**What is Machine Learning?**

In all but the most trivial cases, insight or knowledge you’re trying to get out of the raw data won’t be obvious from looking at the data. For example, in detecting spam email, looking for the occurrence of a single word may not be very helpful. But looking at the occurrence of certain words used together, combined with the length of the email and other factors, you could get a much clearer picture of whether the email is spam or not. **Machine learning is turning data into information. Machine learning lies at the intersection of computer science, engineering, and statistics and often appears in other disciplines**. As you’ll see later, it can be applied to many fields from politics to geosciences. **It’s a tool that can be applied to many problems.** Any field that needs to interpret and act on data can benefit from machine learning techniques. **Machine learning uses statistics**. To most people, statistics is an esoteric subject used for companies to lie about how great their products are. (There’s a great manual on how to do this called How to Lie with Statistics by Darrell Huff. Ironically, this is the best-selling statistics book of all time.) So **why do the rest of us need statistics?** The practice of engineering is applying science to solve a problem. In engineering we’re used to solving a deterministic problem where our solution solves the problem all the time. If we’re asked to write software to control a vending machine, it had better work all the time, regardless of the money entered or the buttons pressed. There are many problems where the solution isn’t deterministic. That is, we don’t know enough about the problem or don’t have enough computing power to properly model the problem. For these problems we need statistics. For example, the motivation of humans is a problem that is currently too difficult to model. In the social sciences, being right 60% of the time is considered successful. If we can predict the way people will behave 60% of the time, we’re doing well. How can this be? Shouldn’t we be right all the time? If we’re not right all the time, doesn’t that mean we’re doing something wrong? Let me give you an example to illustrate the problem of not being able to model the problem fully. Do humans not act to maximize their own happiness? Can’t we just predict the outcome of events involving humans based on this assumption? Perhaps, but it’s difficult to define what makes everyone happy, because this may differ greatly from one person to the next. So even if our assumptions are correct about people maximizing their own happiness, the definition of happiness is too complex to model. There are many other examples outside human behavior that we can’t currently model deterministically. For these problems we need to use some tools from statistics.

Key tasks of machine learning In this section we’ll outline the key jobs of machine learning and set a framework that allows us to easily turn a machine learning algorithm into a solid working application. The example covered previously was for the task of classification. **In classification, our job is to predict what class an instance of data should fall into**. Another task in machine learning is regression. **Regression is the prediction of a numeric value.** Most people have probably seen an example of regression with a best-fit line drawn through some data points to generalize the data points. **Classification and regression are examples of supervised learning**. This set of problems is known as supervised because we’re telling the algorithm what to predict. The opposite of supervised learning is a set of tasks known as unsupervised learning. In unsupervised learning, there’s no label or target value given for the data. A task where we group similar items together is known as clustering. In unsupervised learning, we may also want to find statistical values that describe the data. This is known as density estimation. Another task of unsupervised learning may be reducing the data from many features to a small number so that we can properly visualize it in two or three dimensions. Table 1.2 lists some common tasks in machine learning with algorithms used to solve these tasks. If you noticed in table 1.2 that multiple techniques are used for completing the same task, you may be asking yourself, “If these do the same thing, why are there four different methods?

* **Unsupervised learning** : no labeled data.
* **Supervised learning** : uses labeled data for prediction on unseen points.
* **Reinforcement learning** : Unbounded data

**Supervised Learning**

In supervised learning**, the target is to infer a function or**

**mapping from training data that is labeled**. The training data

consist of input vector X and output vector Y of labels or tags.

A label or tag from vector Y is the explanation of its

respective input example from input vector X. Together they form

a training example. In other words, training data comprises

training examples. If the labeling does not exist for input vec-

tor X, then X is unlabeled data.

**Why such learning is called supervised learning?** The

output vector Y consists of labels for each training example

present in the training data. These labels for output vector

are provided by the supervisor. Often, these supervisors are

humans, but machines can also be used for such labeling.

