

LESSON 5: APPLICATIONS OF ML

Section 1: Lesson Overview

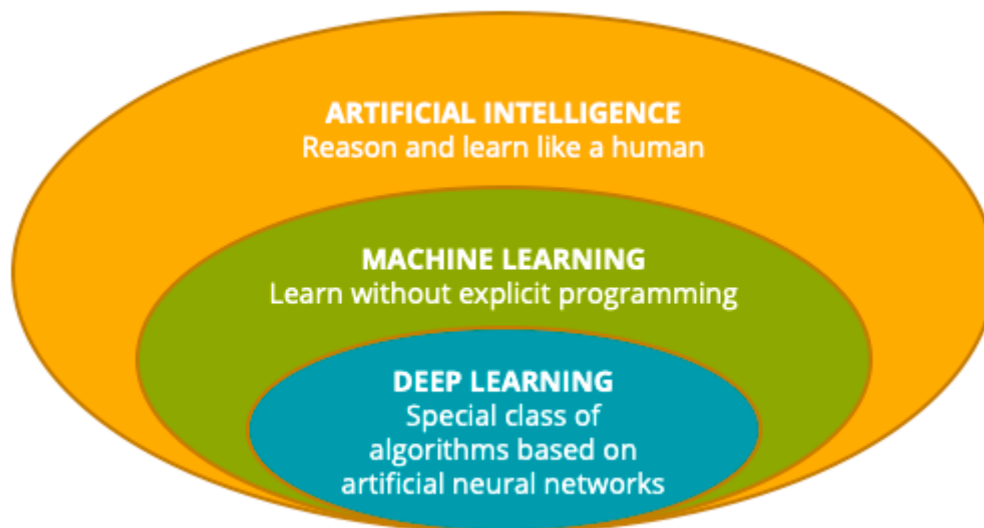
In this lesson, we will first look at **deep learning**. You'll learn about:

- The differences between classical machine learning and deep learning
- The benefits and applications of Deep Learning
- How to train your first neural network model
- Next, you will learn about some of the most important specialized cases of model training, including:
 - Similarity learning and the basic features of a recommendation engine
 - Text classification and the fundamentals of processing text in machine learning
 - Feature learning, an essential task in feature engineering
 - Anomaly detection
 - Time-series forecasting.

Along the way, you will get practice with several hands-on labs, in which you will train a simple neural network, train a recommendation engine, train a text classifier, and get some experience with forecasting

Section 2: Classical ML vs Deep Learning (DL)

Artificial Intelligence (AI) includes Machine Learning (ML), which includes Deep Learning (DL). We can visualize the relationship like this:



All deep learning algorithms are particular cases of machine learning algorithms—but it's not true that all machine learning algorithms are deep learning algorithms.

Example: solving a problem of binary classification (on the Iris dataset). In this example the binary classification is for Iris-setosa. This problem can be solved with multiple approaches:

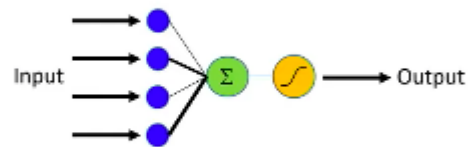
- we can use a **classical ML** approach based, for example, on **regression**, where we aim to train a model that finds the coefficients of a simple linear equation.
- we can also approach the problem by training a **DL model**, which is essentially a **directed graph** that will eventually will get an output that is the result of our binary classification.

Classical Machine Learning & Deep Learning Compared

ML trains
an equation

$$y=mx+b$$

Deep learning
trains a graph

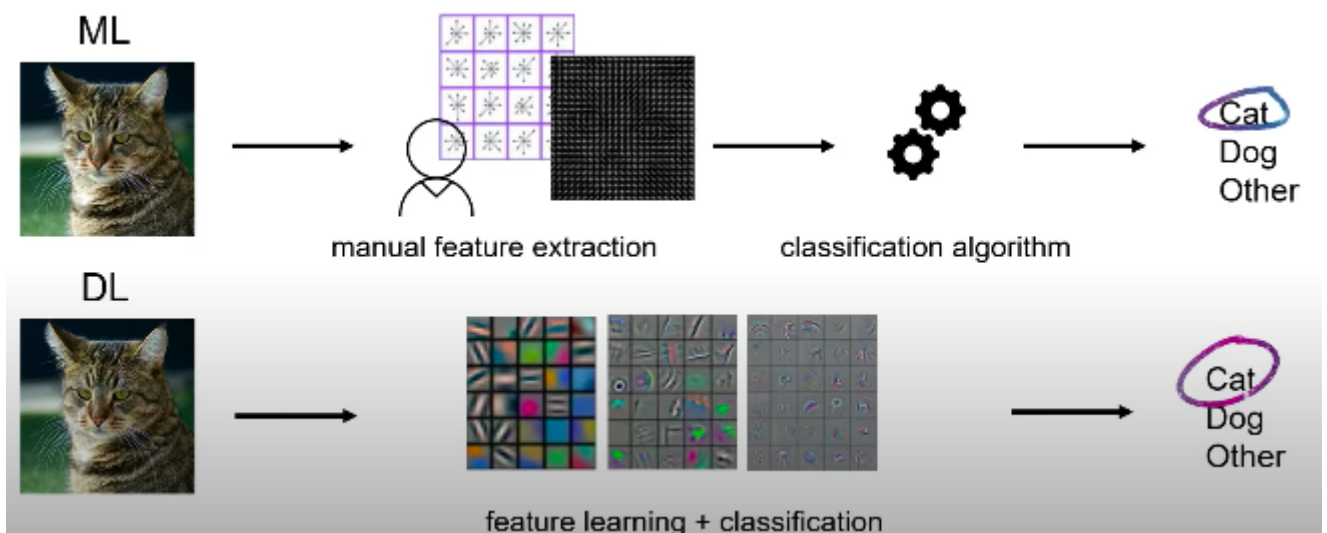


Ultimately both *can* predict a value of 0 (not iris-setosa) or 1 (is iris-setosa)

A More Detailed Comparison

One important characteristic that separates DL from classical ML is its inner capability of learning new features:

- With **classical ML**, you usually run a **manual process to extract new features (feature engineering)**. A selection of those features is then used to train a classification algorithm which finally provides the classification results:
- With **DL**, the **internal layers of the neural network have built-in capabilities that allow them to implicitly learn features** without having to explicitly encode those features. With every layer, different and potentially more complex features are learned, making DL a very good fit for problems like image recognition.

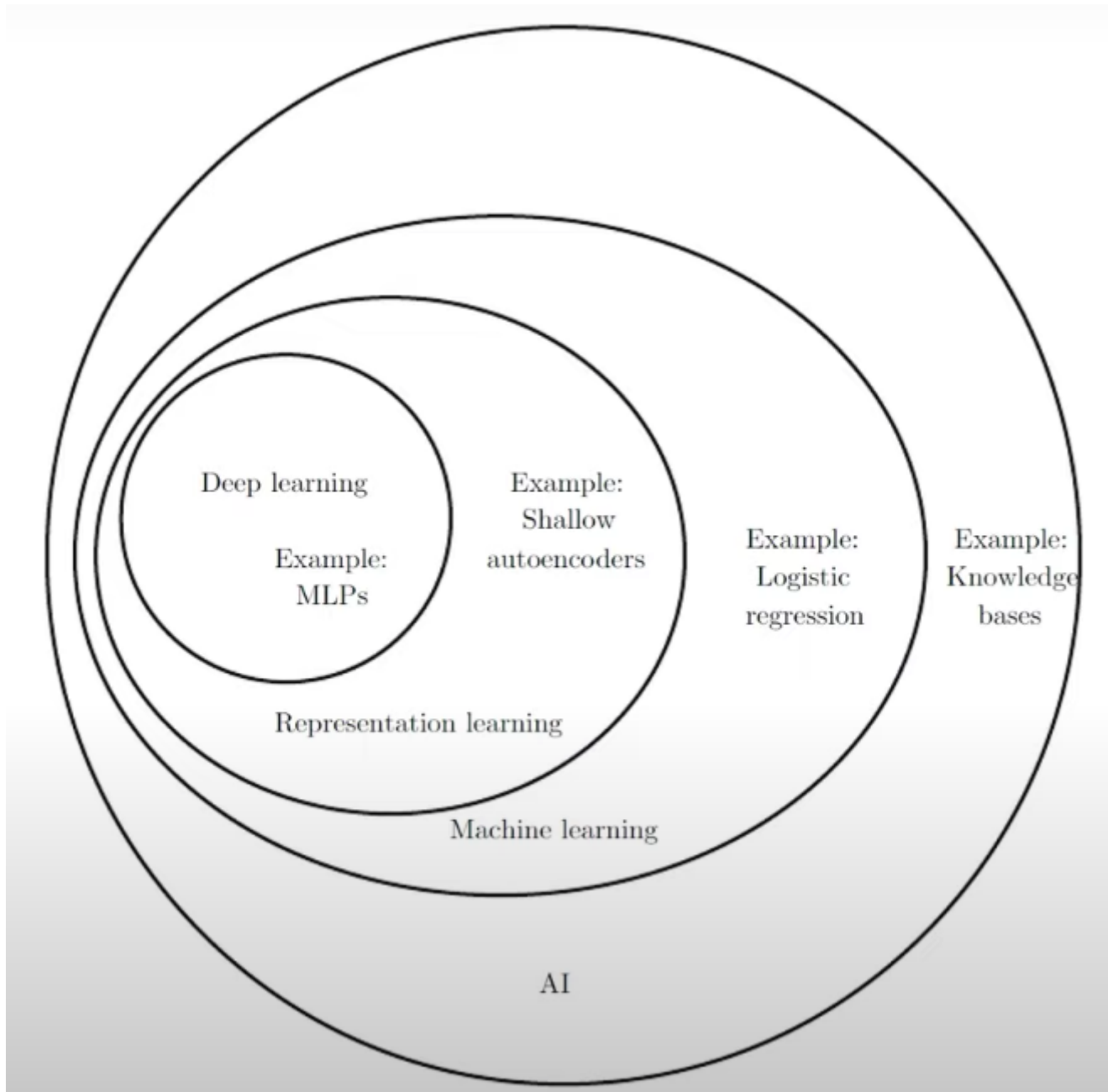


Different problems in ML can be solved using different methodologies. Given the **huge revolution in hardware resources** in the past years, DL has proved to be superior in a certain number of tasks to classical ML.

