Practical Data Science Coursera Specialization

by DeepLearning.AI

Course #1 Analyze Datasets and Train ML Models using AutoML

Course Site

Made By: Matias Borghi

Table of Contents

Summary

In the first course of the Practical Data Science Specialization, you will learn foundational concepts for exploratory data analysis (EDA), automated machine learning (AutoML), and text classification algorithms. With Amazon SageMaker Clarify and Amazon SageMaker Data Wrangler, you will analyze a dataset for statistical bias, transform the dataset into machine-readable features, and select the most important features to train a multi-class text classifier. You will then perform automated machine learning (AutoML) to automatically train, tune, and deploy the best text-classification algorithm for the given dataset using Amazon SageMaker Autopilot. Next, you will work with Amazon SageMaker BlazingText, a highly optimized and scalable implementation of the popular FastText algorithm, to train a text classifier with very little code.

Practical data science is geared towards handling massive datasets that do not fit in your local hardware and could originate from multiple sources. One of the biggest benefits of developing and running data science projects in the cloud is the agility and elasticity that the cloud offers to scale up and out at a minimum cost.

The Practical Data Science Specialization helps you develop the practical skills to effectively deploy your data science projects and overcome challenges at each step of the ML workflow using Amazon SageMaker. This Specialization is designed for data-focused developers, scientists, and analysts familiar with the Python and SQL programming languages and want to learn how to build, train, and deploy scalable, end-to-end ML pipelines - both automated and human-in-the-loop - in the AWS cloud.

Week 1: Explore the Use Case and Analyze the Dataset

Ingest, explore, and visualize a product review data set for multi-class text classification.

Learning Objectives

- **Describe the discipline of practical data science in the cloud**
- **Define the task and use case: Multi-class classification for sentiment analysis of product reviews**
- **Ingest and explore data**
- **Analyze data using visualizations**

Week 2: Data Bias and Feature Importance

Determine the most important features in a data set and detect statistical biases.

Learning Objectives

- **Describe the concept of data bias and compare popular bias metrics**
- **Demonstrate how to detect data bias**
- **Understand feature importance**

Week 3: Use Automated Machine Learning to train a Text Classifier

Inspect and compare models generated with automated machine learning (AutoML).

Learning Objectives

- **Describe the concept of Automated Machine Learning (AutoML)**
- **Discuss how SageMaker Autopilot uniquely implements AutoML**
- **Demonstrate how to train a text classifier with AutoML**

Week 4: Built-in algorithms

Train a text classifier with BlazingText and deploy the classifier as a real-time inference endpoint to serve predictions.

Learning Objectives

- **Summarize why and when to choose built-in algorithms**
- **Describe use case and algorithms**
- **Understand the evolution of text analysis algorithms**
- **Discuss word2vec, FastText and BlazingText algorithms**
- **Transform raw review data into features to train a text classifier**
- **Apply the Amazon SageMaker built-in BlazingText algorithm to train a text classifier**
- **Deploy the text classifier and make predictions**

Week 1: Explore the Use Case and Analyze the Dataset

Introduction to Practical Data Science

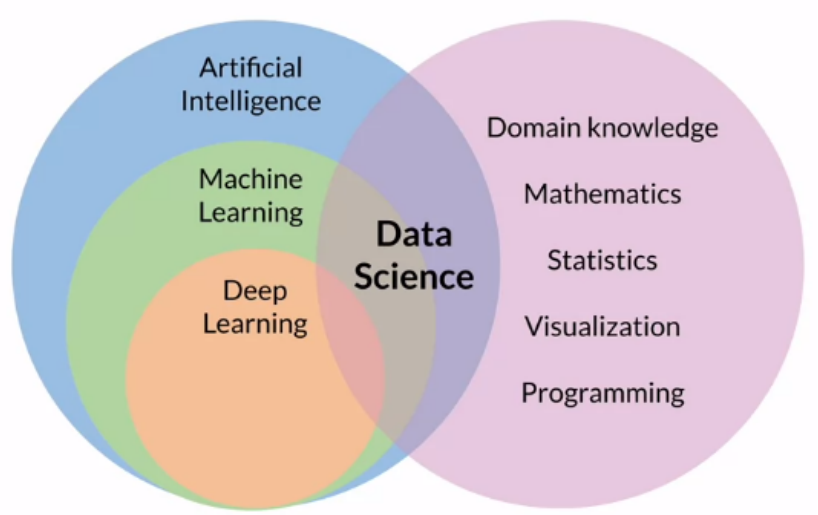
Welcome

I will start this week with a brief introduction to the discipline of practical data science and discuss the benefits of performing data science in the Cloud. Practical data science helps you to improve your data science and machine learning skills, work with almost any amount of data and implement your use cases in the most efficient way for your use case. By moving your data science projects into the Cloud, you are no longer bound by resource limitations, such as your laptop, CPU, or memory. You can run data analysis on virtually any size of data. You can slice your data and run data transformations in parallel. You can switch from CPU to GPU if you want to speed up your model training and you can do much of this in just a few clicks. In the course of this week, I will give you a short overview of the data science and machine learning concepts, you will learn to apply and the toolkits you will use. I will then move on to define the task and the use case you will focus on. To give you our first hint, you will learn how to work with text data. Specifically, you will focus on a multi-class classification for sentiment analysis of product reviews. As always, data science starts with data. I will introduce you this week to the dataset you will work with to implement this text classification task. You will learn how easy it is to ingest the data into a central repository and explore the data using various tools from our practical data science and machine learning toolbox. You will learn how to analyze the data further using interactive queries and learn how to visualize the results. The analysis will reveal important information for future tasks in the model development process, as you will see.

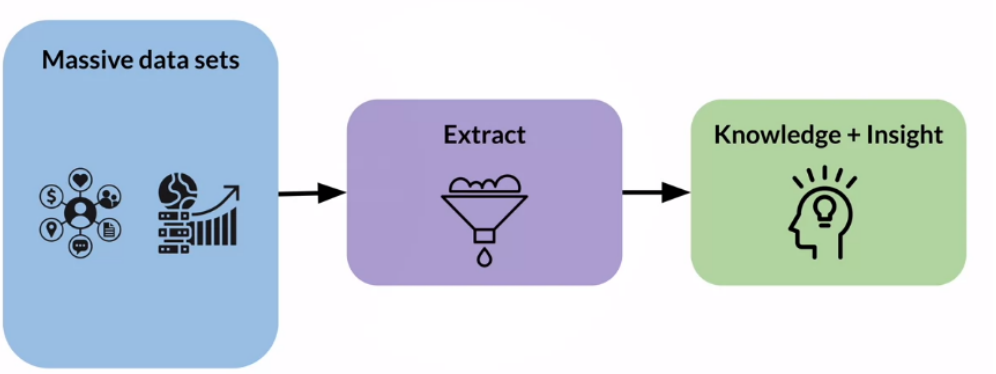
Practical Data Science

I want to briefly define what's behind practical data science in the Cloud and introduce you to the data science and machine learning concepts which you will learn to apply and the toolkits you will learn to use.

Let me start by comparing the terms artificial intelligence, machine learning, deep learning, and data science.



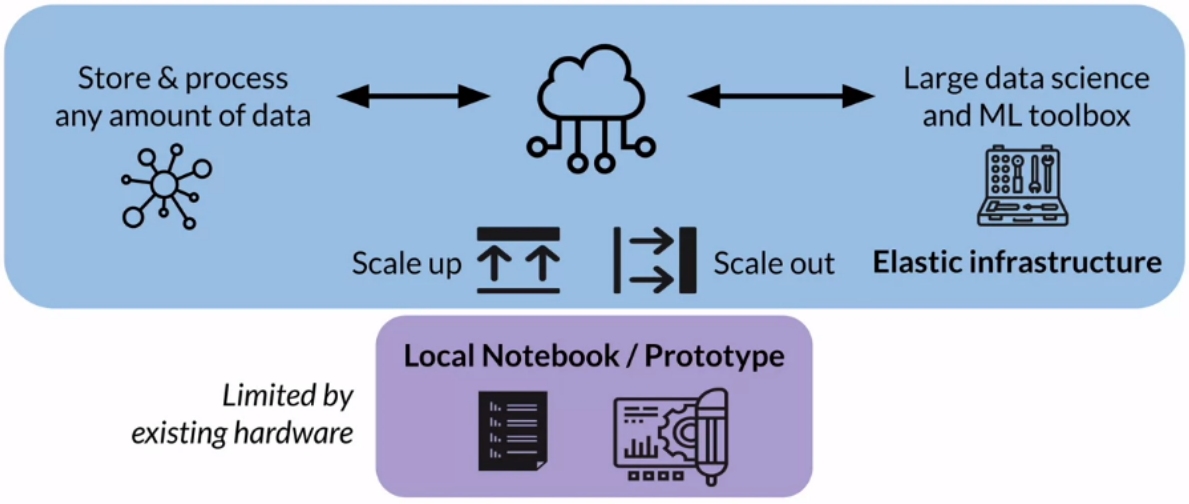
Artificial intelligence or AI is generally described as a technique that lets machines mimic human behavior. **Machine learning** or ML is a subset of AI that uses statistical methods and algorithms that are able to learn from data without being explicitly programmed. Then finally, **deep learning** is yet another subset of machine learning that uses artificial neural networks to learn from data. If you're new at data science, you see that this discipline touches all fields. **Data science** truly is an interdisciplinary field that combines business and domain knowledge with mathematics, statistics, data visualization, and programming skills.



Now, what's practical data science? **Practical data science** helps you to improve your data science and machine learning skills, work with almost any amount of data and implement their use cases in the most efficient way. It's different from working on a local development environment, such as your laptop, with small curated datasets. Practical data science is geared towards handling massive datasets that could originate from social media channels, mobile and web applications, public or company internal data sources, and much more, depending on the use case you're working on. This data is often messy, potentially error written, or even poorly documented. Practical data science tackles these issues by providing tools to analyze and clean

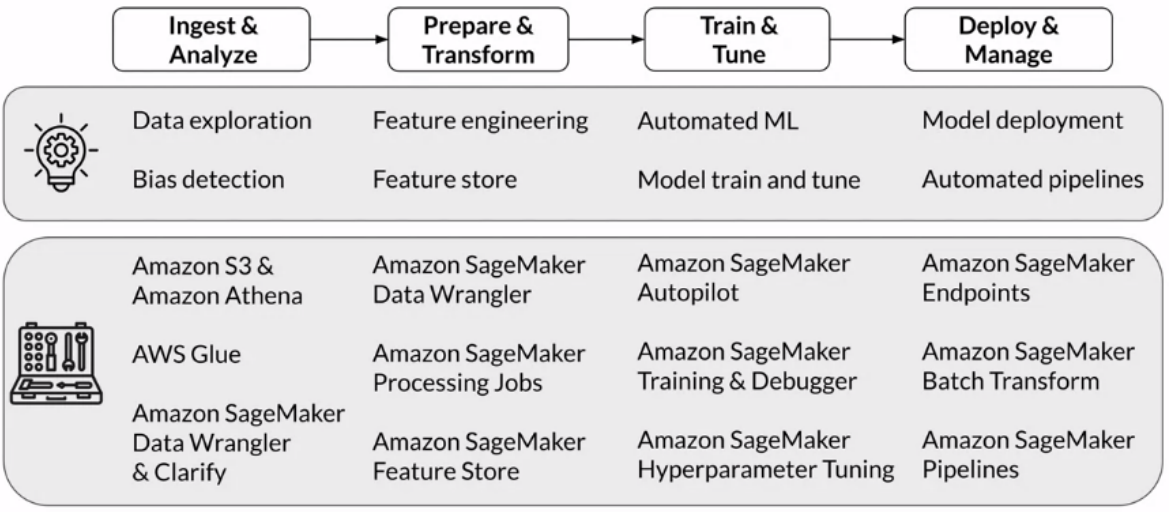
the data and to extract relevant features. This process leads to the ultimate goal of data science, which is knowledge distillation and gaining insight from those large datasets.

What's different about **practical data science in the Cloud**? If you develop data science projects in a local notebook or IDE environment for example, hosted on your laptop or company owned server pool you are limited by the existing hardware resources. For example, you have to carefully watch how much data you process and potentially lodge into memory. How much CPU processing power you have available to train and tune your model. If you need more, you need to buy additional computer resources. This process doesn't allow you to move and develop quickly. One of the biggest benefits of developing and running data science projects in the Cloud is the agility and elasticity that the Cloud offers. Maybe your model training takes too long because it consumes all of the CPU resources of the compute instance you have chosen. You can switch to using a compute instance that has more CPU resources, or even switch up to a GPU based compute instance. This is referred to as scaling up or instead of training your model on a single CPU instance, say you want to perform distributed model training in parallel across various compute instances. This is referred to as *scaling out*. Both scenarios can be accomplished in the Cloud within a few seconds. When the model training is completed, the instances are terminated as well. This means you only pay for what you actually use. This elastic infrastructure allows you to store and process almost any amount of data because the infrastructure scales too much the required resources. You can also innovate faster because you can try new datasets, new models, new code, or new machine learning libraries quickly and without any upfront investments. The Cloud also comes with a large data science and machine learning toolbox, you can choose from to perform your tasks as fast and efficiently as possible.

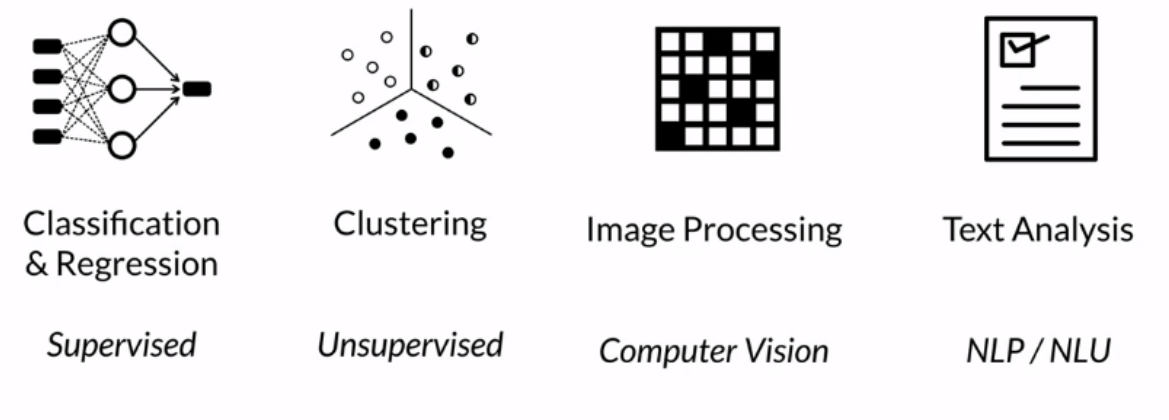


Let's have a look at a typical **machine learning workflow** and of course, every workflow starts with data. In the *ingest and analyze step*, you explore the data and analyze the data for potential statistical bias. Looking at the toolbox for this step, you will use Amazon Simple Storage Service or Amazon S3 and Amazon Athena to ingest, store and query your data. With AWS Glue, you will catalog the data in its schema. For statistical bias detection in data, you will learn how to work with Amazon SageMaker Data Wrangler and Amazon SageMaker Clarify, and don't worry right now, the tools will be explained in much more detail as you move throughout this course and the following courses of this specialization, as you learn how to apply each concept to implement the text classification use case. In the following weeks, you will step into the model development stage, where you'll start with *preparing and transforming* the data for model training. You will learn how to perform feature engineering and learn about the concept and

benefits of a feature story. Again, you will work with a powerful set of tools that are part of the Amazon SageMaker service. In the *model training and tuning stage*, you will learn how to use automated machine learning, cheek rate of first baseline, and a set of best model candidates for the use case. You will also perform custom model training and tuning in a later week. Again, you will work with additional tools part of the Amazon SageMaker service. In the *model deployment and management stage*, you will learn to discuss different model deployment options and strategies and how to orchestrate the model development as an automated pipeline. Similarly, your toolkit will be based on Amazon SageMaker.

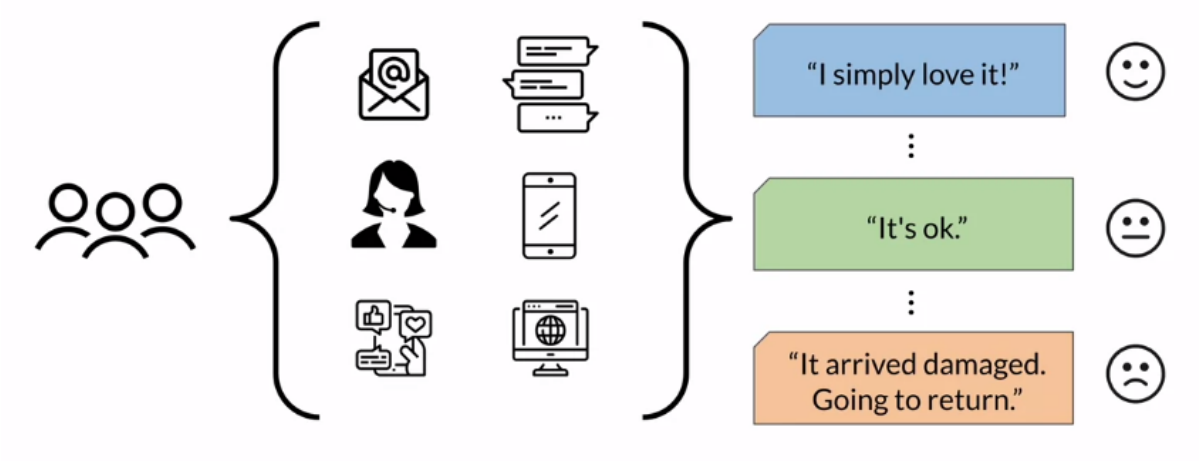


Use case and data set



Many use cases can be classified into one of the major machine learning tasks and learning paradigms. Popular machine learning tasks are *classification and regression problems*, which are examples of **supervised learning**. In supervised learning, you learn by example by providing the algorithm with labeled data. With classification, the goal is to assign the input sample to a defined class. For example, is this email I received spam or not spam? In contrast, regression applies statistical methods to predict a continuous value, such as a house price given a set of related and non related input variables. Another popular task is *clustering* and clustering is an example of **unsupervised learning**. Here the data is not labelled, hence doesn't provide any examples. The clustering algorithm tries to find patterns in the data and starts grouping the data points into

distinct clusters. A clustering use case could be identifying different customer segments for marketing purposes. If you look more at the fields of AI applications, *image processing* is a major task as part of the broader scientific field of **computer vision**. You might need to classify images into pictures of dogs and cats, identify segments in the image to help your car's driver assistance systems to distinguish between speed signs and trees or to detect brand labels in an image. Following computer vision, *text analysis* has regained popularity and research and the industry in recent years. The field of **Natural Language Processing** or NLP or Natural Language Understanding NLU have been studied since the 1950s, but thanks to advances in deep learning and neural network architectures. You can see more advanced NLP tasks such as neural machine translations, sentiment analysis, question answering and others being implemented. And you will learn much more about the field of NLP and text analysis because this is the task you will focus on in this course.