Human judgments are more expensive than machines, but the

higher error rates in data labeled by machines suggest superiority of human judgment. The manually labeled data is a precious and reliable resource for supervised learning. However,

in some cases, machines can be used for reliable labeling.

Two groups or categories of algorithms come under the

umbrella of supervised learning.

They are

1. Regression

2. Classification

***Unsupervised Learning***

In unsupervised learning, we lack *supervisors* or

training data. In other words, all what we have is

**unlabeled data. The idea is to find a hidden structure**

**in this data**. **There can be a number of reasons for the**

**data not having a label**. It can be due to unavailability of

funds to pay for manual labeling or the inherent nature

of the data itself. With numerous data collection devices,

now data is collected at an unprecedented rate. The

variety, velocity, and the volume are the dimensions in

which *Big Data* is seen and judged. To get something

from this data without the supervisor is important.

This is the challenge for today’s machine learning

practitioner.

In the machine learning community, probably ***clustering* (an**

**unsupervised learning algorithm) is analogous to the *walk***

***long enough* instruction of the Cheshire cat.** The *somewhere* of

Alice is equivalent to *finding regularities in the input*.

**Reinforcement Learning**

The reinforcement learning method **aims at using observations**

**gathered from the interaction with the environment to take**

**actions that would maximize the reward or minimize the risk**.

In order to produce intelligent programs (also called agents),

reinforcement learning goes through the following steps:

1. Input state is observed by the agent.

2. Decision making function is used to make the agent

perform an action.

3. After the action is performed, the agent receives reward

or reinforcement from the environment.

4. The state-action pair information about the reward is

stored.

Using the stored information, policy for particular state in

terms of action can be ne-tuned, thus helping in optimal

decision making for our agent.

Reinforcement learning will not be discussed further in this

book.

**How to choose the right algorithm**

With all the different algorithms how can you choose which one to use? First, you need to consider your goal. What are you trying to get out of this? (Do you want a probability that it might rain tomorrow, or do you want to find groups of voters with similar interests?) What data do you have or can you collect? Those are the big questions. Let’s talk about your goal. **If you’re trying to predict or forecast a target value**, then you need to look into supervised learning. If not, then unsupervised learning is the place you want to be. **If you’ve chosen supervised learning, what’s your target value? Is it a discrete value like Yes/No, 1/2/3, A/B/C, or Red/Yellow/Black?** If so, then you want to look into classification. If the target value can take on a number of values, say any value from 0.00 to 100.00, or -999 to 999, or + to -, then you need to look into regression. **If you’re not trying to predict a target value, then you need to look into unsupervised learning. Are you trying to fit your data into some discrete groups? If so and that’s all you need, you should look into clustering**. Do you need to have some numerical estimate of how strong the fit is into each group? If you answer yes, then you probably should look into a density estimation algorithm. The rules I’ve given here should point you in the right direction but are not unbreakable laws. In chapter 9 I’ll show you how you can use classification techniques for regression, blurring the distinction I made within supervised learning. The second thing you need to consider is your data. You should spend some time getting to know your data, and the more you know about it, the better you’ll be able to build a successful application. Things to know about your data are these: Are the features nominal or continuous? Are there missing values in the features? If there are missing values, why are there missing values? Are there outliers in the data? Are you looking for a needle in a haystack, something that happens very infrequently? All of these features about your data can help you narrow the algorithm selection process. With the algorithm narrowed, there’s no single answer to what the best algorithm is or what will give you the best results. You’re going to have to try different algorithms and see how they perform. There are other machine learning techniques that you can use to improve the performance of a machine learning algorithm. The relative performance of two algorithms may change after you process the input data. We’ll discuss these in more detail later, but the point is that finding the best algorithm is an iterative process of trial and error.

