



AWS Academy Natural Language Processing
Module 02 Student Guide
Version 0.1.0
200-ACMNLP-01-EN-SG

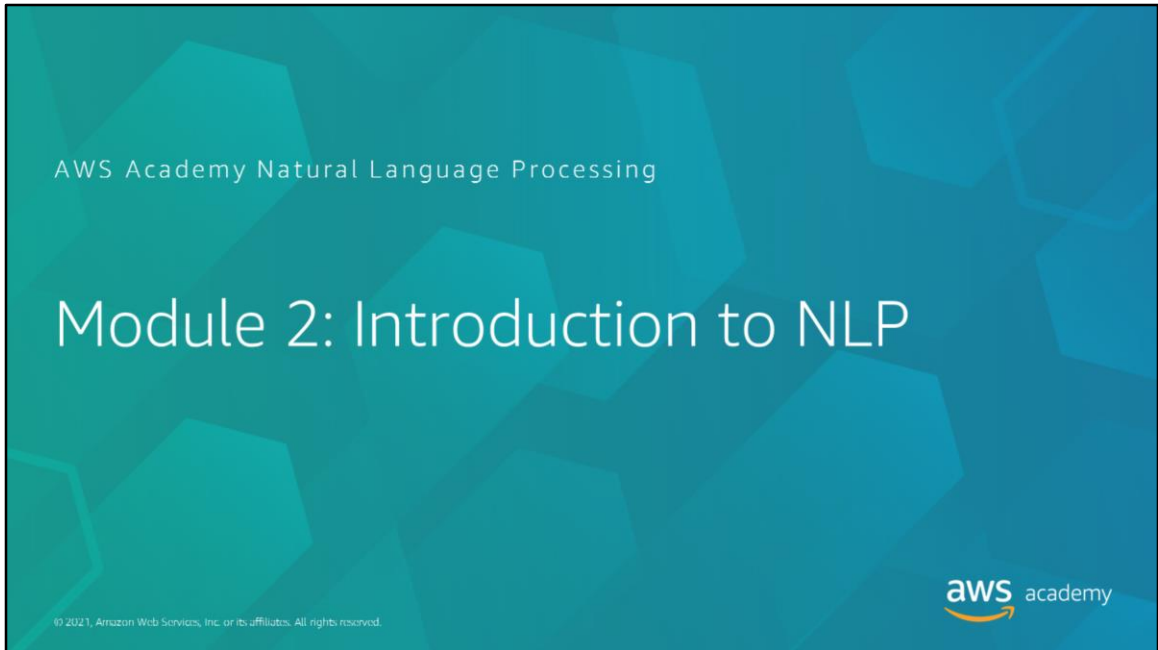
© 2020 Amazon Web Services, Inc. or its affiliates. All rights reserved.

This work may not be reproduced or redistributed, in whole or in part, without prior written permission from Amazon Web Services, Inc. Commercial copying, lending, or selling is prohibited.

All trademarks are the property of their owners.

Contents

Module 2: Introduction to Natural Language Processing	4
---	---



Welcome to Introduction to NLP.

Module overview



Sections

1. NLP and ML
2. Common NLP tasks
3. Walkthrough of an NLP problem
4. NLP architectures

© 2021, Amazon Web Services, Inc. or its affiliates. All rights reserved.

2

In this module, you will learn about the following topics:

1. **NLP and ML** – In this section, you will learn how the machine learning pipeline—which is also known as the ML pipeline—changes for problems in natural language problems, or NLP
2. **Common NLP tasks** – In this section, you will about some of the more common NLP tasks that you will use throughout this course
3. **Walkthrough of an NLP problem** – In this section, you will walk through a typical NLP problem to perform sentiment analysis on customer reviews
4. **NLP architectures** – In this section, you will learn about the history of NLP, and how ML and Deep Learning have improved NLP capabilities

Module objectives



At the end of this module, you should be able to:

- Describe how the ML pipeline can be applied to NLP
- Identify common NLP tasks
- Use the ML pipeline on an NLP problem
- Summarize NLP architectures

© 2021, Amazon Web Services, Inc. or its affiliates. All rights reserved.

5

At the end of this module, you should be able to:

- Describe how the ML pipeline can be applied to NLP
- Identify common NLP tasks
- Use the ML pipeline on an NLP problem
- Summarize NLP architectures

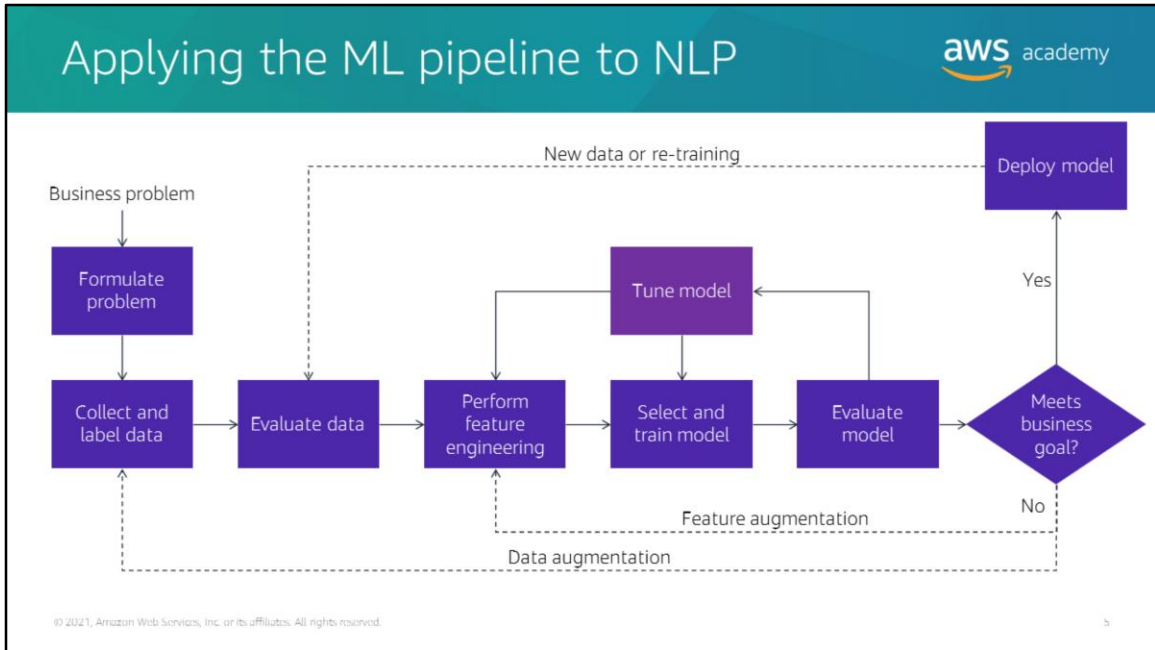
Module 2: Introduction to NLP

Section 1: NLP and ML

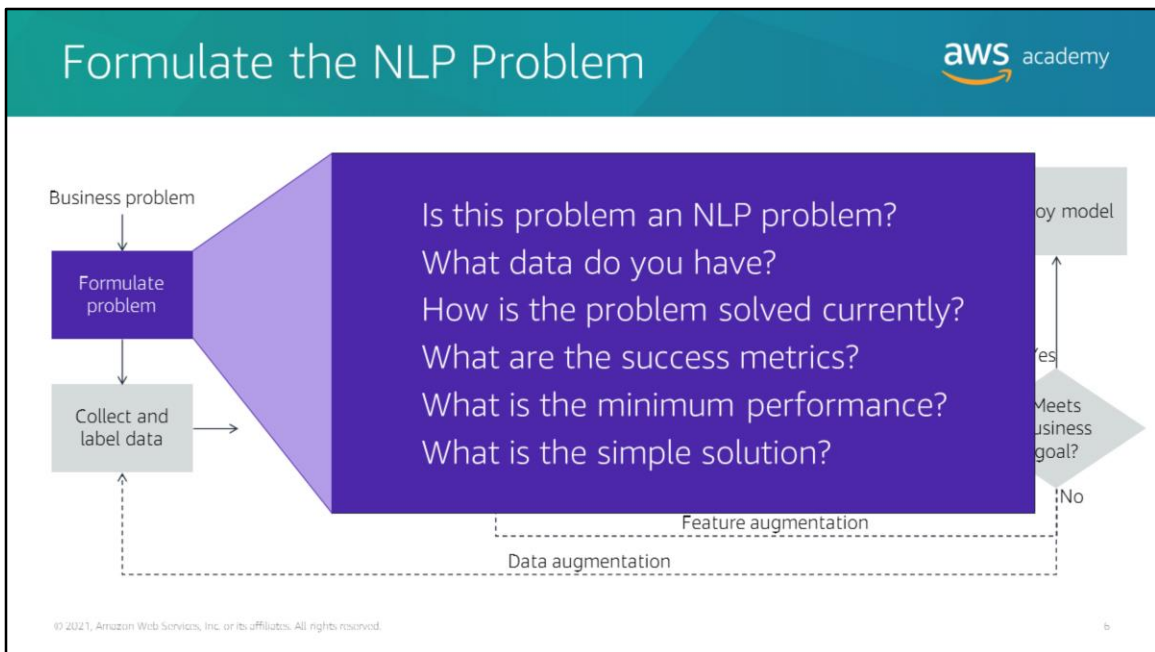
© 2021, Amazon Web Services, Inc. or its affiliates. All rights reserved.



