1. Intro **4**

2. Linear Regression with One Variable **4**
2.1. Notation 4
2.2. Cost Function 4
2.3. Gradient Descent 5

3. Linear Regression with Multiple Variables **6**
3.1. Gradient Descent 6
3.1.1. Feature Scaling 6
3.1.2. Debugging Gradient Descent 7
3.1.3. Feature Engineering 7
3.2. Polynomial Regression 7
3.3. Normal Equation 7

4. Classification - Logistic Regression **8**
4.1. Linear decision boundary model 8
4.2. Non-linear decision boundary model 8
4.3. Cost Function 8
4.4. Loss Function 9
4.5. Simplified Loss Function 9
4.6. Gradient Descent 10

5. Regularization **10**
5.1. Cost Function with Regularization for Linear Regression 10
5.2. Gradient Descent with Regularization 10
5.3. Gradient Descent of Logistic Regression with Regularization 11

6. Neural Networks **11**
6.1. Vectorization 11
6.2. Activation Functions 12
6.3. TensorFlow 13
6.3.1. Round-off errors 13
6.3.2. Multi-label Classification 13
6.3.3. Advanced Optimization 13
6.3.4. Convolutional Layer 13

7. Tips on model building **13**
7.1. Model selection 14
7.2. Bias and Variance 14
7.2.1. Polynomial and Bias/Variance 14
7.2.2. Regularization and Bias/Variance 15
7.2.3. Neural Network and Bias/Variance 15
7.3. Error metrics for skewed datasets 15

8. Decision Tree Model **16**
8.1. Decisions 16
8.2. Decision Tree Learning Process 16
8.3. Regression Tree 17
8.4. Decision Trees vs Neural Networks 17

9. Unsupervised Learning **18**
9.1. Clustering 18
9.1.1. K-means Algorithm 18
9.1.2. Cost Function 18
9.1.3. Choosing number of clusters 18
9.2. Anomaly Detection 19
9.2.1. Anomaly Detection Algorithm with Gaussian (Normal) Distribution 19
9.2.2. Anomaly Detection vs. Supervised Learning 19
9.2.3. Error Analysis for Anomaly Detection 19
9.3. Recommender Systems 20
9.3.1. Cost Function 20
9.3.2. Cost Function when features are unknown 21
9.3.3. Collaborative Filtering 21
9.3.4. Gradient Descent 21
9.3.5. TensorFlow Implementation of Collaborative Filtering 22
9.3.6. Limitations of Collaborative Filtering 22
9.3.7. Content-based filtering neural network 22

10. Reinforcement Learning **23**
10.1. Key Factors 23
10.2. Bellman Equation 24
10.3. Markov Decision Process (MDP) 24
10.4. Algorithm 24
10.4.1. Deep Reinforcement Learning Architecture 25
10.4.2. Greedy policy 25
10.5. Limitations of Reinforcement Learning 25

11. Machine Learning Project Lifecycle **25**
11.1. Scoping 25
11.1.1. Scoping process 26
11.1.2. Ethical consideration 26
11.1.3. Milestones & Resourcing 26
11.2. Data 26
11.2.1. Define Data 26
11.2.2. Establish Baseline 27
11.2.3. Label and Organize Data 27
11.2.4. Meta-data, Data Provenance and Lineage 27
11.2.5. Data Augmentation 27
11.2.6. From Big Data to Good Data 28

11.3. Modeling 28
11.3.1. Iterative Loop of ML Development 28
11.3.2. Key Challenges 28
11.3.3. Tips For Getting Started on ML Project 28
11.3.4. Error Analysis 28
11.3.5. Auditing Framework 29
11.3.6. Experiment Tracking 29
11.4. Deployment 29
11.4.1. Key Challenges 29
11.4.2. Key Ideas 30
11.4.3. Common Deployment Patterns 30
11.4.4. Monitoring 30

1. Intro

Machine learning is the field of study that gives computers the ability to learn without being explicitly programmed. A computer program is said to learn from experience E with respect to some task T and some performance measure P, if its performance on T, as measured by P, improves with experience E.
In general, any machine learning problem can be assigned to one of two broad classifications:

- **Supervised learning** - we are given a dataset and already know what our correct output should look like, having the idea that there is a relationship between the input and the output. Supervised learning problems are categorized into "regression" and "classification" problems:
    - In a **regression** problem, we are trying to predict results within a continuous output, meaning that we are trying to map input variables to some continuous function.
    - In a **classification** problem, we are instead trying to predict results in a discrete output. In other words, we are trying to map input variables into discrete categories.
- **Unsupervised learning** - allows us to approach problems with little or no idea what our results should look like. We can derive structure from data where we don't necessarily know the effect of the variables. We can derive this structure by clustering the data based on relationships among the variables in the data. With unsupervised learning, there is no feedback based on the prediction results.

