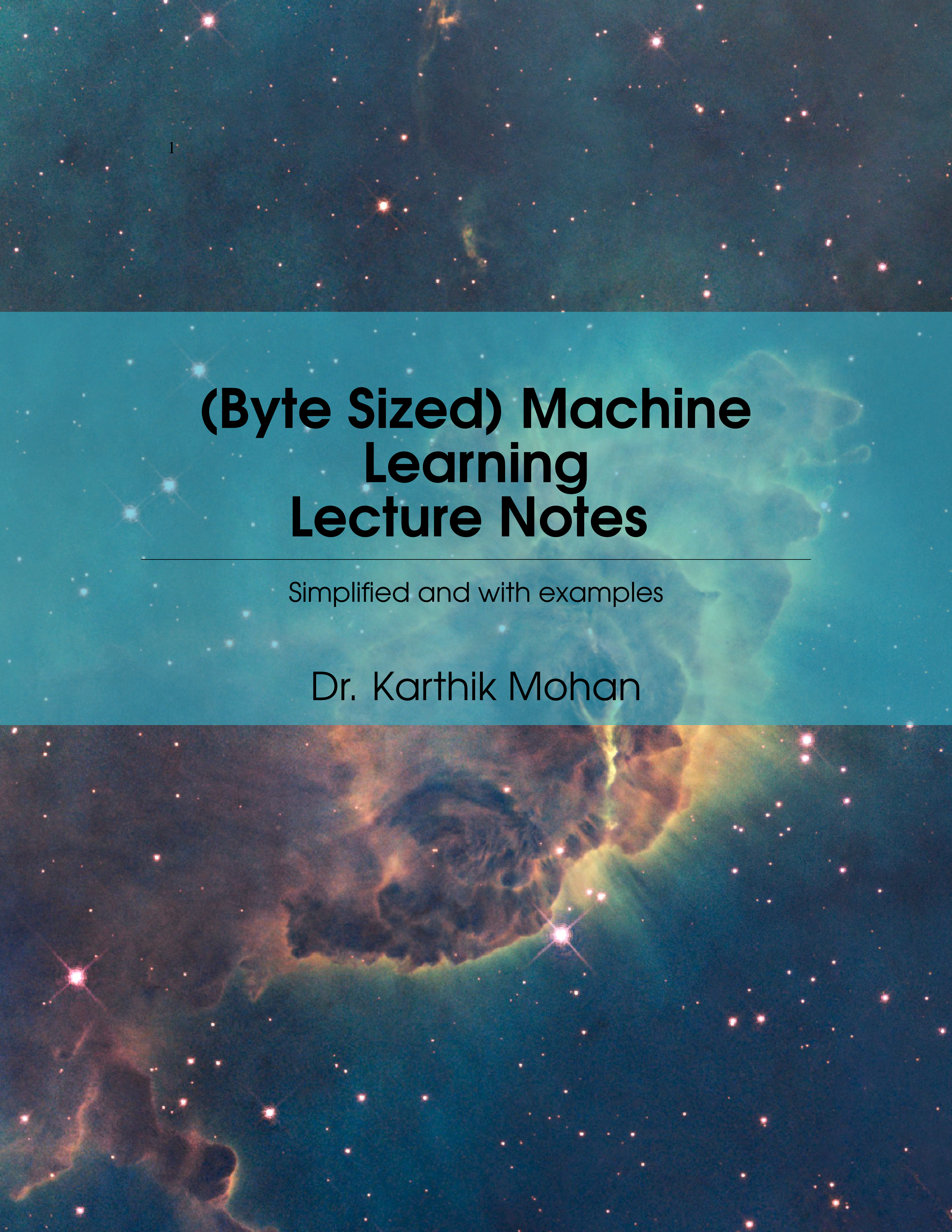


(Byte Sized) Machine Learning Lecture Notes

Simplified and with examples

Dr. Karthik Mohan

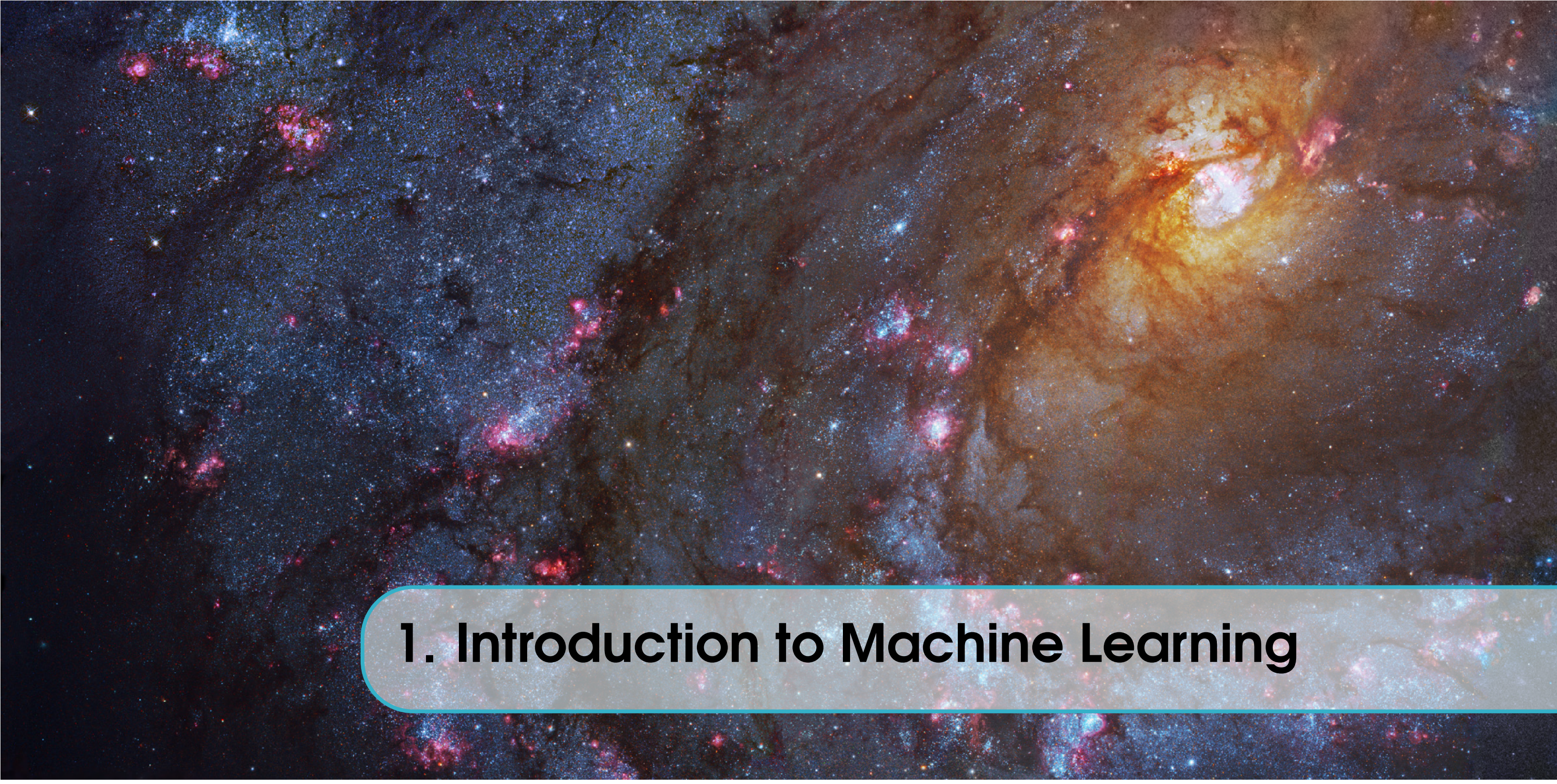
ADVANCED INTRO TO ML, LECTURE NOTES

HTTPS://BYTESIZEML.GITHUB.IO/ML2023/

These lecture notes are part of the course taught at University of Washington, Seattle in Winter 2023.

First release, January 2023

Contents



1. Introduction to Machine Learning

1.1 High-level Motivation

Data is in abundance in today's world. Every leading tech company is focused or moving towards data-driven decision making. Machine Learning in the real-world, bridges the gap between data and making data-driven decisions.

1.2 Motivating Example

Consider that you want to teach a kid to recognize objects, and specifically to recognize apples.

1. So you show the kid a red apple and say 'This is an apple.' The kid picks up on the color red and learns to associate it with apple.
2. You then show the kid a 'red cube' and ask, 'Is this an apple?' - The kid says, 'Yes that's an apple, because it is red in color.'
3. You then clarify, that an apple has to be circular in shape.
4. You then show the kid a red circular ball, and ask, 'Is this an apple?' The kid says, 'Yes, because it is red and circular in shape.'
5. You clarify again, that apples are circular but with a depression at the top.
6. The kid, by now, has understood, that an apple has to be circular in shape but with a depression at the top, and also red in color.
7. With 4 examples, the kid has quickly learned to recognize any apple in the wild!
8. To test the kid's understanding, you show the kid a green apple, and ask if it's an apple? The kid responds, 'No! It's not an apple because it's green in color!'
9. You shake your head and decide you will clarify this to the kid once you finish a business meeting that's about to start in a minute!

1.3 What is Machine Learning?

Here are three distinct but complementary defenitions of machine learning:

