thirty”*	Bible
3	*“ , . : And Rood inhabitants inobled is it keepeth paint re the there turneth”*	Hamlet
4	*“ After Anckle Funerall accent beshrew betimes gentle judged to was wrong”*	Hamlet

Generative vs. Discriminative Approaches: Weighing Pros and Cons

In contrasting logistic regression and Naive Bayes, distinctive advantages and considerations emerge. Logistic regression demonstrate superiority in handling correlated features, assigning weights more accurately, especially in scenarios with numerous interrelated variables, making it a common default choice for larger datasets (Ng and Jordan, 2002). Despite providing less accurate probabilities, Naive Bayes can still make correct classification decisions and excels in simplicity, speed, and effectiveness on smaller datasets or short documents (Wang and Manning, 2012). Its ease of implementation and quick training without an optimization step make it a reasonable approach in specific situations.

Turning to the evaluation of the Naive Bayes generative probabilistic model, its simplicity and efficiency are evident in achieving a high accuracy of 99.52% during real data testing. However, its reliance on the assumption of conditional independence among features becomes apparent in misclassifications, particularly with short and contextually complex sentences. Synthetic data evaluation reveals high accuracy of 99.5% in testing data, showcasing effectiveness in capturing generative patterns (Ng and Jordan, 2002). Yet, caution is warranted due to potential overfitting in synthetic datasets. Naive Bayes excels in scenarios with smaller datasets and straightforward patterns but may struggle in nuanced linguistic contexts.

On the other hand, logistic regression, a discriminative neural network model, exhibits robust performance, achieving an outstanding accuracy of 99.87% during real testing on larger datasets (Kashnitsky, 2018). Its ability to handle correlated features showcases resilience, though computational complexity emerges as a potential drawback. The testing phase maintains commendable accuracy at 98% in testing data, emphasizing model consistency. Logistic regression's balanced probability estimation makes it reliable for discerning linguistic patterns in diverse datasets, even though it may face challenges with computational demands (Scikit-learn Documentation).

Both models demonstrate high accuracy, with strengths and weaknesses that highlight the challenges in distinguishing intricate literary works. The synthetic data experiments underscore potential overfitting issues, emphasizing the need to validate model robustness on real-world datasets. In conclusion, the choice between Naive Bayes and logistic regression hinges on data nature, size, and the desired balance between simplicity and accuracy. Each model offers a valuable tool in natural language processing, contributing to our understanding of their applicability in distinct scenarios.

References

Amidi, A., & Amidi, S. (n.d.). Supervised learning. Stanford University. https://stanford.edu/~shervine/teaching/cs-229/cheatsheet-supervised-learning

Das, B., & Chakraborty, S. (2018). An improved text sentiment classification model using TF-IDF and next-word negation. https://arxiv.org/pdf/1806.06407.pdf

Kashnitsky, Y. (2018). Logistic regression TF-IDF baseline. Kaggle. https://www.kaggle.com/code/kashnitsky/logistic-regression-tf-idf-baseline

Majumder, P. (2023). Creating a movie reviews classifier using TF-IDF in Python. Analytics Vidhya. 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.

Scikit-learn. (n.d.). Logistic regression. Scikit-learn Documentation. https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.LogisticRegression.html

Scikit-learn. (n.d.). Linear models. Scikit-learn Documentation. https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression

Wang, P., (2023). classification_example_logistic_regression.py [code]. Pratt School of Engineering, Duke University. https://sakai.duke.edu/access/content/group/dc8e8288-c9b6-459c-a335-ce6a804c174f/lecture_code/classification_example_logistic_regression.py