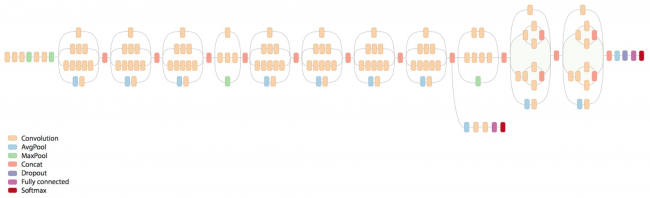
## iOS Machine Learning

Machine learning is a type of artificial intelligence where computers “learn” without being explicitly programmed. Instead of coding an algorithm, machine learning tools enable computers to develop and refine algorithms, by finding patterns in huge amounts of data.

### Deep Learning

Since the 1950s, AI researchers have developed many approaches to machine learning. Apple’s Core ML framework supports neural networks, tree ensembles, support vector machines, generalized linear models, feature engineering and pipeline models. However, neural networks have produced many of the most spectacular recent successes, starting with Google’s 2012 use of YouTube videos to train its AI to recognize cats and people. Only five years later, Google is sponsoring a contest to identify 5000 species of plants and animals. Apps like Siri and Alexa also owe their existence to neural networks.

**A neural network tries to model human brain processes with layers of nodes, linked together in different ways.** Each additional layer requires a large increase in computing power: Inception v3, an object-recognition model, has 48 layers and approximately 20 million parameters. **But the calculations are basically matrix multiplication**, which GPUs handle extremely efficiently. The falling cost of GPUs enables people to create multilayer deep neural networks, hence the term *deep learning*.

[](https://koenig-media.raywenderlich.com/uploads/2017/06/neural-network.png)

A neural network, circa 2016

**Neural networks need a large amount of training data, ideally representing the full range of possibilities**. The explosion in user-generated data has also contributed to the renaissance of machine learning.

*Training the model* means supplying the neural network with training data, and letting it calculate a formula for combining the input parameters to produce the output(s). Training happens offline, usually on machines with many GPUs.

**To *use* the model, you give it new inputs, and it calculates outputs**: this is called *inferencing*. Inference still requires a lot of computing, to calculate outputs from new inputs. Doing these calculations on handheld devices is now possible because of frameworks like Metal.

As you’ll see at the end of this tutorial, deep learning is far from perfect. It’s really hard to construct a truly representative set of training data, and it’s all too easy to over-train the model so it gives too much weight to quirky characteristics.

### What Does Apple Provide?

Apple introduced NSLinguisticTagger in iOS 5 to analyze natural language. Metal came in iOS 8, providing low-level access to the device’s GPU.

Last year, Apple added Basic Neural Network Subroutines (BNNS) to its Accelerate framework, enabling developers to construct neural networks for inferencing (not training).

And this year, Apple has given you Core ML and Vision!

* **Core ML makes it even easier to use trained models in your apps.**
* **Vision gives you easy access to Apple’s models for detecting faces, face landmarks, text, rectangles, barcodes, and objects**.

You can also wrap any image-analysis Core ML model in a Vision model, which is what you’ll do in this tutorial. Because these two frameworks are built on Metal, they run efficiently on the device, so you don’t need to send your users’ data to a server.

Core ML

Integrate machine learning models into your app.

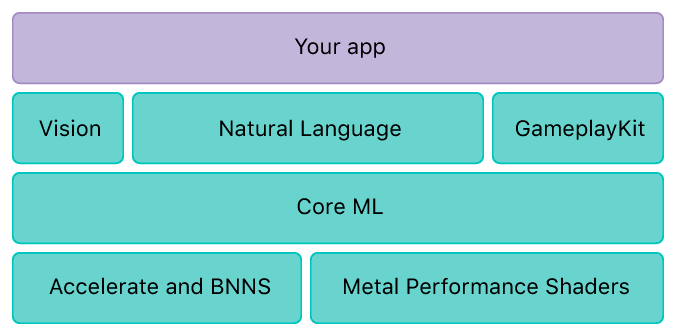
## Overview

With Core ML, you can integrate trained machine learning models into your app.