Section 3: What is DL?

- The first major field to gain the attention of both academia and practitioners was **artificial intelligence (AI)**. The initial goal of AI in the 1950s and 1960s was to create machines that act and think like humans. this ambition was hindered by a lack of computing resources and mathematical models.
- A few decades later the focus shifted to a more narrow focus set of problems, i.e. **learning without explicit programming**. This field was finally named **ML**. ML is a subset of AI that focusses on creating programs that are capable of learning without explicit instructions. Within the algorithms and approaches used by ML, in times several classes of those algorithms have emerged, giving different approaches to solve different classes of problems.

- Inspired from the **organisation of the human brain**, neural nets were initially intensely studied as part of ML, but due to lack of hardware resources, their area was pretty much abandoned. However, around the year 2000, fuelled by the **explosive development of proper hardware** (e.g. GPUs), the field of neural nets came back and **highly complex neural nets with large numbers of hidden layers and nodes** were feasible to implement (**DL**).



The Venn diagram shows how **DL** is a kind of **representation learning**, which is in turn a kind of **ML**, which is used for many but not all approaches to **AI**. Each section of the Venn diagram includes an example of an AI technology. Looking at each section of the diagram (from most external to most internal):

- General AI problem: Knowledge Bases.** The initial use of the term 'knowledge base' was to describe one of the 2 parts of a **knowledge-based system** (the other part being the **inference engine** that reasons about the facts that are represented in the knowledge base). The problem of implementing a knowledge base is outside the realm of ML.
- ML algorithms: logistic regression.**
- Representation learning: Shallow autoencoders.** These are **neural networks** but not deep neural networks. these were the first neural nets to be used and, because of lack of computational resources, **their number of hidden layers and nodes were limited (shallow learners)**

- **DL: MLP (Multi-Layer Perceptron).** Opposed to shallow learners, DL models are neural nets with a **large of hidden layers and nodes** in them.

The diagram is from [Deep Learning, by Ian Goodfellow, Yoshua Bengio, Aaron Courville](https://www.deeplearningbook.org/contents/mlp.html) (<https://www.deeplearningbook.org/contents/mlp.html>). The entire book is available for free through [their website](https://www.deeplearningbook.org/) (<https://www.deeplearningbook.org/>). In case you'd like to dig more into the comparisons we discussed here, the diagram and accompanying discussion of deep learning can be found in the introduction [here](https://www.deeplearningbook.org/contents/intro.html) (<https://www.deeplearningbook.org/contents/intro.html>).

A Word of Caution About "Neural"

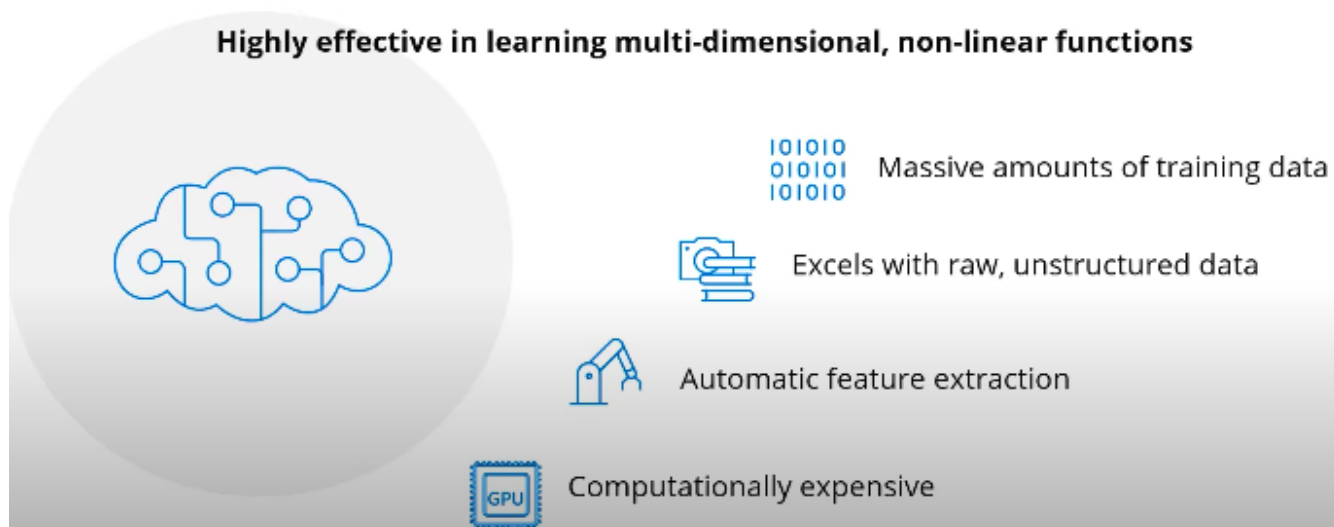
Artificial neural networks were inspired from the human brain. The similarity stops here.

- In the human brain there are about 86-88 billions neurons, with around a 1000 trillion of synapses between them. Information is transmitted using electrochemical signalling, but with a process that still needs to be fully understood.
- Neural networks are a highly simplified mathematical model, inspired from the organisation of the human brain but is not yet able to fully mimic it.

Section 4: Characteristics of Deep Learning

Characteristics of DL:

- Effectiveness in **learning multi-dimensional non-linear functions**. Training a model is nothing more than learning a function. The form of the function is very important in respect to the types of ML algorithms you can use and the outputs they produce.
- Capability of **handling massive amounts of training data** (e.g. hundreds of thousands of very high resolution images).
- It excels with **raw and unstructured data**. DL has an intrinsic capability of learning new features at every single layer of the neural network.
- Capability of **automatic feature extraction**. This happens automatically at every hidden layer of the neural net, without having to explicitly set-up the DL model to do that as it happens as the training part of the model.
- The training of DL models is **computationally expensive** (you will need a significant type and specialised computing resources, e.g. **GPUs or TPUs** = Tensor Processing Units, dedicated hardware that processes steps in the training of DL models). There is a **cost** associated with the training of a DL model, the more complex is your model (i.e. higher numbers of hidden layers and nodes) the higher is the cost, which can become a significant factor impacting on whether to train or re-train the DL model.



Section 6: Benefits & Applications of DL

Benefits of DL

- The **non-parametric** approach taken by neural nets allows them to learn **arbitrarily complex functions**. Most of the ML models that take a parametric approach, instead, are limited in terms of the form of the function they can learn.
- They can **learn complex patterns without explicitly seeing them**. Example of a DL model used in image recognition: you start with the input data which then gets processed by the hidden layers of the neural net. These hidden layers start to learn progressively increasingly complex features from the input data. The DL model can do this without having any prior knowledge of what the image represents
- **Effective across various structures/incarnations of data** (e.g. numbers, images, text). The area of applicability for DL is quite wide.
- It can work on **(very) large datasets**. Some of the other ML algorithms have limitations in regards of the size of the datasets they can process (either intrinsic limitations or limitations given by the fact that the process of training becomes very slow; this does not happen with DL!)
- DL models can be **distributed for parallel training**. You can train a neural network either on a **parallelisation approach on the same machine (e.g. using multiple cores)**, but you can also train it in an **MPP (Massively Parallel Processing) approach** where you distribute the workload on multiple individual machines.
- More recently a new class of DL has emerged, which can learn **time-related patterns**, the **RNNs (Recurrent Neural Networks)**. They are used for example to process speech data (it is a continuous stream of sound/frequencies that are time based).