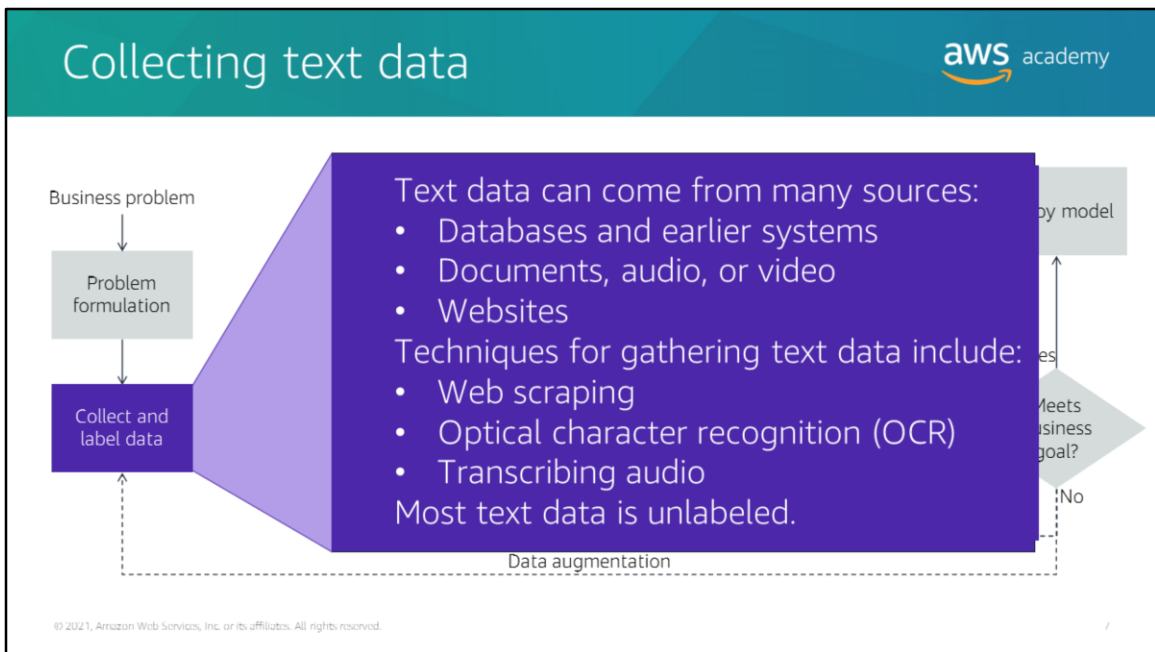
This is Section 1: NLP and ML



You can apply the same ML pipeline that you learned about in AWS Academy Machine Learning Foundations to NLP problems. In this section, you will learn about each stage of the ML pipeline, and learn some of the key differences when you work on solutions with NLP.



The first task for all ML problems is to formulate the problem. In this stage, you make sure that the problem is actually an NLP problem, and that you have a suitable dataset to address the problem. You should also work with business stakeholders to get a clear definition of success metrics and brainstorm on the simplest way to solve the problem.

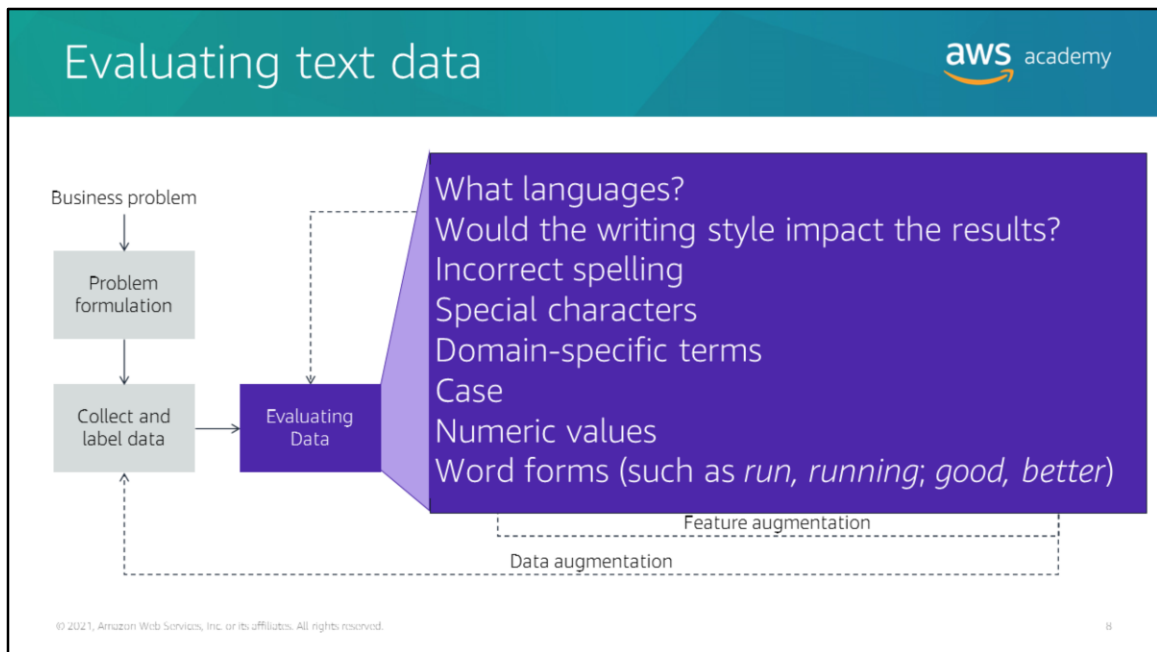


The next step is to collect data. Data for NLP can come from databases or earlier systems, documentation, or websites. Data can be only text, or a combination of text and other data types.

Collecting data from databases or earlier systems for NLP data is no different than collecting regular data. You can use extract, transform, and load (ETL) techniques and AWS services (such as AWS Glue) to perform collect, transform, and ingest from multiple systems into a single location.

Sometimes, you must work with paper documents, which can be either handwritten or typed. These documents need scanning. For handwritten text, you might need to use ML to identify the letters and symbols. For audio and video, you might need to transcribe the speech by using a service, such as Amazon Transcribe. If the data is in different languages or multiple languages, you might need to perform some translation before you process the data further.

Text data is often not labeled for working with ML models. As such, you will use more unsupervised models in your NLP solutions.

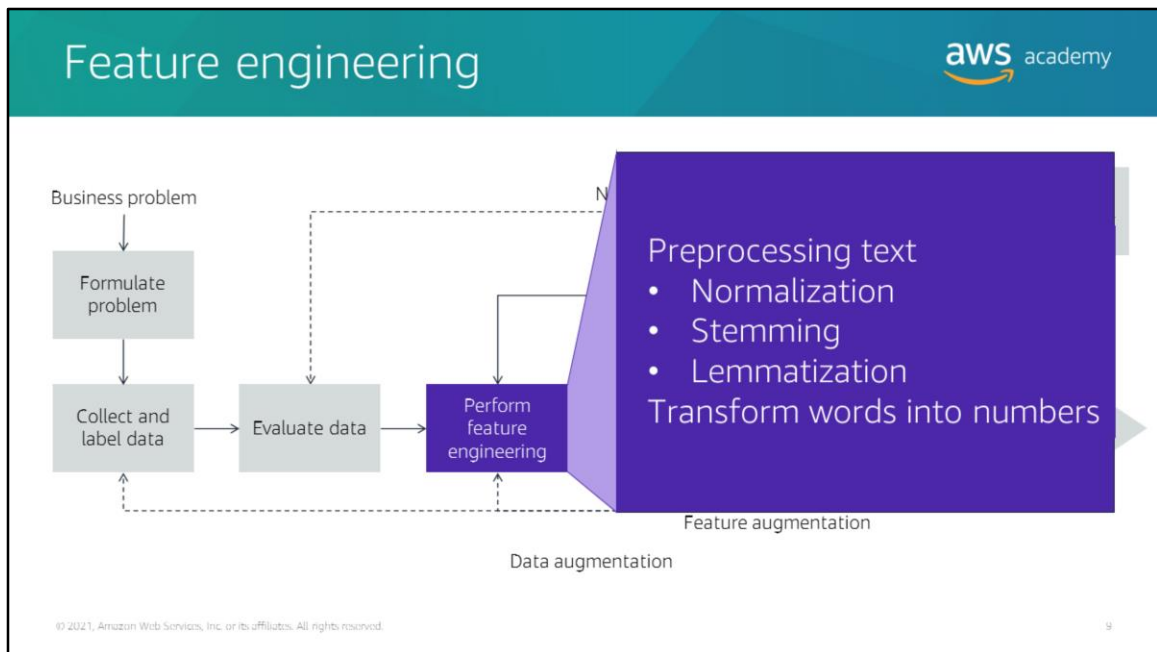


When you evaluate data, you must evaluate the text elements by considering which characteristics might impact your solution. Though this list is not exhaustive, you must determine:

1. **What languages are within the data?** – When you try to solve NLP problems, you must understand the language of the data. Languages have many differences that impact the process. For example, European languages can be structured differently than Asian languages. Having a consistent language will help you solve problems with a higher degree of accuracy.
2. **Writing Style** – Is the text a formal scientific journal, a blog post, or a social media post? The writing style can impact how you process your data. For example, with social media posts, you must decide how to handle emojis, sarcasm, and slang—all attributes that aren't typically a problem in a formal, scientific journal article. You might also expand acronyms or abbreviations, depending on your scenario.
3. **Incorrect spelling** – Having the correct spelling can improve results.
4. **Special characters** – You might need to remove or convert special characters (such as emojis, punctuation, and mathematical formulas) to process them correctly. For example, a thumbs-up emoji and the word *great* might have the same meaning in your problem domain.
5. **Domain-specific terms** – It's important to understand the terms that you have, and how

they might impact the results. For example, a common task removing stopwords, such as *the*, *and*, *do*, and *don't*. This process might cause a problem in sentiment-analysis problems, where *do* and *don't* could change the sentiment of a statement.

6. **Case** – It's common to convert text to lowercase so that *Happy* and *happy* have the same meaning.
7. **Numeric values** – Depending on the problem domain, you might convert numeric values to a text representation, or extract the numeric value into a separate feature.
8. **Word forms** – You must decide whether to convert different forms of a word to a common format. This process could include converting forms of the same word into a common format, such as changing *run*, *runner*, and *running* into *run*. It could also be more sophisticated to convert similar words into a standard word. For example *good*, *great*, *brilliant*, and *better* could all be represented as *good*.



After you evaluate the data, you must perform some processing to extract the features that you need.

First, you must process the text.

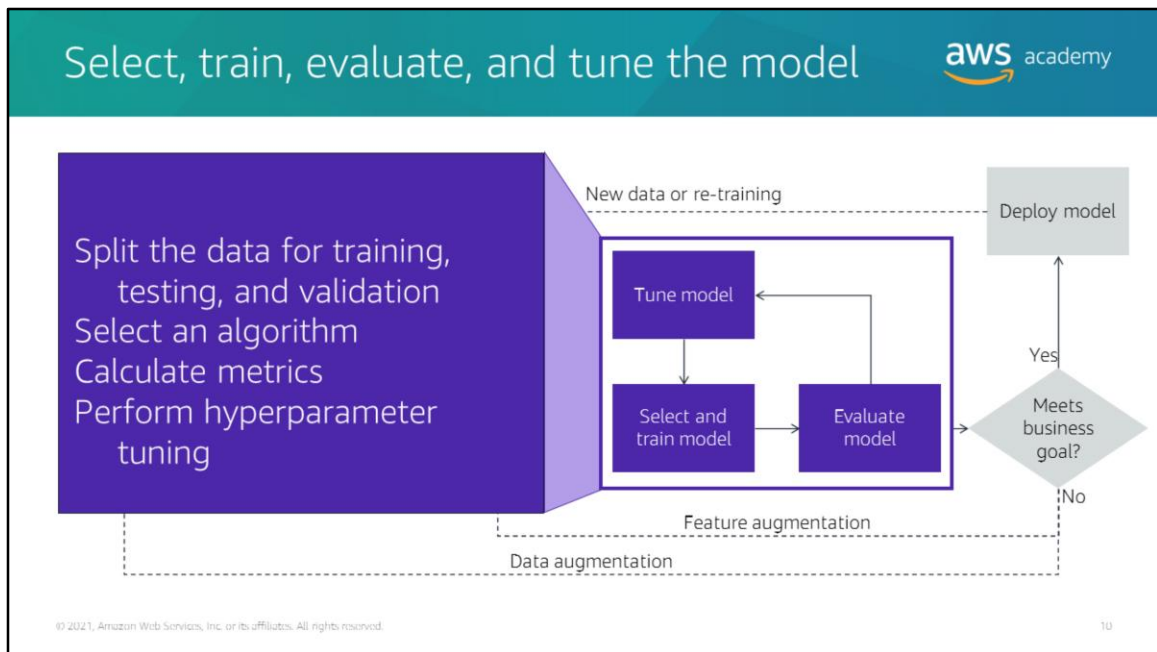
Normalization is a term that's used to identify a range of text-processing steps, such as converting text to lowercase or removing punctuation. You will learn about many of these steps in the next module. However, for now, you can think of normalization as a set of steps for cleaning up the text and making it consistent across your data.

Stemming is a method for changing different forms of a word to a standard form. For example, you could change *run*, *runner*, and *running* to *run*.

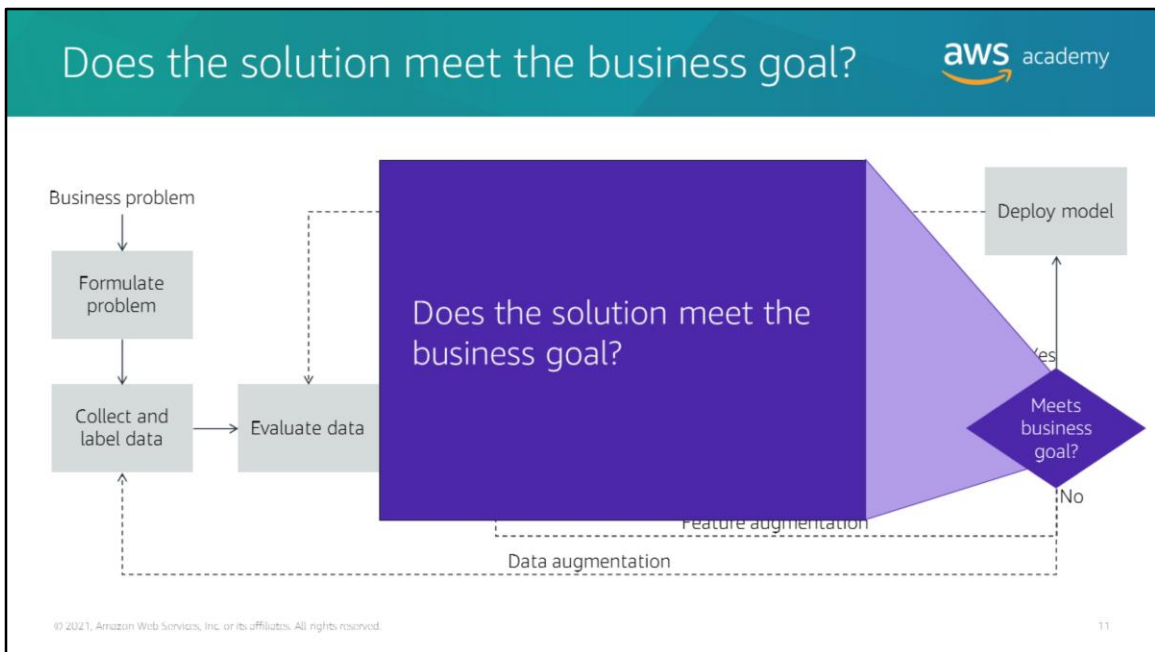
Lemmatization is another technique that can convert similar words (such as *good*, *great*, and *better*) into a single common word (such as *good*).

After you process your data, the final step is to convert the text into numeric values. You could do this conversion in different ways, depending on the problem that you are trying to solve. You will learn about many of these processes throughout the course. For now, however, you can imagine that each word in the text is converted to a numerical column. It's

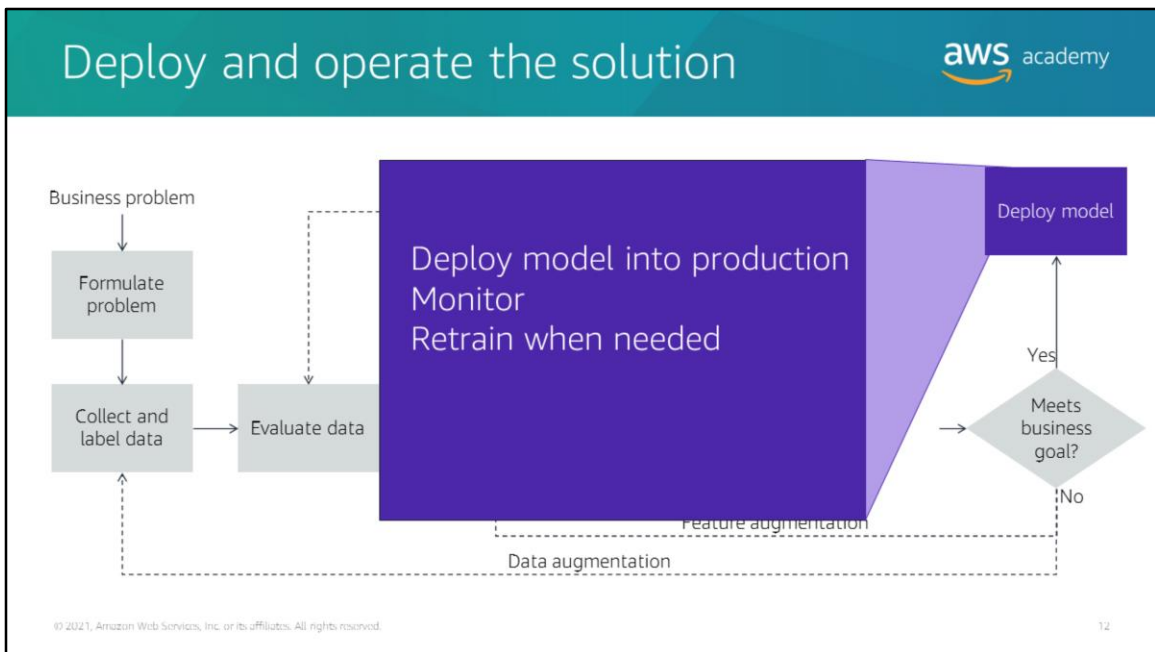
likely that you will often work with datasets that contain hundreds of columns!