More specifically, you will perform multi-class classification for sentiment analysis of product reviews. Assume you work at an e-commerce company selling many different products online. Your customers are leaving product feedback across all the online channels. Whether it is through sending email, writing chat FAQ messages on your website, maybe calling into your support center or posting messages on your company's mobile app, popular social networks or partner websites. And as a business, you want to be able to capture this customer feedback as quickly as possible to spot any change in market trains or customer behavior and then be alerted about potential product issues. Your task is to build an NLP model that will take those product reviews as input. You will then use the model to classify the sentiment of the reviews into the three classes of positive, neutral and negative. For example, a review such as "I simply Love It", should be classified into the positive class.

Multi-class classification is a supervised learning task hence you need to provide your tax classifier model with examples how to correctly learn to classify the products and the product reviews into the right sentiment classes. A great resource to explore

Input feature for model training	Label for model training
Review Text	Sentiment
I simply love it!	1 (positive)
It's ok.	0 (neutral)
It arrived damaged, going to return	-1 (negative)

product reviews are e-commerce sites. You can use the review text as the input feature for the model training and the sentiment as a label for model training. The sentiment class is usually expressed as an integer value for model training such as 1 for positive sentiment, 0 for neutral sentiment and -1 for negative sentiment.

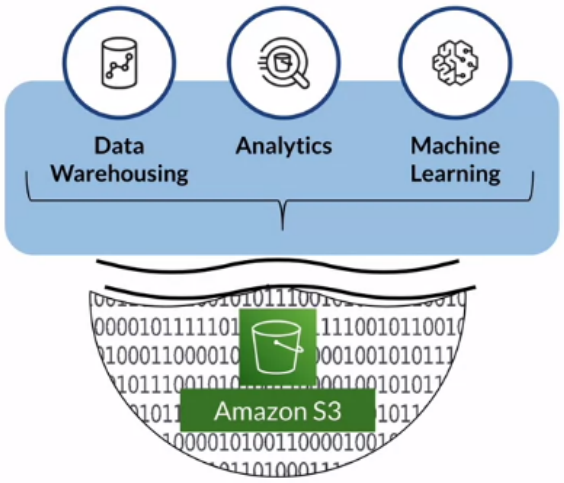
Working with Data

Data ingestion and exploration

One of the largest advantages of performing data science in the cloud is that you can store and process virtually any amount of data. The infrastructure scales elastically with the size of your data. Just think of the product reviews use case you will be working on. Imagine your e-commerce company is collecting all the customer feedback across all online channels. You need to capture sudden customer feedback streaming from social media channels, feedback captured and transcribed through supports under calls, incoming emails, mobile apps, and website data, and much more. To do that, you need a flexible and elastic repository that can start not only the different file formats, such as dealing with structured data, CSV files, as well as unstructured data such as support center call audio files. But it also needs to elastically scale the storage capacity as new data arrives.

- Centralized and secure repository
- Store, discover and share data at any scale
  - structured relational data
  - semi-structured data
  - unstructured data
  - streaming data
- Governance

Cloud based **data lakes** address this problem. Think of a data lake as this centralized and secure repository that can store, discover, and share virtually any amount and any type of data. You can ingest data in its raw format without any prior data transformation. Whether it's structured relational data in the form of CSV or TSV files, semi-structured data such as JSON or XML files, or unstructured data such as images, audio, and video files. You can also ingest streaming data, such as an application delivering a continuous feed of log files or feeds from social media channels into your data lake. A data lake needs to be governed. With new data arriving at any point in time you need to implement ways to discover and catalog the new data. You also need to secure and control access to the data to comply with the political data security, privacy, and governance regulations. With this governance in place, you can now give data signs and machine learning teams access to large and diverse datasets.



- Amazon Simple Storage Service (Amazon S3)
- Object storage
- Durable, available, exabyte scale
- Secure, compliant, auditable

Data lakes are often built on top of objects storage such as **Amazon S3**. You're probably familiar with file and block storage. File storage stores and manages data as individual files organized in hierarchical file folder structures. In contrast, block storage stores and manages data as individual chunks called the blocks. Each block receives a unique identifier, but no additional metadata is stored with that block. With object storage, data is stored and managed as objects, which consists of the data itself, any relevant metadata, such as when the object was last modified and a unique identifier. Object storage is particularly helpful for storing and retrieving growing amounts of data of any type hence it's the perfect foundation for data lakes. Amazon S3 gives you access to durable and high available object storage in the cloud. You can ingest virtually anything from just appear dataset files to exabytes of data. AWS also provides additional tools and services to assist you in building a secure, compliant, and auditable data lake on top of S3. With a data lake in place, you can now use this centralized data repository to enable data warehousing analytics and also machine learning.

AWS Data Wrangler

- Open source Python library
- Connects pandas DataFrames and AWS data services
- Load/unload data from
  - data lakes
  - data warehouses
  - databases

```
!pip install awswrangler

import awswrangler as wr
import pandas as pd

# Retrieving the data directly from Amazon S3
df = wr.s3.read_csv(
     path='s3://bucket/prefix/')
```

Now, let me introduce some additional toolbox items you will be working on this week. One of them is **AWS Data Wrangler**. AWS Data Wrangler is an open source Python library developed by members of the AWS professional services team. The library connects Pandas DataFrame with AWS data-related Services. Pandas is a very popular Python data analysis and manipulation tool. AWS Data Wrangler offers abstracted functions to load or unload data from

data lakes, data warehouses or databases on AWS. You can install the library through the "*pip install awswrangler"* command. Here's a sample code snippet that shows how you can work with AWS Data Wrangler. First, you import the library together with Pandas. If you want to read, for example, CSV data from your S3 data lake into a Pandas DataFrame, you can run this command shown here using the S3 read CSV function, providing the S3 path to your data.

AWS Glue Data Catalog

Another tool you will be using this week is the **AWS Glue Data Catalog**. This data catalog service is used to register or catalog the data stored in S3. Similar to taking inventory in a shop, you need to know what data is stored in your S3 data lake or bucket as an individual container for objects is called. Using the Data Catalog Service, you create a reference to the data basically S3 to table mapping. The AWS Glue table which is created inside an AWS Glue database only contains the metadata information such as the data schema. *It's important to note that no data is moved. All the data remains in your S3 location*. You catalog where to find the data and which schema should be used to query the data. Instead of manually registering the data, you can also use **AWS Glue Crawler**. A Crawler can be used and set up to run on a schedule or to automatically find new data which includes inferring the data schema and also to update the data catalog.

AWS Glue
Data Catalog

Name	reviews
Database	dsoaws_deep_learning
Classification	csv
Location	s3://<bucket>/<prefix>

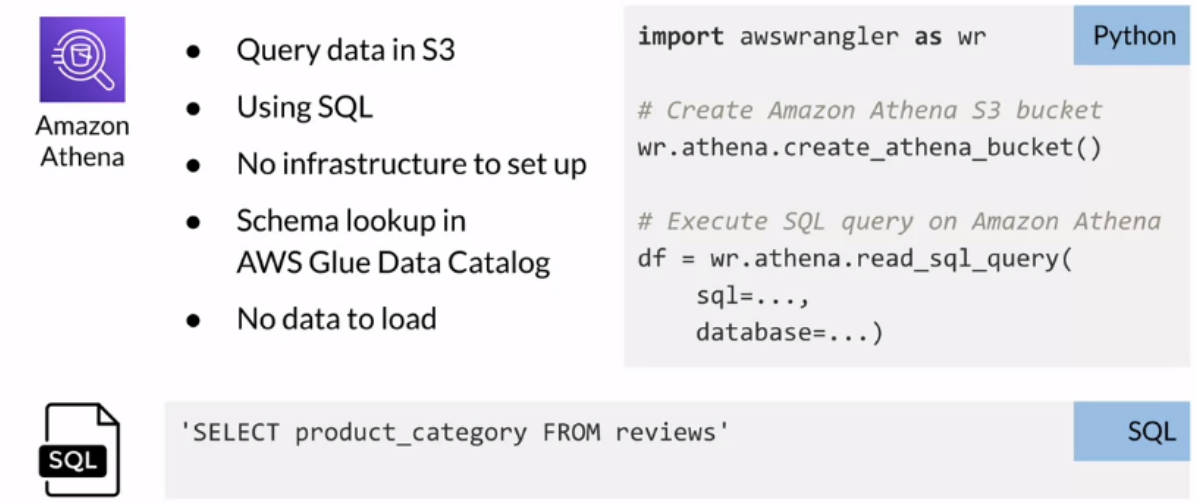
- Creates reference to data ("S3-to-table" mapping)
- Just metadata / schema stored in tables
- No data is moved
- *AWS Glue Crawlers* can be set up to automatically
  - infer data schema
  - update data catalog

Now how can you register the data? You can use the AWS Data Wrangler tool just as I introduced. The first step is to create an AWS Glue Data Catalog database. To do that, import the AWS Wrangler Python library as shown here, and then call the *catalog.create_database* function, providing a name for the database to create. AWS Data Wrangler also offers a convenience function called *catalog.create_CSV_table* that you can use to register the CSV data with the AWS Glue Data Catalog. The function will only store the schema and the metadata in the AWS Glue Data Catalog table that you specify. The actual data again remains in your S3 bucket.

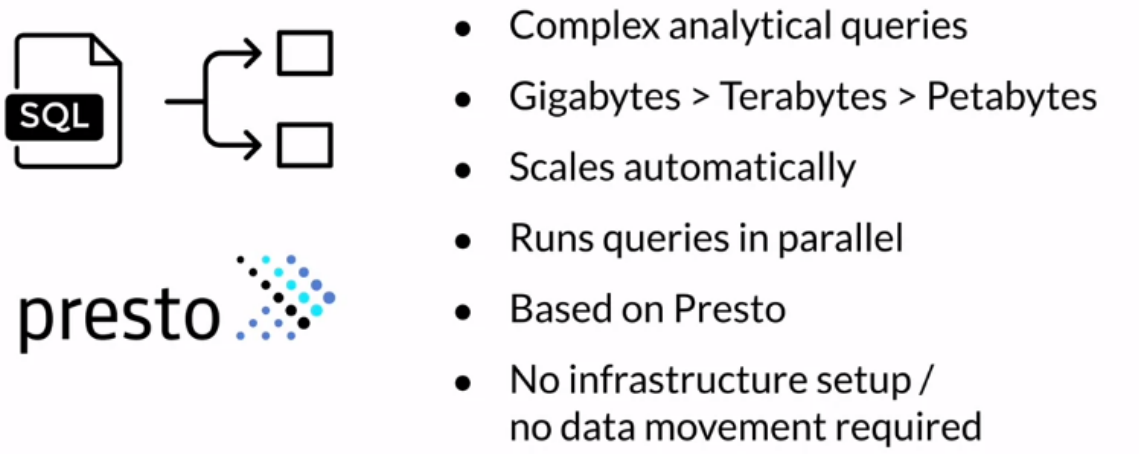
```
import awswrangler as wr

# Create a database in the
# AWS Glue Data Catalog
wr.catalog.create_database(
     name=...)

# Create CSV table (metadata only) in the
# AWS Glue Data Catalog
wr.catalog.create_csv_table(
     table=...,
     column_types=...,
     ...)
```



Now you can query the data stored in S3 using a tool called **Amazon Athena**. Athena is an interactive queries service that lets you run standard SQL queries to explore your data. Athena is serverless which means you don't need to set up any infrastructure to run those queries, and no matter how large the data is that you want to query, you can simply type your SQL query referencing the dataset schema you provided in the AWS Glue Data Catalog. No data is loaded or moved, and here is a sample SQL query. Let's list all product categories from the AWS Glue table called reviews, and imagine this table points to the dataset stored in S3. To run this query, you can use the previously introduced AWS Data Wrangler tool again. From the Python environment you're working in, just use the Data Wrangler, *athena.read_SQL_query* function. Pass in the SQL statement you have written and point to the AWS Glue database which contains the table you'll reference here in the SQL query. Again, this database and table only contains the metadata of your data. The data still resides in S3, and when you run this Python command, AWS Data Wrangler will send this SQL query to Amazon Athena. Athena then runs the query on the specified dataset and stores the results in S3, and it also returns the results in a Pandas DataFrame as specified in the command shown here.



Given this simplicity, you might wonder what is so special about this, and I have to admit the SQL query I've shown here was fairly simple. But just imagine building highly complex analytical queries to run against not just gigabytes, but potentially terabytes or petabytes of data. Using Athena, you don't have to worry about any compute and memory resources to support this query, because Athena will automatically scale out and split the query into simpler queries to run in parallel against your data. Athena is based on Presto, an open source distributed SQL engine developed for this exact use case, running interactive queries against data sources of all sizes. Remember, no installation or infrastructure setup is needed, and no data movement is required. Just register your data with AWS Glue and use Amazon Athena to explore your datasets from the comfort of your Python environment.

Data visualization

Visualizing your data is one of the most effective ways of exploring your data across multiple dimensions at once. Depending on what kind of data you are exploring and what kind of relationships in the data you're looking for. The type of visualizations you use might be different. Let's take a look at a few of those visualizations you will use this week, and the tools you will work with.

Pandas, as introduced earlier, is used for data analysis and data manipulation. **NumPy is** used to perform scientific computing in python. Similarly, matplotlib and Seaborn, are popular python libraries for creating visualizations. **Matplotlib** helps to create static animated and interactive visualizations. **Seaborn** is based on matplotlib, and adds statistical data visualizations.

You will use these tools to visualize the product reviews.

The purpose of exploring and visualizing the data set is to better understand data set characteristics, from simple things such as understanding the number of samples you have available for model training to answering more complex business questions. Let me start with the first one. To get an understanding of how many samples the data set contains, let's run this SQL query using amazon Athena, which will return the number of reviews in each sentiment class. The query result is best suited to be visualized in the bar chart.

SQL Query

```
SELECT sentiment, COUNT(*) AS count_sentiment
FROM dsoaws_deep_learning.reviews
GROUP BY sentiment
ORDER BY sentiment DESC, count_sentiment
```

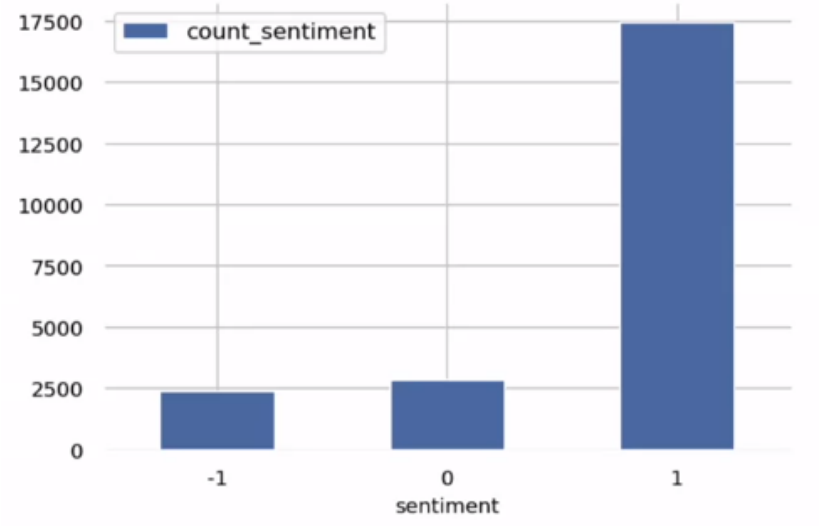
Here is the visualization code. I import the matplotlib library, for a simple bar chart, you can use the pandas data frame that holds the query results, and then call the plot bar function directly. You define the data frame columns which are used as the X and Y axis data, and at any additional data such as title to the plot. You can also use matplotlib to enrich the plot, let's say with labels for the X and Y axis. And when you're done, you can call the show function, and here can see the result in a sample bar chart.

Python visualization code

```
import matplotlib.pyplot as plt
chart = df.plot.bar(
     x="sentiment",
     y="count_sentiment")

plt.xlabel("sentiment")
plt.show(chart)
```

The bar chart also makes it easy to see that the positive sentiment class here, numbered with one, contains many more samples than any other of the classes. You basically have an unbalanced distribution of samples across the sentiment classes. You will learn how to address this in later week.



Here is another interesting visualization, this time I want to show you how to calculate percentiles and visualize data distributions. For this purpose, I will calculate the distribution of the review length or the number of words per review. The SQL query here is pretty straightforward. I split the review text in the column review body by a space character which gives me a list of the individual words, and then I calculate the cardinality which gives me the number of words.

SQL Query

```
SELECT CARDINALITY(SPLIT(review_body, ' ')) as num_words
FROM dsoaws_deep_learning.reviews
```

Before I plot the distribution, I want to calculate the percentiles. You can use the described function on the panda's data frame that contains the review length, and you specify the percentiles to calculate. To visualize the distribution of review length, I choose a histogram. Histograms represent frequency distributions. They show how often each different value occurs.

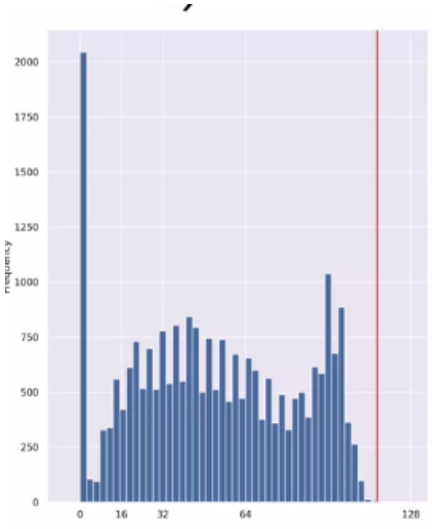
In this case, I want to group the review length in 100 bins, and I add a red marker to highlight the 100th percentile.

Python visualization code

```
summary = df["num_words"].describe(
     percentiles=[0.10, 0.20, 0.30, 0.40, 0.50, 0.60, 0.70, 0.80, 0.90, 1.00])

df["num_words"].plot.hist(
     xticks=[0, 16, 32, 64, 128, 256], bins=100,
     range=[0, 256]).axvline(x=summary["100%"], c="red")
```

Let's have a look at the result, and here we go. On the right, you can see sample results of the percentile calculations. You can see that the shortest review had only one word, and if you remember I grouped all review length into the 100 bins, which are represented here by the blue bars in the histogram. The fewer bins you specify, the fewer bars show up here. The X axis shows the length and the Y axis represents their frequency. And if you look at the 100 percentile highlighted and right here, you can see that all of the reviews have 115 words fewer in this sample. And this information will be very helpful later in the course when you start building the text classifier model.



mean	52.51
std	31.38
min	1.00
10%	10.00
20%	22.00
30%	32.00
40%	41.00
50%	51.00
60%	61.00
70%	73.00
80%	88.00
90%	97.00
100%	**115.00**

Additional reading material

If you wish to dive more deeply into the topics covered this week, feel free to check out these optional references. (You won't have to read these to complete this week's practice quizzes.)