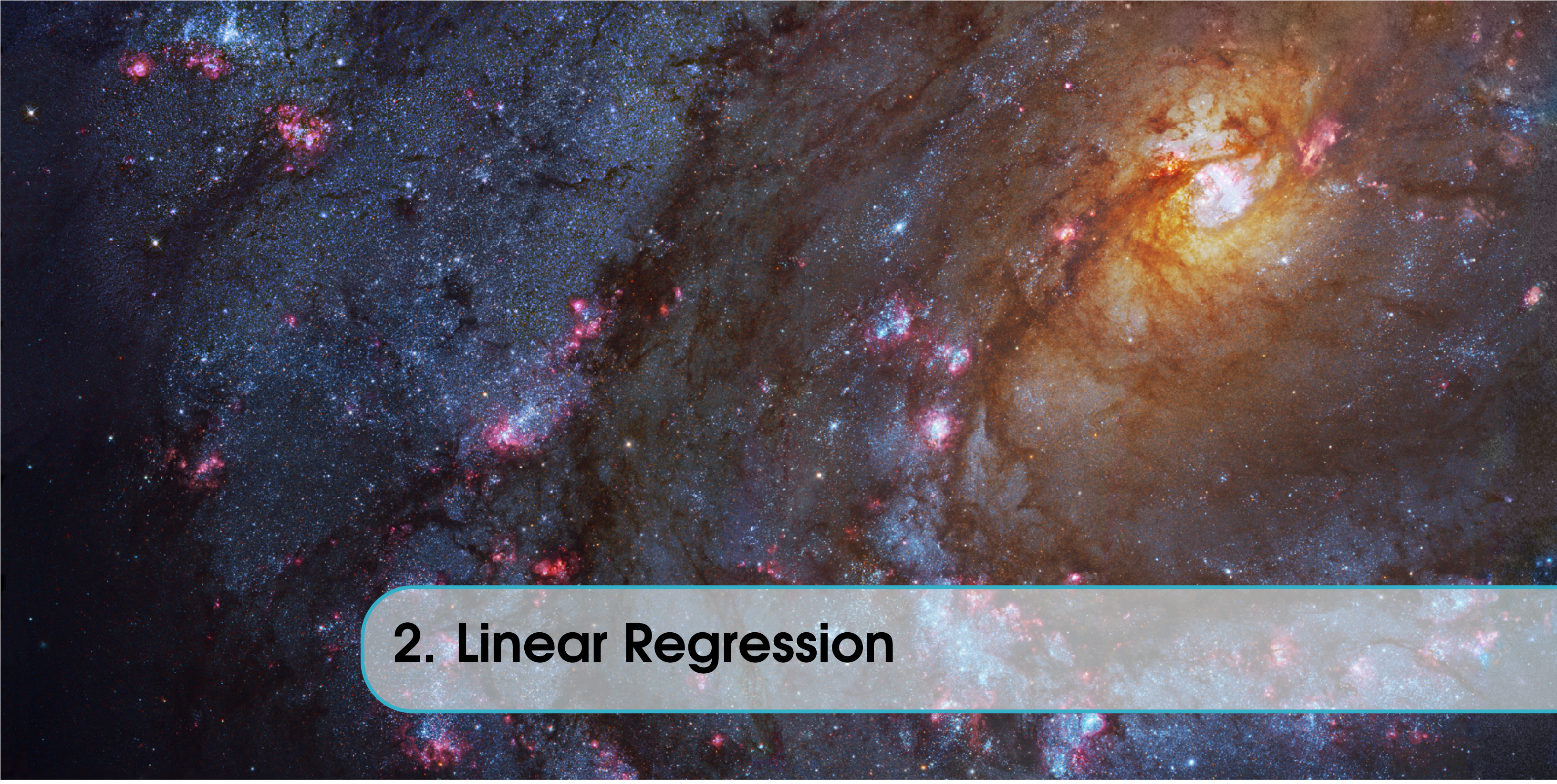
1. Machine Learning is a set of methods, tools, algorithms and frameworks to help us solve real-world problems with data.
2. Machine Learning are tools to learn useful patterns from data sets, which can then be used for decision making in the real-world
3. Machine Learning helps machines learn from data to interact with the real world as humans would (this definition is closer to that of Artifical Intelligence)

1.4 Methods used in Machine Learning

Methods in machine learning can be put in 3 broad categories: Supervised Learning, Unsupervised Learning and Reinforcement Learning. In this course, we focus extensively on the first two, which can be used to solve a majority of business problems that need machine learning.

1.5 Exercises

1. What phenomenon would help us understand why the kid couldn't identify a green apple as an apple and instead mistakenly identify it as a lime?



2. Linear Regression

2.1 Motivating Example

Consider that you want to predict the weight of a person, given their height! How would you do it? Is it generally true that taller a person is, the more they weigh? On an average, you might answer affirmative to this question. However, there are people who are tall and weigh less than average. Conversely, there are people who are short who weight more than average. But on an average, maybe we can derive a relationship between weight and height of a person!

2.2 What is Linear Regression?

In the above example, if the relationship we derive was 'linear' - We get linear regression! Regression is essentially a way to explain an output (weight in this example), as a function of the inputs (height in this example). What's an example of a linear function?

$$f(w) = wx + c$$

Here, f as a function of w is linear in w. w could represent a weighting term, x could be the height of the person and $f(w)$ predicts the height! What if now, instead of just using height to predict the weight, you also use BMI (body mass index) and perhaps resting heart rate (a measure of fitness)? We might get a more accurate fit. Now x is no longer just a scalar, it becomes a vector $\mathbf{x}$ and it represents height, BMI, and resting heart rate. Our output, prediction, is still the weight. So this changes, the linear regression model:

$$f(\mathbf{w}) = \mathbf{w}^T \mathbf{x} + c$$

2.3 Methodology behind Linear Regression

You can fit any line through the data and perhaps it could help explain the relationship between the height and weight of a person. And perhaps, it won't! But the *line of best fit*, would certainly *best* *explain* this relationship between the input and the output. Why? Because it is the line of best fit. Consider Figure 1 below. The red line represents the line of best fit. But look at all the other lines, they may explain the relationship between height and weight, but they certainly don't look like the best fit!

2.4 Delving into the Math behind Linear Regression

Let $\hat{y} = f(\mathbf{w})$ represent the **prediction** of the Linear Regression model. We want, the prediction to be as close to the ground truth y as possible for all possible data points. In other words,

$$\min_{\mathbf{w}\in\mathscr{R}^d} \frac{1}{N}\sum_{i=1}^{N}(y_i - \hat{y}_i)^2$$

also equivalent to,

$$\min_{\mathbf{w}\in\mathscr{R}^d} \frac{1}{N}\sum_{i=1}^{N}(y_i - \mathbf{w}^T\mathbf{x}_i)^2$$

also equivalent to,

$$\min_{\mathbf{w}\in\mathscr{R}^d} \frac{1}{N}||\mathbf{y} - X\mathbf{w}||_2^2$$

where $||\mathbf{z}||_2^2 = \sum_i \mathbf{z}_i^2$ is the *Euclidean Norm* of $\mathbf{z}$.

2.5 Exercises

1. Suppose you have m = 23 training examples with n = 5 features (excluding the additional all-ones feature for the intercept term, which you should add). For the given values of m and n, what are the dimensions of w, X, and y in this equation? The normal equation is

   $$w = (X^TX)^{-1}X^Ty.$$

   (a) w is 5x1, X is 23x5, y is 23x1.
   (b) w is 5x5, X is 23x5, y is 6x1.
   (c) w is 6x6, X is 6x23, y is 6x1.
   (d) w is 6x1, X is 23x6, y is 23x1.
2. Which of the following is the reason for using feature scaling(data normalization)?
   (a) Because the calculation of normal equation will have no matrix inversion problems(i.e, singularity matrix).
   (b) Because solving the normal equation will be more efficient.
   (c) Because when optimizing by gradient descent, the speed of convergence will be faster.
   (d) Because solving the normal equation will be more accurate than without feature scaling.

