Natural Language Processing (NLP) Fundamentals: Maximum Entropy (MaxEnt)

SEPTEMBER 5, 2016   3 COMMENTS

This blog post is part of a series, titled Natural Language Processing (NLP) Fundamentals (https://nadesnotes.wordpress.com/2016/04/03/nlp-fundamentals/).

> **Disclaimer**: I am a developer at Amazon.com for their Alexa technologies, on devices such as Amazon Echo (http://www.amazon.com/echo). As a consequence, there may be hints of Amazon sprinkled throughout this blog series. However, I should note that this blog series is not sponsored

> by Amazon.com, and represent my opinions only, and not those of the company.

As humans, we are used to making decisions. Many times, we are given several pieces of information with which to base our decisions. How do you "piece everything together" to come to a final decision?

Consider for a moment that you are not a human, but a computer. Fast! Imagine how you would answer the following question: "Fried chicken or dog?"

You might notice that the dogs in the picture have certain characteristics, and that the fried chicken might have other characteristics.

In this next post, I will introduce Maximum Entropy (MaxEnt) classification.

Definition

Here's Wikipedia's definition on maximum entropy classification (https://en.wikipedia.org/wiki/Multinomial_logistic_regression) (or, MaxEnt for short):

> "[Maximum entropy classification] is a classification method that generalizes logistic regression to multiclass problems, i.e.

> with more than two possible discrete outcomes."

This is a pretty good formal definition, but I imagine that if I were to read this for the first time without knowing anything about MaxEnt, I might get confused. So here's a hand-wavy explanation that might provide some intuition.

If we are given some data (e.g., a picture) and are faced with making a decision (e.g., "is this a picture of a dog or fried chicken?"), we could think of attributes about the data (e.g., "is there anything resembling a leash in the picture?", "are there 2 black dots that could look like eyes?"). Probably feeling a bit fancy, smart folks like to call these attributes "features". Some of these features might matter more than others.

"Maximum Entropy Classification" is also a fancy name (those silly smart folks (https://www.quora.com/What-is-maximum-entropy-in-the-simplest-terms/answer/Anjan-Nepal)!) for throwing these "features" into a particular kind of math equation, and using that math equation to tell us if we should make a decision. For each feature (e.g., "has a leash", "has two dots that resemble eyes", etc.) that is found for the data, a weight is applied to it, and all of the features are added up. Finally, the weighted sum is normalized to give a fraction between 0 or 1. We can use this fraction to tell us the "score" of how confident we might be in making a decision (e.g., "is this a dog?").

"Alright Nade, enough about fried chicken and dogs. I want to see an NLP example!" No problem, hypothetical reader! I'll do just that, before more formally diving into the math 🙂

An Example

Imagine that it is the year 1999, and you are one of the first employees of a new email startup. You are asked to read several emails, and for each email, it is your job to determine whether or not it is spam.

Email #1:

> From: LeChuck@yahoo.com
>
> Har har!
>
> Allow mes to introduce myself. Myname is Lechuck ,and I got your email from the interwebs mail directory. I threw a dart at

the directory and hit your name. You seam like a good lad based on your name, so I here I am writing this email.

I live out here in this faraway island, and I have some moneys ($122K USD, to be exact) that I need to send overseas. Could you do mes a favor and help me with my moneys transfer?

1) Provide mes a bank account where this money would be transferred to.

2) Send me a picture of yourselfs so I know who to look for and thank when I sail to the US. Click heres to my Facebook [www.lechuck.myfacebook.com/fake.link/give_me_money] and post me your pic.

Monkeys bananas money rich

As reward, I are willing to offer you 15% of the moneys as compensation for effort input after the successful transfer of this fund to your designate account overseas. please feel free to contact ,me via this email address
lechuck@yahoo.com

Email #2:

From: GuybrushThreepwood@gmail.com

Hi Nade,

This is Guybrush from James Logan High School. It's been a while! How have you beens?

I heard from Elaine that you wuld be visiting Melee Island soon, so I thought I should check in to see if u wanted to meet up. Are you avalable for dinner on Friday, September 8? We could catch up and hang out, like old times. I know a bar that serves some horrible grog.

Let me know! If we can't meet up, then no problem; I hope you enjoy your visit!

Cheers,

Guybrush

Easy, right? Great. How would a computer figure this out?