After you define your dataset, you can continue to the next phases. Next, you split the data into multiple sets for training, testing, and validation. Then, you select and train a model. In the next phase, you assess the model against the business goal by using the metrics that the algorithm provides, or by calculating your own metrics. You might then tune the model by updating the model hyperparameters, or by changing how you process the data. You will repeat this process many times, and use the metrics to find the best-performing model.



It's important to determine if the solution meets the business goal. Using metrics can be a good way to determine the effectiveness of the solution. However, you must understand which metric to use, depending on your goal. You will learn about this process in more detail during the walkthrough in the next section.



Finally, you must deploy, monitor, and maintain your model. Amazon SageMaker has features that make this process more straightforward to manage, but these features are out of scope for this course.

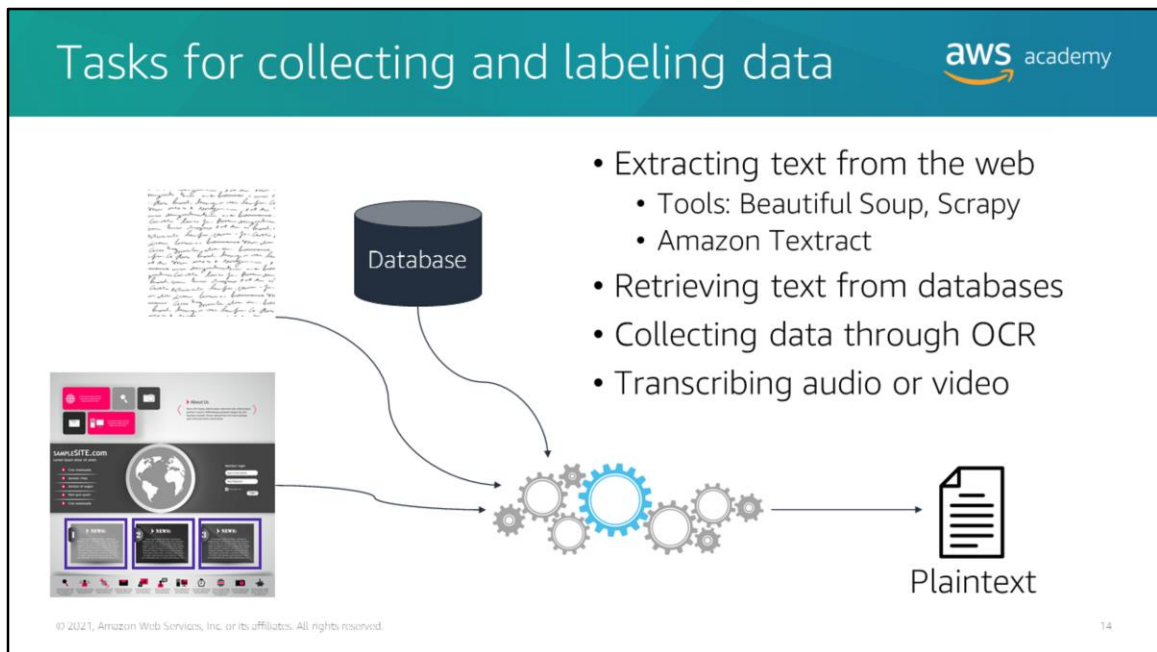
Module 2: Introduction to NLP

Section 2: Common NLP tasks

© 2021, Amazon Web Services, Inc. or its affiliates. All rights reserved.



In this section, you will learn about some of the more common NLP tasks—specifically, how you might process text so you can get ready to train an ML model.



The first step is to collect the data that you might need to solve the business problem. Like most ML problems, the data won't be in the correct format, and you must gather data from multiple sources. These sources might include traditional databases and earlier systems. However, the data could also reside in handwritten documents, or be embedded in a webpage among advertisements. Some Python tools can help you extract data, including BeautifulSoup and Scrapy. AWS also has a service called Amazon Textract, which you can use to extract data from documents. You might also need to translate text or transcribe audio to get the data that you need.

The target data format is typically a text document that you can further manipulate with text processing.

Tasks for text processing



- Removing stopwords
- Performing normalization tasks
 - Removing punctuation
 - Converting numbers to text
 - Converting text to lowercase
- Stemming
- Performing lemmatization
- Tagging parts of speech
- Performing tokenization

© 2021, Amazon Web Services, Inc. or its affiliates. All rights reserved.

15

Depending on the scenario and the state of your data, you can choose to do various tasks with your text data. In the next few slides, you will learn examples of some of these tasks.

Text-processing example



Task 3: Text preprocessing is an important NLP step b4 training a model. Tasks to correct speling, converting words to a better standard form & removing STOP words.

© 2021, Amazon Web Services, Inc. or its affiliates. All rights reserved.

1b

This slide shows the starting text that must be processed.

Text-processing example: Remove stopwords



Task 3: Text preprocessing is an important NLP step b4 training a model. Tasks to correct speling, converting words to a better standard form & removing STOPWORDS.

Remove stopwords

Task 3: Text preprocessing ~~is an~~ important NLP step b4 training ~~a~~ model. Tasks ~~to~~ correct speling, converting words ~~to a~~ better standard form ~~&~~ removing STOPWORDS.

© 2021, Amazon Web Services, Inc. or its affiliates. All rights reserved.1/

In this task, you can observe that the stopwords were removed. This process removes words such as *is*, *an*, and *and*.

Sometimes, you might remove all stopwords. However, depending on the scenario, you might remove only a subset of stopwords. For example, the word *not* can change the sentiment of a statement. Thus, you might not remove that word, depending on your business goal.

Text-processing example: Normalize



Task 3: Text preprocessing is an important NLP step b4 training a model. Tasks to correct speling, converting words to a better standard form & removing STOPWORDS.

Remove stopwords Normalize

task three text preprocessing is an important natural language processing step before training a model tasks to correct spelling converting words to a better standard form & removing stopwords

© 2021, Amazon Web Services, Inc. or its affiliates. All rights reserved.

18

The next task is a set of normalization steps. Observe the following changes:

- The text was converted to lowercase
- The acronym *NLP* was expanded to *natural language processing*
- *b4* was converted to *before*
- Punctuation was removed
- The spelling of *speling* was corrected

Text-processing example: Stemming



Task 3: Text preprocessing is an important NLP step b4 training a model. Tasks to correct speling, converting words to a better standard form & removing STOPWORDS.

Remove stopwords
Normalize

Stemming

task three text preprocessing is an important natural language
processing step before training a model tasks to correct spelling
converting words to a better standard form & removing stopwords

© 2021, Amazon Web Services, Inc. or its affiliates. All rights reserved.

19

In this step, stemming was used to convert words to a common format:

- *preprocessing* was changed to *preprocess*
- *processing* was changed to *process*
- *training* was changed to *train*
- *spelling* was changed to *spell*
- *converting* was changed to *convert*
- *removing* was changed to *remov*
- *words* was changed to *word*

You should notice that *removing* was converted to *remov*. This change is not a spelling error. Instead, it's a common way to stem words like this example.

Text-processing example Lemmatization



Task 3: Text preprocessing is an important NLP step b4 training a model. Tasks to correct speling, converting words to a better standard form & removing STOPWORDS.

Remove stopwords
Normalize

Stemming
Lemmatization

gets the original work
back to its original
from the dictionary
longer time but better
performance

task three text preprocessing is an important natural language
processing step task before training a model tasks to correct
spelling converting words to a better good standard form &
removing stopwords

© 2021 Amazon Web Services, Inc. or its affiliates. All rights reserved.

23

In this step, lemmatization was performed:

- *step* was converted to *task*
- *better* was converted to *good*

keep in mind stemming doesn't look through the dictionary, for example when the "ing" was removed in the remove word the e wasn't inserted back. this is an example of a difference between it and lemmatization.

