ML/DL fundamentals

Prerequisites for ML and DL: part 1

NULL SPACE

0.1 Entropy

Entropy is a measure of information. Consider that we are given a random variable X over our variable of interest (or a variable we would like to predict). In the NLP context it could be **words, letters, POS**. Further we can denote the range of values taken by this random variable as χ. This random variable X is associated with a particular probability function $p(x)$. So, the **entropy** of this random variable X can be given by:

$$H(X) = -\sum_{x \in \chi} p(x) \log_2 p(x) \tag{1}$$

This entropy is traditionally measured in **bits**. Entropy could be thought of as *a lower bound on the number of bits it would take encode a piece of information in the optimal coding scheme*.

0.1.1 An example

Suppose we want to bet on a horse race with eight horses but we convey our message about which horse to bet on, to the bookie, via short messages. Basically we could use the **binary representation** of the horse number as the code. Under this scheme, horse 1 would be 001, horse 2 would be 010 and so on. Since each horse is coded in 3 bits we would be sending 3 bits per race. We will now attempt to do better than this. We can represent the spread of bets placed as the **prior probability** of each horse in the following manner:

Horse 1: $1/2$, Horse 5: $1/64$
Horse 2: $1/4$, Horse 6: $1/64$
Horse 3: $1/8$, Horse 7: $1/64$
Horse 4: $1/16$, Horse 8: $1/64$

With this if we calculate entropy as per the previous formula we can get the lower bound on the number of bits required to send the message:

$$H(X) = -\sum_{i=1}^{8} p(i) \log_2 p(i) \tag{2}$$

$$= -\frac{1}{2}\log\frac{1}{2} - \frac{1}{4}\log\frac{1}{4} - \cdots - \frac{1}{64}\log\frac{1}{64} = 2 \text{ bits} \tag{3}$$

Now we note that a code that averages 2 bits per race can be constructed with short encodings for **more probable horse** and longer encodings for **less probable horses**. The most likely horse could get 0 as a code, followed by $10, 110, 1110$ and so on for the rest.

0.2 A note on classification

The goal of classification is take an observation, extract some of its useful features and then **classify** the observation into one among a set of discrete classes. This is a type of **supervised learning** method wherein we have a dataset of input observations, each associated with some correct output, or a **supervision signal** of sorts. The goal then is to take a new observation and map it to a correct class. It takes an input x and returns a predicted class label y.

0.2.1 Explaining classification from an NLP context

We denote d as a document and c as its associated class. Let us say we have a training dataset of N documents wherein each document is hand labeled with its class - $(d_1, c_1), (d_2, c_2), \cdots, (d_N, c_N)$. Our goal then is to learn a classifier that is able to map a new document d to its correct class $c \in C$. **probabilistic classifiers** tell us the probability of an observation belonging to a certain class as well. Note that **Naive Bayes** is a **Generative classifier** that builds a model of how a class could generate data; it tells us that given an observation, what class would have most likely generated the observation. On the other hand **logistic regression** is more like a **discriminative classifier** that learns the important features from the input data that are most useful to discriminate between different classes.

0.2.2 Introduction to Logit

Logistic regression is a probabilistic classifier that requires a training corpus of M input/ouput pairs $(x^{(i)}, y^{(i)})$. Note that the superscripts denote individual instances in the dataset. Noting down the main components for a Machine learning classification task.

- A **feature representation** of the input is required. For each observation $x^{(j)}$ this could be a vector of features given by: $[x_1, \cdots, x_n]$. Typically, feature i for input j would be denoted as $x_i^{(j)}$.
- The classification function essentially computes the estimated class $\hat{y}$ by computing the conditional probability $p(y|x)$. Note that the **sigmoid** and **softmax** are important tools for classification.
- An objective function is typically used for learning, wherein some measure of error is minimized. We often consider the **cross entropy loss function** here.

- Finally we need an algorithm for optimizing the objective function as well. Typically the **Stochastic gradient descent** is used.
- **Training** involves determining the weights w and b using SGD and Cross entropy loss.
- **Testing** involves computing $p(y|x)$ for a new x and returning the higher probability label y.

0.3 The sigmoid in Classification

As mentioned before as well, our goal is to train a classifier so as to make a **binary decision** about the class of a new input observation. In this matter, the **sigmoid** is used. First consider a single input observation x represented by a vector of features $[x_1, x_2, \cdots, x_n]$. The classifier output could be either 1 or 0. Our aim is to know the probability $P(y = 1|x)$.

Now we note that Logistic regression solves this task by learning a vector of **weights** and a **bias term**. Each weight w_i is a real number, associated with each feature x_i. This weight represents the importance of a particular feature in the classification decision - a high (positive) value would mean that it is very strongly associated with a class and a low (negative) value means it is not. Additionally, the **bias** term can be thought of as the **intercept** and is yet another real number added to the weighted inputs.

After the weights have been learnt, in order to make a decision on a test instance - the classifier multiplies each feature with its corresponding weight, sums up the weighted features and then adds the bias term as well. The resulting number z expresses the weighted sum of evidence for a particular class:

$$z = \left(\sum_{i=1}^{n} w_i x_i \right) + b \tag{4}$$

This could be expressed in the form of dot product notation as well, as follows:

$$z = \boldsymbol{w} \cdot \boldsymbol{x} + b \tag{5}$$

Note that this number could be anything between $-\infty$ and ∞ and is certainly not a probability measure yet. Now to convert z into a probability we pass it through the **sigmoid** function $\sigma(z)$, also known as the **logistic function** given by:

$$y = \sigma(z) = \frac{1}{1 + e^{-z}} \tag{6}$$

This function essentially takes a real values input and maps into the range $[0, 1]$, which is ideal to get a probability measure. Now what we have until now is a

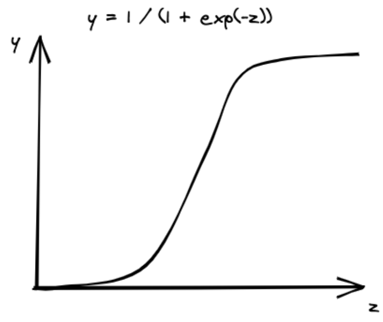

Figure 1: Sigmoid function

sigmoid function that our sum of weighted features is a value between 0 and 1. For this to be a true probability we need to ensure that the two exhaustive cases $P(y=1)$ and $P(y=0)$ actually sum to 1. Therefore we have the following:

$$P(y=1) = \sigma(\boldsymbol{w} \cdot \boldsymbol{x} + b) = \frac{1}{1 + e^{-(\boldsymbol{w} \cdot \boldsymbol{x} + b)}} \tag{7}$$

$$P(y=0) = 1 - \sigma(\boldsymbol{w} \cdot \boldsymbol{x} + b) = 1 - \frac{1}{1 + e^{-(\boldsymbol{w} \cdot \boldsymbol{x} + b)}} = \frac{e^{-(\boldsymbol{w} \cdot \boldsymbol{x} + b)}}{1 + e^{-\boldsymbol{w} \cdot \boldsymbol{x} + b}} \tag{8}$$

As a last step, we can set the **decision boundary** as 0.5. This essentially means that if $P(y=1)$ is greater than 0.5 then we make the decision to assign that observation a label of 1.

References

[1] Daniel Jurafsky, James H Martin - Speech and Language Processing