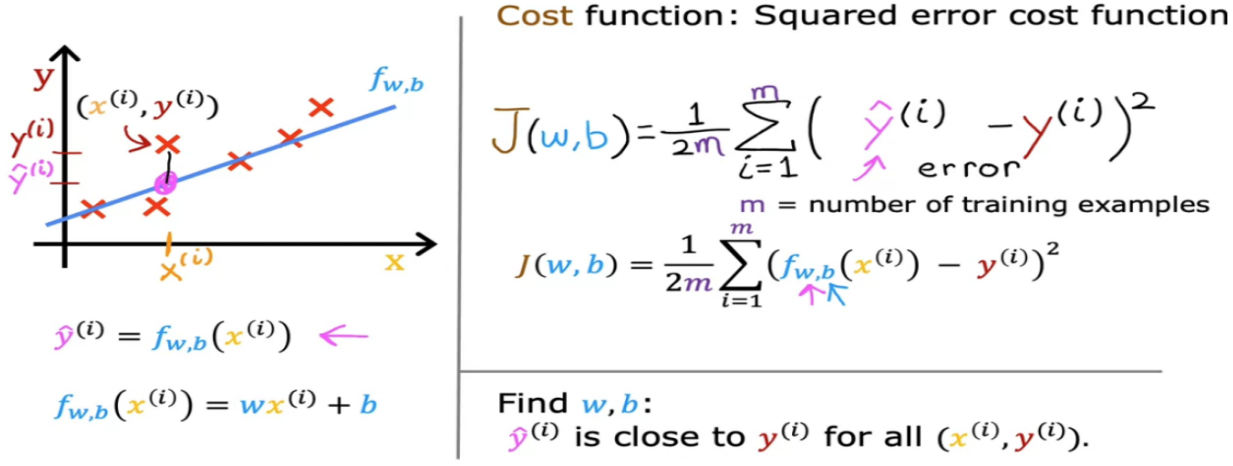
2. Linear Regression with One Variable

2.1. Notation

x = to denote the "input" variables or input features,
y = to denote the "output" or target variable
m = number of training examples
$(x^{(i)}, y^{(i)})$ ith training example
$\hat{y}$ is the estimate or the prediction for y.
$f(x) = wx + b$ - univariate linear regression model
w = weight
b = bias
w, b sometimes referred to as coefficients, weights

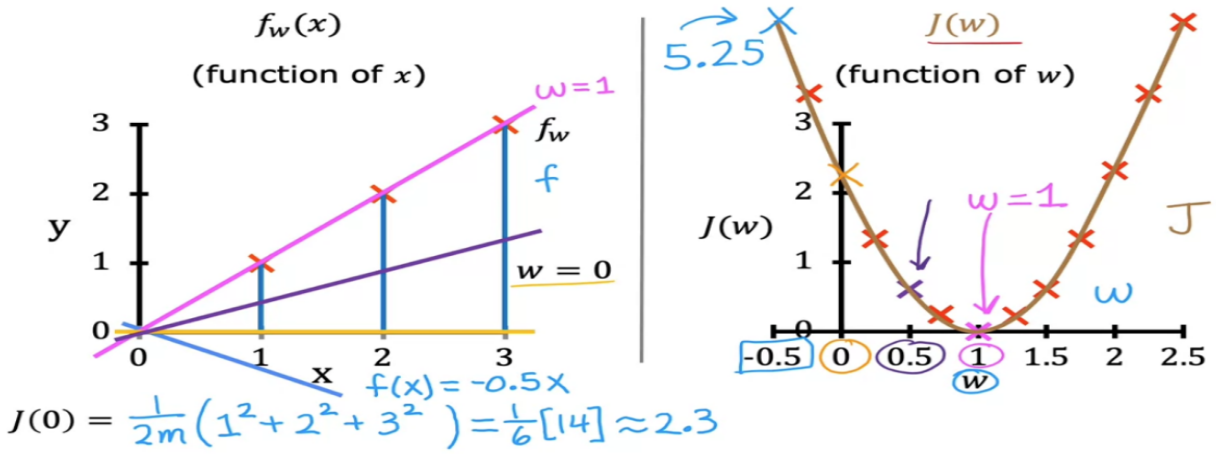
2.2. Cost Function

We can measure the accuracy of our hypothesis function by using a **Cost Function** (Squared error function or Mean squared error). Squared error measures the difference between the model's predictions and the actual true values for y.



2.3. Gradient Descent

Gradient descent will minimize the cost function J of w, b. It is also used in minimizing cost in the neural networks however its cost is not a parabola shaped like square errors. Hence it can have multiple minima and maxima in a neural network. Squared error cost can only have one minimum and therefore it's called convex function.