- AWS Data Wrangler
- AWS Glue
- Amazon Athena
- Matplotlib
- Seaborn
- Pandas
- Numpy

Week 2: Data Bias and Feature Importance

Statistical bias and feature importance

Statistical Bias

What is **statistical bias**? A data set is considered to be biased if *it cannot completely and accurately represent the underlying problem space*. For those of you familiar with statistics, you know that statistical bias is a tendency of a statistic to either overestimate or underestimate a parameter. In this course, you will learn about statistical biases in training data sets, which are imbalances in these training data sets. In these biased data sets, *some elements of a data set are more heavily represented than others*. Let's say for example, you are trying to build a financial services model that can be used to ***detect fraud***. The training data that you will use to build this model is a previous set of credit card transactions that a business has access to. Now for the majority of the time, credit card transactions are not fraudulent, which is really good for the business. But if you're using that biased data set to train a model to detect fraud, then your model is very unlikely to detect fraudulent transactions because it has not seen that many fraudulent transactions before. One way to address this problem is to *add more examples* of how fraud transactions would look like to your training data set. Biased data sets typically lead to biased models, and the biased models could have both business and regulatory consequences for the businesses that use these models.

- Training data does not comprehensively represent the problem space
- Some elements of a dataset are more heavily weighted or represented

Fraud Detection

Biased models

- Imbalances in product review dataset

Let's take another example. Let's construct a ***product's review*** data set. Such a data set could be biased if it contains disproportionately large number of reviews for, let's say, one product category called A and fewer number of reviews for other categories like product category B and C. When you build a product sentiment prediction model with this biased data set, the resulting model could very well detect sentiment of new products that belong to product category A. But for newer products that belong to other categories such as product category B and C, your sentiment model is not going to be really accurate.

Statistical bias causes

What causes **statistical bias**? How does bias even get introduced into your datasets? There could be multiple reasons for that.

The first one that we see here is **activity bias**. This is biases that exist in *human-generated content*, especially on social media. Think about all the data that has been collected over these social media platforms over the last several years. Reality is that a very small percentage of the population is actively participating on these social media platforms. So the data that has been collected over the years on these platforms is not representative of the entire population.

The second one is very similar but slightly different. This is **societal bias**. This is once again, biases in *data that is generated by humans*, but maybe not just on social media. These biases could be introduced because of preconceived notions that exist in society. Data generated by humans can be biased because all of us have unconscious bias.

Sometimes bias can be introduced by the machine learning system itself. Let's say, for example, a machine learning application gives users a few options to select from, and once the user selects an option, the user selection is used as training data to further train and improve the model. This introduces *feedback loops*. Take, for example, a streaming service. You want to watch a movie on the streaming service and the streaming service makes a few recommendations for you and you decide to watch Dancing with Wolves. You like the movie and you rate it high. From then on, the streaming service is recommending you the movies that have wolves in them. It's partly because of the feedback you provided to the service. But in reality, maybe you watched that movie because you like the actress in the movie and you don't even particularly like wolves. Situations like this could result in **selection bias** that includes a feedback loop that involves both the model consumers and the Machine Learning model itself.

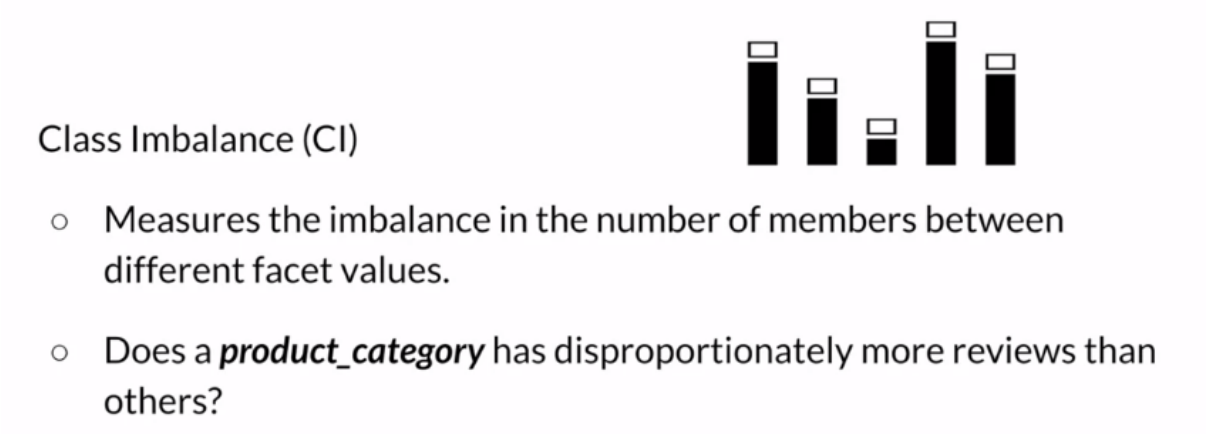
Now, even if you detect some of the statistical biases in your dataset prior to training your model, once the model is trained and deployed, drift can still happen. **Data drift** happens, especially when the *data distribution significantly varies from the distribution of the training data that was used to initially train the model*. This is called data drift and also data shift. There are several different variations of data drift. Sometimes the distribution of the independent variables or the features that make up your dataset can change. That's called *covariant drift*. Sometimes the data distribution of your labels or the targeted variables might change. That's the second one, which is *prior probability drift*. Sometimes the relationship between the two, that is the relationship between the features and the labels can change as well. That's called *concept drift*. Concept drift also called concept shift can happen when the definition of the label itself changes based on a

particular feature like age or geographical location. Take, for example, my experience. Last time when we traveled a few years ago across US on a road trip, we quickly found out that the soft drinks are not called the same across US. So when we stopped for meals and ordered soft drinks, we realized that soda is not called soda across the US. In some areas, it's called pop, and in some areas, it's called soda. Now, if you think about all the geographies across the world, you can only imagine the interesting combinations, different labels you can come up with. With all these issues that could potentially happen with your datasets, it becomes really important to continuously monitor and detect various biases that could be prevalent in your training datasets before and after you train your models.

Measuring statistical bias

Now, you're ready to measure the imbalances and the statistical bias in your dataset using specific metrics. When I talk about these metrics, it's important to understand that these metrics are applicable to a particular facet of your dataset. A **facet** is a sensitive feature in your dataset that you want to analyze for these imbalances. Let's take, for example, the product review dataset. In that dataset or product category could be a facet or a feature interest for you that you want to analyze for imbalances.

The first metric that I will introduce is **class imbalance**. It is very easy intuitively to understand this. Class imbalance, or CI, measures the imbalance in the number of examples that are provided for different facet values in your dataset. When you apply this to the product review dataset, it answers this particular question, does a particular product category, such as product category A, have a disproportionately larger number of total reviews than any other category in the dataset.



The next metric that I will introduce is DPL, this is the difference in proportions of labels. This metric measures the imbalance of positive outcomes between the different facet values. When applied to the product review dataset, what this metric is measuring is if a particular product category, say product category A, has disproportionately higher ratings than other categories. While CI, the metric that we just saw as measuring if a particular category has a total number of reviews higher than any other categories, DPL is actually looking for higher ratings than any other product categories. Now, I'm introducing only a couple of different metrics here, but there are several other metrics that can be used to measure different portions of bias across your data-sets.

Difference in Proportions of Labels (DPL) ✓ ✗

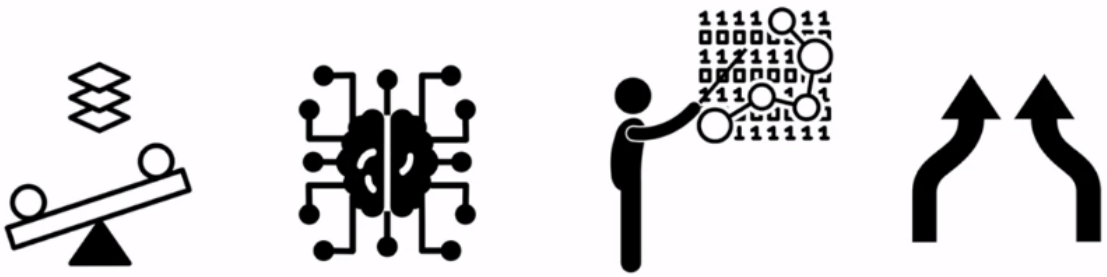
- Measures the imbalance of positive outcomes between different facet values.
- Does a ***product_category*** has disproportionately higher ratings than others

Detecting statistical bias

Next I will talk about two different tools that can help with detection of statistical bias in your training data sets. The two tools are **SageMaker Data Wrangler** and **SageMaker Clarify**. First, I will introduce Data Wrangler. Data Wrangler provides you with capabilities to connect to various different *sources* for your data, *visualize* the data and transform the data by applying any number of *transformations* in the Data Wrangler environment. And detect statistical bias in your data sets and *generate reports* about the bias detected in those data sets. It also provides capabilities to provide *feature importance* calculations on your training data set. You will have a chance to explore the various capabilities of Data Wrangler in different parts of the course. For this section, I will be focused on the ability to detect statistical bias and generate bias reports on the training data sets.

Detect statistical bias with Amazon SageMaker Clarify

Next, I will introduce **Amazon SageMaker Clarify** as a tool to perform statistical bias detection on your datasets. SageMaker Clarify can perform *statistical bias detection* and *generate bias reports* on your training datasets. Additionally, it can also perform bias detection in trained and deployed models. It further provides capabilities for *machine learning explainability*, as well as *detecting drift in data and models*. For now, I'm going to focus on the statistical bias detection and report generation capabilities of Clarify.



Statistical Bias Report | Model Bias Report | Explainability | Drift

To start using Clarify APIs, start by importing the Clarify library from the SageMaker SDK. Once you have the Clarify library, construct the object, SageMaker Clarify Processor using the library. **SageMaker Clarify Processor** is a construct that allows you to scale the bias detection process into a distributed cluster. By using two parameters, *instance type* and *instance count*, you can scale up the distributed cluster to the capacity that you need. Instant count represents the number of nodes that are included in the cluster, and instance type represents the processing capacity of each individual node in the cluster. The processing capacity is measured by the node's compute capacity, memory, and the network [inaudible]. Once you have configured the distributor cluster, next, you specify an *S3 location* where you want the bias report to be saved to. That's the parameter bias report output path.

```
from sagemaker import clarify

clarify_processor = clarify.SageMakerClarifyProcessor(
    role=role,
    instance_count=1,
    instance_type='ml.c5.2xlarge',
    sagemaker_session=sess)

bias_report_output_path = << Define S3 path >>
```

Distributed cluster size

Type of each instance

S3 location to store bias report

Once this step is done, the next step is to configure the data config object on the Clarify library. The **data config object** represents the details about your data. As you can expect, it has the *input and output location of your data*, an S3, as well as the label that you're trying to predict using that dataset. In this case here, the *label* that we are trying to predict is sentiment.

```
bias_data_config = clarify.DataConfig(
    s3_data_input_path=...,
    s3_output_path=...,
    label='sentiment',
    headers=df_balanced.columns.to_list(),
    dataset_type='text/csv')
```

Data Configuration

Next, you configure the **bias config object** on the Clarify library. The bias config object captures the *facet or the featured name* that you are trying to evaluate for bias or imbalances. In this case, you're trying to find out imbalances in the product category feature. The parameter *label values* or threshold defines the desired values for the labels. If the sentiment feature is your label, what is the desired value for that label? That value goes into the parameter label values or threshold.

```
bias_config = clarify.BiasConfig(
    label_values_or_threshold=[...],
    facet_name='product_category')
```

Bias Configuration

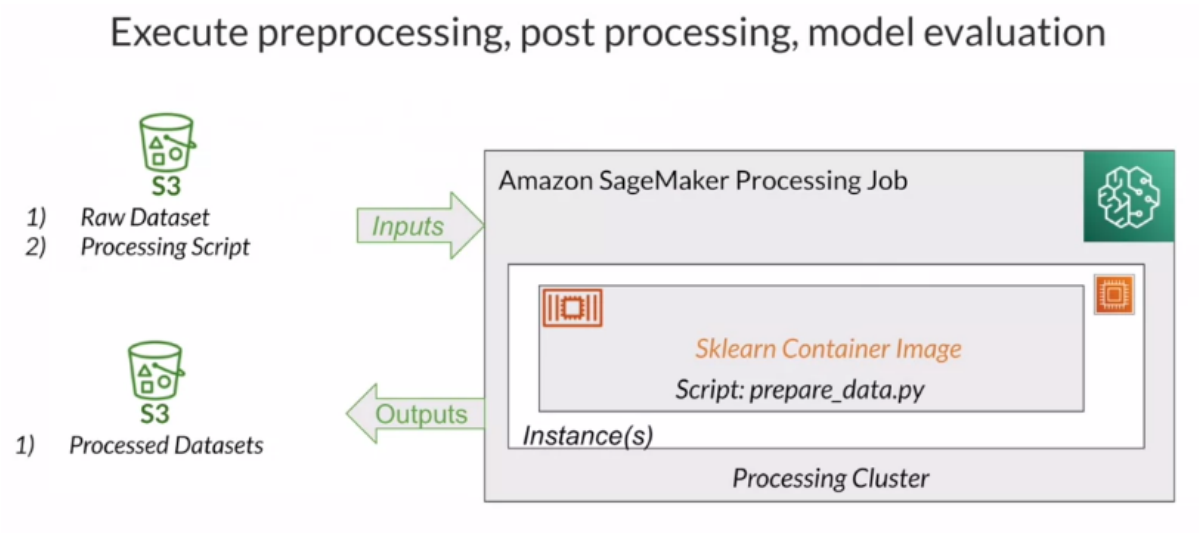
Once you have configured those three objects, you are ready to run the **pre-training bias method** on the Clarify processor. In addition to specifying the *data config* and the *data bias config* that you already configured, you can also specify the *methods* that you want to evaluate for bias. These methods are basically the metrics that you've already learned about to detect bias. The metrics here are the CI, the class imbalance, and the DPL. You can also specify a few other methods here as well. The *wait* parameter specifies whether this bias detection job should block the rest of your code or should it be executed in the background. Similarly, the *logs* parameter specify whether you want to capture the logs or not.

```
clarify_processor.run_pre_training_bias(
    data_config=...,
    data_bias_config=...,
    methods=["CI", "DPL", ...],
    wait=<<False/True>>,
    logs=<<False/True>>)
```

Pre training bias job

Once the configuration of the pre-training bias method is done, you launch this job. In the background, SageMaker Clarify is using a construct called **SageMaker Processing Job** to execute the bias detection at scale. SageMaker Processing Jobs is a construct that allows you to perform any data-related tasks at scale. These tasks could be executing pre-processing, or post-processing tasks, or even using data to evaluate your model. As you can see in the figure here, the SageMaker Processing Job expects the data to be in an S3 bucket. The data is collected from the S3 bucket and processed on this processing cluster which contains a variety of containers in the cluster. By default, containers for scikit-learn, Python, and a few others are supported. You can also have the opportunity to bring your own custom container as well. Once

the processing cluster has processed the data, the transformed data or the processed data is put back in the S3 bucket.



Now, you understand how to use the Clarify APIs and what happens behind the scenes as well. What do you think happens when you execute this run pre-training bias? What do you think the result is going to be? *The result will actually be a very detailed report on the bias on your dataset that has persisted in S3 bucket.* You can download the report and review in detail to understand the behavior of your data.

Approaches to statistical bias detection

Now the question becomes, which one of these tools should you use, in which situation?

The first option, **Data Wrangler**, provides you with more of a UI based visual experience. So, if you would like to connect to multiple data sources and explore your data in more visual format and configure what goes into your bias reports by making selections from drop down boxes and option buttons. And finally, launch the bias detection job using a button click, Data Wrangler is the tool for you. Keep in mind that Data Wrangler is only using a subset of your data to detect bias in that data set.

On the other hand, **SageMaker Clarify** provides you with more of an API based approach. Additionally, Clarify also provides you with the ability to scale out the bias detection process. SageMaker Clarify uses a construct called processing jobs that allow you to configure a distributed cluster to execute your bias detection job at scale. So, if you're thinking of large volumes of data, for example, millions of millions of rows of product reviews and you want to explore that data set for bias. Then, SageMaker Clarify is the tool for you, so that you can take advantage of the scale and capacity offered by Cloud.

Feature importance: SHAP

Next, I will talk about feature importance and the open source framework SHAP that is behind feature importance. In this section, you will also learn about using SageMaker Data Wrangler to calculate feature importance on the product review data set.

Feature importance is the idea of explaining the individual features that make up your training data set using a score called important score. Some features from your data set could be more relevant or more important to your final model than others. Using feature importance, you can rank the individual features in the order of their importance and contribution to the final model. Feature importance allows you to evaluate how useful or valuable a feature is in relation to the other features that exist in the same data set. Let's take for example, the product review data set. It consists of multiple different features and you are trying to build a product sentiment prediction model out of that data set. You would be interested in understanding what features will play a role towards that final model, that is where feature importance comes into picture.

- Explains the features that make up the training data using a score (importance).

- How useful or valuable the feature is relative to other features.
- Predict the sentiment for a product → Which features play a role?

Feature importance is based on a very popular open source framework called **SHAP**, SHAP stands for *shapley additive explanations*.

- **Open Source Framework - SHAP**
  - Shapley values based on game theory.
  - Explain predictions of a ML model
    - Each feature value of training data instance is a player in a game
    - ML prediction is the payout
  - Local vs global explanations
  - SHAP can guarantee consistency and local accuracy.

The framework itself is based on *shapley values*, which in turn is based on game theory. To understand how SHAP works, consider a play or a game in which multiple players are involved and there is a very specific outcome to the play that could be either a win or a loss. Shapley values allow you to attribute the outcome of the game to the individual players involved in the

game. Translating that into the machine learning world, you can use the same concept to *explain the predictions* made by the machine learning model. In this case, the individual players would be the individual features that make up the data set and the outcome of the play would be the machine learning model prediction. So using the same concept, you can explain how the predictions will correlate to the individual feature values that make up your training data set. Using the SHAP framework, you can provide both *local and global explanations*. While the local explanation focuses on indicating how an individual feature contributes to the final model. The global explanation takes a much more comprehensive view in trying to understand how the data in its entirety contributes to the final outcome from the machine learning model. SHAP framework is also very extensive in nature in that it considers all possible combinations of feature values along with all possible outcomes for your machine learning model. Because of this extensive nature, the SHAP framework could be very time intensive, but also because of this extensive nature, SHAP can provide you with guarantees in terms of *consistency and local accuracy*.

Additional reading material

If you wish to dive more deeply into the topics covered this week, feel free to check out these optional references. (You won't have to read these to complete this week's practice quizzes.)