Let's start out with determining some features that might be useful here. Here's a quick list of a few:

1. $f_1(email)$: Email contains spelling/grammatical errors
2. $f_2(email)$: Email asks for money to be transferred
3. $f_3(email)$: Email mentions account holder's name

Let's say that some of these features matter more than the others. Let's give the following weights for each of these features, for when they indicate spam:

1. $w_1(spam)$ (Email contains spelling/grammatical errors): 0.5. Seriously, proofread your emails.
2. $w_2(spam)$ (Email asks for money to be transferred): 0.2. Many email scams somehow involve some kind of bank transfer.
3. $w_3(spam)$ (Email mentions account holder's name): -0.5. If the sender mentions me by name, then maybe it isn't spam. By the way, notice that weights can be negative for a decision.

Here are the weights for each of the features indicating NOT spam:

1. $w_1(notspam)$ (Email contains spelling/grammatical errors): -0.2
2. $w_2(notspam)$ (Email asks for money to be transferred): 0
3. $w_3(notspam)$ (Email mentions account holder's name): 1

(These weights were arbitrarily chosen, just for the sake of discussion. I might explain in a separate post how weights are learned using training data. For now though, perhaps you can satisfy some of your interest by looking up gradient descent (https://en.wikipedia.org/wiki/Gradient_descent). There are other, fancier methods too, that are used in more practical settings, in particular the L-BFGS algorithm (https://en.wikipedia.org/wiki/Limited-memory_BFGS). Also, Andrew Ng also does an awesome machine learning Coursera course (https://nadesnotes.wordpress.com/2016/07/28/learn-machine-learning-with-professor-andrew-ng-on-coursera/), and happens to discuss gradient descent in some fantastic lecture videos; here's one of the videos (https://www.coursera.org/learn/machine-learning/lecture/8SpIM/gradient-descent))

Alright, we have our features, and we have our weights. We're pretty much ready to dive into the math now.

The most scary part about the math equations below is probably that they involve exponents, but after that really just involves multiplication, addition, and division. Alright, let's dive in! 🙂

The Maths. Har Har

Let a feature function, $f_i(x)$, take in an input, x, and return either 0 or 1, depending if the feature is present in x:

$$f(x) = \begin{cases} 1, & \text{if the feature is present in } x \\ 0, & \text{otherwise} \end{cases}$$

Furthermore, for N features, associate each feature function $f_i(x)$ with a weight $w_i(d)$, which is a number that denotes how "important" $f_i(x)$ is compared to other features for a decision, d (In this case, spam or not spam).

We can "model" (in my opinion, this word could be understood as "estimate") the score of a decision d on input x using the following procedure:

1. For each $f_i(x)$ in a set of N features, determine if $f_i(x)$ should be 1 or 0
2. Multiply each $f_i(x)$ with the associated weight $w_i(d)$, which depends on the decision d being evaluated.
3. Add up all of the weight*feature pairs: $sum_d = \sum_{i=1}^{N} w_i(d) * f_i(x)$
4. Throw the sum up into an exponent: $numerator_d = \exp(sum_d)$
5. Divide the sum by a number that will range the score between 0 and 1, and such that the sum of scores across all decisions is 1. It turns out that this is the sum of the numerators for every possible decision d: $denominator = \sum_d \exp(\sum_{i=1}^{N} w_i(d) * f_i(x))$

The procedure above is pretty much the equation below:

$$Score_d(x) = \frac{\exp(\sum_{i=1}^{N} w_i(d) * f_i(x))}{\sum_d \exp(\sum_{i=1}^{N} w_i(d) * f_i(x))}$$

Let's think about how each of the emails would score.

Email #1: LeChuck

Here's the email again, for reference:

> From: LeChuck@yahoo.com
>
> Har har!

> Allow mes to introduce myself. Myname is Lechuck ,and I got your email from the interrwebs mail directory. I threw a dart at the directory and hit your name. You seam like a good lad based on your name, so I here I am writing this email.
>
> I live out here in this faraway island, and I have some moneys ($122K USD, to be exact) that I need to send overseas. Could you do mes a favor and help me with my moneys transfer?
>
> 1) Provide mes a bank account where this money would be transferred to.
>
> 2) Send me a picture of yourselfs so I know who to look for and thank when I sail to the US. Click heres to my Facebook [www.lechuck.myfacebook.com/fake.link/give_me_money] and post me your pic.
>
> Monkeys bananas money rich
>
> As reward, I are willing to offer you 15% of the moneys as compensation for effort input after the successful transfer of this fund to your designate account overseas. please feel free to contact ,me via this email address
> lechuck@yahoo.com

Let's see what features match up.

1. f_1: Email contains spelling/grammatical errors. YES.
2. f_2: Email asks for money to be transferred. YES.
3. f_3: Email mentions account holder's name. NO.