Gradient Descent Algorithm (repeat until convergence)

$w = w - \alpha \frac{\partial}{\partial w} J(w,b)$

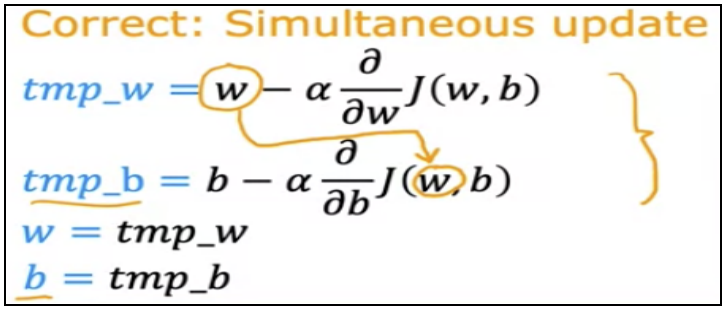
$b = b - \alpha \frac{\partial}{\partial b} J(w,b)$

Notation

α = learning rate

j represents the feature index number.

$\frac{\partial}{\partial \theta_1} J(\theta_0, \theta_1)$ derivative of J



At each iteration j should simultaneously update the parameters $\theta_1, \theta_2, \theta_3, \theta_4 ..., \theta_n$ Updating a specific parameter prior to calculating another one on the $j(th)$ iteration would yield a wrong implementation. When the slope of the random point in J(w,b) is negative, the value of w increases, and when it is positive the value decreases.

On a side note, we should adjust our parameter α to ensure that the gradient descent algorithm converges in a reasonable time. Failure to converge or too much time to obtain the minimum value implies that our step size is wrong. The intuition behind the convergence is that $\frac{d}{d\theta_1} J(\theta_1)$ approaches 0 as we approach the bottom of our convex function. At the minimum, the derivative will always be 0 and thus we get: $\theta_1 = \theta_1 - \alpha * 0$

Gradient Descent Algorithm of Univariate Linear Regression

$w = w - \alpha \frac{\partial}{\partial w} J(w,b)$

$\frac{\partial}{\partial w} J(w,b) = \frac{1}{m} \sum_{i=1}^{m} (f_{w,b}(x^{(i)}) - y^{(i)}) x^{(i)}$

$w = w - \alpha \frac{1}{m} \sum_{i=1}^{m} (f_{w,b}(x^{(i)}) - y^{(i)}) x^{(i)}$

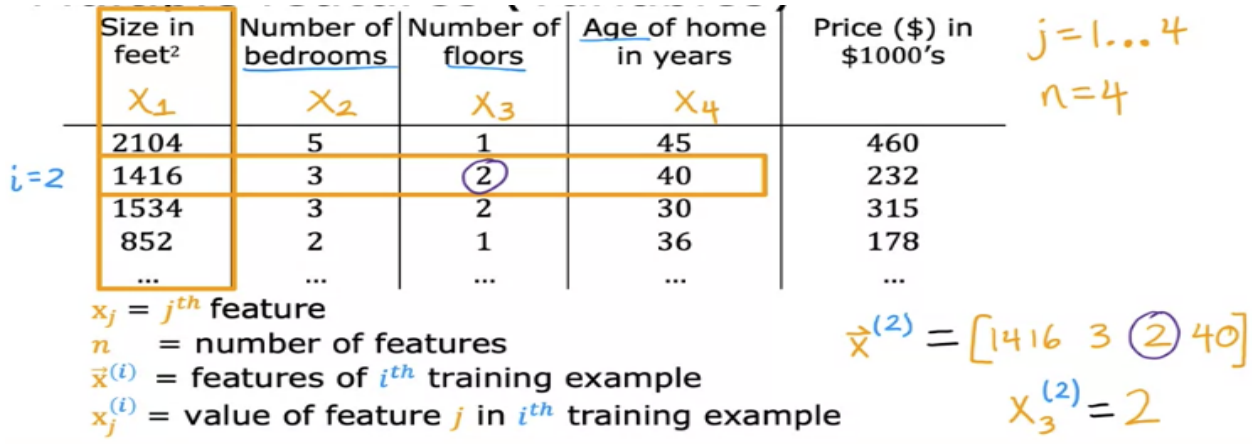
$b = b - \alpha \frac{\partial}{\partial b} J(w,b)$

$\frac{\partial}{\partial b} J(w,b) = \frac{1}{m} \sum_{i=1}^{m} (f_{w,b}(x^{(i)}) - y^{(i)})$

$b = b - \alpha \frac{1}{m} \sum_{i=1}^{m} (f_{w,b}(x^{(i)}) - y^{(i)})$

3. Linear Regression with Multiple Variables

Linear regression with multiple variables is also known as "multivariate linear regression".