Text processing example: End result



Task 3: Text preprocessing is an important NLP step b4 training a model. Tasks to correct spelling, converting words to a better standard form & removing STOP words.

Remove stopwords
Normalize

Stemming
Lemmatization

task three text preprocess important natural language process task
before train model task correct spell convert word good standard
form remov stopword

© 2021, Amazon Web Services, Inc. or its affiliates. All rights reserved.

21

The final result of these steps is shown.

You might think that some of these steps don't make much difference in the example. However, when you perform the same steps across the entire corpus of text, you will usually get better results because you introduced consistency through these processing steps.

Tokenization: Machines want numbers!



task three text preprocess important natural language process task
before train model task correct spell convert word good standard form
remov stopword

Map text to vectors of numbers



task	three	text	preprocess	important	natural	language	process	before
3	1	1	1	1	1	1	1	1

© 2021, Amazon Web Services, Inc. or its affiliates. All rights reserved.

22

The final task in this introduction is converting the text into numerical values. In the diagram, the example converts the text into numerical values by using a vectorizer. You will learn more about this process in the walkthrough, and in later modules.

Module 2: Introduction to NLP


Section 3: Walkthrough of an NLP problem

© 2021, Amazon Web Services, Inc. or its affiliates. All rights reserved.



In this section, you will walk through an NLP ML problem by applying the tasks in the previous section.

Business problem: Is a review positive or negative?



What data do you have?	<ul style="list-style-type: none">• Reviews that were labeled positive or negative
How is the problem solved currently?	<ul style="list-style-type: none">• Nikki spends time each day reading comments to find negative ones
What are the success metrics?	<ul style="list-style-type: none">• Identify all negative reviews so customer service can address them• It's ok to incorrectly identify positive reviews as negative
What is the minimum performance?	<ul style="list-style-type: none">• 90 percent negative reviews detected
Is this an NLP problem?	<ul style="list-style-type: none">• It's a binary classification problem!

© 2021, Amazon Web Services, Inc. or its affiliates. All rights reserved.

24

The first step is to analyze the business problem. In the walkthrough, the business wants to quickly identify negative reviews of products that can then be flagged for a customer service agent to address. They might respond to the comment, if appropriate. Today, the business identifies about 30 percent of negative reviews, but they would like to identify 90 percent of negative reviews.

The problem is solved today by a human resource, Nikki. She spends a large portion of her time reading comments and deciding if they need a response. She only manages to get through a subset of the comments in the time that she allocates to the task each day. Many comments are not being detected, which is causing a negative impact on the business.

Nikki collected the reviews in a spreadsheet, and she labeled the reviews as either positive or negative.

Because the business has a labeled dataset, and there are two outcomes to be predicted, it sounds like this problem could be a binary classification problem!

Obtaining the data



- 70,000 rows in a comma-separated values (CSV) file

Feature	Description
reviewText	Text of the review
summary	Summary of the review
verified	Whether the purchase was verified (True or False)
time	Unix timestamp for the review
log_votes	Logarithm-adjusted votes $\log(1 + \text{votes})$
isPositive	Whether the review is positive or negative (1 or 0)

© 2021, Amazon Web Services, Inc. or its affiliates. All rights reserved.

25

When you examine the data, it's organized into six columns.

- **reviewText** contains the full text of the review, but **summary** contains only a short summary of the review
- **verified** indicates if the customer actually purchased the product
- **time** is the date and time of the review
- **log_votes** are the number of votes adjusted to a logarithm scale
- **isPositive** is a numeric value, with 1 indicating a positive review and 0 indicating a negative review

Sample data

aws academy

	reviewText	summary	verified	time	log_votes	isPositive
0	PURCHASED FOR YOUNGSTER WHO\NINHERITED MY *TOO...	IDEAL FOR BEGINNER!	True	1361836800	0.000000	1.0
1	unable to open or use	Two Stars	True	1452643200	0.000000	0.0
2	Waste of money!!! It wouldn't load to my system.	Dont buy it!	True	1433289600	0.000000	0.0
3	I attempted to install this OS on two differen...	I attempted to install this OS on two differen...	True	1518912000	0.000000	0.0
4	I've spent 14 fruitless hours over the past tw...	Do NOT Download.	True	1441929600	1.098612	0.0
5	I purchased the home and business because I wa...	Quicken home and business not for amatures	True	1335312000	0.000000	0.0
6	The download doesn't take long at all. And it'...	Great!	True	1377993600	0.000000	1.0
7	This program is positively wonderful for word ...	Terrific for practice.	False	1158364800	2.397895	1.0
8	Fantastic protection!! Great customer support!!	Five Stars	True	1478476800	0.000000	1.0

© 2021, Amazon Web Services, Inc. or its affiliates. All rights reserved.

2b

You can observe from this sample data that it's not the type of language that you would get from a scientific journal article. The text contains excessive punctuation, all-uppercase text, and some codes (such as \n).

Exploratory data analysis



What is the distribution of the data?

```
[ ]: df['isPositive'].value_counts()
[ ]: 1.0 43692
      0.0 26308
      Name: isPositive, dtype: int64
```

Do you have missing values?


```
[ ]: df.isna().sum()
[ ]: reviewText      11
      summary       14
      verified       0
      time          0
      log_votes      0
      isPositive     0
      dtype: int64
```

© 2021, Amazon Web Services, Inc. or its affiliates. All rights reserved.

2 /

By examining the structure of the data, you can see the distribution of positive reviews (43,692) and negative reviews (26,308). Some data also appears to be missing.

Text processing



Text Samples

PURCHASED FOR YOUNGSTER WHO\nINHERITED MY "TOO sMALL FOR ME"\nLAPTOP. IDEAL FOR LEARNING A\nFUTURE GOOD SKILL. HER CHOICE\nOF BOOKS IS A PLUS AS WAS THIS BOOK!

I've spent 14 fruitless hours over the past two days fruitlessly attempting to install this software on my computer and nothing I've found has worked. I need the software to type proficiently due to disability, and it will not install. The download itself seems to be a corrupted file, I have a fair amount of computer skills and no amount of tinkering has made the program work, so, judging by other reviews, it must be the program itself.

Remove stop words*

Convert text to lowercase

Remove whitespace

Remove extra spaces and tabs

Remove HTML tags

© 2021, Amazon Web Services, Inc. or its affiliates. All rights reserved.

* See next slide 28

Some of the text processing tasks that you might consider include:

- Removing stopwords
- Converting text to lowercase
- Removing extra white space
- Removing extra spaces and tabs
- Removing HTML tags

Note: You might not want to remove all the stopwords. The next slide explains why.

© 2021 Amazon Web Services, Inc. or its affiliates. All rights reserved.

33

Not removing all stopwords: Explanation



In sentiment analysis, some stopwords are needed!

against, not, do, don't, are, aren't, could, couldn't, did, didn't, does, doesn't, had, hadn't, has, hasn't, have, haven't, is, isn't, might, mightn't, must, mustn't, need, needn't, should, shouldn't, was, wasn't, were, weren't, won, won't, would, wouldn't

© 2021, Amazon Web Services, Inc. or its affiliates. All rights reserved.

29

Consider the following sentences:

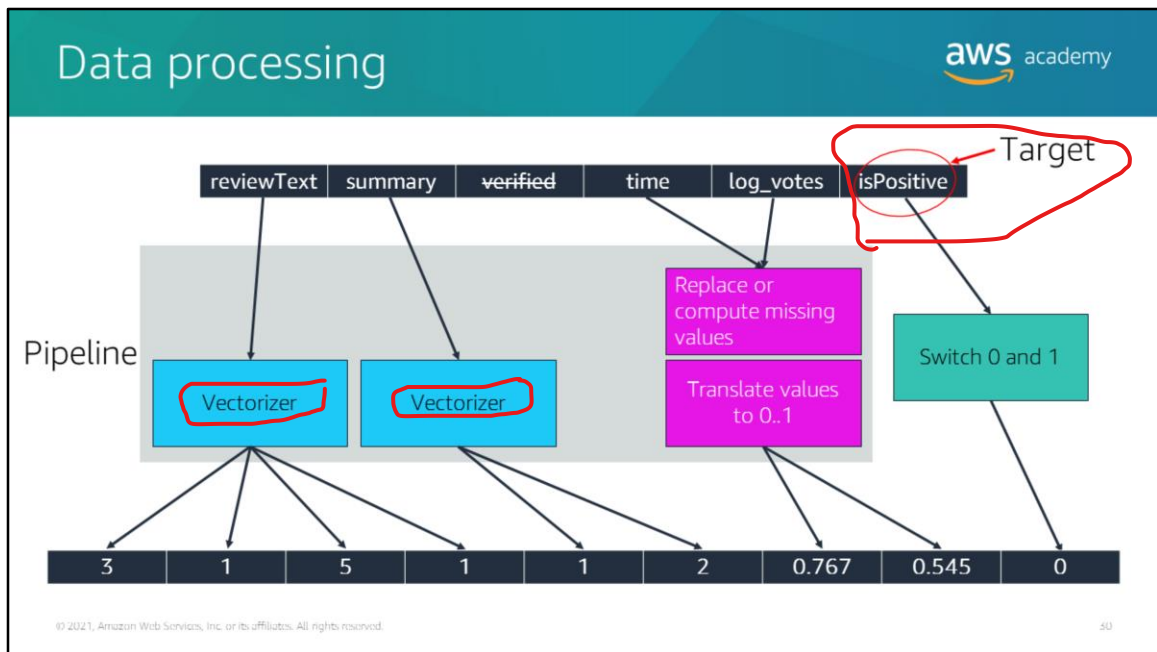
1. I didn't love this product.
2. I love this product.