A trained model is the result of applying a machine learning algorithm to a set of training data. The model makes predictions based on new input data. For example, a model that's been trained on a region's historical house prices may be able to predict a house's price when given the number of bedrooms and bathrooms.

Core ML is the foundation for domain-specific frameworks and functionality. Core ML supports [Vision](https://developer.apple.com/documentation/vision) for image analysis, [Natural Language](https://developer.apple.com/documentation/naturallanguage) for natural language processing, and [GameplayKit](https://developer.apple.com/documentation/gameplaykit) for evaluating learned decision trees. Core ML itself builds on top of low-level primitives like [Accelerate](https://developer.apple.com/documentation/accelerate) and [BNNS](https://developer.apple.com/documentation/accelerate/bnns), as well as [Metal Performance Shaders](https://developer.apple.com/documentation/metalperformanceshaders).



Core ML is optimized for on-device performance, which minimizes memory footprint and power consumption. Running strictly on the device ensures the privacy of user data and guarantees that your app remains functional and responsive when a network connection is unavailable.

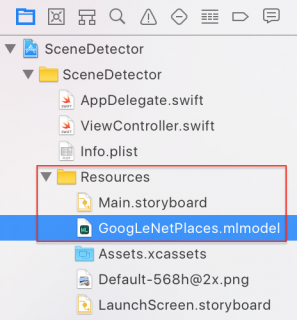
## Integrating a Core ML Model Into Your App

This tutorial uses the **Places205-GoogLeNet** model, which you can download from Apple’s [Machine Learning page](https://developer.apple.com/machine-learning/). Scroll down to **Working with Models**, and download the first one. While you’re there, take note of the other three models, which all detect objects — trees, animals, people, etc. — in an image.

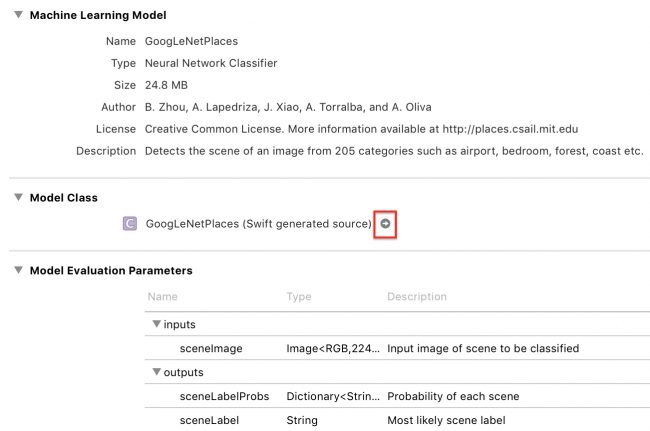
**Note:** If you have a trained model created with a supported machine learning tool such as Caffe, Keras or scikit-learn, [Converting Trained Models to Core ML](https://developer.apple.com/documentation/coreml/converting_trained_models_to_core_ml)describes how you can convert it to Core ML format.

### Adding a Model to Your Project

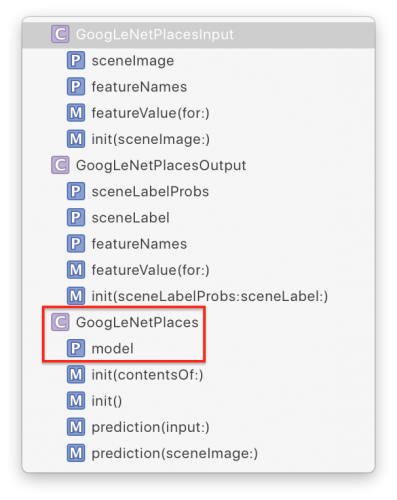
After you download **GoogLeNetPlaces.mlmodel**, drag it from Finder into the **Resources** group in your project’s **Project Navigator**:

[](https://koenig-media.raywenderlich.com/uploads/2017/06/add-model.png)

Select this file, and wait for a moment. An arrow will appear when Xcode has generated the model class:

[](https://koenig-media.raywenderlich.com/uploads/2017/06/model.png)

Click the arrow to see the generated class:

[](https://koenig-media.raywenderlich.com/uploads/2017/06/generated-model-class.png)

Xcode has generated input and output classes, and the main class GoogLeNetPlaces, which has a model property and two prediction methods.