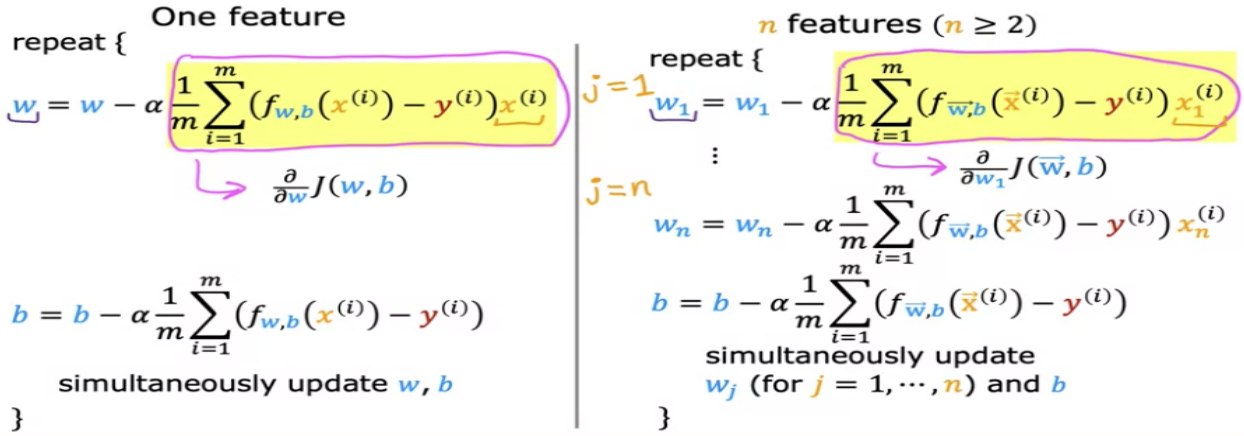
The multivariable form of the hypothesis function accommodating these multiple features are as follows:

$f_{w,b}(x) = w_1 x_1 + w_2 x_2 + w_3 x_3 + \ldots + w_n x_n + b$

Vectorization will both make your code shorter and also make it run much more efficiently.

f = np.dot(w,x) + b

3.1. Gradient Descent



3.1.1. Feature Scaling

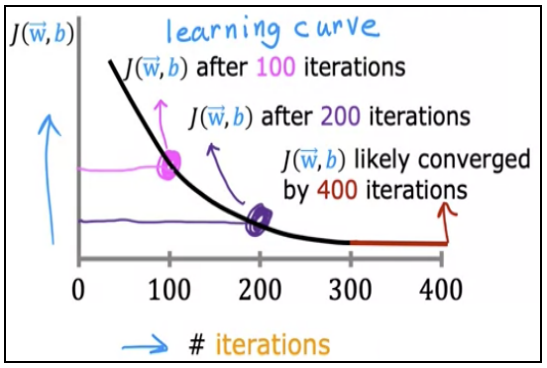
Techniques to find gradient descent faster. Rule of thumb ideal range $-1 \leq x_1 \leq 1$ but max $-3 \leq x_1 \leq 3$

- **Feature scaling** involves dividing the input values by max of the input variable, resulting $-1 < x_1 \leq 1$
  - E.g min=0,max=2000 $x_1 = \frac{size}{2000}$
- **Mean normalization** involves subtracting the average value for an input variable and dividing it by the range of input variables (i.e. the maximum value minus the minimum value). All features are centered around 0, can be negative and positive values $-1 \leq x_1 \leq 1$
  - E.g min=0,max=2000 $x_1 = \frac{size-\mu}{2000-0}$
- **Z-score normalization** just like mean normalization just divides by standard deviation not range.
  - E.g min=0,max=2000 $x_1 = \frac{size-\mu}{\sigma_1}$

3.1.2. Debugging Gradient Descent

Learning curve chart

Make a plot with a number of iterations on the x-axis and J(θ)/J(w,b) on the y-axis. J(w,b) should decrease after every iteration. The number of iterations to reach convergence varies. If the learning rate α is sufficiently small, then J(θ) will decrease on every iteration.



Values of α to try:

0.001 -> 0.01 -> 0.1 -> 1

Then multiply by 3

0.003 -> 0.03 -> 0.3 -> 3

Automatic convergence test

Declare convergence if J(θ) decreases by less than ε in one iteration, where ε is some small value such as 10^{-3}. However, in practice, it's difficult to choose this threshold value.

To summarize:

If α is too small: slow convergence.

If α is too large: may not decrease on every iteration and thus may not converge.

3.1.3. Feature Engineering

Feature engineering is using intuition to design new features, by transforming or combining original features. We can combine multiple features into one. E.g.: Let's assume we have Frontage (x_1) and Depth (x_2) of the house. By combining x_1 and x_2 we can create a new variable Area x_3 and improve our model.

3.2. Polynomial Regression

Polynomial regression will let you fit curves, non-linear functions, to your data. We can change the behavior of the curve of our hypothesis function by making it a quadratic, cubic, square root function, or any other form.