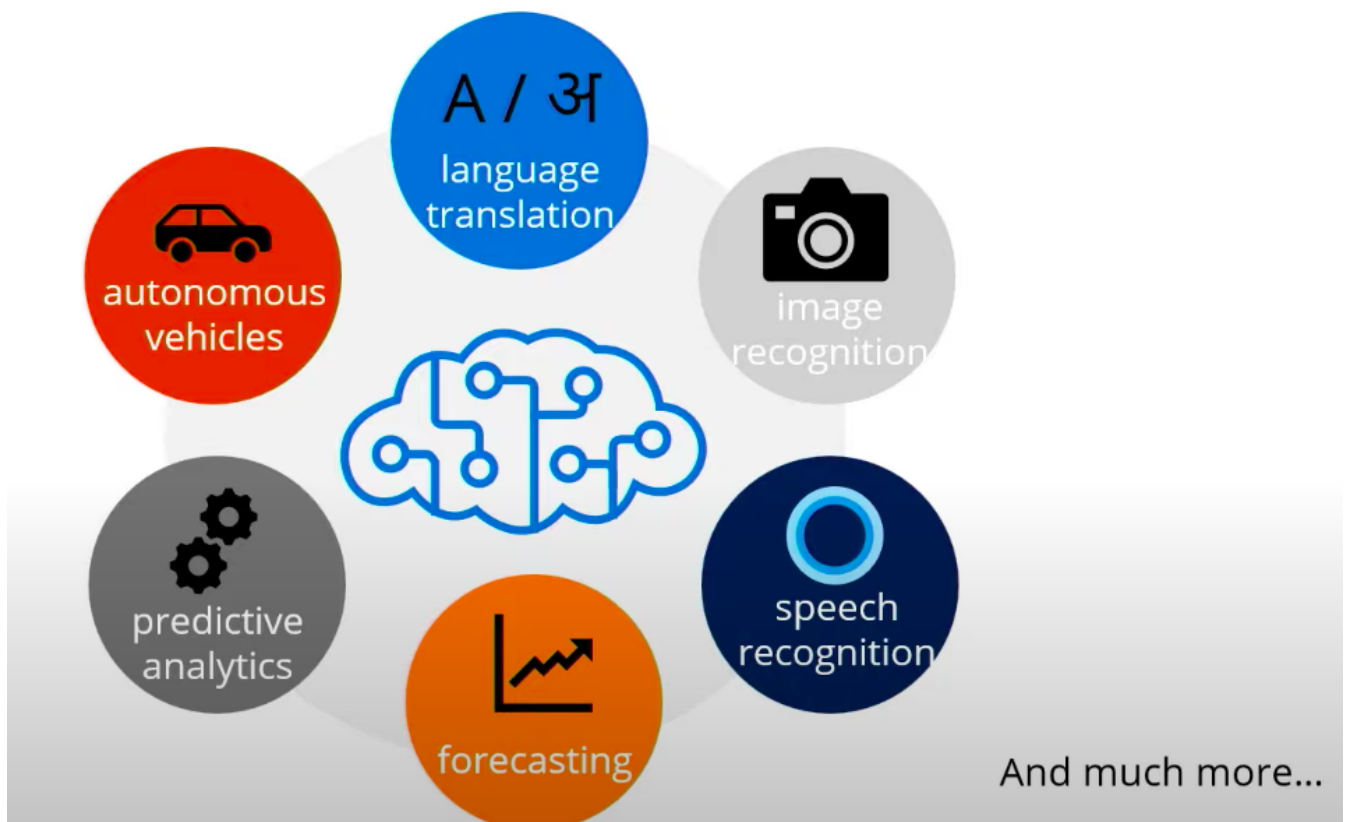
Some of the ML models that are based on DL approaches have been able to reach performance comparable to humans in specific activities (e.g. speech recognition or language translation).

Applications of DL

Fields where DL is applied:

- **Language translation**. This is one of the fields where DL models have proved to be equal or superior to human performance
- **Image recognition**. DL is able to automatically extracting increasingly complex features from images.
- **Speech recognition**. Speech input data is converted into a time-series based information (in numeric format). Learning the different patterns that occur over time is key to perform effective speech recognition.

- DL is increasingly used also in other types of **time-series based ML tasks**, such as **forecasting**
- **Predictive analytics**
- **Autonomous vehicles** (for example combining some of the other fields: image recognition, speech recognition and reinforcement learning)



Example of applying DL in **text analytics**; you can apply DL to understand:

- **Semantics**, i.e. detecting meaning
- **Sentiment**
- **Summarisation**: extraction of key phrases, entities and topics
- **Classification**: you can do binary or multi-class single/multi-label classification
- **Clustering**: e.g. identifying similarities between terms or documents, documents clustering on the basis of a metric of your choice
- **Search**



Section 9: Lab (Train a Simple Neural Net)

Although neural networks are widely known for use in deep learning and modeling complex problems such as image recognition, they are easily adapted to regression problems. Any class of statistical models can be termed a neural network if they use adaptive weights and can approximate non-linear functions of their inputs. Thus neural network regression is suited to problems where a more traditional regression model cannot fit a solution.

Neural network regression is a **supervised learning method**, and therefore requires a **tagged dataset**, which includes a label column. Because a regression model predicts a numerical value, the label column must be a numerical data type.

In this lab we will be using a subset of NYC Taxi & Limousine Commission - green taxi trip records available from [Azure Open Datasets \(https://azure.microsoft.com/en-us/services/open-datasets/\)](https://azure.microsoft.com/en-us/services/open-datasets/). The data is enriched with holiday and weather data. Based on the enriched dataset, we will configure the prebuilt Neural Network Regression module to create a regression model using a customizable neural network algorithm. We will train the model by providing the model and the NYC taxi dataset as an input to Train Model. The trained model can then be used to predict NYC taxi fares. We will do all of this from the Azure Machine Learning designer without writing a single line of code.

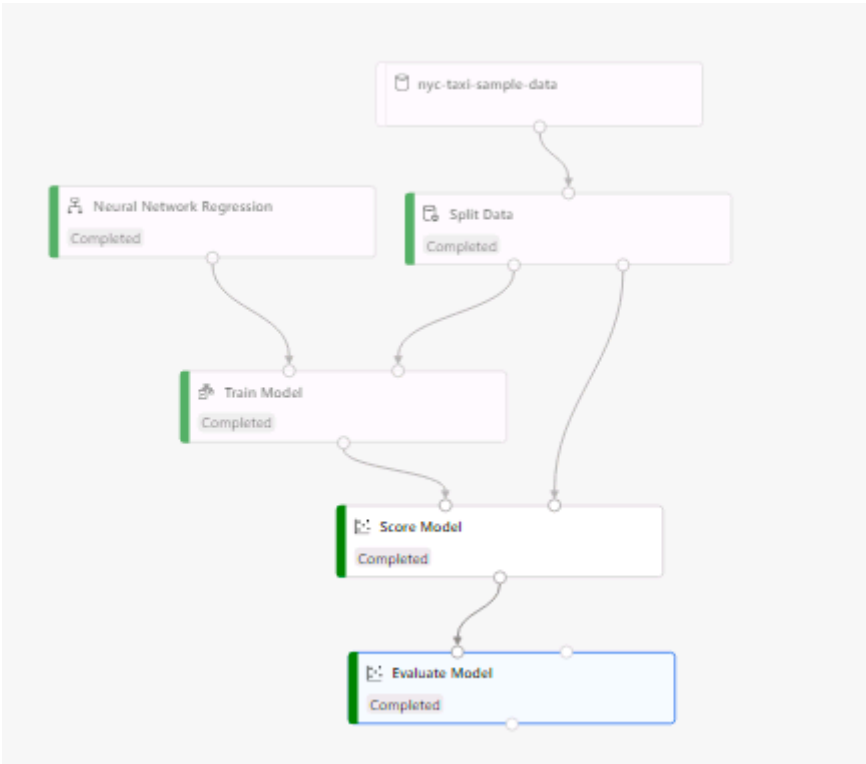
You can learn more about the training algorithm used in this lab by selecting [link \(https://docs.microsoft.com/en-us/azure/machine-learning/algorithm-module-reference/neural-network-regression\)](https://docs.microsoft.com/en-us/azure/machine-learning/algorithm-module-reference/neural-network-regression).

Neural Network Regression Parameters:

- Create trainer mode: Single Parameter
- Hidden layer specification: fully connected
- Number of hidden nodes: 100
- Learning rate: 0.01 (this is one of the core hyperparameters in every single neural net and controls the speed at which the internal weights of the neural net get updated as part of the learning process)
- Number of learning iterations: 100

Note: Because the number of nodes in the input layer is determined by the number of features in the training data, in a regression model there can be only one node in the output layer.