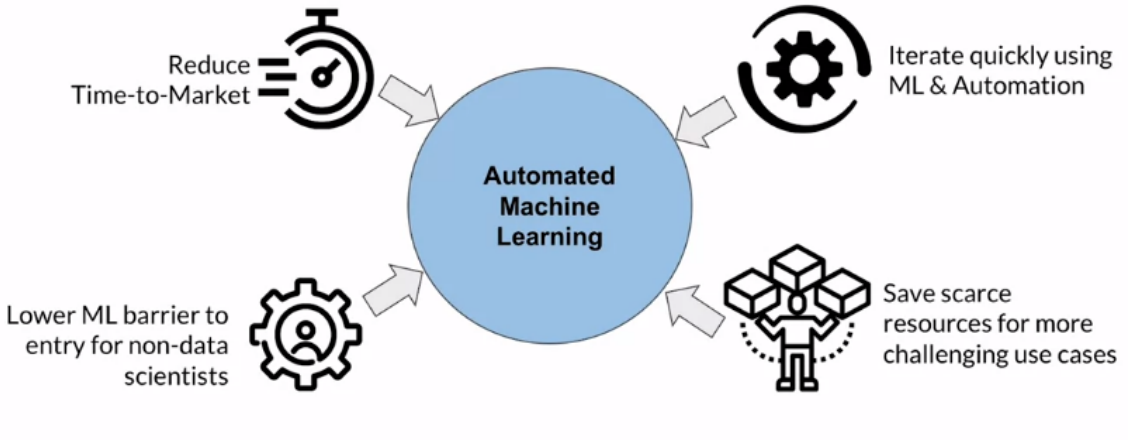
- Measure Pretraining Bias - Amazon SageMaker
- SHAP

Week 3: Use Automated Machine Learning to train a Text Classifier

Automated Machine Learning

Automated Machine Learning (AutoML)

So why are we going to use auto ml for this use case when you're trying to build machine learning models to solve everyday problems, it's common to run into model building challenges for a number of different reasons. First, the steps involved in creating a machine learning model typically involve multiple iterations that can often result in **increased time to market**.



Second, machine learning can also require **specialized skill sets** that can be challenging to find or staff from your existing teams. Also, machine learning experiments or **iterations typically take much longer** than traditional development life cycles. This is largely due to the time it takes to get model performance feedback and the time it takes to run through the numerous experiments using different combinations of data transformations, algorithms and hyper parameters until you find that model that is performing and meeting at least your minimum objective metric that you've identified for success. The nature of machine learning development can also make it difficult to iterate quickly, not only from a workflow perspective, but also from a resource perspective. This also includes scarce computing resources. If you're limited by on premises compute resources or by scarcity of human resources that have data science skill sets.

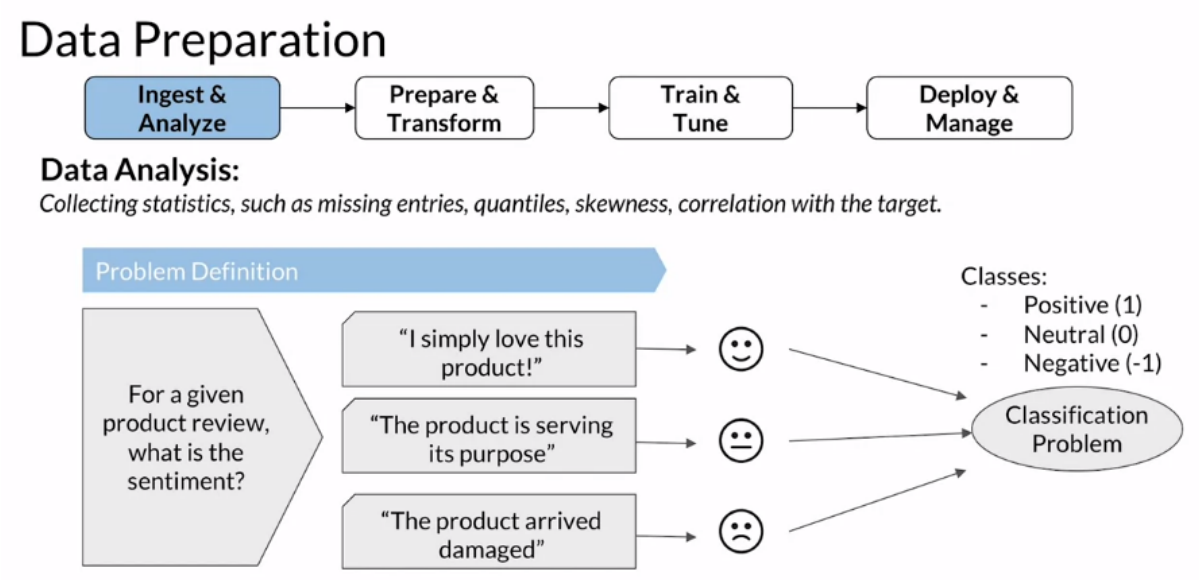
So what are some ways that you can work around some of these challenges? This is where **AutoML** comes in. I'll use the term automated machine learning and auto ml interchangeably throughout this week. But the key thing to note is that I'm referring to a concept, not a specific tool or implementation of auto ml. So conceptually, auto ml uses machine learning to automate many of the tasks in the machine learning workflow, allowing you to address some of those challenges. I just discussed one, it can **reduce time to market** by automating resource intensive tasks like data transformation, feature engineering and model tuning. 2nd, auto ml can enable your non data scientists to **build machine learning models without requiring that deep data science skill set**. 3rd, auto ml let's you **iterate quickly** by using machine learning and automation to perform the majority of the task in your model building workflows. I also talked

about the challenge of scarce compute resources where you may be constrained by on premises resources for your different experiments. Using auto ml in combination with cloud computing also **addresses potential compute resource challenges** due to capacity constraints, which can also cause those longer iterations, because you can't train or tune your models in parallel. Finally, auto ml let's data scientists **focus on those really hard to solve machine learning problems**.

AutoML Workflow

Let's dive into some of the tasks within each of these workflow stages to better understand automl capabilities.

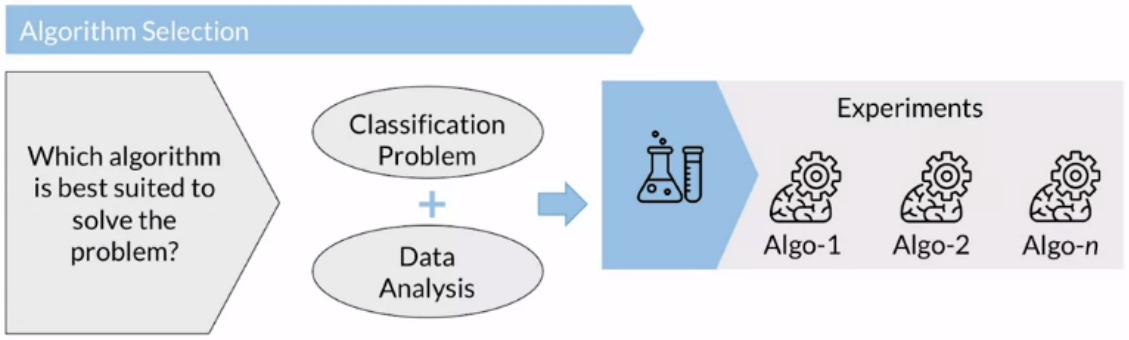
In the first week of this course, we talked about **data ingestion and exploration** for the machine learning problem that you're trying to solve.



In this case you're ingesting product review data that is labeled meaning the data set includes the target that you're trying to predict. So in this case your data has the target classes of positive, neutral or negative for each product review on input based on your data, you can further refine your problem definition to **determine what class of problem you're trying to solve**. So is this a regression problem or is this a classification problem? In this case, you can see that you're really trying to determine whether a specific product review is either positive, neutral or negative, meaning you have 3 different classes that you're trying to predict. So in this case you have a classification problem.

Once you've narrowed down your machine learning problem, you want to **identify potential algorithms** that you want to try as part of your training experiments, after you've done some analysis on your data and determine the type of machine learning problem that you're trying to solve for. You can then look at which algorithm or algorithms that are best suited for your data and the problem you're trying to solve, when you perform that data analysis you want to understand your data. So getting insight into things like data distribution, attribution correlation or do you have potential quality issues in your data? Like missing data, then based on your machine learning problem combined with your data analysis you're able to identify the algorithm or the algorithms that you'd like to try for your experiments after you've done these things. You

can then outline some experiments, in this case let's say you decided to use *xgboost* for your first experiment which you'll also see inside the lab for this week as well. Xgboost is an implementation of gradient boosted decision trees and can be used for classification problems as well as regression. However, selecting the right algorithm or algorithms is only part of the process for each algorithm. There are also **a number of hyper parameters that you need to consider** as you tune your model for optimal performance. Also, each algorithm can have different expectations in terms of the format of the data that it expects on input for training. So let's take a look at a few considerations related to the product review data set.



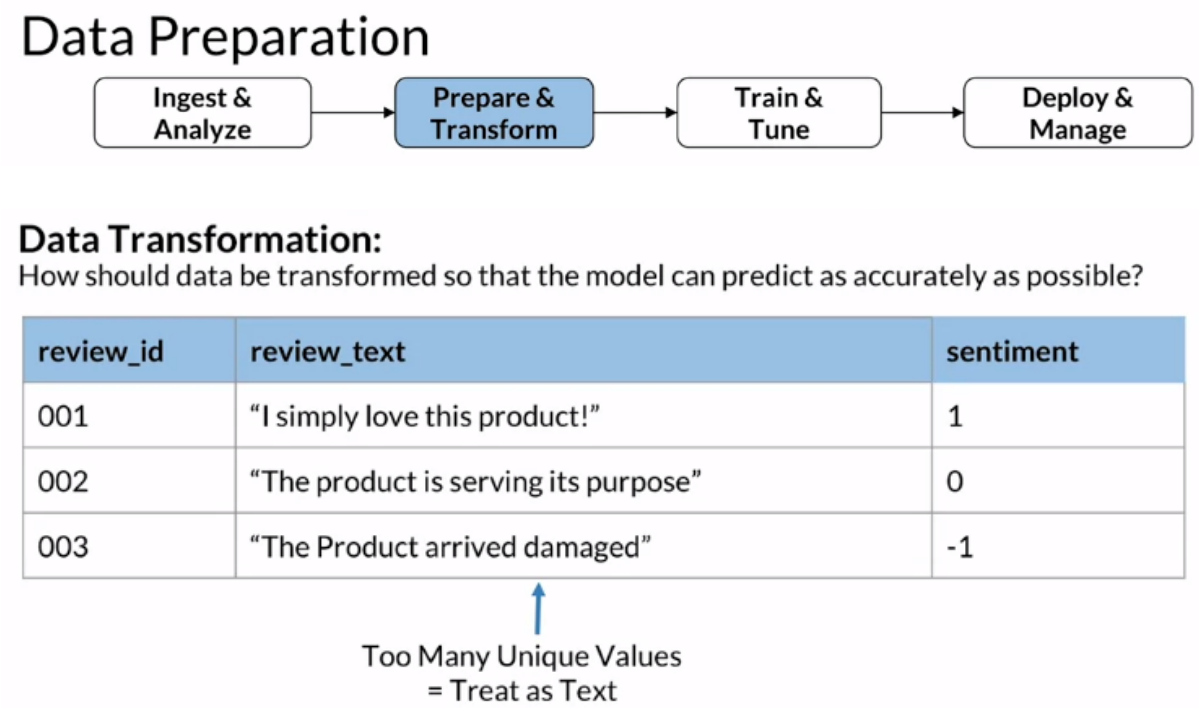
The **schema** for the product review data set includes three attributes on input including first *review id*. Which is just a numeric unique identifier for a specific review. The *review text* which is your text based or categorical data that contains the actual product review and finally the *sentiment* for the review. This is your target or your label that you're trying to predict. This attribute. In this case it is numeric. These numeric values represent the classes where 1 is positive, 0 is neutral and -1 is negative. So looking at your dataset schema, what transformations are you going to need to make so that you can help ensure that your selected algorithm in this case xgboost can accept and understand your data on input for training.

Dataset Schema Detection

Numeric	Categorical	Numeric
review_id	review_text	sentiment
001	"I simply love this product!"	1
002	"The product is serving its purpose"	0
003	"The Product arrived damaged"	-1

Looking at your data in combination with some of the statistics that you previously captured and analyzed. You know that *review text is categorical but you also know that it contains too many unique values to be impactful to your algorithm* if you tried something like *one hot encoding*. So in this case you want to treat your attributes text and apply text processing transformations instead, text transformation is a pretty broad topic because it can include some additional data processing like using *tokenization* to convert sentences to words, removing **stop words** or words such as *the* or *is*, which may not be impactful to the overall performance of your model. You then typically perform some type of *feature extraction* where you map your textual data, two vectors

and there are a number of different techniques to do this and text transformations can often take a lot of time and effort to optimize. In the lab. This week, you'll use one technique called *term frequency inverse document frequency* or TFIDF. I'll cover that a bit more later in this session. Finally your last attribute is your label, which in this case we have three unique classes and it's already a numeric format which looks good for our algorithm.



And if you remember, data preparation also includes looking for **class imbalance** or signals of **data bias**. Using statistical bias detection techniques, to balance data, you'll want to determine how you plan to handle class imbalance, which can involve things like *changing your performance metric, applying resampling techniques, generating synthetic data or even changing your selected algorithm*, as an example. Xgboost tends to handle class imbalance well, but it also supports additional hyper parameter tuning to further tune for data imbalance, in classification problems. So like you see here where the number of positive reviews is significantly larger than the neutral or negative reviews.

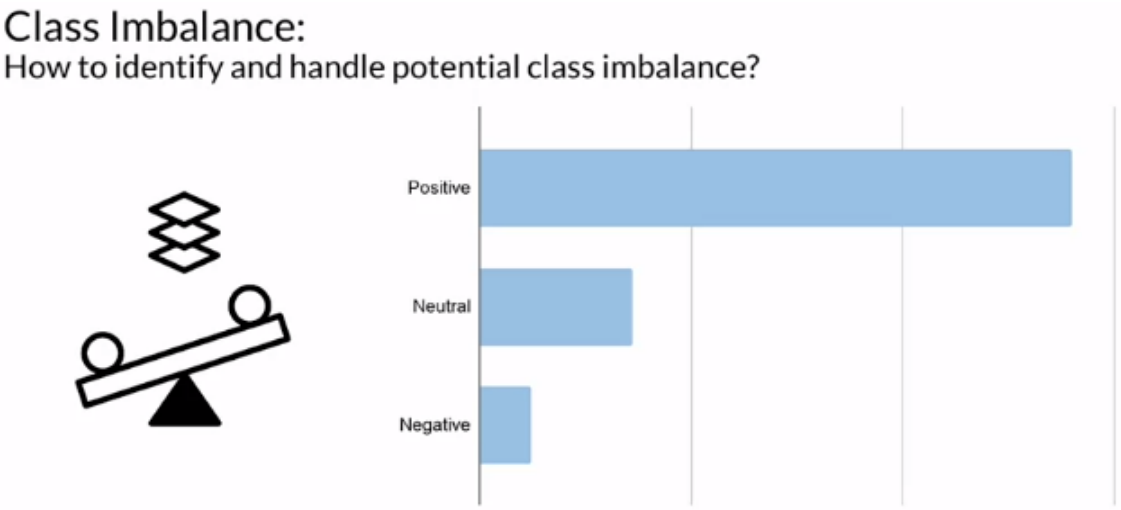
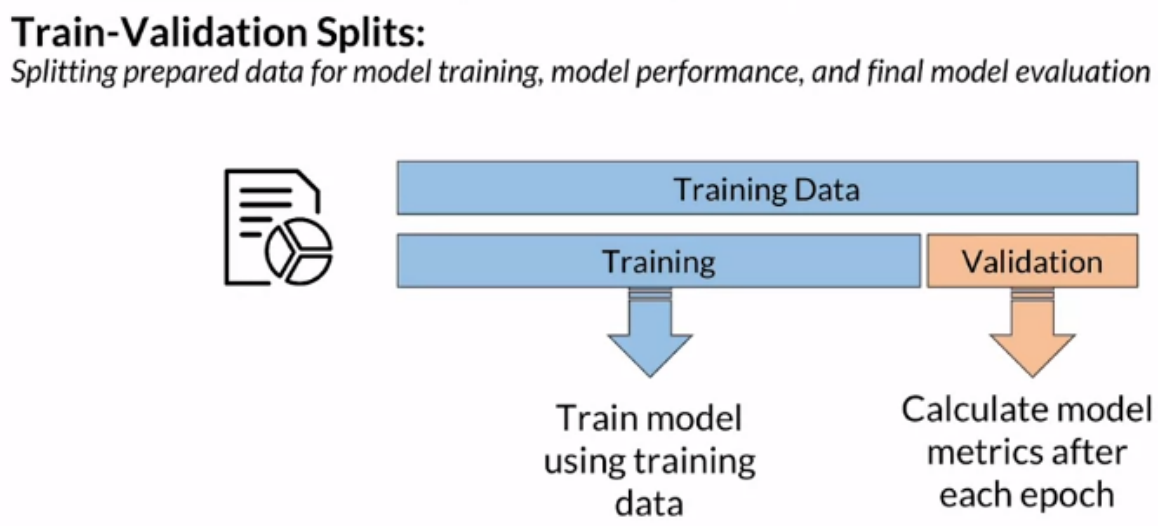


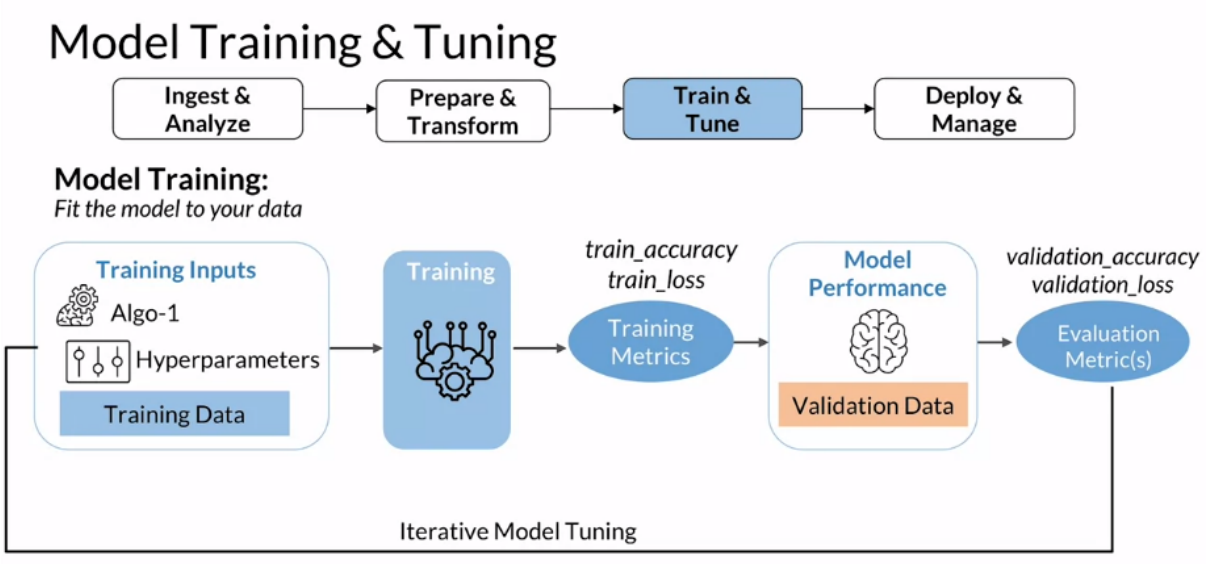
Figure out how to handle problems like class imbalance can consume a lot of your cycles and require multiple experiments using different combinations of data transformations and training tasks before finding that optimal combination of data transformations, algorithm and hyper parameter that gives you the results that you need.