3. Data Pre-processing & Overfitting

3.1 Motivation

Raw data is something you download from a webpage as a “data set”. **Pre-Processed data** is where a sequence of transformations are applied to the data to get it into shape before its ready to go through a ML model! ML models can’t usually be applied to raw data unless they are already in a good shape for training. Some amount of pre-processing is usually necessary to get the raw data ready for ML training

3.2 Types of pre-processing

1. **Missing Data Imputation:** If the data has missing values for some of the examples of an attribute/feature - It can be filled (e.g. through mean or median imputation) for numeric attributes. There are other methods for data imputation that can also be used depending on the data set and the percentage of missing data in the data set.
2. **Numeric vs Categorical Features:** Some attributes are inherently numerical and can be treated as such. For instance, “square footage” is a numeric attribute when we want to do housing price prediction. Other attributes are inherently categorical. For instance, “location” of a house. There are also attributes that fall in the gray area - And can be *modeled* as a numeric feature or a categorical feature. For instance, “number of bedrooms” can be a categorical feature or can also be modeled as a numeric feature. Treating features as categorical or numeric is referred to as *modeling choice*.
3. **Data Normalization:** This is another important pre-processing that is done on data sets, esp. when attributes/features have a big variability in the magnitude. For instance, population of a neighborhood can run in *tens of thousands*, whereas the number of bedrooms in a home is single digit. In this scenario, each of the features can be normalized to be within a certain range, e.g. 0 and 1 or -1 and 1.
4. **Feature Selection/Feature Pruning:** This can also be a pre-processing step that can be applied before ML models are run. Sometimes, you may have access of *hundreds of thousands* of attributes as possible features for your model. This may heavily increase the complexity of

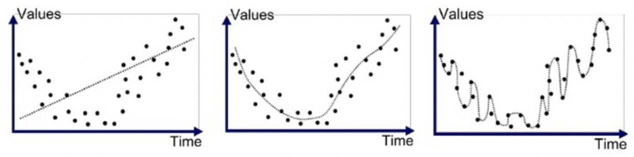


Figure 3.1: Fitting curve to a set of 2D points

your model and may also prove unnecessary. In this case, one can do attribute/feature selection through pruning the features *by frequency of usage* and/or by *relevance/importance/correlation to the prediction*.

3.3 Data Splits

In Machine Learning, data sets are split randomly into *train*, *validation*, and *test*. Why? *train data* gets used for training, *test data* is a proxy for *unseen data* on which we want to evaluate our ML model on. And validation data allows us to fine-tune our trained model so the model doesn't *overfit* and can perform well on unseen data. The typical percentage of splits between train/validation/test are 80:10:10 or 70:10:20

3.4 Overfitting

Overfitting happens for a Machine Learning model, when it performs well on *train data* but has a significantly worse performance on *test data*.

3.5 Preventing Overfitting

There are a few popular strategies that can be used to avoid overfitting:

1. Increase the data set size, N
2. Decerease the feature set size, d through *feature selection* or *feature pruning*
3. *Regularization* using ℓ_1 (sparsity promoting) or ℓ_2 regularization. When ℓ_1 regularization is used for linear regression, it's referred to as *Lasso Regression*. When ℓ_2 regularization is used for linear regression, it's referred to as *Ridge Regression*.
4. ℓ_1 *regularization and feature selection*: Lasso is a regularizer that also does feature selection for free!
5. *Overfitting strategies for Deep learning*: These look a little different, and we will discuss it when we get to the chapter on Deep Learning. *Dropouts* and *Early Stopping* are some strategies to prevent overfitting with deep learning.

3.6 Exercises

1. You are given a set of 2-D points (time, values), and you want to fit a curve to the points such that the curve can capture the relationship between time and values. In the 3 plots below, which of the following best describes how well the curves fit the points?
   (a) 1st plot fits well, 2nd plot overfits, 3rd plot underfits.
   (b) 1st plot overfits, 2nd plot underfits, 3rd plot fits well.