The first sentence is a negative comment. The second sentence is a positive comment. If you removed all the stopwords, you might have the following sentences:

1. love product
2. love product

Though the original sentences included one negative comment and one positive comment, the processed sentences express the same intent. Thus, when you remove stopwords, make sure that you account for the problem that you are trying to solve.

Another factor to consider is the intent or underlying emphasis of the text. Sarcasm and irony are particularly difficult for machine learning as the context is usually not in the text.



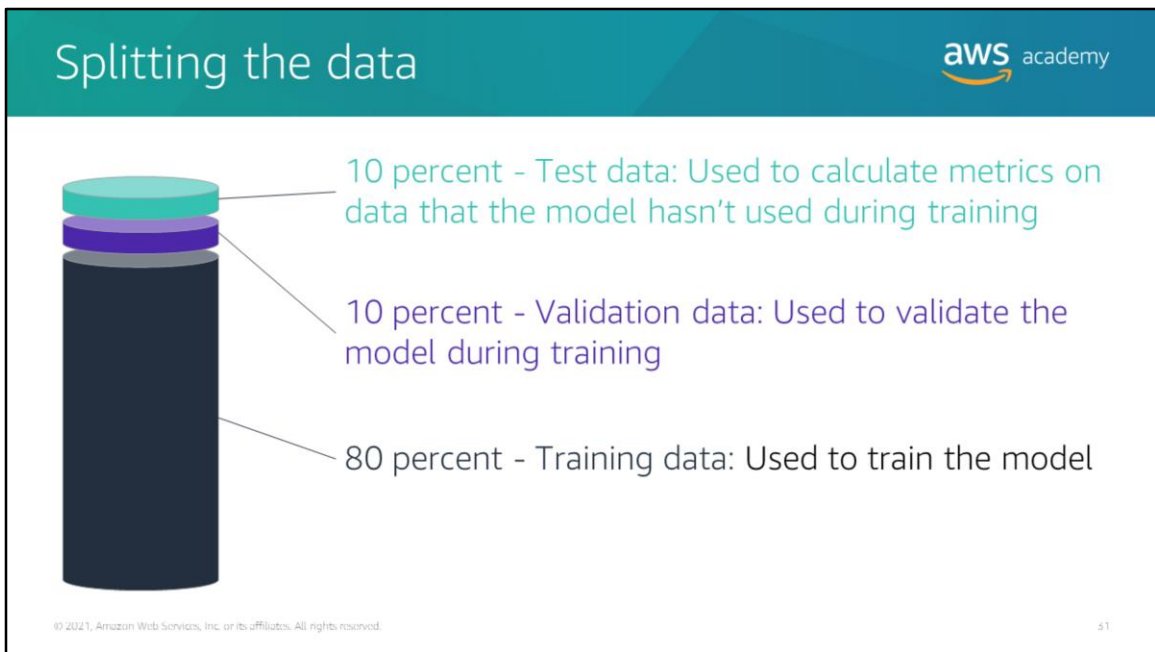
After you process the text, the next step is to get the data into the format for ML.

For the text columns, you could use a vectorizer to convert the text into multiple numerical columns.

For the numeric data, you might want to fill in missing values, and convert the numeric values to a common scale.

Because multiple tasks are performed on the data, a common technique in ML is to create a processing pipeline. You can define this processing pipeline in your Python code to make it easier to process data. You will learn how to construct a processing pipeline in the lab for this module.

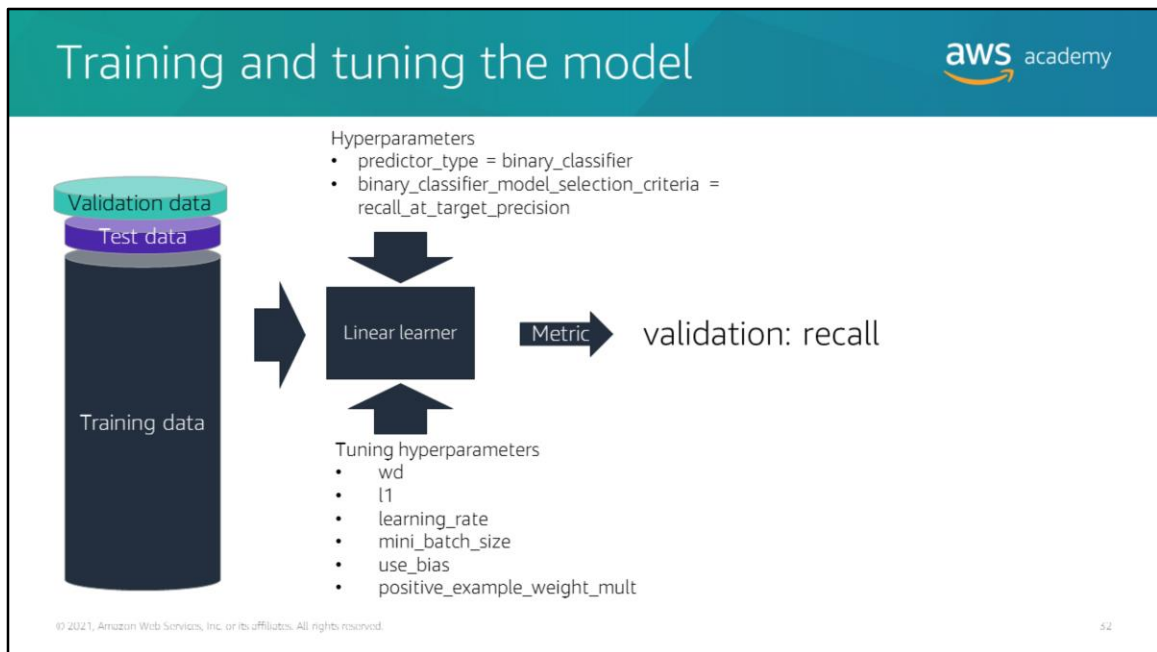
The last processing task is to switch the 0 and 1 in the **isPositive** column. Why? Your business goal is to correctly identify the negative reviews, which are represented by 0. Metrics that are produced by algorithms are usually geared towards identifying the positive value. To use the built-in metric, you must switch these values.



The next step is to split the data into multiple sets. These sets are usually categorized as training data, validation data, and testing data.

Training data, which includes both features and labels, is fed into the algorithm that you selected to produce your model. The model is then used to make predictions over the validation dataset. The *validation dataset* is where you might notice things that you will want to tune and change. Then, when you are ready, you run the *test dataset*—which includes only features, because you want the labels to be predicted. You can reasonably expect to see the performance that you get with the test dataset in production.

A common split when you use the holdout method is as follows: use 80 percent of the data for training, 10 percent for validation, and 10 percent for testing. Or, if you have a large amount of data, you can split the data into 70 percent for training, 15 percent for validation, and 15 percent for testing.



The final step is to train the model.

The walkthrough uses the linear learner algorithm that's built into Amazon SageMaker.

Linear learner requires you to set a few hyperparameters:

- **predictor_type** – This problem is a binary classification problem, so the type must be set to **binary_classifier**.
- **binary_classifier_model_selection_criteria** – The business case requires that you correctly identify the correct (positive) classes, so this criterion is set to **recall_at_target_precision**. This setting is why you switched the 0 and 1 in the **isPositive** column.

The output of training the model will be a trained model and a set of metrics. The metric that's of greatest interest to you is **validation: recall**. This metric can be used when you tune the model with the hyperparameters, perhaps by using a hyperparameter tuning job in Amazon SageMaker. You would also use this metric to measure the impact of changing some of the text-processing steps (perhaps changing the vectorizer that you use, or incorporating other steps).

Module 2 Guided Lab: Applying ML to an NLP Problem

55



© 2021, Amazon Web Services, Inc. or its affiliates. All rights reserved.

You will now complete Module 2 Guided Lab: Applying ML to an NLP Problem.