This leads me to the next part of your workflow and your final prepare and transform task. Once you've done your data transformations, you can then use your process data set to **create your training and validation data sets**, for this. You reserve the largest portion of your data for training your model. So this is the data that the model learns from and you can use it to calculate model metrics such as training accuracy and training loss. The validation data set is a second dataset or holdout data set created from your fully processed training data set and you'll use it to evaluate your model performance, usually after each epoch or full pass through the training set. The purpose of this evaluation is to fine tune the model hyper parameters and determine how well your model is able to generalize on unseen data, here you can calculate metrics like validation accuracy and validation loss.



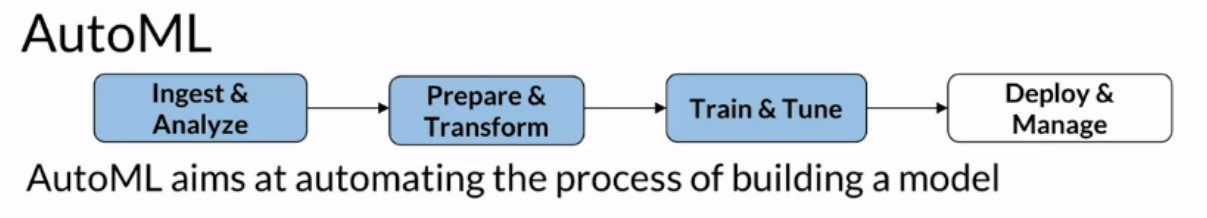
After you have your data sets ready, you can now move on to **model training and tuning**. Model training and validation is highly iterative and typically happens over many experiments, during this step. Your goal is to determine *which combination of data algorithm hyper parameters. Results in the best performing model*. For each combination that you choose. You need to train the model and evaluate it against that holdout data set.

Then you repeat these steps until you have a model that is performing well according to objective metric, whether that's accuracy or something like an F1 score depending on what you're optimizing for. As you can imagine, all of these iterations can take a lot of computation and you typically want to be able to rapidly iterate without running into bottlenecks. So this is where training at scale comes in, the cloud gives you access to on demand resources that allow you to train an experiment at scale without wait time or scheduling. Training time on, on premises resources that are often constrained or limited by GPU, CPU or storage, without resource limitations. You can also further optimize your training time using capabilities like distributed training or taking advantage of parallel processes.



So I just covered the high level steps from data preparation to model training and tuning. And as you can see, there's a lot of work that goes into understanding your data, determining the best algorithm or algorithms to use, performing your data preparation or transformations and finally training and tuning for each of your experiments. Each combination of data algorithm in hyper parameters can consume many human hours and compute hours before getting to a model that is performing well.

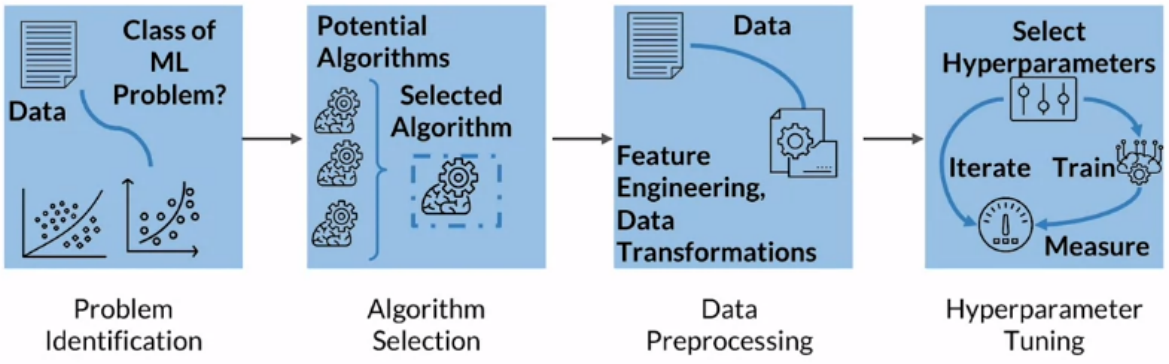
So this is where **AutoML** comes in. There are a lot of different implementations of automl, but in general, automl allows you to reduce the need for data scientists to build machine learning models because it uses machine learning to automate the machine learning workflow tasks that are highlighted here in blue and which I just covered in detail for a product review case. So does this mean that we no longer need data scientists, not at all. Data scientists are still critical, but the goal is to allow data scientists to focus on those really hard to solve machine learning problems. This can also include having data scientists refine the data transformations or the code that's generated by automl, to further optimize those results that are produced through automated machine learning. So I'll spend some time now talking about the kinds of tests that automated machine learning is designed to accomplish.



Without AutoML you first provide your labeled data set which includes the target that you're trying to predict. Then AutoML is going to automatically do some analysis of that data and determine the type of machine learning problem. So is this a binary classification, multi class classification or regression problem? Automl will then typically explore a number of algorithms and automatically select the algorithm that best suits your ML problem and your data.

Once AutoML selects an algorithm, it will automatically explore various data transformations that are likely to have an impact on the overall performance of your model. And then it will automate the creation of those scripts that will be necessary to perform those data transformations across

your tuning experiments. Finally, automl will select a number of hyper parameter configurations to explore over those trading iterations to determine which combinations of hyper parameters and feature transformation code results in the best performing model, AutoML capabilities, reduce a lot of the repetitive work in terms of building and tuning your models through the numerous iterations and experiments that are typically required.



Build models without any ML expertise

- Empower more people in your organization: software developers, business people
- Let experts focus on **hard problems**

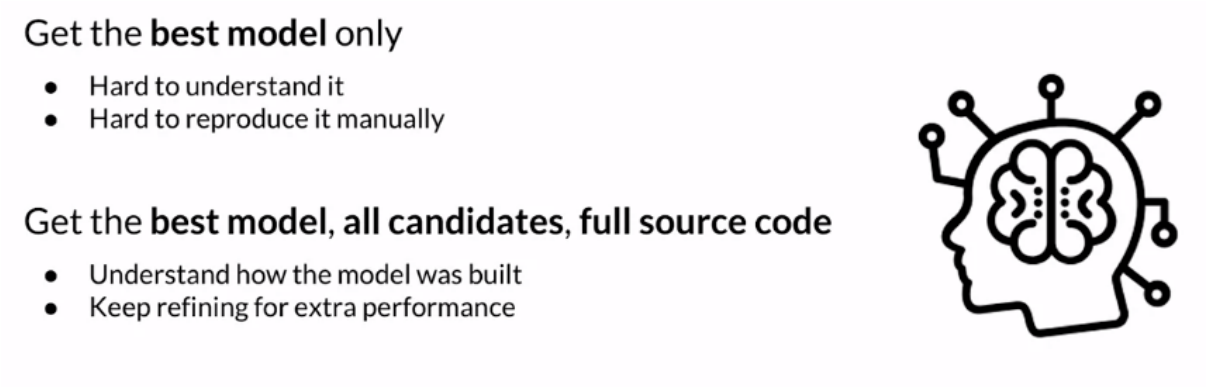
Experiment and build models at scale

- Thousands of data sets can be modeled without human intervention
- Let experts focus on **new problems**

Automate the majority of the work, then tweak

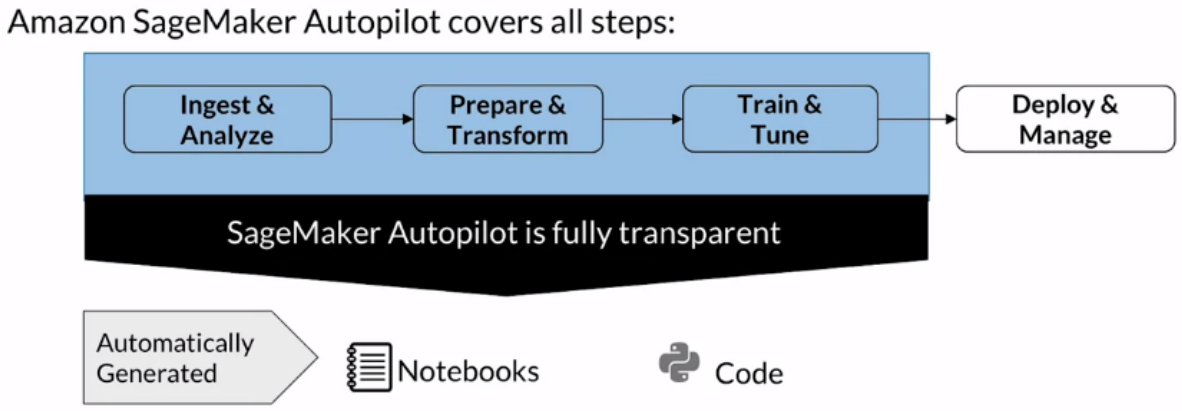
- Data cleaning, feature engineering, feature selection, etc.
- Let experts focus on high value tasks such as **domain knowledge**, and **error analysis**.

However, there are symptom considerations when selecting an implementation of automl, depending on the implementation of automl, that you choose or you're deciding to use. There may be a balance in terms of iterating faster but still maintaining the transparency and control that you may be looking for. Some implementations of automl provide limited visibility into the background experiments, which may produce a really performant model, but that model is often hard to understand, explain or reproduce manually. Alternatively, there are implementations of automl that not only provide the best model, but they also provide all of the candidates and the full source code that was used to create that model. This is valuable for being able to understand and explain your model, but it also allows you to take that model and potentially further optimize it for extra performance by doing things like applying some of that additional domain knowledge or doing some additional feature engineering on top of the recommended feature engineering code.



Amazon SageMaker Autopilot

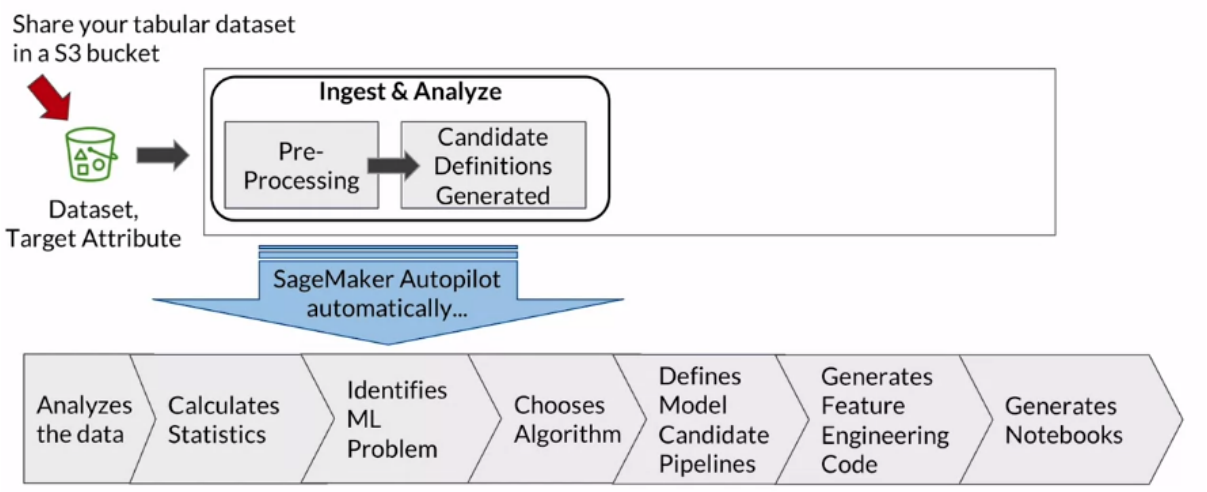
Autopilot is fully transparent, meaning it will automatically generate and share the feature engineering code and generate Jupiter notebooks that walk you through how the models were built. This includes the data processing as well as the algorithms, the hyperparameters, and the training configuration. You can then use those automatically generated notebooks to reproduce the same experiment or perform modifications to that example to continue to refine your model.



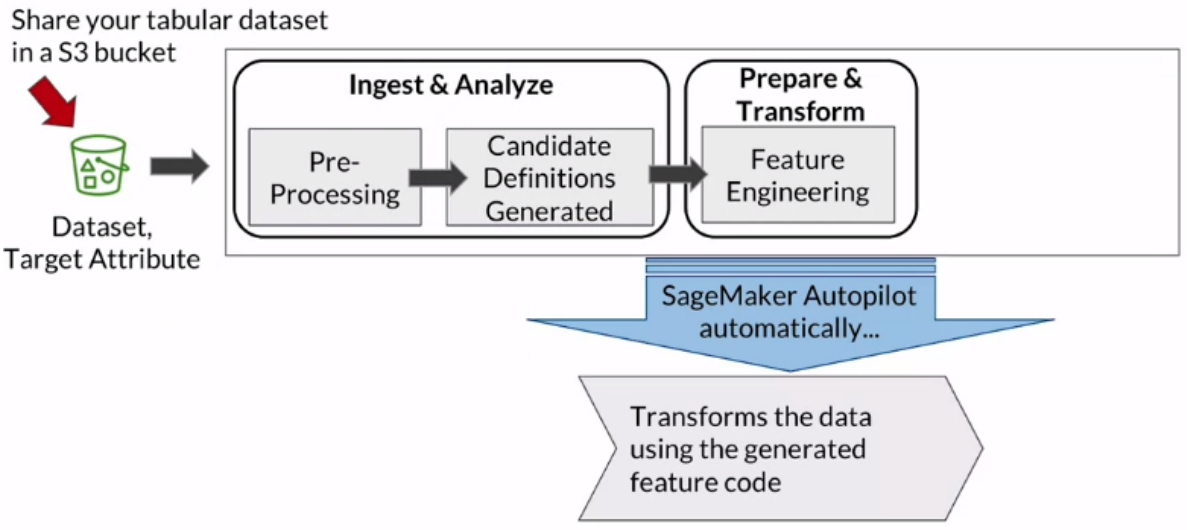
Let's look at how Autopilot works at a high level.

First, you need to upload your tabular data set to a bucket in Amazon Simple Storage Service, or **Amazon S3**, which is an Object Storage Service on AWS. That data set must contain your *target attribute*, which in this example that we've been working with, is your product review sentiment. You tell autopilot what the target attribute is when creating your autopilot experiment. Autopilot will then analyze your data, identify the machine learning problem if you did not predefine it, and then choose the ML algorithm that most closely matches the input data and the target that you're trying to predict. Autopilot supports several built-in algorithms that include linear learner, XGBoost, and a deep learning algorithm. The available algorithms cover the problem types of regression, binary classification, and multi-class classification. Autopilot will then generate feature engineering code to transform your data into the format that is expected by the algorithm. After the data analysis step, autopilot will *generate two types of notebooks* that describe the plan that autopilot allows to create the candidate models. First, there's a *data exploration Notebook* that describes what Autopilot learned about your data. This Notebook also identifies areas of investigation that may indicate potential issues in your source data that could require further analysis or impact your model performance. The second type of Notebook is a *candidate*

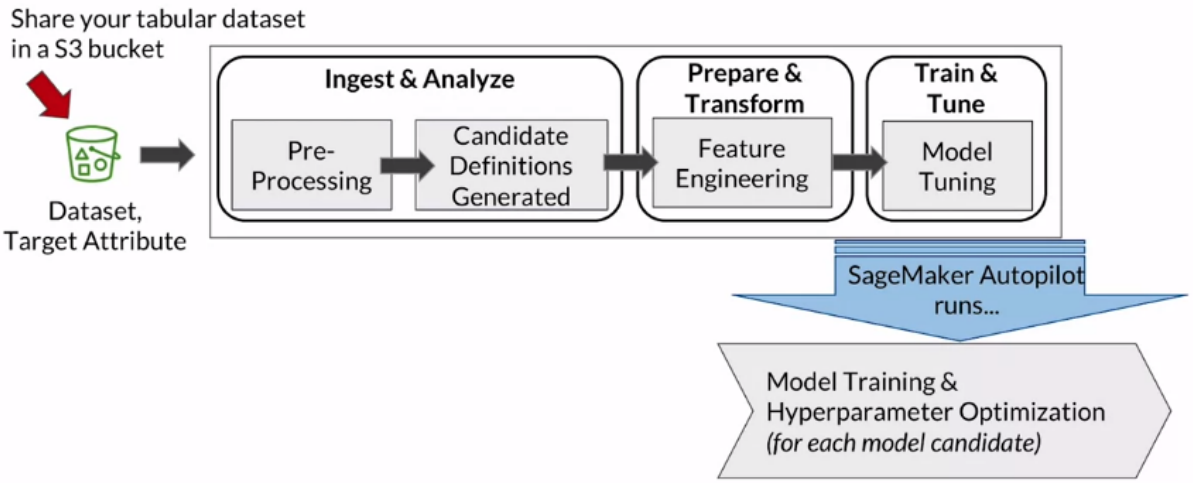
generation Notebook, which contains each suggested data preprocessing step, the algorithm and the hyperparameter ranges that will be used for the tuning job. I previously talked about the handling of class imbalance. Sagemaker Autopilot can provide models that are more accurate even when data sets are highly imbalanced, and have as few as 500 data points. You can also direct SageMaker Autopilot to use the area under the curve, or the area under the receiver operating characteristic curve as the objective metric to create even more accurate models for classification problems.



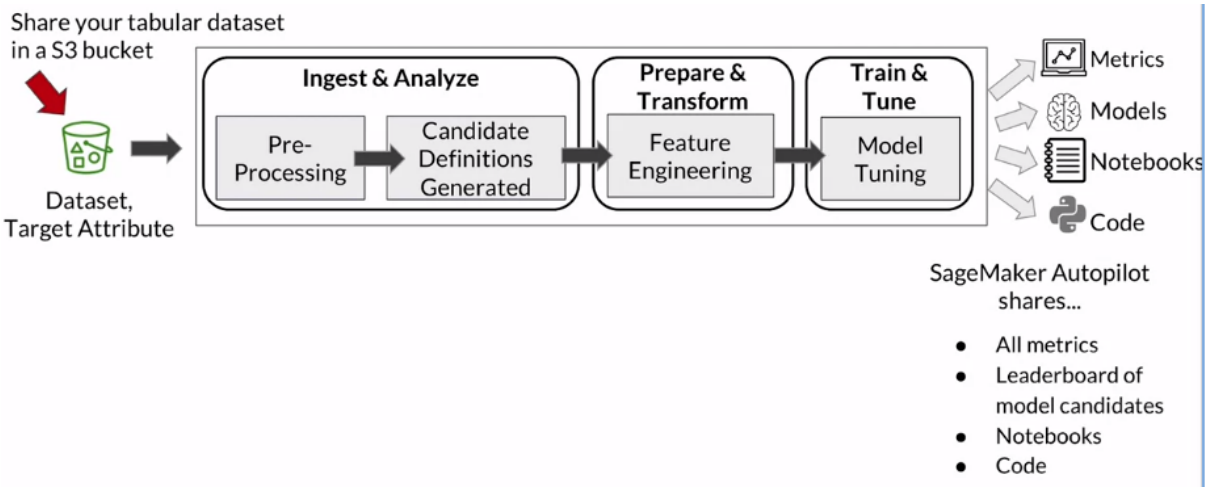
When you set up AutoPilot, you can use it in different ways. You can choose to use it completely on autopilot, as the name suggests, which will automatically flow through the next steps that you see here, starting with *running the Data transformations* using the generated feature transformation code. Optionally, you can choose to run it with different degrees of human intervention as well. As an example, you can set up autopilot so that it does not automatically run the feature engineering code in the data analysis step. This is good if you want to be able to first look at the candidate notebooks and then decide which ones to train and tune.