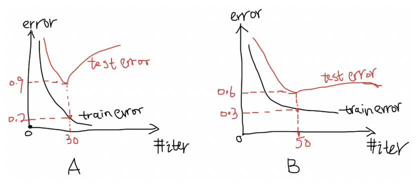
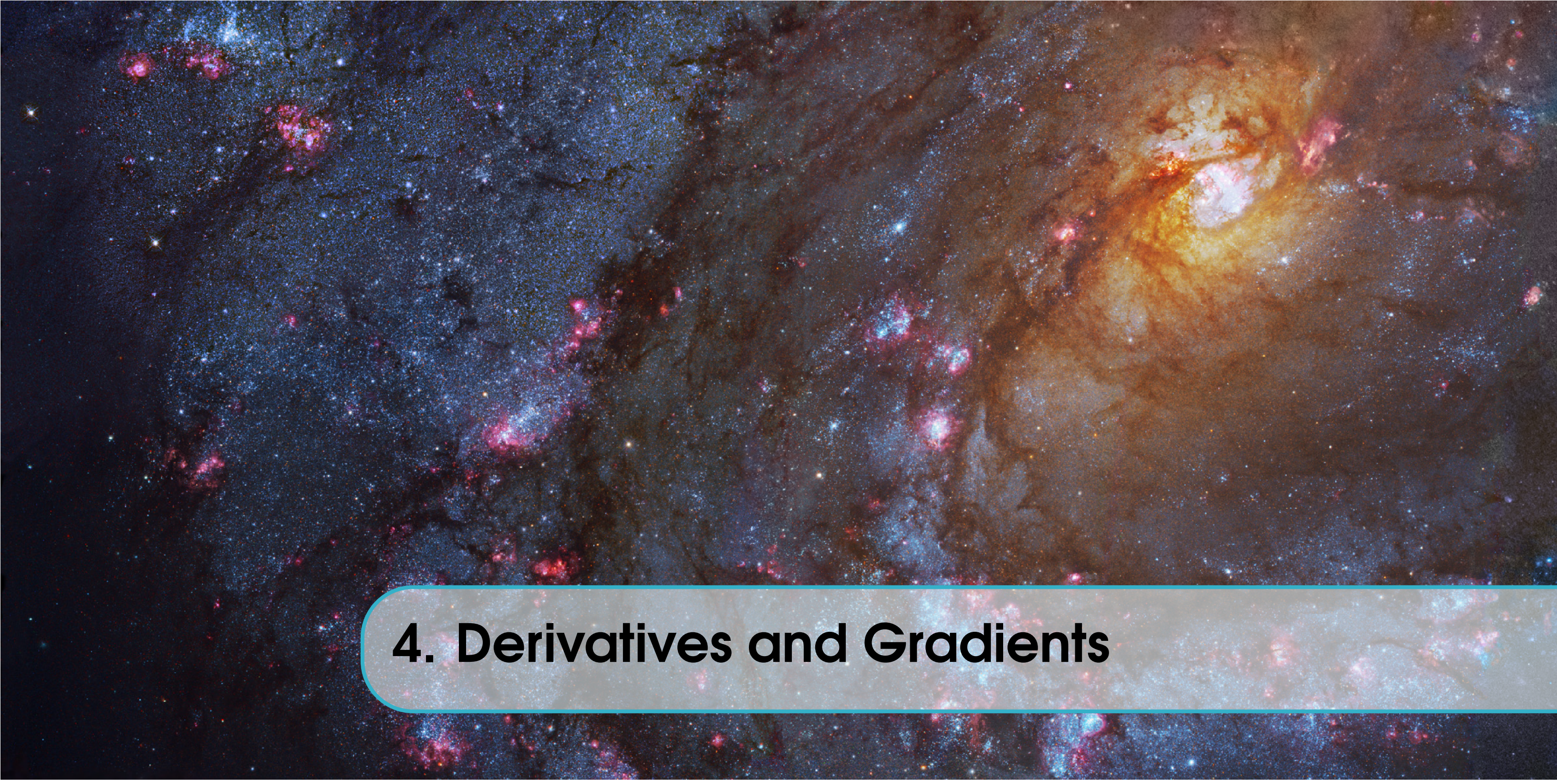


Figure 3.2: Training and Test error curves for two models

   (c) 1st plot underfits, 2nd plot fits well, 3rd plot overfits.
   (d) 1st plot underfits, 2nd plot overfits, 3rd plot fits well.

2. You trained 2 models - A and B on a dataset, the training and test error with respect to number of iterations are shown below, which of the following is the best description for 2 models?
   (a) model A is a better fitting than B because it has lower training error.
   (b) model A is performing better because it takes fewer iterations to converge.
   (c) model B is less overfiting than model A.
   (d) model B is performing better only because the gap between training and test error is smaller.



4. Derivatives and Gradients

4.1 Motivation

Believe it or not, the word, **gradient** is fundamental to most popular machine learning algorithms you will find on the market. Take any library, like *sci-kit learn* or a deep learning framework like *PyTorch*, under the hood - They use *gradients*, as a fundamental to learn from the data! Your favorite object detection method in ML, learns from data through *gradient descent*. So let's learn more on computing gradients.

4.2 What is a gradient?

Gradient is nothing but a collection of partial derivatives of a function. Derivative of a single variable function is something you would have encountered in a basic calculus course. For instance, what is the derivative of $f(w) = w^2$? It's $2w$. Now extrapolate derivative of function of single variable $w \in \mathbb{R}$ to a derivative of a function with respect to a vector, $\mathbf{w} \in \mathbb{R}^d$ and we get the *gradient*.

So what's the gradient of $f(\mathbf{w}) = ||\mathbf{w}||_2^2$? It's $2\mathbf{w}$. The gradient of a function with respect to the vector has the same dimensions as the vector!

4.3 Hiking and Gradients!

Gradients have a cool property: At any given point in space, the gradient of a fucntion tells you the direction to move in, so you increase the function value in the fastest possible way! Imagine, you were to go hiking up a mountain. Usually, hiking trails have switchbacks - These are not the steepest ways to ascend the mountain. However, they are safer! Imagine, you shortcut the switchbacks and went straight up the mountain - Now, that would be the direction in which the gradient would be pointing to!

4.4 Gradient Descent

The main idea with gradients is that the gradient points to the direction of *steepest ascent*. So *negative gradient* points to the direction of steepest descent. So if you were feeling adventurous while hiking down a mountain - You would probably following the path of negative gradient!
Gradient Descent Algorithm is an iterative algorithm that keeps taking steps in the direction of negative gradient until you hit a *local minimum*.

