LEXALYTICS

WHITE PAPER

Machine Learning for Natural Language Processing and Text Analytics

Lexalytics, Inc., 48 North Pleasant St. Unit 301, Amherst MA 01002 USA | 1-800-377-8036 | **www.lexalytics.com**

Introduction

Machine learning is everywhere, from helping us make better toast to researching drug discovery and designs. Sometimes the term is used interchangeably with artificial intelligence (AI), but they're not the same thing. While all AI involves machine learning, not all machine learning is AI.

Lexalytics' core text analytics engine, Salience, can be considered a "narrow" AI: It uses many different types of machine learning to solve the task of understanding and analyzing text, but is focused exclusively on text. We'll be looking at the machine learning and natural language processing (NLP) elements that Salience is built upon.

We'll discuss the different aspects of text analytics and how Lexalytics, a company with more than a decade of experience in machine learning, applies machine learning to solve problems in natural language processing.

3 KINDS OF TEXT ANALYTICS SYSTEMS

1. Rules-based (pure NLP)
2. Machine learning-based (pure ML)
3. Hybrid (a combination of ML and NLP)

For further reading, you can consult our white papers "Build vs. Buy," which talks about the economics of machine learning in a text analytics context, and "Tune First, Then Train," which discusses our philosophy of customization for better accuracy and more-relevant results. When taken together with this paper, these resources offer a more complete view of text analytics solutions.

TABLE OF CONTENTS

Machine teaching (aka learning)

creates a learning framework and provides data that the machine can learn from.

MACHINE LEARNING IS REALLY MACHINE TEACHING

Before we start delving into the different aspects of text analytics, let's clarify some basic machine learning concepts.

Most importantly, "machine learning" really means "machine teaching." We know what the machine needs to learn, so our task is to create a learning framework and provide properly-formatted, relevant, clean data that the machine can learn from.

The goal is to create a system where the model **continuously improves** at the task you've set it. Input is key. Unlike algorithmic programming, a machine learning model is able to generalize and deal with novel cases. If a case resembles something the model has seen before, the model can use this prior "learning" to evaluate the case.

When we talk about a "model," we're talking about a mathematical representation. **A machine learning model is the sum of the learning that has been acquired from the training data.** The model changes as more learning is acquired.

3 MAJOR PARTS TO MACHINE LEARNING

1. Training data
2. Model algorithm
3. Hyper-parameters

The output of this system is a machine learning model.

If you were baking a cake:

- the training data would be the ingredients
- the time and temperature would be the hyper-parameters
- the cake would be the model

Lexalytics Hyper-Parameter Optimization Video | 3:35

Once the model is created (baked), we can run it against new data to evaluate what it's learned, and whether further adjustments are needed.

However, making adjustments isn't just a matter of writing a line of code that tells the model what to do. That kind of direct approach is known as "**algorithmic programming**" – what most people call "coding." With machine learning, we need to convince the model that it wants to do what we want it to do.

Algorithmic programming

is simply a matter of writing a line of code that tells the model what to do.

Writing a line of code is clearly the more precise, concise approach – and one that's going to almost certainly be less work than machine learning. We talk about this in the white paper "Tune First, Then Train."

However, coding isn't always the right solution. Machine learning is much better than coding at dealing with novel cases and learning from the experience.

In the next section we'll review the main classes of machine learning.

SUPERVISED, UNSUPERVISED, AND SEMI-SUPERVISED MACHINE LEARNING

There are three relevant classes of machine learning: supervised learning, unsupervised learning, and semi-supervised learning. Lexalytics uses all three depending on the problem we're trying to solve.

Supervised learning

Supervised learning means feeding a machine learning model a dataset that has been annotated in some way. For example, we might collect 10,000 customer support comments and mark them up based on which are related to software and which are related to hardware. In doing so, we're showing the machine what information it needs to evaluate each comment.

Supervised learning

means feeding a machine learning model an annotated dataset.

This is the most direct way of teaching a model what you want it to do. It's also the most work. At Lexalytics, we use supervised learning for NLP tasks like sentiment analysis and for certain methods of categorization.

For example, we train sentiment analysis models on hand-scored examples because the perspective of the sentiment analysis can change based on context. Consider the following:

"SuperBank lost US$100,000,000 last month."

Well, were they expected to lose US$200,000,000? US$50,000,000? The sentiment of this statement very much depends on who is looking at it.