Pipeline in Azure Studio



Scores results:

totalAmount	Scored Labels
9.8	10.836934
8.3	8.776441
12.8	14.678989

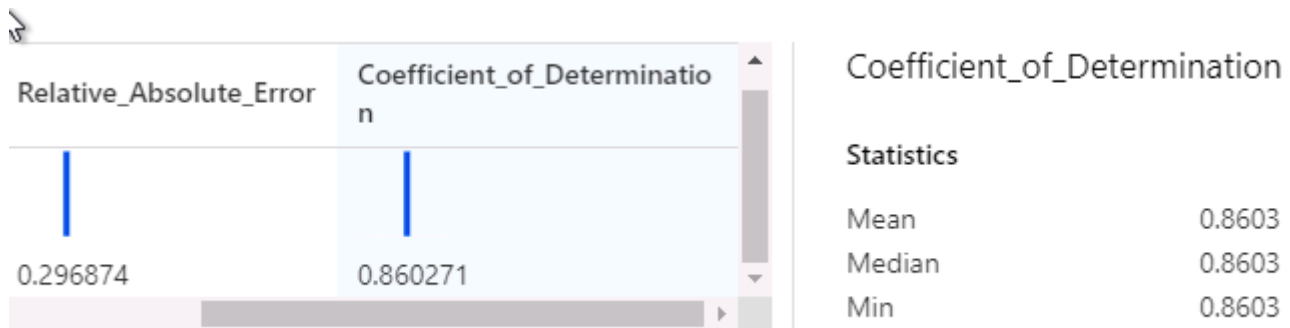
Evaluating the results:

Mean_Absolute_Error	Root_Mean_Squared_Error	Relative_Squared_Error
2.147296	3.847992	0.139729

Mean_Absolute_Error

Statistics

Mean	2.1473
Median	2.1473
Min	2.1473



(Note: for a good model the coefficient of determination R^2 should approach 1)

Section 9: Specialized Cases of Model Training

In this section, we'll have a look at some of the specialized cases of model training. But first, let's review the main types of machine learning approaches that we introduced back in the first lesson.

A Review of Approaches to Machine Learning

As you now know, there are three **main approaches to machine learning model training**:

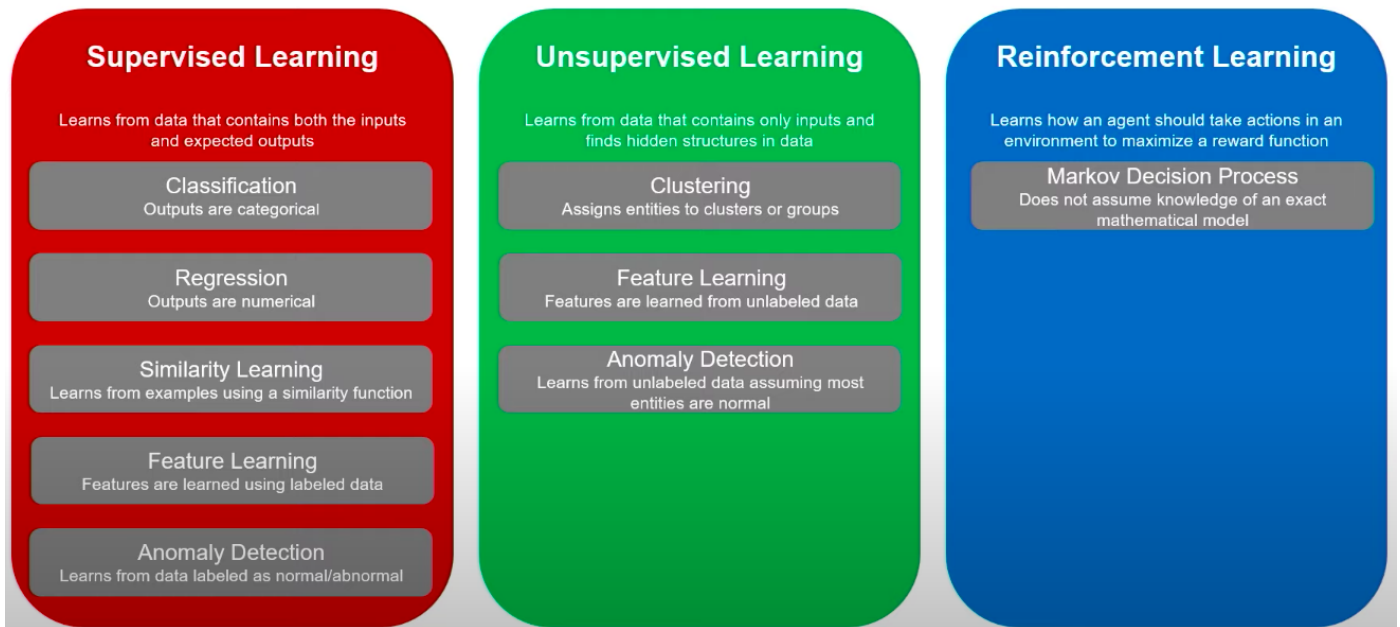
- **Supervised ML**: learns from data that contains both input and the **expected output**. The main class of problems that it solves are: **classification**, **regressions**, **similarity learning**, **feature learning** and **anomaly detection**
- **Unsupervised ML**: the data **only contains the inputs** (there is not teaching of how the output should look like) and the model tries to find **hidden structures** in it without prior information. The most common classes of problems solved are: **clustering**, **feature learning** and **anomaly detection**
- **Reinforcement learning**: it learns how an agent should take action in an environment (taking into account the outcome of their actions in that particular environment) in order to maximise the reward function. The most common class of problems it solves is: **Markov decision process** There are also other type of approaches with are combinations of the major ones (e.g. semi-supervised learning, a combination of supervised and unsupervised)

There are three **major classes of problems** that you encounter in mode training:

- **Classification**
- **Regression**
- **Clustering**

In order to choose the best algorithm for training you can either consider the ML approach that is best to be used or the class of problem you are trying to solve. Some problems can be approached with different types of ML (e.g. anomaly detection can be approached with either supervised or unsupervised ML).

Approaches to Machine Learning



Specialised Cases of Model Training

There are some variations of these generic classes of problems in ML that are in themselves important enough to be treated separately:

- **Similarity learning**: a **supervised approach** that learns the **similarity function**. Widely used in **recommender systems** for example on commercial websites.
- **Text classification**: a special case of a **supervised approach** to classify texts. This problem has found many applications in business cases.
- **Feature learning**: it can be both a **supervised/classification** and an **unsupervised/clustering** problem. It is very important in feature engineering.
- **Anomaly detection**: as well it can be both a **supervised/classification** and an **unsupervised/clustering** problem. It is special because it is usually a binary classification with a high imbalance between the 2 classes (anomaly cases are usually much smaller in numbers than the normal cases).
- **Forecasting**: a **supervised approach**. It is the problem of predicting a feature value in a time-series based scenario

Specialized case	Approach
Similarity Learning	Supervised
Text Classification	Supervised (classification)
Feature Learning	Supervised (classification) Unsupervised (clustering)
Anomaly Detection	Supervised (classification) Unsupervised (clustering)
Forecasting	Supervised

Section 11: Similarity Learning

- It is closely **related to both classification and regression**, however an important difference is that it uses a different type of objective function.
- It is most widely used in **recommendation systems** (with the aim to understand the behavioural patterns of a user in order to provide them recommendations on items, etc)
- Often used also in **solving verification problems** (e.g. speech verification, face verification and the like)

Similarity Learning as a Supervised Learning Approach

- As a **classification problem**: the **similarity function** maps pairs of entities to a **finite number of similarity levels** (ranging from 0/1 to any number of levels). Thus, the similarity function is not a continuous functions but a **discrete function** and its output is a certain level of similarity.
- As a **regression problem**: the **similarity function** maps pairs of entities to **numerical values**. A **variation** of this approach, where the supervision form is weakened from an exact measure to an **ordering measure** (i.e. **ranking similarity learning**). This approach is better fit for real-life large-scale problems (e.g. millions or tens of millions of website users, ten thousands of products, etc...), with predictions needed in **real time**). In this case, **relaxing** a little bit the output requirement (i.e. producing an **ordering measure** rather than an exact measure) allows to significantly increase the performance of the ML model.

Recommender Systems

One of the most common uses of similarity learning is in creating recommender systems.

- The main aim is to **recommend** one or more **items** to the **system users**.
- Examples of **items** are: movies, restaurant, book, song, ...
- A **user** might be a person, a group of persons or any other entity with item preferences.

Two **major approaches**:

- **Content based**: it makes use of **features** for both users (e.g. age, gender, region, ...) and items (author, manufacturer, ...)
- **Collaborative filtering**: it **only uses identifiers** for users and items, but not their properties.
- The recommender properties (**ratings**) are calculated from a **matrix** (a very large matrix of preferences between users and items)
- The **ratings** can be either **explicit** (i.e. recorded in a system, such as when you give N stars to a movie) or **implicit** (for example calculated taking into account history of purchases, history of browsing in a specific website, ...)

Section 12: Lab (Train a Simple Recommender)

The main aim of a **recommendation system** is to recommend one or more items to users of the system. Examples of an item to be recommended, might be a movie, restaurant, book, or song. In general, the user is an entity with item preferences such as a person, a group of persons, or any other type of entity you can imagine.

There are two **principal approaches** to recommender systems:

- The **content-based approach**, which makes use of **features** for both users and items. Users can be described by properties such as age or gender. Items can be described by properties such as the author or the manufacturer. Typical examples of content-based recommendation systems can be found on social matchmaking sites.
- The **Collaborative filtering approach**, which uses **only identifiers** of the users and the items. It is based on a **matrix of ratings** given by the users to the items. The main source of information about a user is the

list the items they've rated and the similarity with other users who have rated the same items.