Otherwise, to use it in *true autopilot mode*, you can let Autopilot automatically run through the data transformation pipeline, perform the model training and tuning for each of the candidates, and then provide you with the best candidate based on the experiments it runs.



Finally, autopilot shares a leaderboard of all the candidate models, including their metrics, with a ranked list of recommendations to determine the best-performing model. Autopilot also provides that complete visibility into the feature engineering code, the algorithm, and the optimized hyperparameters that were used, which is designed to give you that control and transparency that I discussed earlier.



Running experiments with Amazon SageMaker Autopilot

In this section, I'll cover running experiments using SageMaker Autopilot, and an experiment just refers to an autopilot job.

In this scenario, you're going to use Autopilot to create a **classification model** that will predict one of three classes based on product review input text. In this case, you want to predict the sentiment which includes a label of either negative one for negative, zero for neutral and one for positive for each review on input. You're going to let all

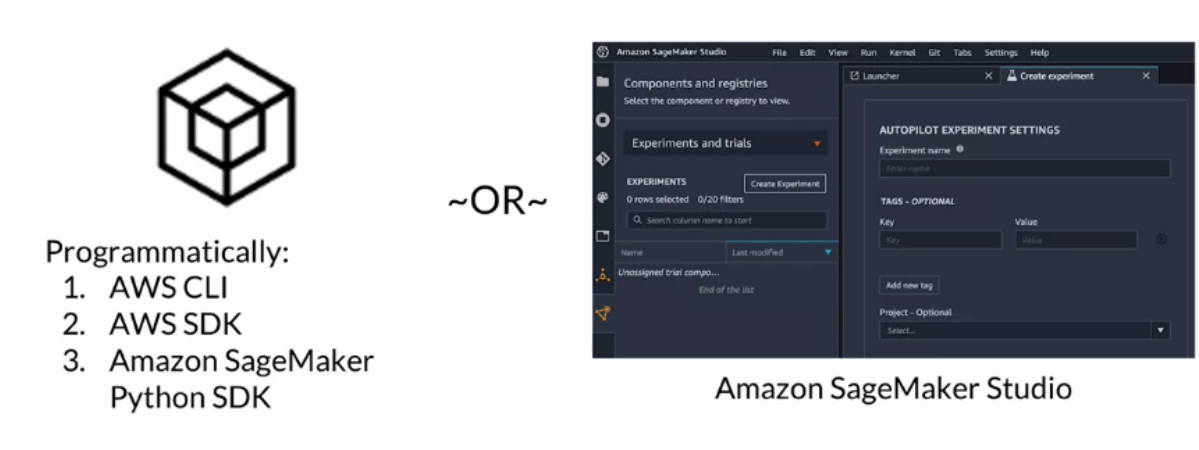
Use Case: Analyze Customer Sentiment

Goal: Use SageMaker Autopilot to find the optimal feature transformations, algorithm, and hyperparameters to produce a best performing model allowing us to predict our label (sentiment) based on product reviews (review_body)

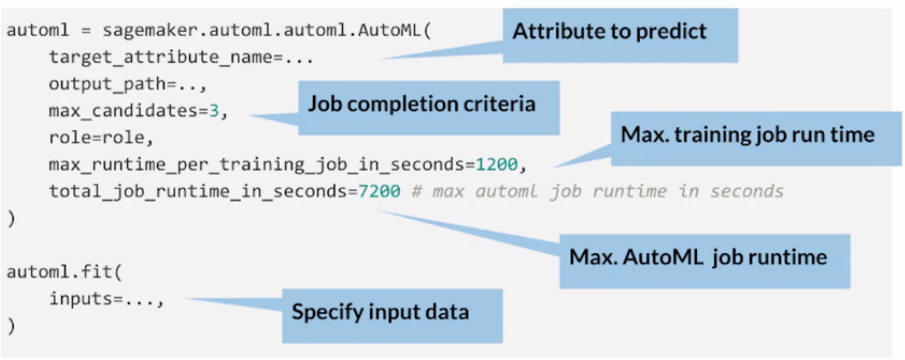
sentiment	review_body
-1	This is bad.
0	This is OK.
1	This is great!

the Autopilot find the right combination of feature transformations, algorithm and hyperparameters that generate the best-performing model.

As I go through Autopilot and as you work with it, an important thing to note is that you can interact with Autopilot in multiple ways, including one programmatically, going through the **SageMaker API** and using the AWS Command Line Interface, the AWS SDKs or the SageMaker Python SDK. Or you can also interact with Autopilot through **SageMaker Studio**, which is an integrated workbench for end-to-end machine learning workflow activities. Whether you're interacting with Autopilot programmatically or through Studio, you're still calling the same set of APIs, and you'll get to see both options in this session, but you'll focus on working with Autopilot programmatically in the lab.

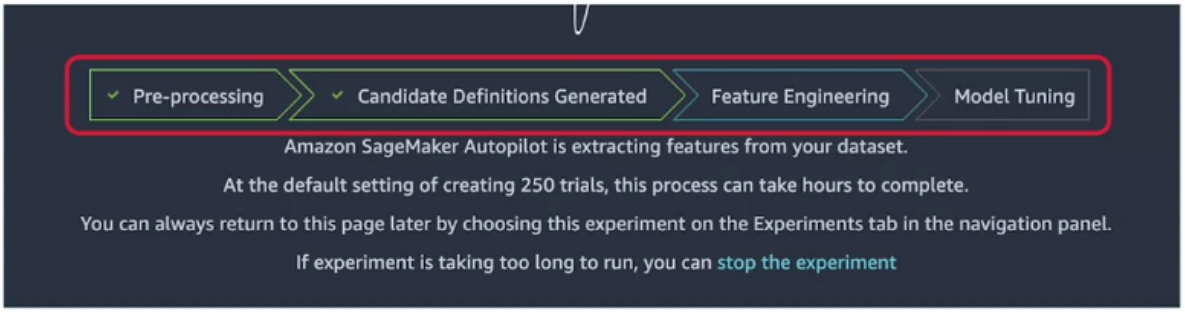


To get started, I'll show you how to **configure your Autopilot job programmatically**. First, you'll configure the Autopilot *job completion criteria*, which is really defining how long a job is allowed to run and how many candidates a job is allowed to produce. Some key configuration items you're able to define here include *max_runtime_per_training_job_in_seconds*, which allows you to set a maximum time in seconds that a training job is allowed to run. Second, *max_candidates*, which indicates the maximum number of times you want to allow a training job to run, and then finally, *max AutoML job runtime* in seconds, which indicates the maximum time in seconds that an AutoML job is allowed to run. Once you've configured the Autopilot job completion criteria, you want to also define your *input and output data*. Here you'll specify your input data. This is the location of your S3 bucket that contains your dataset. You'll also need to specify your *target attribute*. Again, this is the attribute that you're trying to predict, which is sentiment in this use case. You also need to specify an S3 bucket that you want to use for your output artifacts, which I'll talk a little bit more about in the next video. But this would include items such as your generated notebooks, as well as your model artifacts.



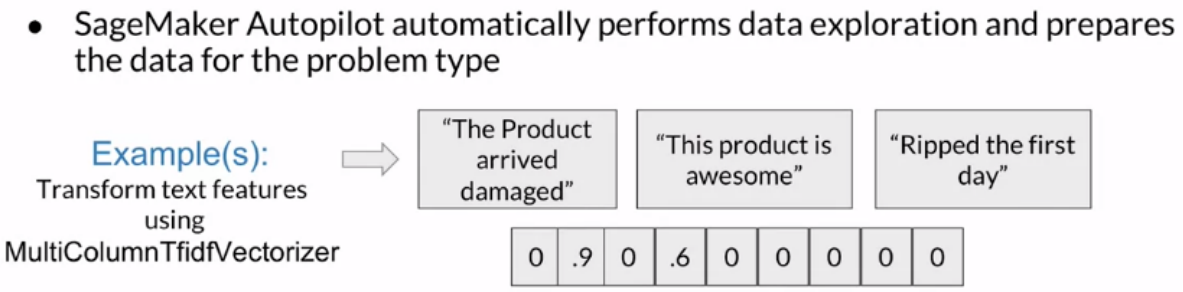
An **optional configuration** that you don't see here is *generate candidate definitions only*, which indicates the capability that I discussed earlier, where you can direct Autopilot to only generate the data exploration and candidate notebooks so that you can review or modify them instead of running them in complete autopilot mode. Once you've configured your job, you now want to launch your Autopilot job.

To launch your Autopilot job, you'll specify a name for your job and refer to the configurations that I just walked through. You'll also use a role that is defined in AWS Identity and Access Management, or IAM. The rule must be set up to allow permissions required for the job to issue the API calls that are needed to complete your Autopilot tasks. I just walked you through the steps to programmatically configure and launch your Autopilot job. The same steps can also be performed in more of a click-through experience directly inside Amazon SageMaker Studio as well. Once your job is submitted, you can then monitor the progress of your job either through *DescribeAutoMLJob API* call, or through your studio environment.



API → *DescribeAutoMLJob*

Based on the product review data, including the target label for sentiment that you provided on input, Autopilot is going to automatically generate the **feature engineering** code required to take that data and transform it into the format expected by the algorithm selected by Autopilot. The product review data contains text and there's a high-variance in that text. Autopilot is going to convert those text or categorical values into numeric values that the selected algorithm can understand. There are several different ways to perform feature engineering and transformations for text data. Autopilot uses the SageMaker Python module, which is an extension built on top of scikit-learn. The module is available on GitHub, but it contains several submodules for various transformations that are used by Autopilot for the automatic generation of feature engineering code.

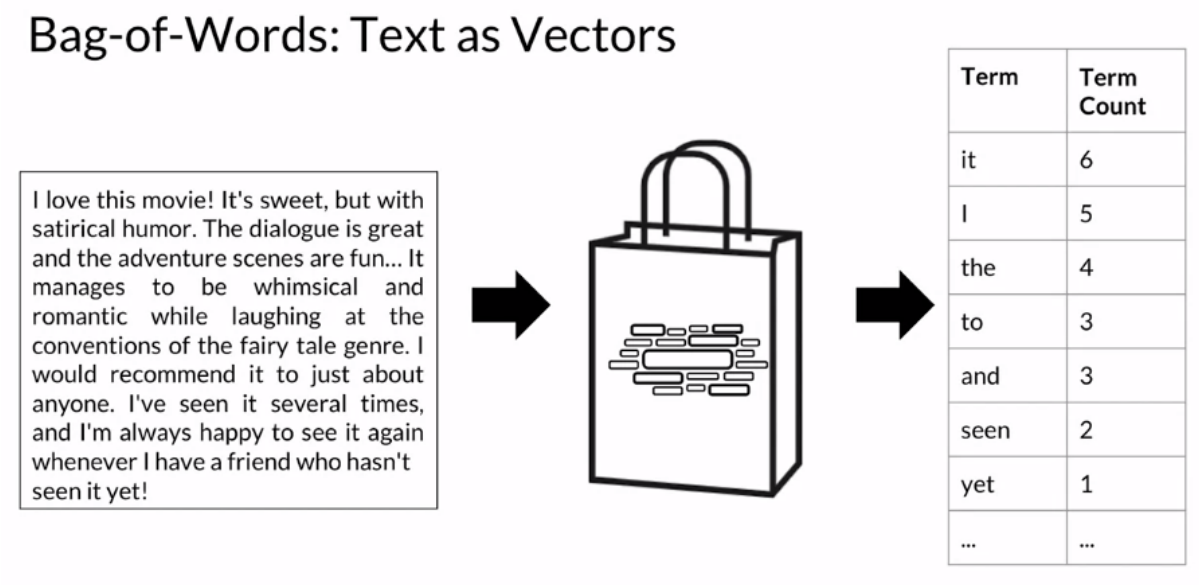


In this example, you'll see a submodule that is focused on feature extraction of text data called *MultiColumnTfidfVectorizer*. MultiColumnTfidfVectorizer converts collections abroad documents to a matrix of term frequency, inverse document frequency, or Tfidf features. At a high level with

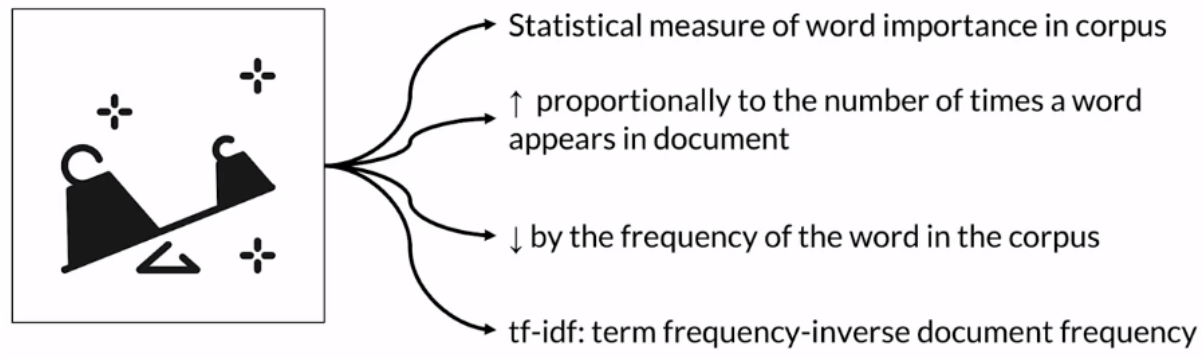
Tfidf, you're evaluating *how relevant a word is to a document or a collection of documents*. This is done by calculating how often the words appear and the inverse document frequency of those words across a set of documents. With the MultiColumnTfidfVectorizer, it converts collections of raw documents to a matrix of Tfidf features.

- SageMaker Autopilot will automatically tune MultiColumnTfidfVectorizer parameters

Let's dive a little bit deeper here so that you understand what's happening in the background. For processing texts with machine learning tools, you need to transform the document into a vector of features. A typical approach is to put all text in a bag and then sample from it without replacement. This way you can tally how many times a given word appears and produce a vector of counts. This approach is called **Bag-of-Words**.



Once you produce a vector using bag-of-words, you want to quantify how important a word is to a document relative to others in the corpus. What you're aiming for here is a concise and reliable statistical description of a word's weight. This weight has to increase proportionately to the number of times that it happens in a given text, but it should decrease if the word is very frequent in that corpus because it will last specificity. The Tfidf is a concise measure that will achieve exactly that.



t	term
d	document
D	corpus

Let's do a divide and conquer approach and start with the formula for **term frequency**, or TF. T refers to a *term* or a word, lowercase d to a **document**, and uppercase D to the **corpus**.

The TF is the *relative frequency of a word in a document against all other words in that document*. Because every document is different in length, it is possible that a term would appear many more times in long documents than in shorter ones. More concisely, it's the ratio of the word frequency in a document over the same of the frequencies of all words in that specific document.

$$\mathrm{tf}(t,d) = \frac{f_{t,d}}{\sum_{t' \in d} f_{t',d}}$$

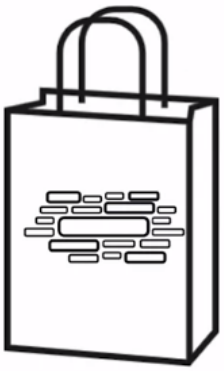
$$\mathrm{idf}(t,D) = \log\left(\frac{|D|}{|\{d \in D : t \in d\}|}\right)$$

Inverse document frequency, or IDF, measures *how important a term is by looking at the whole corpus*. The numerator is just the total number of documents in a corpus. The denominator tallies up all the instances of a term in the corpus. As you will see, very common words like is, if, or the, may appear many times, but they have very little importance. Thus, you need to scale down the weight of frequent terms while you scale up the weight of rare ones. This is the whole point of IDF which puts this quotient on a log scale. The only caveat with this formulation is that it might lead to division by zero.

Let's do an example. In a document containing 200 words where the word food appears ten times. The term frequency or the TF for food is then 10 divided by 200, which equals 0.05. If the corpus has 1 million documents and the word food appears in 1000 of these documents, the inverse document frequency, or IDF, is log 1 million divided by 100, or 3. The Tfidf weight is the product of these quantities, 0.05 times 3, which equals 0.15.

$$\text{tf-idf}(t,d,D) = \mathrm{tf}(t,d) \cdot \mathrm{idf}(t,D)$$

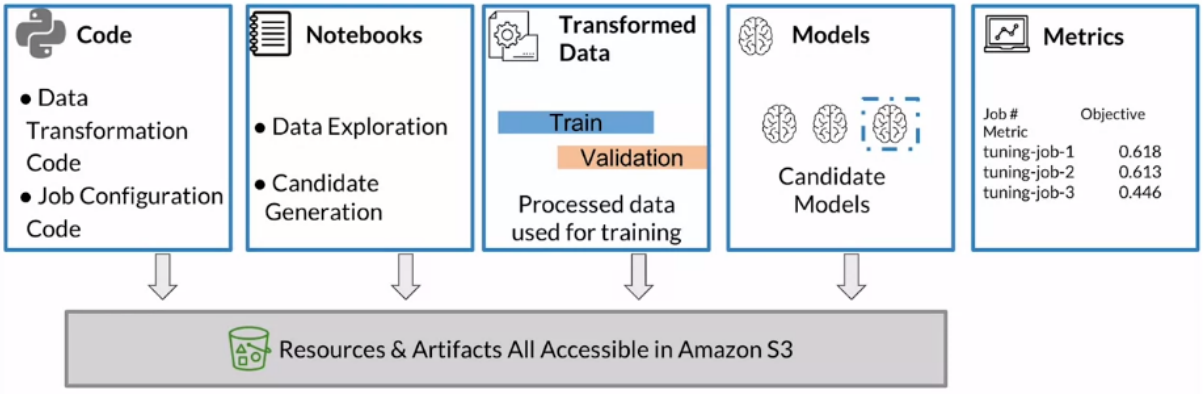
I love this movie! It's sweet, but with satirical humor. The dialogue is great and the adventure scenes are fun... It manages to be whimsical and romantic while laughing at the conventions of the fairy tale genre. I would recommend it to just about anyone. I've seen it several times, and I'm always happy to see it again whenever I have a friend who hasn't seen it yet!



Term	TF / IDF
it	0.06
I	0.05
the	0.01
to	0.03
and	0.03
seen	0.04
yet	0.01
...	...