4.5 Exercises

1. Let $f(\mathbf{w}) = \mathbf{1}^T\mathbf{w} + \|\mathbf{w}\|_2^2$. What's the gradient of f with respect to $\mathbf{w}$? Note that $\mathbf{1}$ is a vector of ones.
   (a) $2\mathbf{w}$
   (b) $\mathbf{1}+\mathbf{w}$
   (c) $\mathbf{1}+2\mathbf{w}$
   (d) $\mathbf{1}$
2. Let $f(w) = \|w - w^0\|_2^2$. Find the gradient of f with respect to w. (Hint: Use first principles to derive this. Compute $\frac{\partial}{\partial w_1} f$ to begin with).
3. You run gradient descent for 15 iterations, with learning rate = 0.3 and compute loss function J(w) after each iteration. You find that the value of J(w) **decreases slowly** and is still decreasing after 15 iterations. Based on this, which of the following conclusions seems most plausible?
   (a) is large, try to decrease =0.1 would be better.
   (b) is small, try to increase =1.0 will make the model converge faster.
   (c) is appropriate and no need to change.
   (d) J(w) cannot converge no matter what you pick.
4. Suppose you have a dataset with m = 50 examples and n = 200000 features for each example. You want to use multivariate linear regression to fit the parameters w to our data. Should you prefer gradient descent or the normal equation?
   (a) Normal equation because it is computationally efficient.
   (b) Gradient descent because it's more efficient to compute.
   (c) Normal equation because the number of examples are small.
   (d) Gradient descent because it's more accurate than normal equation.

5. Gradient Descent and its variants

5.1 Motivation

Gradient Descent is one of the fundamental algorithms in *Mathematical Optimization*. Since most of machine learning is based on *minimizing a loss function*, gradient descent is also fundamental to machine learning and *how machines consume and learn from data*. In this chapter, we look at different variations of gradient descent algorithms and also *learning rate schedulers* for these algorithms.

5.2 (Batch) Gradient Descent

Batch Gradient Descent, as the name implies, is an *optimization method* for learning that goes through the entire batch of training data in one pass or iteration. For the algorithm to converge to a local optimum solution, it may however take several iterations probably in the order of hundreds. Let us say we want to minimize $L(w)$ - Loss Function and find the best $\hat{w}$ that does that.

1. **Initialize** $w = w_0$ (random initialization)
2. **Gradient Descent** $w \leftarrow w - lr * \nabla L(w)$
3. **Iterate** Repeat step 2 until w converges, i.e.

$$\|w^{k+1} - w^k\| / \|w^k\| \leq 10^{-3}$$

Here, lr represents the learning rate. $\nabla L(w)$ is the gradient of loss function L

5.3 Stochastic Gradient Descent

Stochastic Gradient Descent (SGD), is a randomized optimization algorithm for optimizing any machine learning *loss function* for which a gradient can be computed. SGD takes a step in the direction of negative gradient, learning from one data point at a time (as compared to Batch GD, which learns from all data points in each step). The *randomization or stochasticity* comes from how the single data point gets picked! Let $L(w) = \sum_{i=1}^{N} L_i(w)$ where L_i is a function of only the *ith* data point (x_i, y_i) and parameter w.

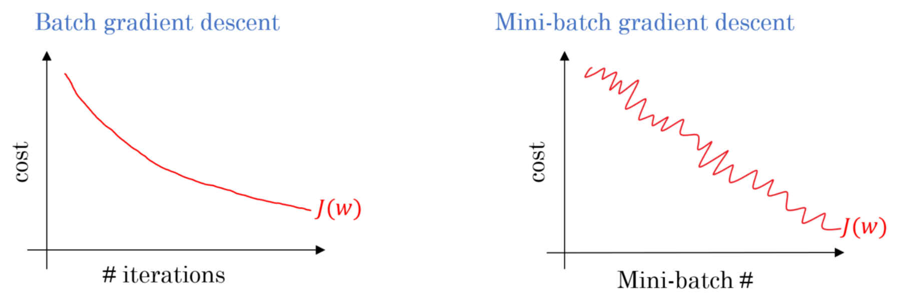


Figure 5.1: GD vs mini-batch SGD

1. **Initialize** w^0 (randomize)
2. Pick index i at random between 1 and N!
3. **Gradient Descent** $w^{k+1} \leftarrow w^k - lr * \nabla L_i(w^k)$
4. **Iterate** Repeat step 2 and 3 until w converges, i.e.

$$\|w^{k+1} - w^k\| / \|w^k\| \leq 10^{-3}$$

5.4 mini-Batch SGD

Let $L(w) = \sum_{i=1}^{N} L_i(w)$ where L_i is a function of only the ith data point (x_i, y_i) and parameter w. Let B be the number of batches and k be the batch size.

1. **Initialize** $w = w_0$ (randomize)
2. Pick a batch of k data points at random between 1 and N: $i_1, i_2, \dots, i_k$!
3. **Gradient Descent** $w^{k+1} \leftarrow w^k - lr * \sum_{j=1}^{k} \nabla_w L_{i_j}(w^k)$
4. **Iterate** Repeat step 2 and 3 until w converges, i.e.

$$\|w^{k+1} - w^k\| / \|w^k\| \leq 10^{-3}$$

5.5 GD vs mini-Batch SGD

How does gradient descent compare against mini-batch SGD? We share a comparison of the covergence behavior in training in Figure 5.1. You can also look at a comparison of GD and mini-batch SGD along different axis in Table 5.1. In summary, mini-batch SGD carries over all the good properties of batch GD while also not having the exhorbitant memory footprint that GD can have with large scale datasets (100k+ data points).