The **SVD recommender module** in Azure Machine Learning designer is based on the **Single Value Decomposition algorithm**. It uses identifiers of the users and the items, and a matrix of ratings given by the users to the items. It's a typical example of **collaborative recommender**.

Lab Overview

In this lab, we make use of the Train SVD Recommender module available in Azure Machine Learning designer (preview), to **train a movie recommender engine**. We use the collaborative filtering approach: the model learns from a collection of ratings made by users on a subset of a catalog of movies. Two open datasets available in Azure Machine Learning designer are used the IMDB Movie Titles dataset joined on the movie identifier with the Movie Ratings dataset. The Movie Ratings data consists of approximately 225,000 ratings for 15,742 movies by 26,770 users, extracted from Twitter using techniques described in the original paper by Doods, De Pessemier and Martens. The paper and data can be found on [GitHub](https://github.com/sidooms/MovieTweetings) (<https://github.com/sidooms/MovieTweetings>).

We will both **train the engine and score new data**, to demonstrate the different modes in which a recommender can be used and evaluated. **The trained model will predict what rating a user will give to unseen movies**, so we'll be able to recommend movies that the user is most likely to enjoy. We will do all of this from the Azure Machine Learning designer without writing a single line of code.

Train SVD Recommender pre-built module parameters:

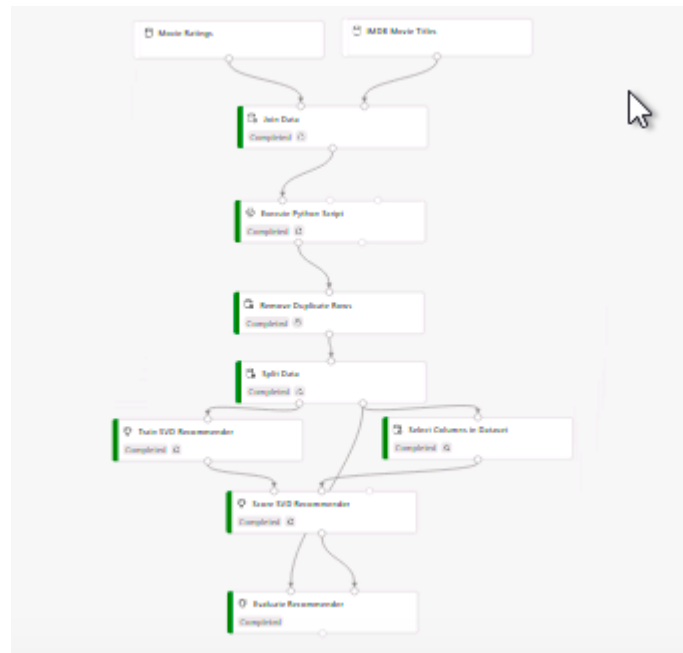
- **Number of factors:** 200. This option specifies the number of factors to use with the recommender. With the number of users and items increasing, it's better to set a larger number of factors. But if the number is too large, performance might drop.
- **Number of recommendation algorithm iterations:** 30. This number indicates how many times the algorithm should process the input data. The higher this number is, the more accurate the predictions are. However, a higher number means slower training. The default value is 30.
- **Learning rate:** 0.001. The learning rate defines the step size for learning.

Score SVD Recommender pre-built module parameters:

- **Recommender prediction kind:** Rating Prediction. For this option, no other parameters are required. When you predict ratings, the model calculates how a user will react to a particular item, given the training data. The input data for scoring must provide both a user and the item to rate.

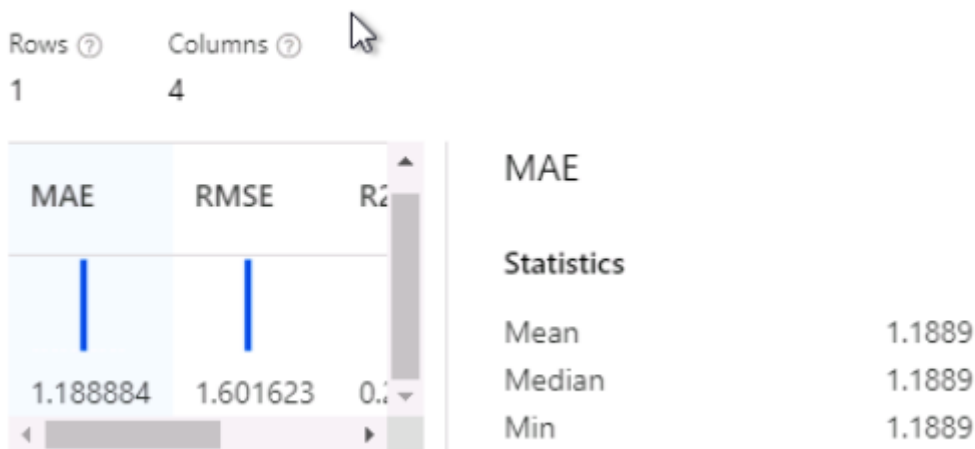
You can learn more about the SVD algorithm used in this lab by selecting [Train SVD Recommender](https://docs.microsoft.com/en-us/azure/machine-learning/algorithm-module-reference/train-svd-recommender) (<https://docs.microsoft.com/en-us/azure/machine-learning/algorithm-module-reference/train-svd-recommender>).

Module pipeline



Model evaluation

Evaluate Recommender result visualization



Section 15: Text Classification

Before we can do text classification, the text first needs to be translated into **some kind of numerical representation**, a process known as **text embedding**. The resulting numerical representation, which is usually in the form of **vectors**, can then be used as an input to a wide range of **classification algorithms**.

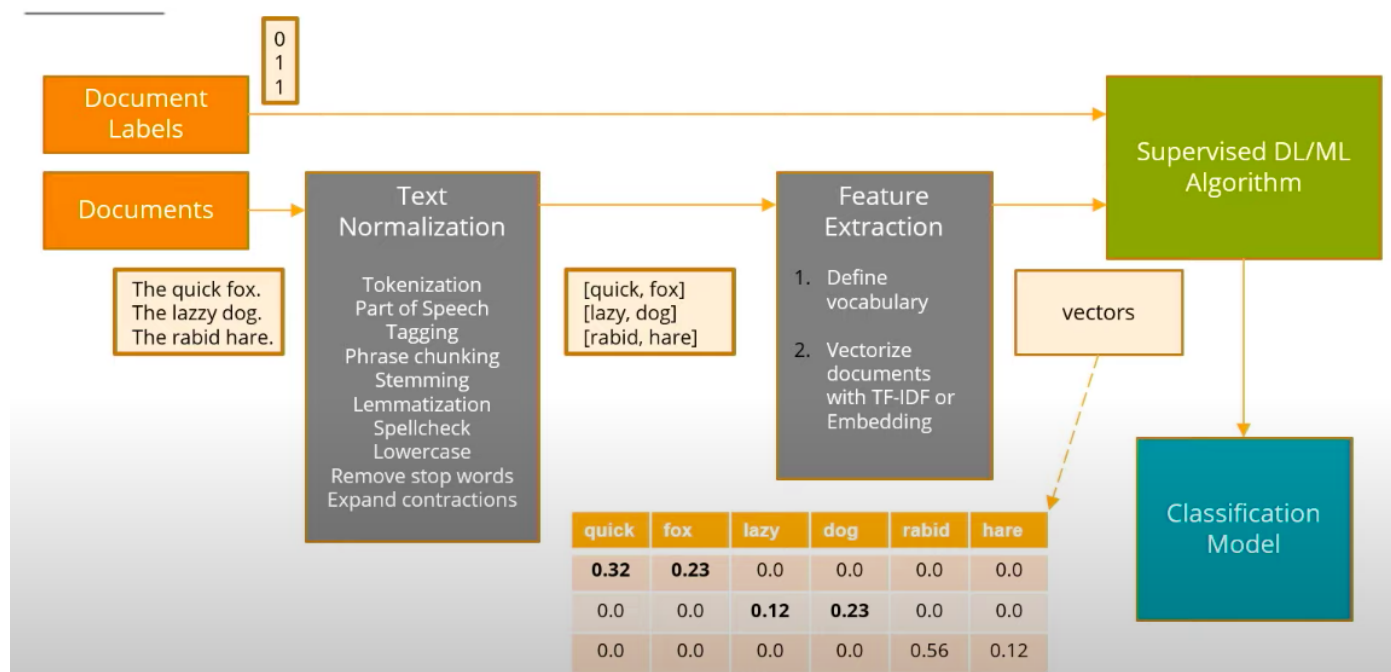
Text embedding comes into 2 **major forms**:

- **Word embedding**: we try to transform every single word in our text into a numerical vector with a number of features or a certain dimension, and then use this representation in the model training process. A variation is **sentence embedding** where we take the single word embeddings and combine them to create a **single embedding at the sentence level**. It gives us a **deep understanding of the text structure** and allows us to carry out complex tasks such as **ML translation**.
- **Scoring**: it aims to calculate a **score** related to the **importance** of that **particular word** in the text. This approach is simpler than word embedding and enables us to understand **global properties** of the text (e.g. the **sentiment**), but it does not allow more complex tasks such as ML translation.