Another example would be "This perfume smells like my grandmother." Do you love your grandmother?

Ultimately, any extraction that requires that the machine understand your perspective needs to be supervised somehow, and this requires lots of work.

Unsupervised learning

is where the machine takes content and is told to find patterns within it.

Semi-supervised learning

is the combination of unsupervised and supervised learning.

Unsupervised learning

Unsupervised learning is where we hand the machine a whole bunch of content and tell it to find the patterns. This is how we built the syntax parser in Salience: We took 40GB of text and had the parser analyze every sentence to understand how subjects and verbs fit together. Consider the following:

> **"I threw the ball over the mountain."**

One way to understand syntax is to parse the entire sentence, like you're doing a sentence diagram from 6th grade. Those are quite computationally intensive (along with being irritating for 6th graders), and so you can't do that for high-volume content – it just takes too long for each document to process.

But what if you were to process a bunch of content ahead of time to come up with a set of relationships that shows how words like "ball," "threw" and "mountain" were typically related across millions and billions of sentences.

As a human, you naturally know that it is far more likely that "threw" is acting on "ball," than it is likely that "threw" is acting on "mountain." You don't throw mountains, you throw balls.

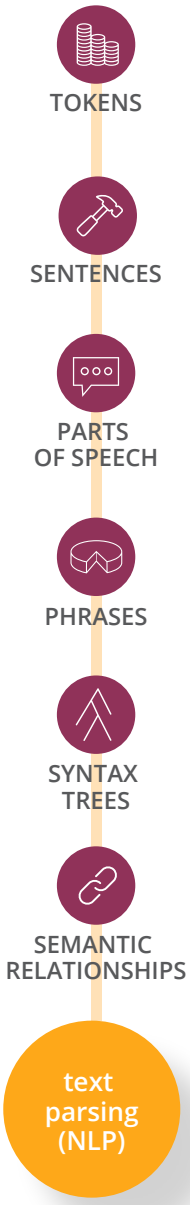
That sort of probabilistic relationship can be extracted using unsupervised learning. The syntax matrix was an excellent candidate for unsupervised learning, as it involved discovering generally applicable patterns from a very large corpus of content. Because it is a matrix, it can be evaluated really fast for each sentence, unlike a full parser.

As the amount of content created every day grows exponentially, unsupervised techniques become more and more valuable.

Semi-supervised learning

Semi-supervised learning is a combination of unsupervised and supervised learning techniques. With this approach we'll have both marked-up supervised content and un-marked data. The machine learning model uses the marked-up content to generalize and make assertions about the rest of the data.

Now that we've reviewed the machine learning essentials, let's look at how to combine machine learning and algorithmic natural language processing to build a high-performing text analytics AI.



Depending on what's optimal for the language, each of these steps is machine learning or NLP code.

HAPPIER BY THE DOZEN: THE MORE MODELS, THE MERRIER

Machine learning models are very good at performing single tasks, such as determining the sentiment polarity of a document or the part-of-speech for a given word. However, models are not good at tasks that require layers of interpretation. Take the following sentence:

> **"Lexalytics is the best text analytics company ever."**

Besides agreeing with its obvious truth, what might we want to know about this sentence? First, we want to know whether it contains any entities (companies, people, products and so on). Second, we want to know whether there's any sentiment associated with those entities. Third, we want to know whether a particular industry is being discussed. Finally, we might ask whether any sentiment is being expressed towards that industry.

One single machine learning model can't do all of that. You'll need at least four separate models:

1. Identify and name any entities (Lexalytics)
2. Determine the sentiment associated with that entity (positive)
3. Industry classification (text analytics)
4. Industry sentiment (neutral)

If you just train a single model, you can only solve #1 or #3. Calculating the sentiment needed for #2 or #4 requires first knowing which entity you're trying to associate the sentiment with. If you only have a single model for sentiment, you'll end up rating the whole sentence as positive. Additionally, if you're only using keywords to look for the term "text analytics," you'll rate this sentence as positive for that phrase, which isn't true.

AI Assembler (our proprietary software)

builds NLP machine learning models and manages dependencies between them.

Not only do you need at least four models to solve this task, but these models are interdependent and have to interact with each other. To create this kind of multi-model solution, we developed proprietary AI building software.