For example, if our hypothesis function is $f_{w,b}(x) = w_1 x + b$ then we can create additional features based on x_1 to get

The quadratic function - $f_{w,b}(x) = w_1 x + w_2 x_1^2 + b$

The cubic function - $f_{w,b}(x) = w_1 x + w_2 x_1^2 + w_3 x_1^3 + b$

The square root function - $f_{w,b}(x) = w_1 x + w_2 \sqrt{x} + b$

3.3. Normal Equation

Normal Equation is an alternative to gradient descent. The minimization explicitly and without resorting to an iterative algorithm. In the normal equation method, we will minimize J by explicitly taking its derivatives with respect to the θj 's, and setting them to zero. This allows us to find the optimum theta without iteration. There is no need to do feature scaling with the normal equation. The normal equation formula is $\theta = (X^T X)^{-1} X^T y$

Gradient Descent	Normal Equation
Need to choose alpha	No need to choose alpha
Needs many iterations	No need to iterate
$O(kn^2)$ complexity	$O(n^3)$ need to calculate inverse of $X^T X$
Works well when n is large	Slow if n is very large

4. Classification - Logistic Regression

Binary classification is a classification problem where there are only two possible outputs. Usually labeled as False or True, 0 or 1. Category is also known as class e.g negative class or positive class. Logistic regression fits a S shaped curve to the dataset. The output predicts the probability of positive class.

$$g(z) = \frac{1}{1+e^{-z}}$$

$$f_{w,b}(x) = g(wx + b) = \frac{1}{1 + e^{-(wx+b)}}$$

The decision boundary is the line where you're just almost neutral about whether y is 0 or y is 1.

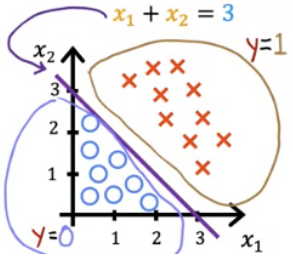
4.1. Linear decision boundary model

$$f_{\vec{w},b}(\vec{x}) = g(z) = g(\underset{1}{w_1}x_1 + \underset{1}{w_2}x_2 + \underset{-3}{b})$$

Decision boundary $z = \vec{w} \cdot \vec{x} + b = 0$

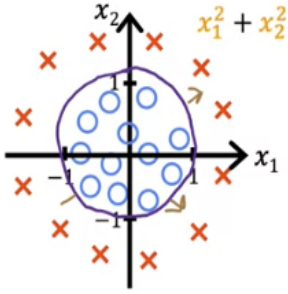
$z = x_1 + x_2 - 3 = 0$

$x_1 + x_2 = 3$



$$f_{w,b}(x) = g(w_1x_1 + w_2x_2 + b) = \frac{1}{1 + e^{-(w_1x_1+w_2x_2+b)}}$$

4.2. Non-linear decision boundary model



$x_1^2 + x_2^2 \geq 1$

$$f_{\vec{w},b}(\vec{x}) = g(z) = g(\overbrace{\underset{1}{w_1}x_1^2 + \underset{1}{w_2}x_2^2 + \underset{-1}{b}}^{z})$$

decision boundary $z = x_1^2 + x_2^2 - 1 = 0$

$x_1^2 + x_2^2 = 1$

$$f_{w,b}(x) = g(w_1x_1^2 + w_2x_2^2 + b) = \frac{1}{1 + e^{-(w_1x_1^2+w_2x_2^2+b)}}$$

4.3. Cost Function

Logistic regression has non-convex cost and squared error cost will not work. Therefore we need a loss function.

Notation

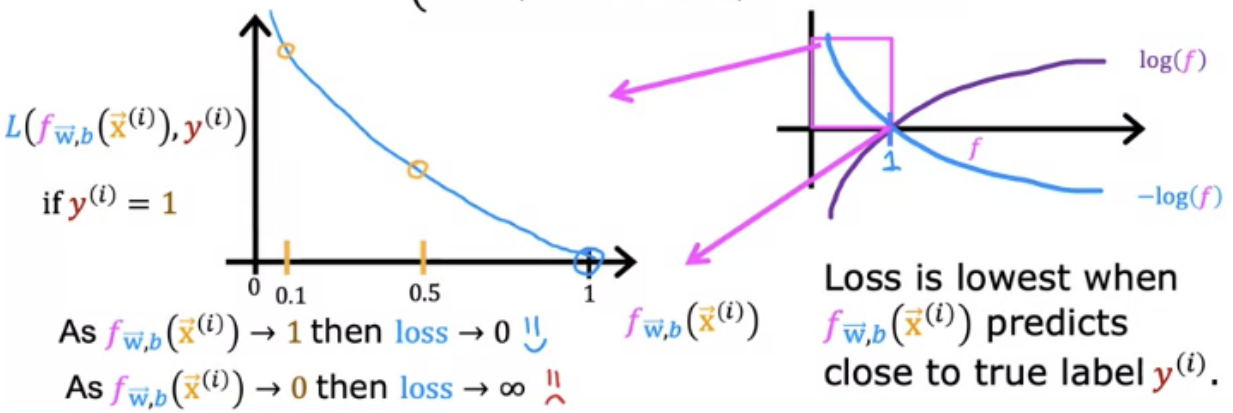
L - loss

$$J(w,b) = \frac{1}{m}\sum_{i=1}^{m}[(L(f_{w,b}(x^{(i)}), y^{(i)})]$$

$$J(w,b) = -\frac{1}{m}\sum_{i=1}^{m}[y^{(i)}log(f_{w,b}(x^{(i)})) + (1 - y^{(i)})log(1 - f_{w,b}(x^{(i)}))]$$ **Explained below**