5.6 Learning Rate Schedulers

Learning rate schedulers tell you what the learning rate looks like in each iteration of GD/SGD. Learning rate is one of the most fundamental hyper-parameters in traininig a ML model. Often, learning rates are already optimized under the hood and you may not need to do anything about it when using a library in sci-kit learn or Pytorch. Learning rates effect how much a ML model learns and how quickly it converges to the local optimum. Too small a learning rate and there isn't sufficient progress and too big a learning rate can also make the model learning stall. Some of the

LightBlue Factor	GD	Mini-batch SGD
Data	All per iteration	Mini-batch (usually 128 or 256)
Randomness	Deterministic	Stochastic
Error reduction	Monotonic	Stochastic
Computation	High	Low
Memory big data	Intractable	Tractable
Convergence	Low relative error	Few "passes" on data
Local Minima traps	Yes	No

Table 5.1: GD vs mini-batch SGD along different axis

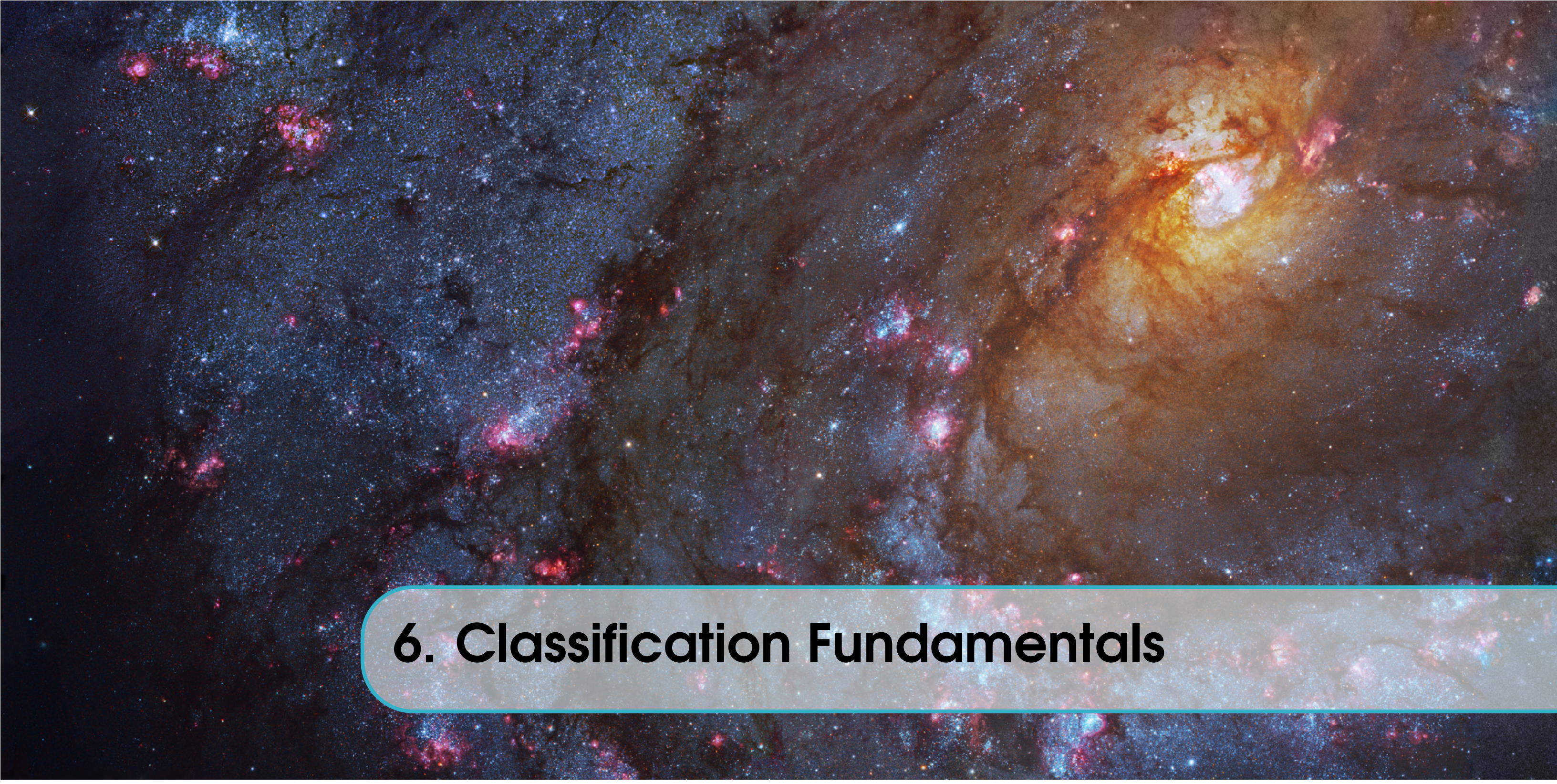
learning rate schedulers apply the same learning rate for every parameter. Others, such as ADAM apply a different learning rate for different parameters - These are called adaptive learning rates.

1. **Constant Learning Rate:** Here, a constant step size is used through out the training. For instance, $s = 1$ where $w \leftarrow w - s \times \nabla l(w)$ and l represents the loss function.
2. **Diminishing Leanring Rate:** A simple diminishing learning rate is $s^t = 1/t$ where t is the iteration number. So the gradient descent update looks like: $w^{t+1} \leftarrow w^t - s^t \nabla l(w^t)$
3. **Learning Rate with Restarts:** This is diminishing learning rate with a restart i.e. set $s^t = s^0$ when $t\%T = 0$, where T is the spacing between restarts. For example: If $T = 20$, after 20 iterations, the learning rate is reset to its initial value. This learning rate scheduler ensures that the learning rate doesn't become too small.
4. **ADAM:** Another popular learning rate scheduler for SGD is ADAM. This one ensures that the learning rate is normalized based on the gradient magnitude.