This tool, "AI Assembler," is used to build our features like sentiment, named entity extraction, intention analysis and more. We also use AI Assembler to build custom machine learning models used by our customers and partners. Among other things, AI Assembler manages dependencies between models, allowing us to easily upgrade one model and then re-build other models as necessary.

Multi-level granular sentiment analysis is difficult due to the model complexity and dependencies. Lexalytics is one of the few companies that actually provides this service – most companies simply provide document sentiment and call it done. Truth is, solving for entity and category sentiment is very difficult, and multiplies the amount of work required. We do it because our customers are making business critical decisions, and they need context-rich insights to make informed decisions that drive business growth.

To borrow from another industry, imagine if you have a wonderful spy satellite. Do you want to just be able to say, "There are a bunch of people there" or, "There's a known terrorist there, and he's holding a gun?"

Practical efficiencies:

When we need NLP code or rules, we write them; when we need models, we build them.

CODING VS. LEARNING: MAKING THE CASE FOR EACH

Let's use the same sentence for this next example. Let me hear you say:

"Lexalytics is the best text analytics company ever."

Now look at the period at the end of the sentence. Periods are important in English because they frequently denote the end of a sentence. It's important to be able to break sentences apart so that you can figure out which statements go together.

However, periods also denote other things, like "Dr." for doctor, or "Mr." for mister. How can a machine tell whether the period is denoting the end of a sentence or a form of address? We could train a machine learning model for this task by marking up examples of each. But this isn't necessarily the most efficient approach. After all, there are only a few cases in the English language where the period is used for anything other than denoting the end of a sentence. It is more efficient, faster computationally, and more precise to just hard-code these cases using NLP code or other algorithms.

Where there are cases that are best handled by NLP code, we write NLP code or use rules. When we need to build models, we build models.

BLACK BOX/CLEAR BOX: LOOKING INSIDE THE DATA

It is important to understand not just what decision a model has made, but **why it has made that decision.** There are two reasons for this:

1. There's often other **business-relevant information** encoded in the "why." For example, knowing simply that certain survey results are full of negative feedback is not particularly useful. We want to know **which** phrases were scored negatively.

2. We might want to **adjust the scoring** somehow. If we can't see why the model is making a decision, we can't really affect the decision and it can be hard to figure out what to do next.

It's often difficult to see how or why a model is making a decision. This is particularly true of **deep learning** models. Despite their popularity, they are profoundly black box algorithms.

One solution to this black box problem is to break big, general models into **smaller, targeted models**.

A model's internal decisions are often hidden from view, so you should make sure that the model isn't doing too much in the first place.

For example, if we were using one big model that analyzes an entire document at once, we'd only be able to work at the document level.

Instead, Lexalytics utilizes a **pipeline interaction** between our models. We start with tokenization, move on to parts of speech, then to phrases, and all the way up until we've deconstructed the document at every level. When there's an issue in the pipeline, this approach helps us see exactly where it is. Then, we can make adjustments to individual components, such as retraining the part of speech tagger or tuning its configuration files. Using smaller models gives us more flexibility to determine where an issue is, as well as how to fix it.

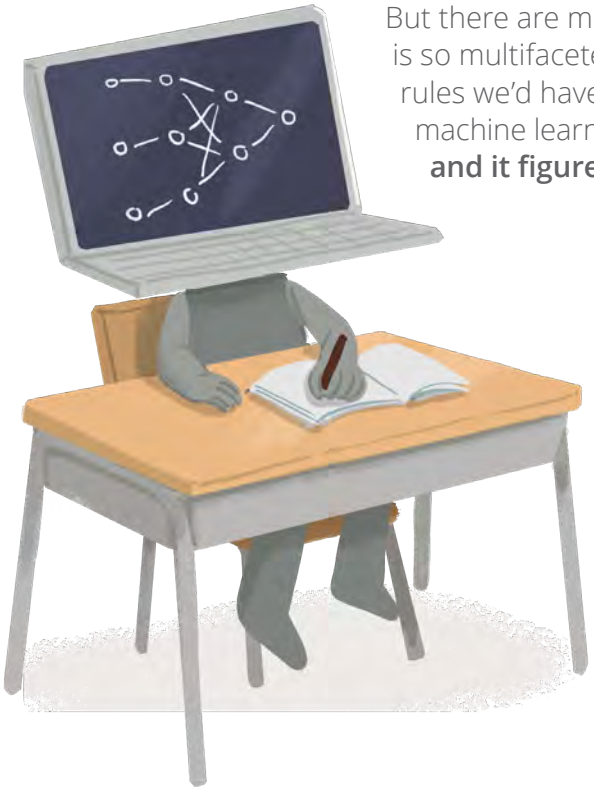
Take the phrase "Good Morning America." It looks innocuous, but it's not. If your part-of-speech tagger fails to apply "proper noun" to the phrase "Good Morning America," this phrase won't be denoted as being an entity.