4.4. Loss Function

$$L\left(f_{\vec{w},b}\left(\vec{x}^{(i)}\right), y^{(i)}\right) = \begin{cases} -\log\left(f_{\vec{w},b}\left(\vec{x}^{(i)}\right)\right) & \text{if } y^{(i)} = 1 \\ -\log\left(1 - f_{\vec{w},b}\left(\vec{x}^{(i)}\right)\right) & \text{if } y^{(i)} = 0 \end{cases}$$



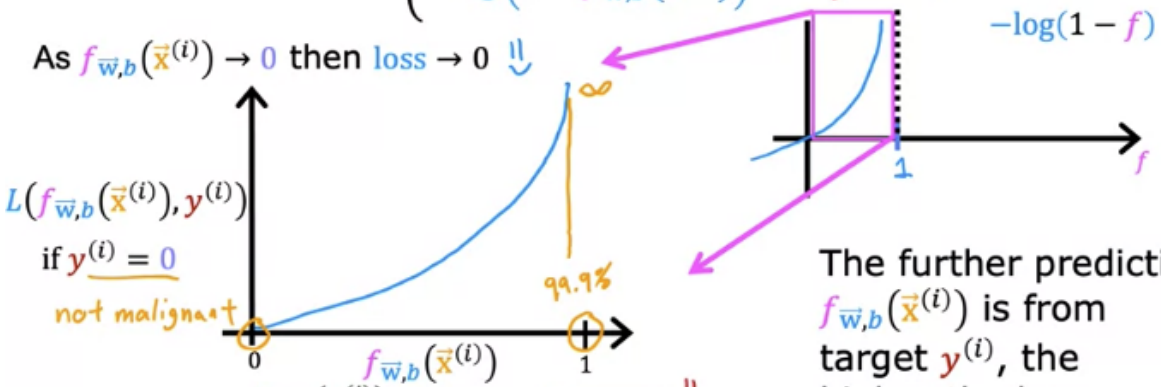
As $f_{\vec{w},b}\left(\vec{x}^{(i)}\right) \to 1$ then loss $\to 0$

As $f_{\vec{w},b}\left(\vec{x}^{(i)}\right) \to 0$ then loss $\to \infty$

Loss is lowest when $f_{\vec{w},b}\left(\vec{x}^{(i)}\right)$ predicts close to true label $y^{(i)}$.

$$L\left(f_{\vec{w},b}\left(\vec{x}^{(i)}\right), y^{(i)}\right) = \begin{cases} -\log\left(f_{\vec{w},b}\left(\vec{x}^{(i)}\right)\right) & \text{if } y^{(i)} = 1 \\ -\log\left(1 - f_{\vec{w},b}\left(\vec{x}^{(i)}\right)\right) & \text{if } y^{(i)} = 0 \end{cases}$$

As $f_{\vec{w},b}\left(\vec{x}^{(i)}\right) \to 0$ then loss $\to 0$



As $f_{\vec{w},b}\left(\vec{x}^{(i)}\right) \to 1$ then loss $\to \infty$

The further prediction $f_{\vec{w},b}\left(\vec{x}^{(i)}\right)$ is from target $y^{(i)}$, the higher the loss.

4.5. Simplified Loss Function

$$L\left(f_{\vec{w},b}\left(\vec{x}^{(i)}\right), y^{(i)}\right) = \begin{cases} -\log\left(f_{\vec{w},b}\left(\vec{x}^{(i)}\right)\right) & \text{if } y^{(i)} = 1 \\ -\log\left(1 - f_{\vec{w},b}\left(\vec{x}^{(i)}\right)\right) & \text{if } y^{(i)} = 0 \end{cases}$$

$$L\left(f_{\vec{w},b}\left(\vec{x}^{(i)}\right), y^{(i)}\right) = -\underbrace{y^{(i)}}_{0}\log\left(f_{\vec{w},b}\left(\vec{x}^{(i)}\right)\right) - \underbrace{(1 - y^{(i)})}_{(1-0)}\log\left(1 - f_{\vec{w},b}\left(\vec{x}^{(i)}\right)\right)$$

if $y^{(i)} = 1$:

$$L\left(f_{\vec{w},b}\left(\vec{x}^{(i)}\right), y^{(i)}\right) = -1\ \log\left(f(\vec{x})\right)$$

if $y^{(i)} = 0$:

$$L\left(f_{\vec{w},b}\left(\vec{x}^{(i)}\right), y^{(i)}\right) = \underbrace{-\ (1-0)\log\left(1 - f(\vec{x})\right)}$$