The resulting **numerical representations** of text (usually as **vectors**) can be fed into **classification ML** algorithms.

Training a Classification Model with Text

Let's have a look at a ML pipeline for this kind of problem:



We want to train the model to **classify a collection of documents**:

- 1) We start with a **set of documents** and their **associated labels** (e.g. 0/1).
- 2) The **documents** need to be **normalised** (e.g. identify parts of speech, remove stop words, transform every single word into their canonical form = lemmatization, ...).
- 3) Application of **feature extraction**, as a **two-step process**: (i) Define a **vocabulary** (identifying individual words based on their frequency and record them as a vocabulary). (ii) **Vectorise** the **documents** using different approaches (using **word embedding** or **word scoring**; one typical example of the latter approach is **TF-IDF** which calculates the relative-importance scores of the words in your text).
- 4) Once everything is transformed into vectors, you can use this **numerical dataset** to **train your supervised deep learning or classical ML algorithm**.
- 5) At the end you get a **classification model**.

An important part of the pipeline is the process of vectorizing the text, using a technique such as **Term Frequency-Inverse Document Frequency (TF-IDF) vectorization**

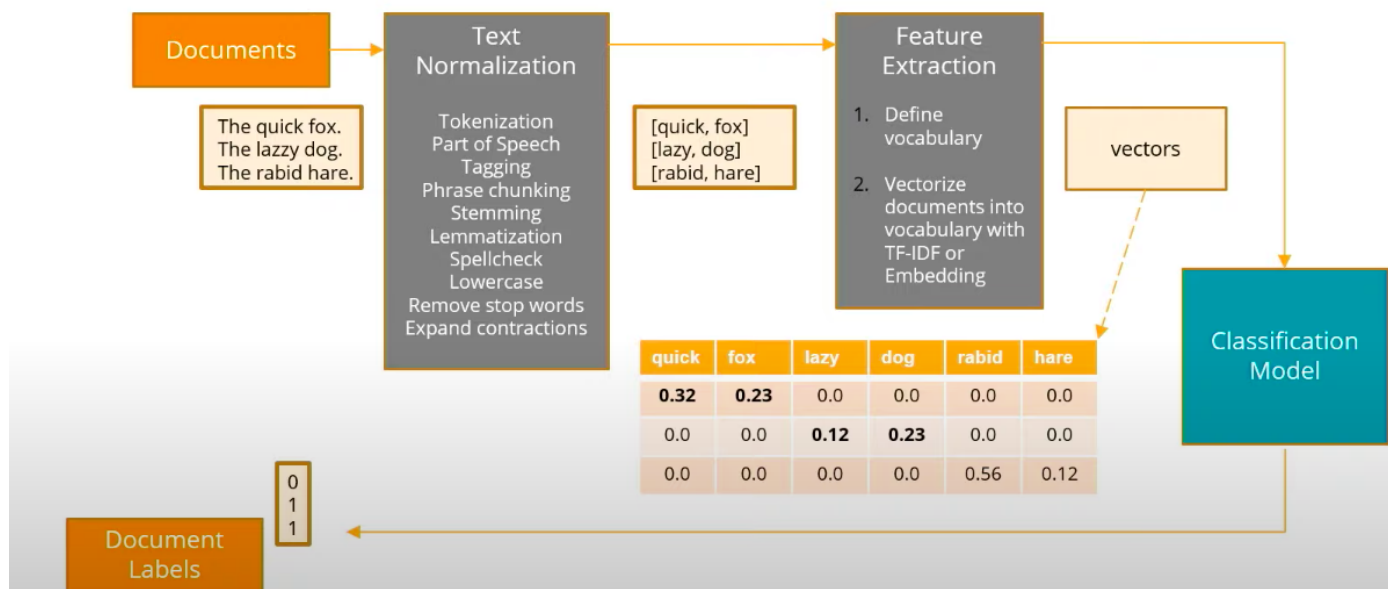
(<https://en.wikipedia.org/wiki/Tf%E2%80%93idf> (<https://en.wikipedia.org/wiki/Tf%E2%80%93idf>)). We discussed text vectorization in some detail back in the introduction lesson when we described text data—so you may want to go back to that section if you feel like you need a review of the concept.

Predicting a Classification from Text

Once we have trained the classification model we then need to use it in an appropriate way to predict classification from a new piece of text. This happens in a way very similar to that used for training the text classification model:

- 1) **Text normalisation**.
- 2) **Feature extraction**. The **same vocabulary used in the training process** needs to be used here as well. You use the **same word embedding process** as used in the training.
- 3) The model then will **classify** your input text dataset, providing it with an **output label**

Predicting a classification from text



Section 16: Lab (Train a Simple Text Classifier)

In text classification scenarios, the goal is to assign a piece of text, such as a document, a news article, a search query, an email, a tweet, support tickets, customer feedback, user product review, to predefined classes or categories. Some examples of text classification applications are: categorizing newspaper articles into topics, organizing web pages into hierarchical categories, spam email filtering, sentiment analysis, predicting user intent from search queries, support tickets routing, and customer feedback analysis.

Lab Overview

In this lab we demonstrate how to use text analytics modules available in Azure Machine Learning designer (preview) to build a simple text classification pipeline. We will create a training pipeline and initialize a multiclass logistic regression classifier to predict the company category with Wikipedia SP 500 dataset derived from Wikipedia. The dataset manages articles of each S&P 500 company. Before uploading to Azure Machine Learning designer, the dataset was processed as follows: extracted text content for each specific company, removed wiki formatting, removed non-alphanumeric characters, converted all text to lowercase, known company categories added. Articles could not be found for some companies, so that's why the number of records is less than 500.

Steps for Building the Pipeline:

- 1) Add Wikipedia SP 500 Sample Datasets
- 2) **Preprocess text** for following steps: using the **Text Analytics/Preprocess Text** module. The dataset input of this prebuilt module needs to be connected to a dataset that has at least one column containing text. Default configuration:

- Remove stop words
- Use lemmatization
- Detect sentences
- Normalise case to lowercase
- Remove numbers
- Remove special characters
- Remove duplicate characters
- Remove email addresses
- Remove URLs
- Normalise backslashes to slashes
- Split tokens on special characters

3) **Split the dataset** into training set (0.5) and test (0.5) set.

4) Convert the plain text of the articles to integers with **Feature Hashing** module, on the training set (hashing bitsize = 10 / N-grams = 2). The goal of using feature hashing is to **reduce dimensionality**; also it makes the **lookup of feature weights faster at classification time** because it uses hash value comparison instead of string comparison.

5) Featurize unstructured text data with **Extract N-Gram Feature** from Text module, on the training set:

- Vocabulary mode = create (you're creating a new list of n-gram features)
- N-grams size = 2 (the maximum size of the n-grams to extract and store)
- Weighting function: TF-IDF Weight (This function calculates a term frequency/inverse document frequency score and assigns it to each n-gram. The value for each n-gram is its TF score multiplied by its IDF score).
- Check the option to Normalize n-gram feature vectors. If this option is enabled, each n-gram feature vector is divided by its L2 norm.

6) **Remove text columns** from dataset.

7) Add the **Train Model Modules**

8) Use the **Multiclass logistic regression** algorithm (<https://docs.microsoft.com/en-us/azure/machine-learning/studio-module-reference/multiclass-logistic-regression> (<https://docs.microsoft.com/en-us/azure/machine-learning/studio-module-reference/multiclass-logistic-regression>))

9) Process the test sets for analysis (note for the test associated to the Extract N-Gram Feature training set you need to use as **Vocabulary mode = Read-only** and connect to it as its second input the N-Gram Feature module used for training so that they both use the **same dictionary**)

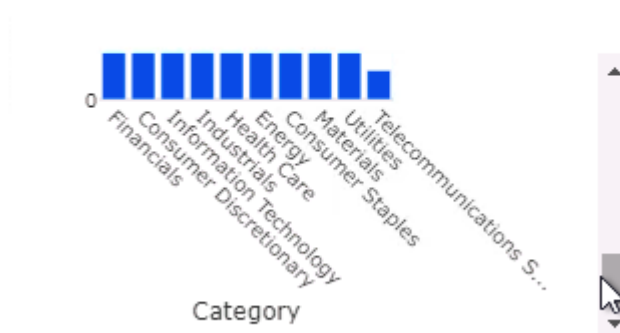
10) **Score** each model.

11) **Evaluate** both models.