It's somewhat obvious here what we'll classify this as, but let's at least go through the exercise.

Spam Score

$$Score_{spam}(email_{LeChuck}) = \frac{\exp(\sum_{i=1}^{N} w_i(spam) * f_i(email_{LeChuck}))}{\sum_d \exp(\sum_{i=1}^{N} w_i(d) * f_i(email_{LeChuck}))} =$$

$$\frac{\exp(0.5*1+0.2*1-0.5*0)}{\exp(0.5*1+0.2*1-0.5*0)+\exp(-0.2*1+0*1+1*0)} = 0.71$$

Not Spam Score

$$Score_{notspam}(email_{LeChuck}) = \frac{\exp(\sum_{i=1}^{N} w_i(notspam) * f_i(email_{LeChuck}))}{\sum_d \exp(\sum_{i=1}^{N} w_i(d) * f_i(email_{LeChuck}))} =$$

$$\frac{\exp(-0.2*1+0*1+1*0)}{\exp(0.5*1+0.2*1-0.5*0)+\exp(-0.2*1+0*1+1*0)} = 0.29$$

LeChuck's spam score (0.71) is higher than the not spam score (0.29), so this is spam. No surprise there. Sorry LeChuck.

Email #2: Guybrush

Here's Guybrush's email below, for reference:

> From: GuybrushThreepwood@gmail.com
>
> Hi Nade,
>
> This is Guybrush from James Logan High School. It's been a while! How have you beens?
>
> I heard from Elaine that you wuld be visiting Melee Island soon, so I thought I should check in to see if u wanted to meet up. Are you avalable for dinner on Friday, September 8? We could catch up and hang out, like old times. I know a bar that serves some horrible grog.
>
> Let me know! If we can't meet up, then no problem; I hope you enjoy your visit!
>
> Cheers,
>
> Guybrush

Once again, let's see what features match up.

1. f_1: Email contains spelling/grammatical errors. Yes… "beens"? "u"? "avalable"? C'mon Guybrush.
2. f_2: Email asks for money to be transferred. No.
3. f_3: Email mentions account holder's name. Yes.

Looks like Guybrush isn't a perfect email writer, but let's see if our system will classify his note as spam.

Spam Score

$$Score_{spam}(email_{Guybrush}) = \frac{\exp(\sum_{i=1}^{N} w_i(spam) * f_i(email_{Guybrush}))}{\sum_d \exp(\sum_{i=1}^{N} w_i(d) * f_i(email_{Guybrush}))} =$$

$$\frac{\exp(0.5*1+0.2*0-0.5*1)}{\exp(0.5*1+0.2*0-0.5*1)+\exp(-0.2*1+0*1+1*1)} = 0.31$$

Not Spam Score

$$Score_{notspam}(email_{Guybrush}) = \frac{\exp(\sum_{i=1}^{N} w_i(notspam)*f_i(email_{Guybrush}))}{\sum_d \exp(\sum_{i=1}^{N} w_i(d)*f_i(email_{Guybrush}))} =$$

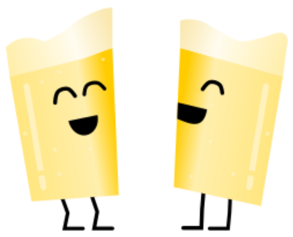
$$\frac{\exp(-0.2*1+0*1+1*1)}{\exp(0.5*1+0.2*0-0.5*1)+\exp(-0.2*1+0*0+1*1)} = 0.69$$

Guybrush's spam score (0.31) is lower than his not spam score (0.69), so the system would not classify Guybrush's email as spam. Looks like I'm going to go meet with him! … and drink some horrible grog.

Adding More Decisions

You may have noticed that MaxEnt actually supports scores for multiple (>=2) decisions. Here, adding another decision isn't difficult, but going through every step in detail will end up leading to an even longer post, so I'll just summarize what you would have to do.

Imagine that we want to also decide if an email comes from a newsletter, kind of like:

> From: info@ScummBar.com
>
> 
>
> Ahoy mateys!
>
> From 9/10 to 9/17, enjoy 50% off on all grog and other horrible beverages at Scumm Bar. Click here [www.scummbar.fake.link.com] to see our full drink selection, as well as our autumn seasonal foods.
>
> See you soon!
>
> The Idiots at Scumm Bar

To add this decision, we would need to do the following:

- Define additional features that would select for newsletter-type content (e.g., contains image, mentions discounts, email address belongs to a known business, etc.)
- For each decision (spam, not spam, and newsletter), re-determine weights for features and biases, if any.