4.6. Gradient Descent

repeat {

looks like linear regression!

$$w_j = w_j - \alpha \left[\frac{1}{m} \sum_{i=1}^{m} (f_{\vec{w},b}(\vec{x}^{(i)}) - y^{(i)}) x_j^{(i)} \right]$$

$$b = b - \alpha \left[\frac{1}{m} \sum_{i=1}^{m} (f_{\vec{w},b}(\vec{x}^{(i)}) - y^{(i)}) \right]$$

} simultaneous updates

Same concepts:

- Monitor gradient descent (learning curve)
- Vectorized implementation
- Feature scaling

Linear regression $f_{\vec{w},b}(\vec{x}) = \vec{w} \cdot \vec{x} + b$

Logistic regression $f_{\vec{w},b}(\vec{x}) = \frac{1}{1 + e^{(-\vec{w} \cdot \vec{x} + b)}}$

5. Regularization

Regularization encourages the learning algorithm to shrink the values of the parameters without necessarily demanding that the parameter is set to exactly 0. It lets you keep all of your features, but prevents the features from having an overly large effect. If you have a lot of features, you may not know which are the most important features and which ones to penalize. So the way regularization is typically implemented is to penalize all of the features (all the Wj parameters) that will usually result in fitting a smoother, simpler, less weekly function that's less prone to overfitting.

5.1. Cost Function with Regularization for Linear Regression

$$\min_{\vec{w},b} J(\vec{w}, b) = \min_{\vec{w},b} \left[\overbrace{\frac{1}{2m} \sum_{i=1}^{m} (f_{\vec{w},b}(\vec{x}^{(i)}) - y^{(i)})^2}^{\text{mean squared error}} + \overbrace{\frac{\lambda}{2m} \sum_{j=1}^{n} w_j^2}^{\text{regularization term}} \right]$$

fit data

Keep w_j small

$\lambda = 0$

λ balances both goals

5.2. Gradient Descent with Regularization

repeat {

$$w_j = w_j - \alpha \left[\frac{1}{m} \sum_{i=1}^{m} \left[(f_{\vec{w},b}(\vec{x}^{(i)}) - y^{(i)}) x_j^{(i)} \right] + \frac{\lambda}{m} w_j \right]$$

$$b = b - \alpha \frac{1}{m} \sum_{i=1}^{m} (f_{\vec{w},b}(\vec{x}^{(i)}) - y^{(i)})$$

} simultaneous update $j = 1 \ldots n$

$$w_j = \underbrace{1 w_j - \alpha \frac{\lambda}{m} w_j}_{w_j \left(1 - \alpha \frac{\lambda}{m}\right)} - \underbrace{\alpha \frac{1}{m} \sum_{i=1}^{m} (f_{w,b}(\vec{x}^{(i)}) - y^{(i)}) x_j^{(i)}}_{\text{usual update}}$$

shrink w_j

$\alpha \frac{\lambda}{m}$

$0.01 \frac{1}{50} = 0.0002$

$w_j (1 - 0.0002)$

0.9998

5.3. Gradient Descent of Logistic Regression with Regularization

$$J(\vec{w},b) = -\frac{1}{m}\sum_{i=1}^{m}\left[y^{(i)}\log\left(f_{\vec{w},b}(\vec{x}^{(i)})\right) + (1-y^{(i)})\log\left(1-f_{\vec{w},b}(\vec{x}^{(i)})\right)\right] + \frac{\lambda}{2m}\sum_{j=1}^{n} w_j^2$$

$\min_{\vec{w},b}$

Gradient descent

repeat {

$w_j = w_j - \alpha \frac{\partial}{\partial w_j} J(\vec{w},b)$ $\quad j = 1 \ldots n$

$b = b - \alpha \frac{\partial}{\partial b} J(\vec{w},b)$

}

Looks same as for linear regression!

$$= \frac{1}{m}\sum_{i=1}^{m}\left(f_{\vec{w},b}(\vec{x}^{(i)}) - y^{(i)}\right) x_j^{(i)} + \frac{\lambda}{m} w_j$$

logistic regression

$$= \frac{1}{m}\sum_{i=1}^{m}\left(f_{\vec{w},b}(\vec{x}^{(i)}) - y^{(i)}\right)$$