Pipeline




For Feature Hashing model:



Score Model result visualization

x

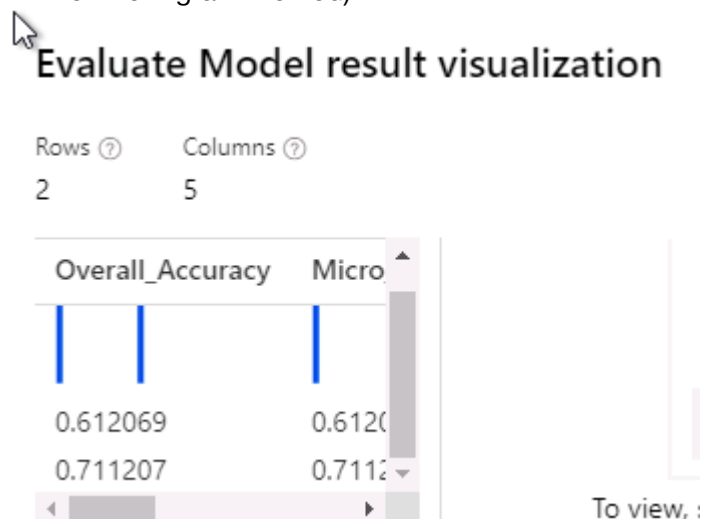


A screenshot of a PivotTable in Excel. The PivotTable is set to show a bar chart for the 'Category' field. The chart displays a series of blue bars of varying heights, representing the distribution of data across categories. The PivotTable structure is visible in the background, with 'Category' as the row label and a numerical value as the data series.



Models' evaluations

We compare the performance of the result generated with feature hashing method and n-gram method (e.g. overall accuracy higher = 0.711 for the n-gram method).



Section 18: Feature Learning

Feature engineering is one of the core techniques that can be used to increase the chances of success in solving machine learning problems. As a part of feature engineering, **feature learning** (also called **representation learning**) is a technique that you can use to **derive new features** in your dataset. Let's have a look at how this technique works. It is often the case that existing features are not enough to obtain high quality ML models. feature/representation learning is used to **transform sets of inputs** into **new inputs** potentially more useful in solving a given problem.

Supervised and Unsupervised Approaches

Feature learning is one of the machine learning techniques that can be done in **both supervised and unsupervised** ways. Let's have a look at both approaches:

- **Supervised features learning:** new features are learned using **data already been labelled**. Examples:
 - Datasets that have **multiple categorical features with high cardinality**. In this case, classical approaches like on-hot encoding fail to produce good results. **Deep learning encoders** can be used to **embed features** into **numerical vectors**. When you have multiple categories and you do simple embedding of the categories, such as 1,2,3,... you **introduce bias** in the data because you are implicitly assuming that there is an order between the categories. One way around this would be one-hot encoding, which however is not feasible for categorical columns with many categories.
 - **Image classification**. The original form of data may not be really suited for a classification task. **Supervised feature learning** then occurs, for example, more or less **automatically** within the **hidden layers** of a **deep neural net**.
- **Unsupervised features learning:** you are trying to learn new features **without** having input labelled data. The typical form of unsupervised feature learning is **clustering** (the **cluster identifier** can be considered a **new feature**). Besides clustering, several types of ML algorithms used in feature learning:
 - **PCA** (more a stats approach rather than a ML approach)
 - **Independent component analysis**
 - **Autoencoder (deep learning)**
 - **Matrix factorisation**

Section 19: Applications of Feature Learning

Some prominent applications of feature learning include:

- **Image classification:** deep learning models with their hidden layers are capable of automatically learning those features.
- **Image search:** a similar approach to image classification is used to search for images
- **Feature embedding:** for categorical features with high cardinality (difficult to model with traditional approaches such as one-hot encoding) (more on feature embedding here:
<https://towardsdatascience.com/neural-network-embeddings-explained-4d028e6f0526>
(<https://towardsdatascience.com/neural-network-embeddings-explained-4d028e6f0526>))

Image Classification with Convolutional Neural Networks (CNNs)

A **CNN** is a special case of a **deep learning neural net** with a combination of hidden layers with specific abilities in **learning hidden features**. A typical CNN will consist of out of a stack of **2 types of hidden layers**:

- **Convolution layer(s):** used to learn some **local patterns**. These are **translation invariant** (once the model learns to recognise a certain type of local pattern somewhere in the image, it can also recognise it somewhere else in the image). This learning approach makes CNNs very different from other types of neural nets (e.g. densely connected neural nets) which learn global patterns instead and need to be re-trained to recognise the same pattern at different locations. The local patterns are learned by application of a special **mathematical operation**, i.e. using a **3x3 matrix (kernel)** applied over the input data. Specific kernels in image processing are used for specific tasks (e.g. edge detection, sharpening, normalisation, blur approximation ...)
- **Max-Pooling layer:** it **down-samples** the data and make the learning **translation-invariant**. It uses a **special kernel**, resulting in a representation that is translational-invariant (enabling, for example, the recognition of a human face even from different angles). Its **output** uses a **densely connected layer** to produce the final classification.

This approach in image classification using CNN is very powerful both because of the use of the convolutional layers (to learn local patterns) and for the CNN capability of learning **hierarchies of patterns**: a usual example of CNN will have **multiple stacks** of Conv layer + Max-pooling layer, enabling the learning of **increasingly complex local patterns** (e.g. from learning simpler local structures such as nose, eyes, ears to learning more complex structures like faces, then face patterns etc...).

Image classification with CNNs

CNN =
Convolutional Neural Network

Step 1 – learn local patterns

Step 2 – down-sample and make translation invariant

Step 3 – densely connect all outputs to learn classification

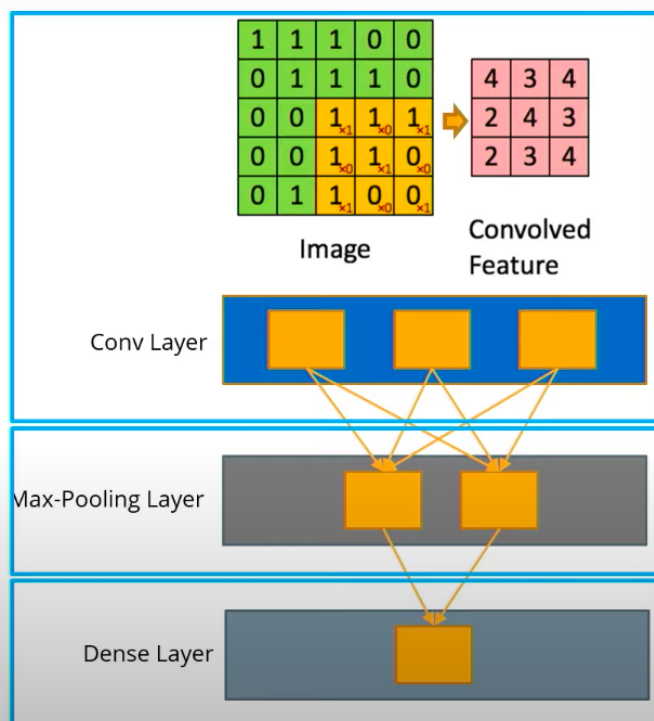


Image Search with Autoencoders

An **Autoencoder** is a very special type of **neural net** used for **unsupervised learning**. Its most important characteristic is that it is trained to **reproduced as accurately as possible its own inputs**. Autoencoders aim to reproduce labelled data at their output. Autoencoders typically start with a **very large number of inputs** and then **each layer is progressively narrower** than the previous one, until the **minimum layer width** is reached. Then, layers start to have **increasingly larger widths**, until the last one which has the **same number of outputs as there are inputs** in the first layer.

The **middle layer** (the one with the smallest width) is important for two reasons:

- It marks the **border between the 2 internal components** of the autoencoder (the **encoder** is **left** of the middle layer, while the **decoder** is **right** of the middle layer).
- It produces the **feature vector**, a compressed (i.e. **dimensionally reduced** and not meaning reduced in size, autoencoders are not used in compressing images!) representation of the input. The autoencoder is then able to **translate the image into an n-dimensional vector** (**n** is the **width** of the **middle layer**).

The **left part** of the autoencoder (the **encoder**) is a **trained deep learning model** that can be used to **encode inputs**.

The **n-dimensional vectors** produced by the encoder can then be **compared** to each other using some **distance metric** such as the **Euclidean distance**.

Image search with autoencoders

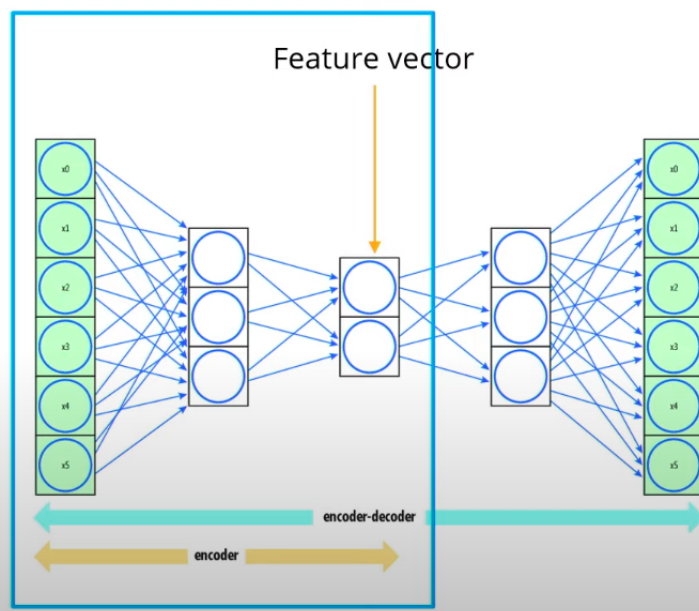
**Autoencoder =
neural net for unsupervised learning**

Trains to reproduce its inputs

Produces the feature vector

The left half (the encoder) can be used to embed inputs into feature vectors

A distance measure is used to identify similar input images



Section 20: Anomaly Detection

Datasets often contain a **small number of items that deviate significantly from the norm**. These **anomalies** can be of interest, since they may be the result of bad data, unusual behavior, or important exceptions to the typical trends. Anomaly detection is a machine learning technique concerned with finding these data points. Anomaly detection is basically a **classification problem** dealing with a **severe imbalance** between the **classes** that need to be predicted. The aim is: given a **set of entities**, train a model to **detect anomalies**. It is used, for example in banking to detect frauds.