You have to know that it's an entity (TV Show) and not a greeting. If you don't, you might interpret "good" as being positive, rather than just part of the entity name.

TUNE FIRST, THEN TRAIN: EFFICIENCY BEFORE COMPLEXITY

We have a whole white paper devoted to this discussion, but let's review the key points. We talked above about how machine learning is really machine teaching, and how changing how a model interprets something means having to convince it to do that. To achieve this, you have to have data, and enough of it, that supports the changes needed to make the model behave differently.

This is different than tuning. **Tuning** is a type of written instruction. With tuning, you might tell a model that the airport term "gate change" carries -0.5 sentiment points, and that this value should be used every time the model sees the phrase. This new command will instantly apply to everything that matches the entry. **Training**, on the other hand, requires the model to parse a significant amount of data before it starts to apply ("learn") the change. Additionally, the more the model "wants" to score something a particular way, the more that you're going to have to work to change it. Old habits are hard to unlearn for machine learning systems, too.



Assuming the right use case, tuning will always be faster. But there are many cases when the use of a particular word is so multifaceted or ambiguous that the number of tuning rules we'd have to put into place is prohibitive. This is where machine learning shines. **Give the model enough examples, and it figures out the rules for itself.**

Training a model

can have many unforeseen side-effects.

There are more potential side-effects of training that you must be aware of.

Say we're scoring a bunch of documents to try to effect change on a particular phrase. Each of those documents contains more than that one phrase, and the other phrases in each document will also be affected by our scoring and re-tuning. This is particularly true of common phrases, which appear often enough that they end up influencing the model.

Imagine that you're scoring news stories from 2008. 2008 was truly awful for business and economics as a whole. If you are focused on scoring financial results from businesses, you'll be marking a lot of content as negative. Then, machine learning algorithms will weigh the phrases in the content in proportion to their occurrence.

Unfortunately, that leaves some collateral damage: "first quarter," "second quarter," "third quarter," and "fourth quarter." These are neutral terms, but they occurred in frequent conjunction with negative financial news. So, the machine learning algorithm will weight those phrases as being negative. That will end up negatively impacting your results for years to come.

Lexalytics has put a number of checks and balances into our text analytics system to handle situations like this. Sometimes you just need to be able to reach in and tell the software that "first quarter" is really just neutral, despite what it might think.

CHART SOURCE: ThomsonOne; Bullion Management Group Inc., http://bmg-group.com/2008-financial-crisis/

Contact us

to explore how Lexalytics can help your business at ***lexalytics.com/contact***

SUMMARY / CONCLUSION

Text analytics is arguably one of the most complex tasks for an AI. Language is messy and complex. Meaning varies from speaker to speaker and listener to listener. Machine learning provides a rich solution set for handling this complexity, but must be implemented in a way that's relevant to the problem – and hand-in-hand with natural language processing code.

Moreover, although it's necessary to use machine learning, it's not sufficient to use a single type of model, like a big "unsupervised learning" system. Certain aspects of machine learning are very subjective, and need to be trained or tuned to match your perspective. Lexalytics combines many types of machine learning along with pure natural language processing code. We have no prejudice for one algorithm over another except in how they help us provide the best possible text analytics system to our customers.

Lexalytics processes BILLIONS of unstructured documents every day, GLOBALLY.

We transform unstructured text into usable data and powerful stories.

Our on-premise Salience® engine, SaaS Semantria® API, and end-to-end Lexalytics Intelligence Platform® combine natural language processing with artificial intelligence to reveal context-rich patterns and insights within comments, reviews, surveys, and other text documents.

Data analytics and data analyst companies rely on Lexalytics to build better products, share insights between engineering, marketing, PR, and support teams, and drive business growth.

For more information, visit www.lexalytics.com or call 1-800-377-8036