GoogLeNetPlacesInput has a sceneImage property of type CVPixelBuffer. *Whazzat!?*, we all cry together, but fear not, the Vision framework will take care of converting our familiar image formats into the correct input type. :]

The Vision framework also converts GoogLeNetPlacesOutput properties into its own results type, and manages calls to prediction methods, so out of all this generated code, *your* code will use only the model property.

### Wrapping the Core ML Model in a Vision Model

Finally, you get to write some code! Open **ViewController.swift**, and import the two frameworks, just below import UIKit:

import CoreML

import Vision

Next, add the following extension below the IBActions extension:

// MARK: - Methods

extension ViewController {

func detectScene(image: CIImage) {

answerLabel.text = "detecting scene..."

// Load the ML model through its generated class

guard let model = try? VNCoreMLModel(for: GoogLeNetPlaces().model) else {

fatalError("can't load Places ML model")

}

}

}

Here’s what you’re doing:

First, you display a message so the user knows something is happening.

The designated initializer of GoogLeNetPlaces throws an error, so you must use trywhen creating it.

VNCoreMLModel is simply a container for a Core ML model used with Vision requests.

The standard Vision workflow is to create a model, create one or more requests, and then create and run a request handler. You’ve just created the model, so your next step is to create a request.

Add the following lines to the end of detectScene(image:):

// Create a Vision request with completion handler

let request = VNCoreMLRequest(model: model) { [weak self] request, error in

guard let results = request.results as? [VNClassificationObservation],

let topResult = results.first else {

fatalError("unexpected result type from VNCoreMLRequest")

}

// Update UI on main queue

let article = (self?.vowels.contains(topResult.identifier.first!))! ? "an" : "a"

DispatchQueue.main.async { [weak self] in

self?.answerLabel.text = "\(Int(topResult.confidence \* 100))% it's \(article) \(topResult.identifier)"

}

}

VNCoreMLRequest is an image analysis request that uses a Core ML model to do the work. Its completion handler receives request and error objects.

You check that request.results is an array of VNClassificationObservation objects, which is what the Vision framework returns when the Core ML model is a *classifier*, rather than a predictor or image processor. And GoogLeNetPlaces is a classifier, because it predicts only one feature: the image’s scene classification.

A VNClassificationObservation has two properties: identifier — a String — and confidence — a number between 0 and 1 — it’s the probability the classification is correct. When using an object-detection model, you would probably look at only those objects with confidence greater than some threshold, such as 30%.

You then take the first result, which will have the highest confidence value, and set the indefinite article to “a” or “an”, depending on the identifier’s first letter. Finally, you dispatch back to the main queue to update the label. You’ll soon see the classification work happens *off* the main queue, because it can be slow.

Now, on to the third step: creating and running the request handler.

Add the following lines to the end of detectScene(image:):

// Run the Core ML GoogLeNetPlaces classifier on global dispatch queue

let handler = VNImageRequestHandler(ciImage: image)

DispatchQueue.global(qos: .userInteractive).async {

do {

try handler.perform([request])

} catch {

print(error)

}

}

VNImageRequestHandler is the standard Vision framework request handler; it isn’t specific to Core ML models. You give it the image that came into detectScene(image:) as an argument. And then you run the handler by calling its perform method, passing an array of requests. In this case, you have only one request.

The perform method throws an error, so you wrap it in a try-catch.

**References :**

**Machine Learning in Action**  PETER HARRINGTON

**INTRODUCTION TO MACHINE LEARNING** Nils J. Nilsson

Documentation Of CoreML :

**https://developer.apple.com/documentation/coreml**