The **class imbalance** between abnormal and normal cases (i.e. **N abnormal entities << N normal entities**, i.e. anomalies are several normal of magnitude smaller than normal entities) makes the training challenging because some of the typical ML algorithms used for binary classification cannot work efficiently.

Supervised & Unsupervised Approaches

- **Supervised anomaly detection: binary classification problem** where entities need to be classified as either normal or anomalies. It is based on using a training dataset that has **already been labelled** as normal/anomaly (e.g. in a bank environment you will need a historical dataset with transactions that have been proven to be fraudulent).
- **Unsupervised anomaly detection:** the problem is **identifying 2 major groups/clusters** of entities (normal and abnormal), thus this is defined as a **clustering problem**. It is based on using a training dataset **without labels** (normal/anomaly). The final task is to create a model able to define what normal and abnormal mean and detect this pattern in new items.

Applications of Anomaly Detection

- **Condition monitoring (industrial maintenance) and failure prevention**
- **Fraud detection** (e.g. in banking or telecommunications)
- **Intrusion detection** (e.g. any abnormal type of activity in network communication)
- **Anti-virus and anti-malware protection** (i.e. abnormal operations manifesting on a certain computer/device)
- **Data preparation (outlier detection)**

Specific example of anomaly detection, so that we can get a more concrete idea of what the process might look like. In this particular example, we'll consider what **anomaly detection** might look like when **applied to machinery maintenance**.

- In this example we will train an **autoencoder** (a deep learning model) to **recognise normality**, which then will be able to identify anomalies.
- A set of **machine parameters** (e.g. reading from sensors) are passed as **inputs** through the **autoencoder**. Once we have a **set of readings** that we know to represent **normal conditions** of operations, we use them to **train** the autoencoder.
- Once the autoencoder is trained and you apply it to the **real stream of data**, when **failure occurs** the autoencoder will be able to **identify it**.
- **Failure identification** is done using the **threshold value**, one of the metrics used to train the autoencoder (in the example is the Mean Average Error = MAE, the red horizontal line). The autoencoder trained on normal data will contain values for this threshold metric; the **highest possible value** contained in the autoencoder for the threshold metric and this will be the **threshold for normality**.
- When you get **new data**, you process it through the autoencoder and calculate their metric (e.g. MAE) and compare it with the maximum allowed for normal data. If it is above this threshold, the data is classified as anomaly.

One interesting ability of this approach is to **predict early failure**. In the vast majority of cases, complex systems like industrial machinery are **not** failing suddenly, but produce subtle signs of failure and if you are able to detect them early (e.g. looking where the autoencoder curve is starting to deviate from normality but is not abnormal yet) you can take actions early (e.g. stopping the machine before it completely fails)

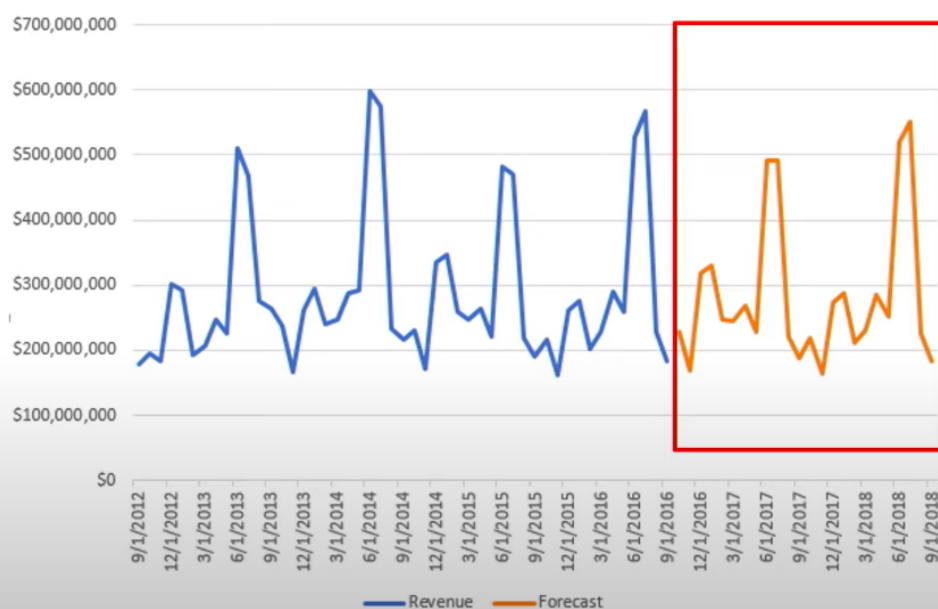
Section 22: Forecasting

Forecasting is a class of problems that deals with **predictions in the context of orderable datasets**.

A typical example of a **forecasting problem** would be: Given a **set of ordered data points** (such as sales data over a series of dates), we want to **predict the next data points in the series** (such as what sales will look like next week). Forecasting deals with sets of events that can be **ordered in time**. This ordering most of the time relates to **actual dates** or **moments in time (time-series datasets)**, but it can also simply refer to **orderable sets of items**.

Usually the dataset contains **numerical values** that can be **ordered** based on some kind of column that **provides explicit ordering information** (a date-time column in the graph below).

Given N time-ordered single-valued data points, predict the next M points



In some situations you might have more than one property for every single data point you have (one numerical will be the target property and the other categorical/continuous properties can be used to enhance the power of the underlying ML model).

Types of Forecasting Algorithms

- **ARIMA (AutoRegressive Integrated Moving Average)**: it is an evolution of **ARMA (AutoRegressive Integrated Moving Average)** algorithm, initially designed to provide some kind of description for a random time-based processing (such as time series).
- **Multi-variate regression**: the problem of forecasting can be also modelled as a regression problem (predicting a numerical value for the next iteration of the time series). Being a multivariate approach, you can also take into account other properties of the entities that occur in the time series
- **Prophet**: algorithm initially developed by Facebook. It works best with time-series with a **strong seasonal effect** (e.g. sales for a pharmacy, some medications are very seasonal such as for flu)
- **ForecastTCN**: based on **TCNs (Temporal Convolution Networks)** and developed by Microsoft. It is a one-dimensional convolutional network because of its specific approach based on time series.
- **RNNs (Recurrent Neural Networks)**: they are class of networks with **additional connections between the nodes**. The classical structure of a neural net is feed-forward, with all data/connections flowing in one direction (input -> output). Here there are additional **backwards facing connections** that create **cycles** in the net, obtaining the RNNs. RNNs can effectively **learn time-based patterns**. There are several types of RNNs (e.g. long short term memory, gated recurrent unit) and they have revolutionised areas like **speech recognition, text-to-speech synthesis** and **machine translation**, all fields highly connected to time series based data

Section 23: Lab (Forecasting)


In this lab you will learn how the **Automated Machine Learning** capability in Azure Machine Learning (AML) can be used for the **life cycle management of the manufactured vehicles** and how AML helps in **creation of better vehicle maintenance plans**. To accomplish this, you will **train a Linear Regression model to predict the number of days until battery failure** using Automated Machine Learning available in AML studio.



Regression algorithm parameters:

- **Primary metric*: Normalized root mean squared error
- **Exit criterion** > Metric score threshold: 0.09
- **Validation** > **Validation type**: k-fold cross validation, **Number of Cross Validations**: 5
- **Concurrency** > Max concurrent iterations: 1




Model evaluation

In the models list, notice at the top the iteration with the **best normalized root mean square error score**. Note that the normalized root mean square error measures the error between the predicted value and actual value. In this case, the model with the lowest normalized root mean square error is the best model.

Run 1  Completed

 Refresh  Cancel

Details Data guardrails Models Outputs + Logs Child runs Snapshot

Properties	Best model summary
<div>Status<div> Completed</div></div>	<div>Algorithm name<div>MaxAbsScaler, LightGBM</div></div>
<div>Created<div>Jul 20, 2020 9:40 AM</div></div>	<div>Normalized root mean squared error<div>0.043281  View all other metrics</div></div>
<div>Duration<div>1m 13.54s</div></div>	<div>Sampling<div>100% </div></div>
<div>Compute target<div>aml-compute</div></div>	<div>Registered models<div>No registration yet</div></div>
<div>Run ID<div>AutoML_fa204a60-0f47-4225-acea-a78322d519d4</div></div>	<div>Deploy status<div>No deployment yet</div></div>