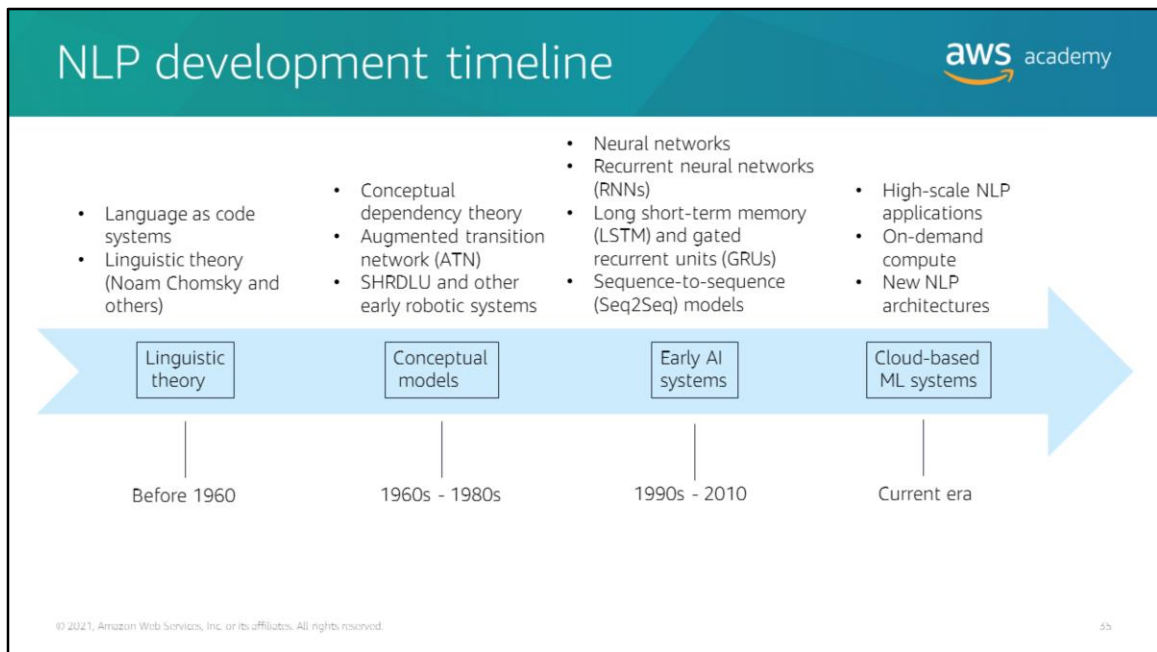
Module 2: Introduction to NLP

Section 4: Evolution of NLP architectures

© 2021, Amazon Web Services, Inc. or its affiliates. All rights reserved.



In this section, you will learn about the evolution of NLP architectures.



In this section, you learn about three NLP architectures—recurrent neural networks (RNNs), sequence-to-sequence models, and transformers. Before you learn about these architectures, it's important to understand some NLP history, and the relationship between NLP and neural networks.

NLP problems have been studied for a long time. As far back as the fifteenth century, philosophers and mathematicians proposed creating a system for translating between languages with code. During and shortly after World War II, mathematicians proposed using some of the encoding techniques that were developed for military purposes to create automated translation machines. One of the well-known artifacts from this period is the *Turing test* that was developed by Alan Turing in 1950. The Turing test evaluates whether a machine can exhibit intelligent behavior that can't be distinguished from human thought.

The lack of a linguistic model was a major obstacle for many of these early approaches to NLP. This situation started to change in the 1950s. Noam Chomsky published *Syntactic Structures* in 1957, which introduced the concept of a *universal grammar*. It was a major step forward in the field of linguistics.

NLP became recognized as a computer science problem in the 1960s. Several significant

conceptual frameworks were developed between the 1960s and 1980s. Two of the more significant advances were the *conceptual dependency model* and the *augmented transition network (ATN)*. The conceptual dependency model was developed by Roger Schank at Stanford University in 1969. It introduced the idea of deriving meaning from human language independently from the words that are included in the input. ATNs were developed in a similar timeframe, and were influenced by Schank's work. ATNs introduced the idea of using recursion to parse sentences.

These early frameworks were applied to automated translations, early robots, and specific domains. For example, the SHRDLU system was developed by Terry Winograd at the Massachusetts Institute of Technology (MIT) in the late 1960s. It could respond to human language commands to manipulate images of wooden blocks.

The 1990s and the first decade of the twenty-first century saw significant growth in AI and the use of neural networks. Some important early developments included:

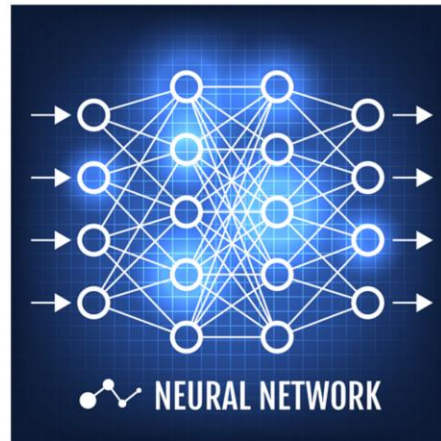
- *Recurrent neural networks*, which can pass the learning from one layer of the network to another layer
- *Long short-term memory (LSTM)* and *gated recurrent units (GRUs)*, which can optimize neural networks
- *Sequence-to-sequence (Seq2Seq) models*

These early neural networks provided an important foundation for the NLP systems that are now in use every day. Running neural networks in large-scale production was not economically feasible. However, the growth of cloud computing—and its ability to quickly scale on demand to very high compute capacity—made it possible to develop NLP applications that millions of users interact with daily (for example, Amazon Alexa or Google Translate). New NLP architectures have also been developed in the current era. For example, *transformers* are a new architecture that evolved from Seq2Seq models.

Neural networks



- Neural networks discover relationships in a dataset by adjusting weights for the connections between a collection of nodes
 - Mimics human thinking
 - Is used in unsupervised ML systems
- Generally, neural networks are divided into three layers –
 - Input layer: Captures the dimensions in the dataset
 - Hidden layer: Consists of weighted nodes and activation functions that perform functions specific to the problem domain
 - Output layer: Provides the results of the processing



© 2021, Amazon Web Services, Inc. or its affiliates. All rights reserved.

5b

Neural networks imitate the way that human thinking occurs by creating a collection of interoperating nodes and adjusting the weights between the nodes as inputs are processed. The nodes are divided into three distinct layers:

- The input layer captures the dimensions in the dataset.
- The hidden layer consists of weighted nodes that perform functions specific to the problem domain
- And finally, the output layer provides the results of the processing

Neural networks are applied to various AI problems. You can use a neural network for many types of NLP problems, such as speech recognition or machine translation.

Neural network optimization



- Neural networks must be tuned
 - Find the number of input nodes
 - Determine the best weights for the hidden layer
- Gradients measure the changes to the output relative to the changes in inputs
 - Measures the slope of a function
 - Networks with high gradients learn faster



© 2021, Amazon Web Services, Inc. or its affiliates. All rights reserved.

5/

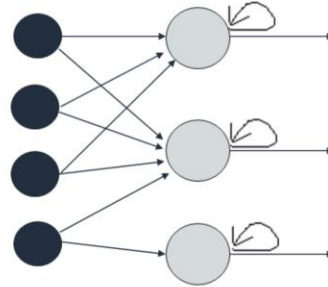
You must run neural networks multiple times to find the correct number of nodes and the correct weights for the functions in the hidden layer. During the tuning process, you will find that some weights outperform others. If you track the improvements in output as you change the weights, you will have a series of number pairs that match the change in weights to the improvement of the function. If you plotted these changes, you would have a line. The gradient would be the slope of the line. A gradient that's near zero would mean your model is not improving. Networks with higher gradients learn faster.

Here's an analogy for the gradient. Imagine that you are climbing a mountain. Each early step up the mountain would take you closer to your destination than your later steps because the mountain has a steeper slope towards the bottom. If you measured the distance between the start and end of each step, you would have a series like the series for the change in weights of the neural network function.

NLP architectures: RNNs



- Recurrent Neural Networks
 - Can remember what they learned in previous nodes
 - Are commonly used for NLP problems such as machine translations and chatbots
- RNNs combine short-term memory with *long short-term memory (LSTM)* or *gated recurrent units (GRUs)*



© 2021, Amazon Web Services, Inc. or its affiliates. All rights reserved.

58

Although neural networks mimic human thought, they have several limitations. For example, humans don't need to evaluate every single input from scratch. Humans can process an input because they already have certain knowledge. When you read this text or listen to a lecture, your mind can process the input from previous cycles. If you think of this series of inputs to your mind as a long-running sequence, then you have the idea of a *feed-forward network*. As new inputs arrive, the feed-forward network passes the output to the next series of nodes.