5.7 Exercises

1. **Gradient Descent Plots for Linear Regression:**
Consider the linear regression model: $y = Xw + e$ where X is the data-matrix, w are the parameters, e is measurement error/noise and y is the target that we want to predict. In the housing prices example, y is the selling price of a home, X is a matrix of features where, each row represents a home. We will consider using random numerical data for this exercise. Let X be generated using a standard normal distribution (`randn function in numpy`) where $X \in \mathbb{R}^{2000\times 1000}$. Generate, $\tilde{w} \in \mathbb{R}^{1000}$ based on a standard normal as well. Let $y = X\tilde{w} + e$, where $e \sim 0.05 * \mathcal{N}(0,1)$. So here e is a 5% random noise added to the data. Your raw data is now (X, y). Split (X, y) into train, validation and test (random splits). Train a linear regression model using, $(X_{train}, y_{train}$. Implement (vanilla) Gradient Descent from scratch for this (do not use sci-kit learn). Plot the loss function $\frac{1}{N}||X_{train}w - y_{train}||_2^2$ as a function of number of iterations. Note that the gradient of $||Xw - y||_2^2$ is $2X^T(Xw - y)$. What *learning rate scheduler* seems to work best for training (definitely try constant learning rate and diminishing learning rates). If you have set things up right, you should the loss function go down with number of iterations of Gradient Descent. How does the validation loss change with/without the use of ℓ_2 regularization? Let $\hat{w}$ be your estimated parameter vector. What's the value: $||\hat{w} - \tilde{w}||_2 / ||\tilde{w}||_2$, and is this relative error small for your model?
2. Notice that the GD algorithm you implemented in the previous exercise converges perhaps in 100 iterations or less. Why do people use mini-batch SGD in practice if GD has good convergence properties?

(a) It’s hard to compute gradients for gradient descent
(b) It’s computationally expensive to train GD as compared to mini-batch SGD
(c) It’s memory inefficient or intractable to load a million data points for GD as compared mini-batch SGD
(d) None of the above



6. Classification Fundamentals

6.1 Binary vs Multi-Class Classification

Classification is a machine learning problem, where we want to categorize objects into distinct classes. For instance, if we have an *self-driving car* that wants to look at objects on the road - It could probably categorize them into cars, trucks, pedestrians or other. As another example, an email server may classify incoming emails into *spam* or *not spam*. Spam classification is an example of *binary classification*, as there are exactly two possible classes. The self-driving car application is an example of *multi-class classification* as there are more than 2 classes to categorize objects into. Classification differs from *Regression* in that, the target type is categorical instead of numeric, as for regression. This small change in the problem definition, changes the algorithms and methods that get used for classification as compared to regression.

6.2 Multi-Class vs Multi-Label Classification

Multi-Class classificaiton refers to classifying a data point into a *single, distinct class*. *Multi-Label classification*, however can attribute more than one class to a given data point. As an example, in data sets that have images of either a human, a dog or a cat - The machine learning problem can be modeled as a multi-class classification problem with 3 classes.

6.3 Linear Models

Linear models in machine learning refer to models where the output can be expressed $w^T x$ where, w is the weight/parameter vector and x are the features. Such a model is called linear, because the output is a linear function of the features. Linear models are some of the simplest models for classification or regression. They also make for good baselines (i.e. starter models) - *Linear Regression for Regression problems* and *Logistic Regression for Classification problems*. Due to being so simple, linear models rarely overfit, unless there is a paucity or scarcity of data as comapred to the number of parameters. However, linear models can underfit, or in other words - They may not be able to leverage the full power of data as compared to non-linear models.

	Predicted Positive	Predicted Negatives
Positives (P)	TP (True Positives)	FN (False Negatives)
Negatives (N)	FP (False Positives)	TN (True Negatives)

Table 6.1: Matrix of Positives and Negatives

6.4 Evaluating Classifiers

Consider the matrix of positives and negatives as in Table 6.1. The rows represent the ground truth - How many positive examples and how many negative examples exist in the test set. The columns represent the predictions - How many examples were *predicted positive* and how many were *predicted negative*. The interplay between the rows and columns leads to *true positives* (TP), *false negatives* (FN), *false positives* (FP), and *true negatives* (TN).
Accuracy is an easy way to measure the performance of a classifier - How many predictions did the model get right divided by the total number of examples. In Table 6.1, this refers to the sum of the diagonals divide by the total number of examples, i.e $Accuracy = \frac{TP+TN}{P+N}$. However, *accuracy* can be a wrong measure when there is class imbalance.

1. **Precision (Pr)** = TP/(TP + FP)

2. **Recall (R)** = TP/(TP + FN) = TP/P

3. **F1-score** = $\frac{2 \times Pr \times R}{Pr+R}$

4. **Accuracy (Acc)** = $(TP+TN)/(P+N)$

6.5 Exercises

1. **Confusion Matrix:** Let's say we computed a Confusion Matrix for a Spam Classifier on an imbalanced data set (more non-spam emails or negatives than spam emails or positives) and we obtained:

	Predicted Positive	Predicted Negatives
Positives (P)	50	100
Negatives (R)	250	500

Then, **Accuracy**, **Pr**, **R** and **F1** are as follows:

(a) $61\%, 0.16, 0.33, 0.22$
(b) $51\%, 0.33, 0.16, 0.22$
(c) $51\%, 0.16, 0.33, 0.22$
(d) $61\%, 0.33, 0.16, 0.22$

2. **Multi-Class or Multi-Label?** A social media website wants to automatically tag all the relevant people on every photo that users post to their wall. This is a use-case for which one of the following machine learning problems:

   (a) Multi-class classification
   (b) Multi-label classification
   (c) Linear Regression
   (d) Non-Linear Regression
3. **Class Imbalance:** Which of the following strategies will *not* help handle the issues related to class imbalance in datasets:
   (a) Up-sampling the minority class
   (b) Down-sampling the majority class
   (c) Down-sampling both the majority and minority class
   (d) Up-sampling the majority class
   (e) Using accuracy as a metric for classification