Amazon SageMaker Autopilot: evaluating output

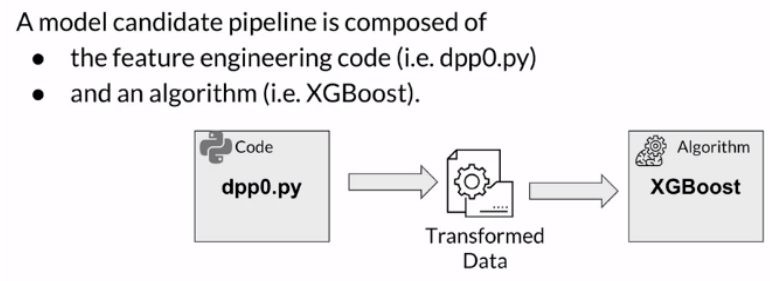
After your autopilot job has completed, there are several resources, and artifacts that are automatically generated.



First, autopilot generates your **data transformation code**. It will also generate your **configuration code** for setting up your data transformation, and training jobs as well. Autopilot also generates **data exploration**, and **candidate generation notebooks**. These notebooks provide visibility into the data exploration activities that autopilot performs to analyze your data. They also provide the steps for each candidate model, which includes the **data preprocessors**, the **algorithm**, and the **algorithm hyper parameter settings** selected by autopilot. These artifacts can be used to provide visibility into each candidate. And, can also be used to perform any additional refinements of your model as well. As a result of the data transformation code, autopilot is going to do those transformations. And produce the training, and validation data sets that will be used for training, and evaluation of your model.

All of these resources and artifacts are stored, and accessible in the S3 bucket that you specified when configuring your autopilot job. Autopilot will run through a number of experiments, training a number of models with different combinations of data transformation code, algorithm and hyper parameter configurations. The goal of all of these experiments, automated through autopilot, is to ultimately identify the best performing candidate model. Trained model artifacts are also stored in S3.

Finally, autopilot produces a **leaderboard with metrics** correlating to each candidate, to easily identify which candidate is performing the best. The notebooks are accessible in S3, but you can also access the notebooks from the studio console as well. So, bringing it all together, autopilot generates multiple model candidate pipelines. And, when I refer to a model candidate pipeline in the context of autopilot, what I'm referring to is a pipeline created automatically by autopilot. That combines the feature engineering code, the algorithm and the algorithm hyper parameter ranges into model candidate pipelines.



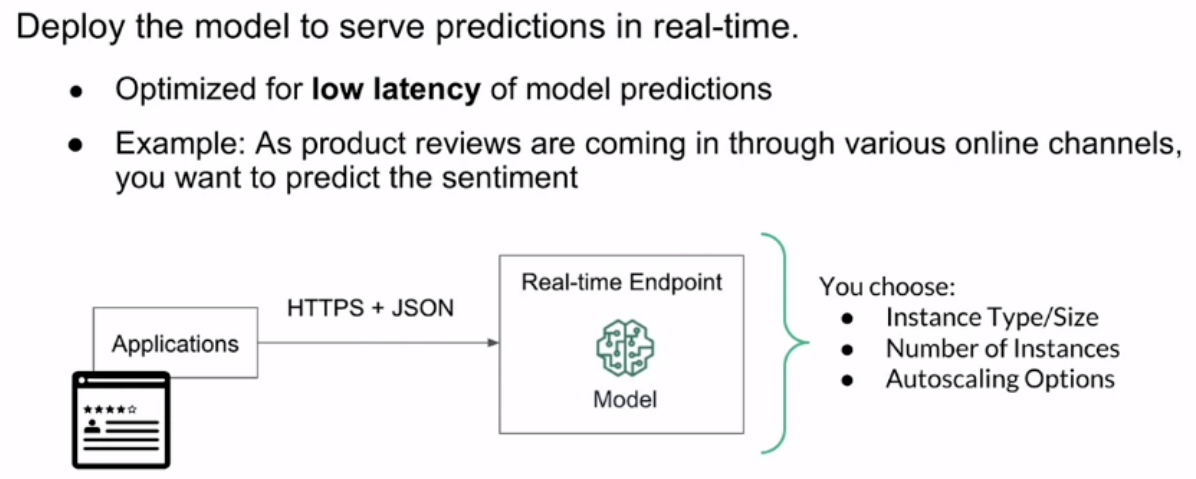
Let's take a look at those model candidate pipelines, and see what they look like. For each of the pipelines that are generated, multiple hyper parameter tuning jobs will be done to find the algorithm hyper parameters

that perform best within that candidate pipeline. Remember, autopilot automatically generates the tuning job configurations that define the hyper parameter ranges that will then be used by those tuning jobs.

Model hosting

Once you use autopilot to find that best performing model, how can you take that model and then deploy it for consumption?. In this section I'll specifically cover deploying your model for use with the real time use case. But keep in mind that **Sagemaker supports both batch and real time deployments**.

In this case, you need the model to be persistently available to be able to serve real time requests for prediction. So a common use case here would be where product reviews are coming in from various online channels, whether it's through a website, social media or email, but you want to be able to predict sentiment in real time. By doing so, you can quickly change your organization's direction with actions such as responding to negative reviews by then automating back in triggers to engage a customer support engineer or provide visibility into potential product issues where a product should be removed from the catalog in a timely manner. Serving your predictions in real time requires a model serving stack that not only has your trained model but also a hosting stack to be able to serve those predictions. This typically involves some type of a proxy, a web server that can interact with your loaded serving code and your trained model. Your model can then be consumed by client applications through real time, invoke employment API requests with Sagemaker model hosting, you simply choose the instance type as well as the count combined with the doctor container image that you want to use for inference and then Sagemaker takes care of creating the endpoint. In deploying that model to the endpoint, you can also configure automatic scaling to scale your endpoint to meet the demands of your workload by taking advantage of on demand capacity when it's needed.



So once you've compared the results across your candidate pipelines, you can then deploy the best performing model. But there's a few things to keep in mind.

A **PipelineModel** actually has multiple containers that are needed for inference, including first, a **data transformation container**. This container will perform the same transformations on your data set that were used for training so that you can ensure your prediction request data is in the correct format for inference. Second, an **algorithm container**, this is the container that contains

the train model artifact that was selected as the best performing model based on your hyper parameter tuning jobs and finally, an **inverse label transformer container**. This container is used to post-process your prediction into a readable value by your application that consumes the output. So let's take a look at what the PipelineModel looks like when you deploy the candidate pipeline to a Sagemaker hosted endpoint.

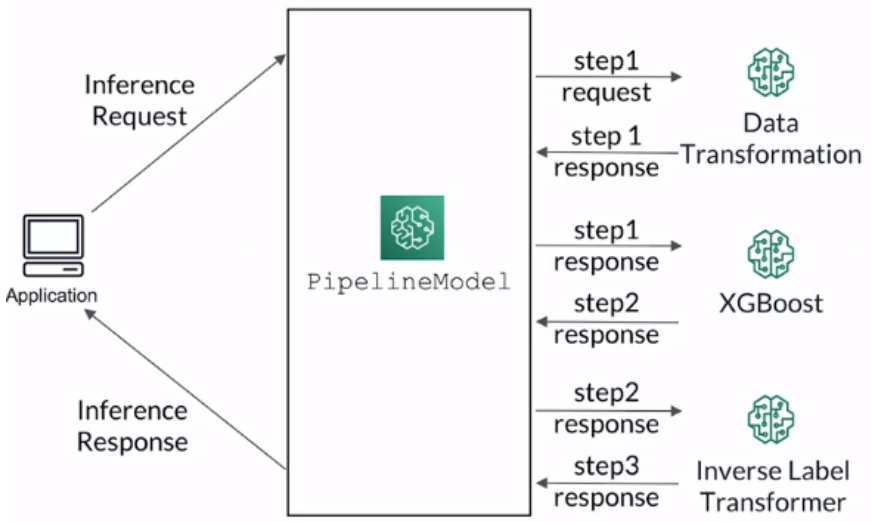
```
pipeline_model.deploy(initial_instance_count=1,
                      instance_type='ml.m5.2xlarge',
                      endpoint_name=pipeline_model.name,
                      wait=True)
```

Congratulations! Now you could visit the sagemaker endpoint console page to find the deployed endpoint (it'll take a few minutes to be in service).

The `PipelineModel` has multiple containers of the following:

- **Data Transformation Container:** a container built from the model we selected and trained during the data transformer sections
- **Algorithm Container:** a container built from the trained model we selected above from the best HPO training job.
- **Inverse Label Transformer Container:** a container that converts numerical intermediate prediction value back to non-numerical label value.

In the first picture that I showed it was a single model behind an endpoint. In that case you need to typically perform your data pre-processing and post processing for prediction as a secondary process or from your consuming application. When you choose to deploy a candidate pipeline generated by autopilot, it gets deployed using a Sagemaker hosting feature called **inference pipeline**. With inference pipeline, you're able to host your *data transformation model*, your *product classification model* and your *inverse label transformer* behind the same endpoint. This allows you to keep your training and inference code in sync and allows you to abstract those transformations away from your consuming applications. When an inference request comes in, the request is sent to the first data transformation model and then the remaining models are sequentially run with that final model. In this case, the inverse label transformer sends the final influence result back to your client application.



In this section, I briefly covered model hosting on Sagemaker, specifically focusing on real time persistent endpoints and the ability for you to deploy the candidate pipeline model generated by autopilot with a simple configuration. This allows you to host your model using Sagemaker managed endpoints, managed endpoints mean you don't have to manage the underlying infrastructure that's hosting your model and you can focus on machine learning.

Additional reading material

If you wish to dive more deeply into the topics covered this week, feel free to check out these optional references. (You won't have to read these to complete this week's practice quizzes.)

- Amazon SageMaker Autopilot

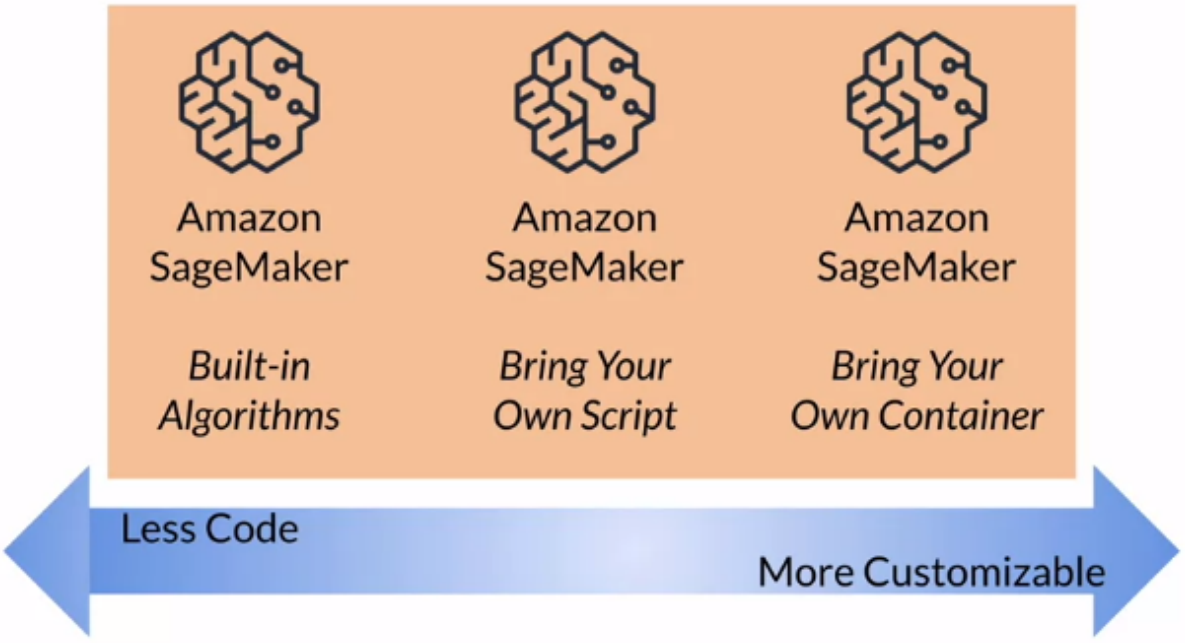
Week 4: Built-in algorithms

Built in algorithms

Introduction

This week you will learn how to quickly train and deploy models using built-in algorithms. The big advantage of building algorithms is that they require no coding. To start running experiments, you can just provide the algorithms with your input data set, any model hyper parameters and define the compute resources such as the number and type of compute instances to use. Another benefit of building algorithms is that many of them support GPUs and parallelization across multiple, compute instances without any additional configuration. This means if you are working with a large data set and you want to distribute your model training, you don't have to worry about writing that code either. You simply increase the number of compute instances to run the model training on and the building algorithm implementation will take care of the rest. Knowing and leveraging these shortcuts is essential as you evolve your skills in data science and machine learning or ml you want to make sure to focus your time where it's used best. Solving the more complex problems that require domain knowledge and expertise rather than spending half a day writing distributed model training code especially for well known and simple machine learning tasks.

When to choose built-in algorithms vs custom code?



Use cases and algorithms

Let me give you an overview of use cases and highlight which algorithms are suitable for solving the problem.

Let's start with classification and regression problems.

Classification covers both binary and *multi-class classification*. For example, a typical *binary classification* use case would be you want to predict whether an email is spam or not spam. Your input data in this case would be tabular with a label, whether the training data sample is indeed spam or not spam. For such classification problems, you could choose between the *XGBoost* or the *K-Nearest Neighbors* algorithm. XGBoost, which is short for extreme gradient boosting, is a popular and efficient open-source implementation of the gradient boosted trees algorithm. The XGBoost algorithm performs really well because of its robust handling of a variety of data types, relationships, and distributions, and the variety of hyperparameters that you can fine tune. You can use XGBoost also for regression and ranking problems. The K-Nearest Neighbors or k-NN algorithm is an index-based algorithm. For classification problems, the algorithm queries the K points that are closest to the sample point and returns the most frequently used label off their class as the predicted label.

Let's move on to **regression** problems. Here, the goal is to predict a numeric or continuous value, such as estimating the belly of a house, given input features such as location, number of rooms, property tax rates, and other data. For regression tasks, you can choose between the *Linear Learner* or the *XGBoost* algorithm. The Linear Learner algorithm extends up on typical linear models by actually training many models in parallel, each with slightly different types of parameters and then returns the one with the best fit. A third problem type is time-series forecasting. Imagine you want to predict sales on a new product, given previous sales data. For time-series forecasting, you can leverage the *DeepAR Forecasting* algorithm. The DeepAR Forecasting algorithm is a supervised learning algorithm for forecasting scalar, meaning one-dimensional time series, using recurrent neural networks or RNNs.

Example problems and use cases	Problem types	Input format	Built-in algorithms
Predict if an item belongs to a category: an email spam filter	Binary/multi-class classification	Tabular	XGBoost, K-Nearest Neighbors (k-NN)
Predict a numeric/continuous value: estimate the value of a house	Regression	Tabular	Linear Learner, XGBoost
Predict sales on a new product based on previous sales data	Time-series forecasting	Tabular	DeepAR Forecasting

Let's move on to **clustering** tasks. Clustering is an example of unsupervised learning. Here, the data is not labeled. The clustering algorithm tries to find patterns in the data and starts grouping data points into those distinct clusters. One prominent problem type which is addressed by clustering, is dimension reduction in the feature engineering step. Assume you want to predict the mileage of a car. In this use case, the color of the car should not be any relevant input feature, and then can be dropped. You can use the *principal component analysis* or PCA

algorithm for this task. PCA is an unsupervised machine learning algorithm that attempts to reduce the dimensionality or the number of features within a dataset while still retaining as much information as possible. Another popular problem type for clustering is anomaly detection. For example, anomalies could manifest as unexpected spikes in time-series data, such as an unexpected high or low number of requested chairs, maybe due to weather conditions or a large event in town. Anomalies can be detected using the *Random Cut Forest* algorithm or RCF. RCF is an unsupervised algorithm for detecting anomalous data points within a dataset. RCF associates an anomaly score with each data point. Low score values indicate that the data point is considered normal. However, high values indicate the presence of anomalies in the data. One of the most popular algorithms for clustering or grouping of data is *K-Means*. K-Means could be used, for example, if you want to group customers in high, medium, or low spending groups based on transaction data. K-Means is an algorithm that trains a model that groups similar objects together. For example, suppose you want to create a model to recognize handwritten digits, and you choose the MNIST dataset for training. You can think of the MNIST database or Modified National Institute of Standards and Technology database as the Hello World dataset of Compute Vision. The dataset provides thousands of images of handwritten digits from 0-9. In this example, you might choose to create 10 clusters, one for each number. As part of model training, the k-means algorithm would then group the input images into one of those 10 clusters. Another clustering problem type is topic modeling. Here, you are working with text data specifically as input. For example, say you want to organize a set of documents into topics based on words and phrases used in those documents. Two built-in algorithms could help you implement this, *Latent Dirichlet Allocation*, which is also known as LDA, or a *Neural Topic Model*, which is also known as NTM. Although you can use both the NTM and LDA algorithms for topic modeling, they are distinct algorithms and can be expected to produce different results based on the same input data. LDA is a generative probability model, which means it attempts to provide a model for the distribution of outputs and inputs based on latent variables. In statistics, latent variables are variables that are not directly observed, but are inferred from other variables in the training dataset. This is opposed to the discriminative models, which attempt to learn how inputs map to the outputs. NTM uses a deep learning model rather than a pure statistical model. I would recommend you try both LDA and NTM and explore which one works better on your specific data.

Example problems and use cases	Problem types	Input format	Built-in algorithms
Drop weak features such as the color of a car when predicting its mileage.	Feature engineering: reduce dimensions	Tabular	Principal Component Analysis (PCA)
Detect abnormal behavior	Anomaly detection	Tabular	Random Cut Forest (RCF)
Group high/medium/low-spending customers from transaction histories	Clustering / grouping	Tabular	K-Means
Organize a set of documents into topics based on words and phrases	Topic modeling	Text	Latent Dirichlet Allocation (LDA), Neural Topic Model (NTM)

Let's have a look at popular **image processing** use cases. Content moderation refers to the ability to review user-generated content and decide whether the content is appropriate to display, or whether the content should be removed. This use case could be implemented in various ways and doesn't apply only to images. For the image use case, you can use the built-in *image classification* algorithm to classify the image into one of your defined output categories. The

built-in image classification algorithm can be run in two modes; full training or transfer learning. In full training, the network is initialized with random weights and trained on user data from scratch. In transfer learning mode, the network is initialized with pre-trained weights, and just the top fully connected layer is initialized with the random weights. Then the whole network is fine tuned with new data. In this mode, training can be achieved even with a smaller dataset. This is because the network is already trained and therefore can be used in cases without sufficient training data. Another use case working with image data is detecting people or objects and images. The *object detection* algorithm detects all instances of predefined objects within the images categorized as the object, and also adds a bounding box indicating the location and scale of the object in given images. The third use case describes, for example, how self-driving cars identify objects in their path, which is realized with *semantic segmentation*. Semantic segmentation is different from the image classification and object detection in that it classifies every pixel in an image. This leads to information about the shapes of the objects contained in the image. The segmentation output is represented as a grayscale image called a segmentation mask that has the same shape as the input image. Classifying each pixel is fundamental for understanding scenes, which is critical to an increasing number of Computer Vision applications, such as self-driving vehicles, but also medical imaging diagnostics, and robot sensing.