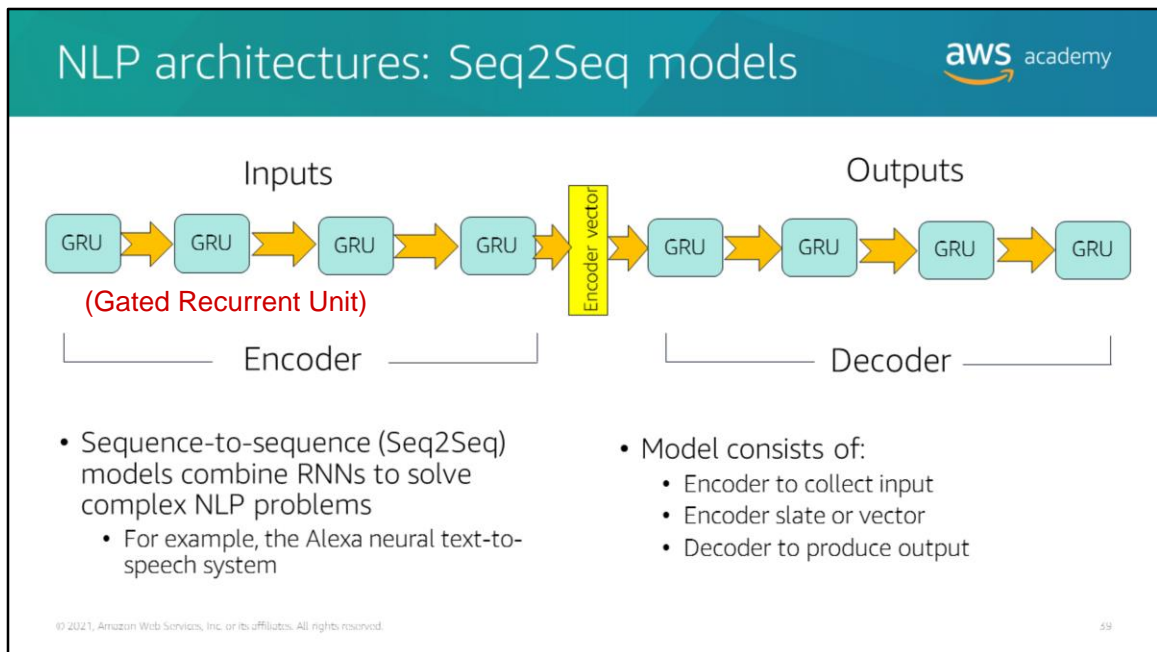
The problem with a feed-forward network is that you need a way for the individual nodes in the network to use what they learned. The learnings from one point in the sequence can be used to evaluate input in the next point. This situation is why *recurrent neural networks (RNNs)* were developed. RNNs can remember what they learned by using *long short-term memory (LSTM)*.

Here's an example of why you would need LSTM. Imagine that you are creating a typing application that suggests words as you type, and you are using a simple neural network. When you type the words *government* and you got to the letter *V*, you want the application to suggest the rest of the word (*ernment*). A recurrent neural network would be able to provide this sort of suggestion. When you get to the letter *V*, the neural network would be able to remember the previous letters because they are recent input.

Now let's look at a more complex problem. Instead of predicting subsequent letters in a word, you want your application to suggest words based on the context of the sentence. For example, if a user writes "Dear Joe" in an email it needs to "remember that the recipient's name is Joe so it can suggest the name when appropriate. RNN's are not very good at "remembering" this sort of information. LSTM was introduced to develop networks that could handle the problem of retaining information over a longer sequence in order to make predictions about subsequent items in the sequence.

After working with neural networks, researchers in the 1990s discovered some problems with managing the gradients. For example, accumulating gradients between multiple cycles can result in large updates to the weights in the hidden layer. This situation is known as an *exploding gradient* and it can lead to an unstable network. A similar problem is known as "vanishing gradients". Vanishing gradients occur because the incremental changes to weights become so small that the impact on the network performance can not be measured.

Gated recurrent units (GRUs) were developed to handle these problems.



Google first introduced sequence-to-sequence (Seq2Seq) models in 2014. The main problem that they were designed to address was how to map inputs to outputs in situations where the inputs and output sequences are not the same length. For example, suppose that you want to translate the question *How are you?* between English and Spanish. In English, the input is three words (*How are you?*), but the output is only two words in Spanish (*¿Cómo estás?*).


Seq2Seq models work well for translations because you must transform the sequence of words from one language into a sequence of different words in another language. LSTM-based modules are generally used for a Seq2Seq model because they can give meaning to the sequence, while they also remember the parts that are important.

A Seq2Seq model consists of:

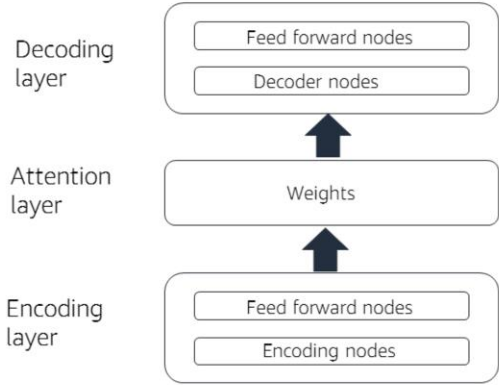
- An *encoder* that captures the input to the model. It consists of either LSTM or GRU cells.
- An *encoder slate* (also known as an *encoder vector*) that calculates the value for the nodes in the hidden layer.
- A *decoder* that predicts the output based on the input.

Seq2Seq models are used by many systems that respond to human speech, such as Amazon Alexa.

NLP architectures: Transformers



- Seq2Seq model + attention mechanism
 - No RNN required
- Attention adds a layer of weights that are used to determine the most important aspects of the input
 - For example, if *order* is important, the attention layer increases the weight for *order* when it evaluates the input



The diagram illustrates the Transformer architecture's flow. At the bottom is the **Encoding layer**, which contains **Feed forward nodes** and **Encoding nodes**. An upward arrow leads to the **Attention layer**, which contains **Weights**. Another upward arrow leads to the **Decoding layer**, which contains **Feed forward nodes** and **Decoder nodes**.

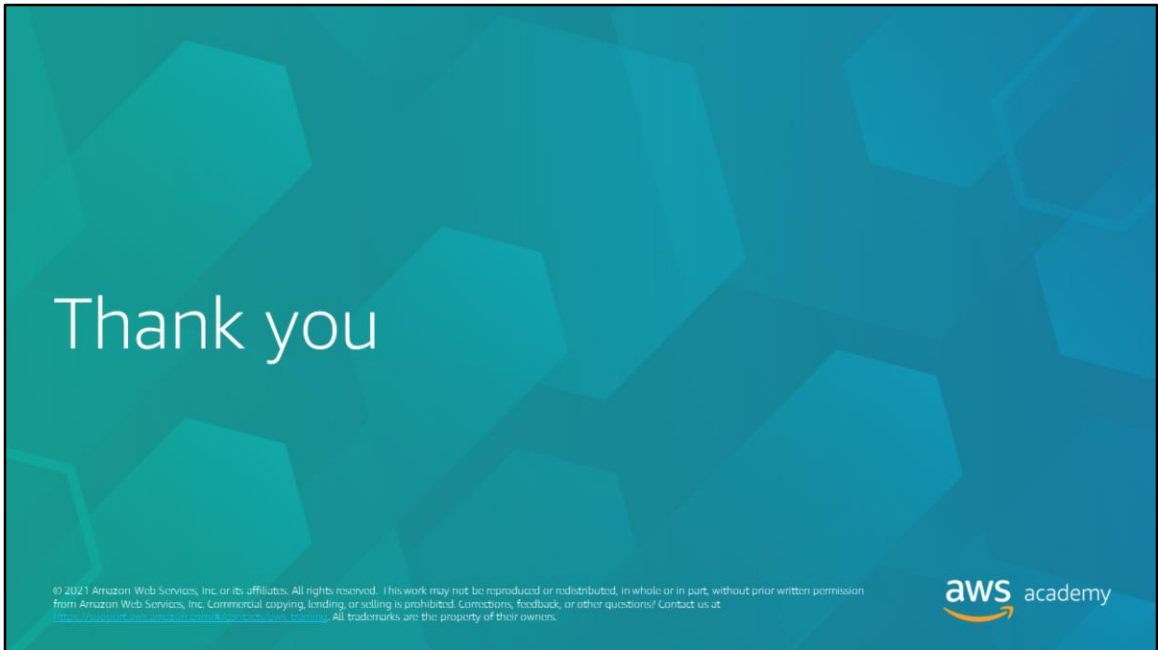
© 2021, Amazon Web Services, Inc. or its affiliates. All rights reserved.

40

The transformer architecture for NLP provides a way to create a Seq2Seq model without using an RNN. It introduces the idea of *attention*. This feature is another way that neural networks have been adapted to behave more like human thought. When you read or hear a sentence, your brain can evaluate each word in the sequence. Your brain knows that word order is important, and it can immediately infer the meaning of the word in relation to the previous words. The task for the neural network designer is how to give the neural network this knowledge.

The transformer architecture is similar to the Seq2Seq model. However, it replaces the ability of the RNN to retain knowledge between iterations with a layer that retains the position of every word in a sequence.

The *Bidirectional Encoder from Transformers (or BERT)* is a model that builds on the idea of a transformer to fine-tune pre-trained NLP models. It has been used to improve performance for various NLP tasks.



Thank you for completing this module. In the next module, you will learn more about how to process text for an NLP application.