Applications

MaxEnt classification is one of the more classical machine learning tasks, and solves problems beyond natural language processing. Here are a few:

- Sentiment analysis (e.g., given a product review, what does the reviewer like and dislike about the product?)
- Preferences (e.g., Given a person's demographics, who will a person vote for? Would they prefer Superman, Batman, or the Teenage Mutant Ninja Turtles? etc.)
- Diagnosis (e.g., Given characteristics of several medical images and patient history, what medical condition is a person at risk of having?)

Real-World Usage

The Python-based Natural Language Toolkit (NLTK) (http://www.nltk.org/) provides a library for Maximum Entropy classification (http://www.nltk.org/api/nltk.classify.html?highlight=maxent#module-nltk.classify.maxent).

The demo looks at a list of names (~8000) and uses a handful of single letter-based features to determine if the name is male or female.

Invoking the demo itself is simple:

```
In [1]: from nltk.classify import maxent
In [2]: maxent.demo()
```

Output:

```
Training classifier…
==> Training (100 iterations)

Iteration … Log Likelihood … Accuracy
——————————————
```

```
1 … -0.69315 … 0.374
2 … -0.61426 … 0.626
3 … -0.59970 … 0.626
4 … -0.58597 … 0.627
5 … -0.57305 … 0.633

[…]

96 … -0.33073 … 0.809
97 … -0.33030 … 0.809
98 … -0.32989 … 0.809
99 … -0.32948 … 0.810
Final … -0.32908 … 0.810

Testing classifier…
Accuracy: 0.7980
Avg. log likelihood: -0.6085

Unseen Names P(Male) P(Female)
——————————————-
Octavius *0.9342 0.0658
Thomasina 0.0478 *0.9522
Barnett *0.6464 0.3536
Angelina 0.0077 *0.9923
Saunders *0.7839 0.2161
```

(For the extra curious, the demo happens to be learning weights using Improved Iterative Scaling (http://www.cs.cmu.edu/~aberger/pdf/scaling.pdf), or IIS. NLTK's MaxEnt Classification training (http://www.nltk.org/api/nltk.classify.html?highlight=maxent#nltk.classify.maxent.MaxentClassifier.ALGORITHMS) also supports GIS (https://en.wikipedia.org/wiki/Generalized_iterative_scaling), MEGAM (http://www.umiacs.umd.edu/~hal/megam/index.html), and TADM (http://tadm.sourceforge.net/))

I probably would not say that 80% is remarkably accurate, though I am intrigued, considering the surprisingly simple feature set. Here is the list of features being used:

```python
def names_demo_features(name):
    features = {}
    features['alwayson'] = True
    features['startswith'] = name[0].lower()
    features['endswith'] = name[-1].lower()
    for letter in 'abcdefghijklmnopqrstuvwxyz':
        features['count(%s)' % letter] = name.lower().
        features['has(%s)' % letter] = letter in name.
    return features
```

Notice that the features here contain a bias feature ("always on"), and then simply checks the start/end letters of a name, and what letters are in the name.

Conclusion

MaxEnt is a useful and easy-to-understand tool to help computers make decisions based off of "features" on your data. These features, once properly weighted, feed into scores for making each decision. MaxEnt also provides a good introduction for logistic regression / classification problems, and is likely used for several practical, real-world problems today.

Finally, understanding MaxEnt opens the doors toward understanding Maximum Entropy Markov Models (MEMMs) and Conditional Random Fields (CRFs).

More Reading

Christopher Manning – Maxent Estimation and Discriminative Models (https://web.stanford.edu/class/cs124/lec/Maximum_Entropy_Classifiers.pdf) (lecture slides)

Charles Sutton and Andrew McCallum – An Introduction To Conditional Random Fields for Relational Learning (pp. 4-5) (https://people.cs.umass.edu/~mccallum/papers/crf-tutorial.pdf) (35 page tutorial actually on CRFs, but with good introductory material on MaxEnt and MEMMs)

Christopher Potts, Sentiment Symposium Tutorial: Classifiers (Maximum Entropy) (http://sentiment.christopherpotts.net/classifiers.html#maxent)

Kamal Nigam, John Lafferty, Andrew McCallum – Using Maximum Entropy For Text Classification (http://www.kamalnigam.com/papers/maxent-ijcaiws99.pdf) (paper)

Natural Language Toolkit (NLTK) – MaxEnt Classifier (http://www.nltk.org/api/nltk.classify.html?highlight=maxent#module-nltk.classify.maxent) (code documentation)

Questions? Comments?