Example problems and use cases	Problem types	Input format	Built-in algorithms
Content moderation	Image classification	Image	Image Classification
Detect people and objects in an image	Object detection	Image	Object Detection
Self-driving cars identify objects in their path	Computer vision	Image	Semantic Segmentation

Let's come back to the field of **text analysis**. Typical use cases here are, for example, a translating text. Let's say you want to convert Spanish to English. A built-in algorithm that you could use for this purpose is *sequence to sequence*. The sequence to sequence algorithm is a supervised learning algorithm where the input is a sequence of tokens, for example, text or audio, and the output is generated as another sequence of tokens. You can use the same algorithm to summarize texts. Imagine you want to summarize a long research paper into just a short abstract. You can also use the sequence to sequence algorithm for speech-to-text conversations. Let's say you want to transcribe call center conversations. Finally, the use case you will be working on is text classification, classifying product reviews into sentiment classes. You can use the *blazing text* algorithm.

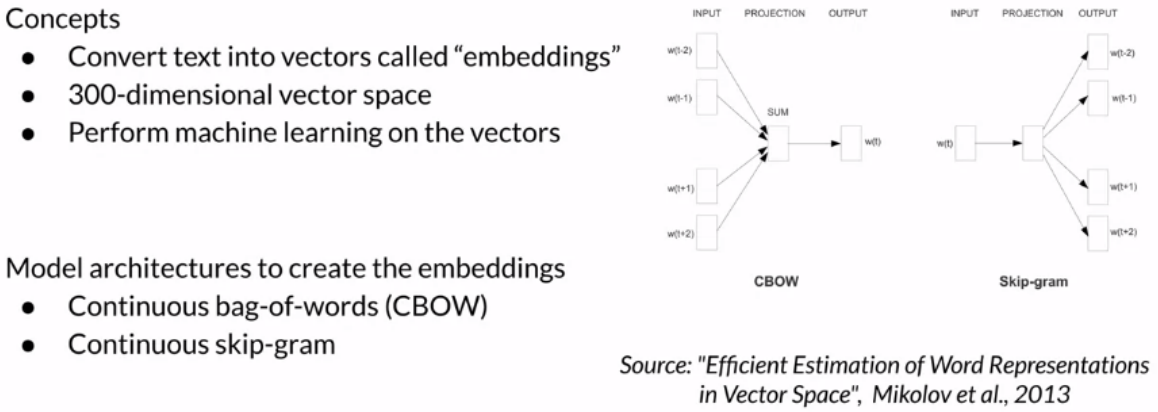
Example problems and use cases	Problem types	Input format	Built-in algorithms
Convert Spanish to English	Machine translation	Text	Sequence-to-Sequence
Summarize a research paper	Text summarization	Text	Sequence-to-Sequence
Transcribe call center conversations	Speech-to-text	Text	Sequence-to-Sequence
Classify reviews into categories	Text classification	Text	BlazingText

Text analysis

As a scientific field, text analysis or **natural language processing**, also known as NLP has been around for a long time. In fact, early work dates back to the 1950s, some even further in the last decade or so. However, there has been enormous progress in the field due to the evolution and the advancement of machine learning algorithms for text analysis. Here, I like to talk a little bit about that recent evolution and where we are now in the field of text analysis.

A very simple **bag of words** model was already introduced in the 1950s to count the currents of each word in a document.

I want to spend a few minutes walking you through some of the more recent advancements starting in 2013. In 2013, a research team led by Thomas Mikolov at Google introduced the **Word2Vec** algorithm. Word2Vec to back *converts text into vectors also called Embeddinds*. Each of those vectors consists of 300 values. Hence it represents a 300 dimensional vector space. You can then use those vector representations as inputs to your machine learning. Use cases for example, applying nearest neighbor classification or clustering algorithms. And Word2Vec is famous for the two different model architectures as you see here, which it implements to generate the word embeddings, continuous bag of words or CBOW, and continuous skip gram. The architectures are based upon shallow to layer neural networks. Another approach back in 2013, CBOW predicts the current word from a window of surrounding context words, continuous skip gram uses the current word to predict the surrounding window of context words. One challenge though with Word2Vec is that it tends to run into what's called out of vocabulary issues, because its vocabulary only contains three million words. The *vocabulary* is a set of known words that the model learned in the training phase. Out of vocabulary words are words that were not present in the text data said the model was initially trained on so if the word is not found in its vocabulary, the model architecture assigns a zero to that word which is basically discarding the word.

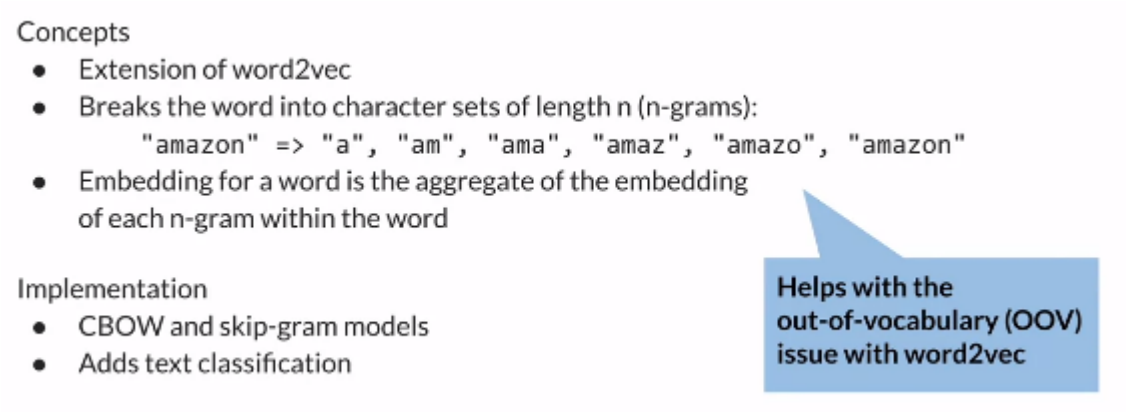


Source: "Efficient Estimation of Word Representations in Vector Space", Mikolov et al., 2013

In 2014, a research team led by Jeffrey Pennington at Stanford University introduced **GloVe** or global vectors, forward representations. GloVe novel approach In 2014 used the regression model to learn word representations through unsupervised learning.

In 2016, a research team led by Piotr Bojanowski at Facebook AI Research published their work on **FastText**. And let's have a closer look at FastText. FastText builds on Word2Vec but it treats each word as a set of sep-words called the character n-grams. And this helps with the out of

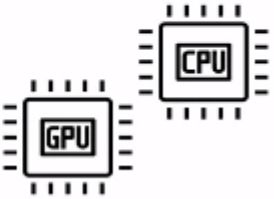
vocabulary issue that I mentioned for Word2Vec. Here are examples how FastText divides the word into smaller character sets. Now, even if the word Amazon is not in the vocabulary, chances are that the character said “am is”. The embedding that fast tech Lawrence for a word is the aggregate of the embeddings of each n-gram with the word. FastText uses the same CBOW and skip gram models but it adds support for text classification use cases. With a character n-gram representation of words, FastText increases the effect of vocabulary of Word2Vec beyond the three million words.



Another large milestone in the evolution of text analysis was the Introduction of the **Transformer** Architecture in 2017 in a paper called “Attention Is All You Need”. Published at the 2017 conference on neural information processing systems, Vaswani et al from Google brain and collaborators at Google research and the university of Toronto introduced a novel neural network architecture based on a self attention mechanism. The concept of attention had been studied before for different model architectures and generally refers to one model component capturing the correlation between inputs and outputs. In NLP terms, the attention would map each word from the model's output to the words in the input sequence, assigning them weights depending on their importance towards the predicted word. The self attention mechanism in this new transformer architecture focuses on capturing the relationships between all words in the input sequence and thereby significantly improving the accuracy of natural language understanding tasks such as machine translation. While the transformer architecture marked a very important milestone for NLP, other research teams kept evolving alternative architectures.

Also in 2017 Khare and Gupta from AWS introduced **BlazingText**. BlazingText provides highly optimized implementations of the Word2Vec and text classification algorithms. Blazing text scales and accelerates Word2Vec using multiple CPUs or GPUs for training. Similarly, the BlazingText implementation of the text classification algorithm extends FastText to use GPU acceleration with custom CUDA kernels. CUDA, or compute unified device architecture, is a parallel computing platform and programming model developed by NVIDIA and to give you an idea of the scope of the acceleration, using blazing text, you can train a model on more than a billion words in a couple of minutes using a multi core CPU or GPU. Bazing text creates character n-gram embeddings using the continuous bag of words and skip gram training architectures. BlazingText also allows you to save money by stopping your model training early. Let's say that the validation accuracy stops increasing, blazing text also optimizes the IO for datasets stored in Amazon simple storage service or Amazon S3. Later this week, you will use the blazingText algorithm to train the text classifier model.

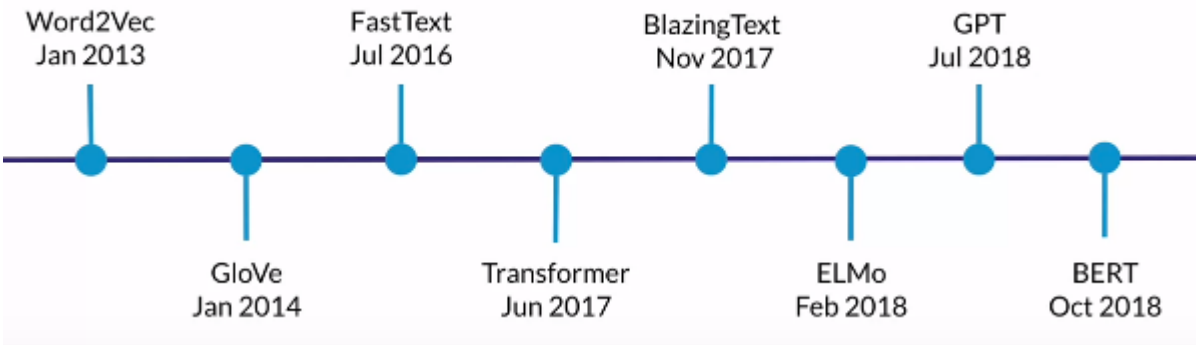
- Scales and accelerates Word2Vec using multiple CPUs or GPUs for training
- Extends FastText to use GPU acceleration with custom CUDA kernels
- Creates n-gram embeddings using CBOW and skip-gram
- Saves money by early-stopping a training job
    - when the validation accuracy stops increasing
- Optimized I/O for datasets stored in Amazon S3



In 2018, Mathew E. Peters at the Ellen Institute for artificial intelligence along with collaborators from the University of Washington published the **ELMO algorithm** which is short for embeddings from language models. In ELMO, word vectors are learned by a deep bidirectional language model. ELMO combines forward and backward language models and is thus able to better capture syntax and semantics across different linguistic contexts.

Later in 2018, a research team led by Alex Redford at Open AI released **GPT** in a paper called “Improving language understanding by generative pre-training”. GPT is based on the transformer architecture but performs two training steps. First, GPT learns a language model from a large unlabeled text corpus, and second GPT performs a supervised learning step with labeled data to learn specific NLP tasks such as text classification. GPT is only trained and hands predicts context from left to right, which is often referred to as unidirectional.

Shortly after GPT, a research team led by Jakov Devlin at Google AI Language published **BERT** or bidirectional encoder representations from transformers. BERT, in contrast to GPT, is truly bidirectional. In the unsupervised training step, BERT learns representations from unlabeled text, from left to right and right to left contexts jointly. This novel approach created interest in BERT across the industry and has led to many variations of BERT models, some of which are focused on specific languages such as French, German or Spanish. There are also BERT models that focus on a specific text domain such as scientific text and up to today, BERT is still among the most popular NLP models.



Train a text classifier

Are you ready to train a text classifier using a built-in algorithm? Let's see what you need to do. Just as a quick summary, you will apply the build-in algorithm to train a model that classifies product reviews into the three sentiment classes as positive, neutral, and negative.

Before you start training the model, you need to transform your training data to the input format BlazingText requires. Specifically append the sentiment classes 1, 0, and --1 to the label identifier as shown here and remove any column headers.

```
sentiment,review_body
1,"i simply love it"

0,"it's ok"

-1,"it arrived damaged. going to return"
```

```
__label__1 "i simply love it ."

__label__0 "it's ok ."

__label__-1 "it arrived damaged ."
```


```
def tokenize(review):
    return nltk.word_tokenize(review)
```

The BlazingText algorithm also requires the text to be tokenized into one sentence per line. You can use Python's Natural Language Toolkit or NLTK to perform exactly that step.

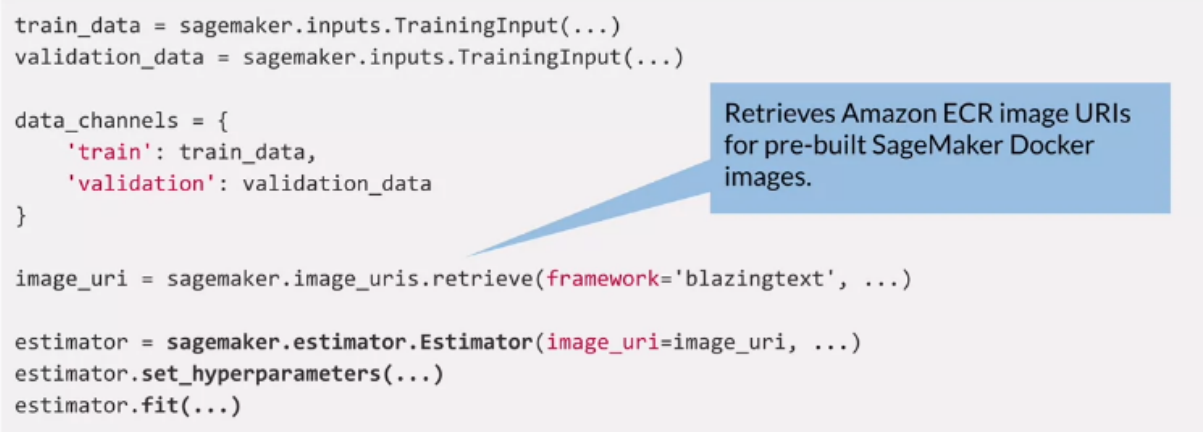
Finally, just upload the training data into an S3 bucket. You can tune SageMaker BlazingText text classification models with the hyper-parameters shown here.

Parameter Name	Recommended Ranges or Values	Description
epochs	[5-15]	Number of complete passes through the dataset
learning_rate	[0.005-0.01]	Step size for the numerical optimizer
min_count	[0-100]	Discard words that appear less than this number
vector_dim	[32-300]	Number of dimensions in vector space
word_ngrams	[1-3]	Number of words n-gram features to use
early_stopping	True or False	Stop training if validation accuracy stops improving
patience	[5-15]	Number of epochs before early stopping

Parameters are what a model learns, hyper-parameters control how the model learns those parameters. The tunable hyper-parameters for BlazingText include the **number of epochs** that

correspond to the number of complete passes through the dataset. The **learning rate** is the step size used by the numerical optimizer. **Min_count** lets you configure the removal of words that appear fewer times than what you specify here. The **vector_dim** is the number of dimensions in the vector space to use. The **word_ngrams** is the number of words in a word n-gram, and this is an important parameter and can have a significant impact on accuracy. **Early_stopping** defines whether to stop training if, for example, validation accuracy doesn't improve after the number of epochs specified in the patient's parameter.

Now, in preparation for training, you need to configure the algorithms data input channels to point to the uploaded training and validation files in S3. You can retrieve the correct SageMaker model training image for BlazingTexts via the *sagemaker.image_uris.retrieve* call. SageMaker offers pre-built docker images for building algorithms that contain all model code. The docker images are stored in a Docker Container Registry, the Amazon Elastic Container Registry or Amazon ECR. The image_uris.retrieve function will then retrieve the correct image from the Container Registry. You simply have to specify the framework as shown here. Then you pass that image_uris together with any additional settings to a SageMaker Estimator object and finally, you can start training the BlazingText text classifier by calling Estimator fit.

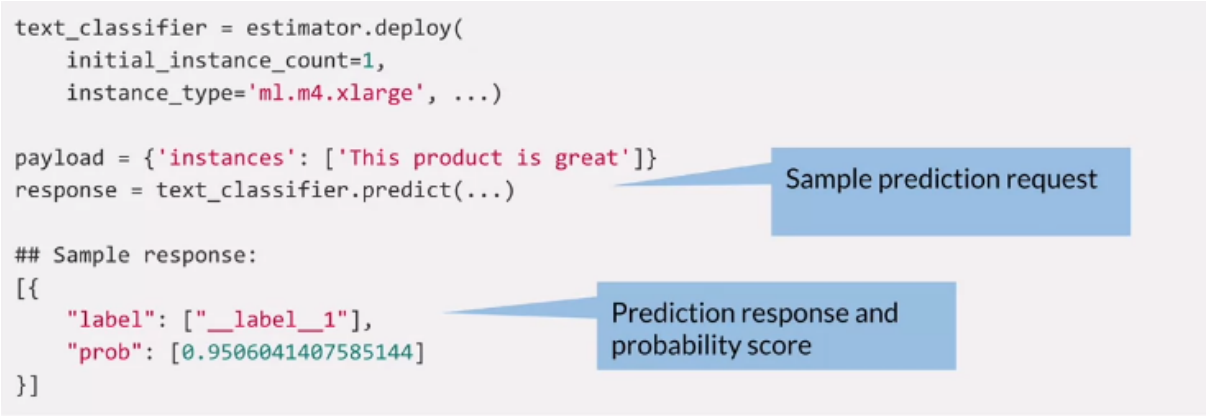


Are you curious to see the result? Here are a sample of model evaluation metrics showing the results, training accuracy and validation accuracy.

time	metric_name	value
00.0	train:accuracy	0.4865
10.0	train:accuracy	0.5220
20.0	validation:accuracy	0.5364

Deploy the text classifier

Let's discuss how you can deploy the text classifier on a rest endpoint and make predictions. With Sagemaker, you can now simply call *estimator.deploy*, to deploy our trained model on a rest endpoint. The endpoint is an Amazon elastic compute cloud, or Amazon EC2 compute instance managed by Sagemaker. You can choose the instance type and the number of instances to serve the model predictions.



Sagemaker will take care of creating those instances and deploying the train model. And by the way, you can increase the instance count here to anything larger than 1 to scale out your model hosting environment. Sagemaker also creates and manages a rest API that you can use to send prediction requests to, and receive prediction results from. You can see the deployed model endpoints in your AWS management console. From your notebook environment, you can also run prediction requests using the text classifier predict call shown here. The request format needs to be in Javascript object notation or JSON, with instances as the key, and the value set to a sample product review sentence. The model response is another JSON string with a predicted label class and the probability score of the prediction.

Additional reading material

If you wish to dive more deeply into the topics covered this week, feel free to check out these optional references. (You won't have to read these to complete this week's practice quizzes.)

- Word2Vec algorithm
- GloVe algorithm
- FastText algorithm
- Transformer architecture, "Attention Is All You Need"
- BlazingText algorithm
- ELMo algorithm
- GPT model architecture
- BERT model architecture
- Built-in algorithms
- Amazon SageMaker BlazingText