don't have to regularize

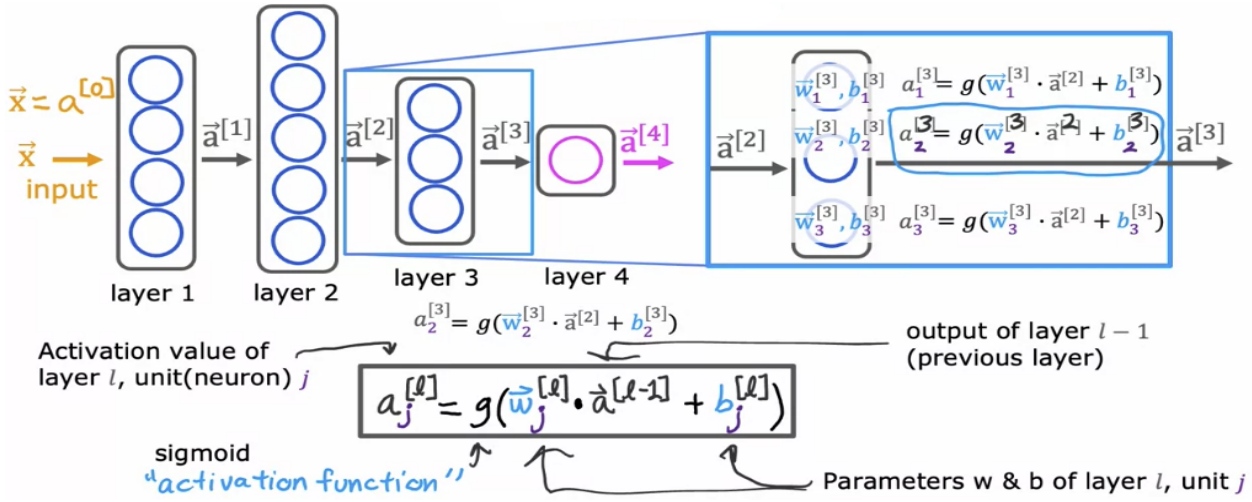
6. Neural Networks

The original motivation was to write software that could mimic how the human brain or how the biological brain learns and thinks. The artificial neural network uses a very simplified Mathematical model of what a biological neuron does. The neural network does feature engineering and makes the learning problem easier for itself. Multilayer perceptron is a neural network with multiple layers.

Notaion

$a = f(x)$ = activation

$a^{[l]}$ = activation of layer l



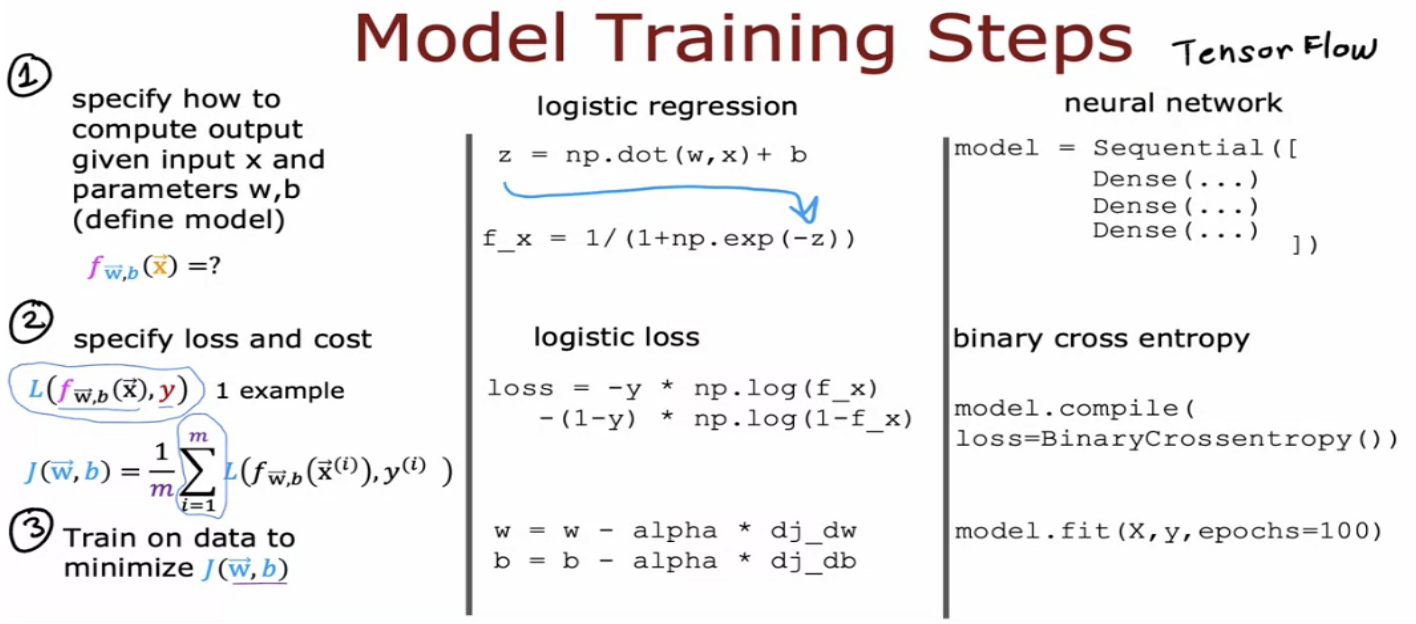
Forward propagation neural network algorithm making computations left to right. (usually predicting)

Back propagation neural network algorithm making computations right to left. (usually training the model)

6.1. Vectorization

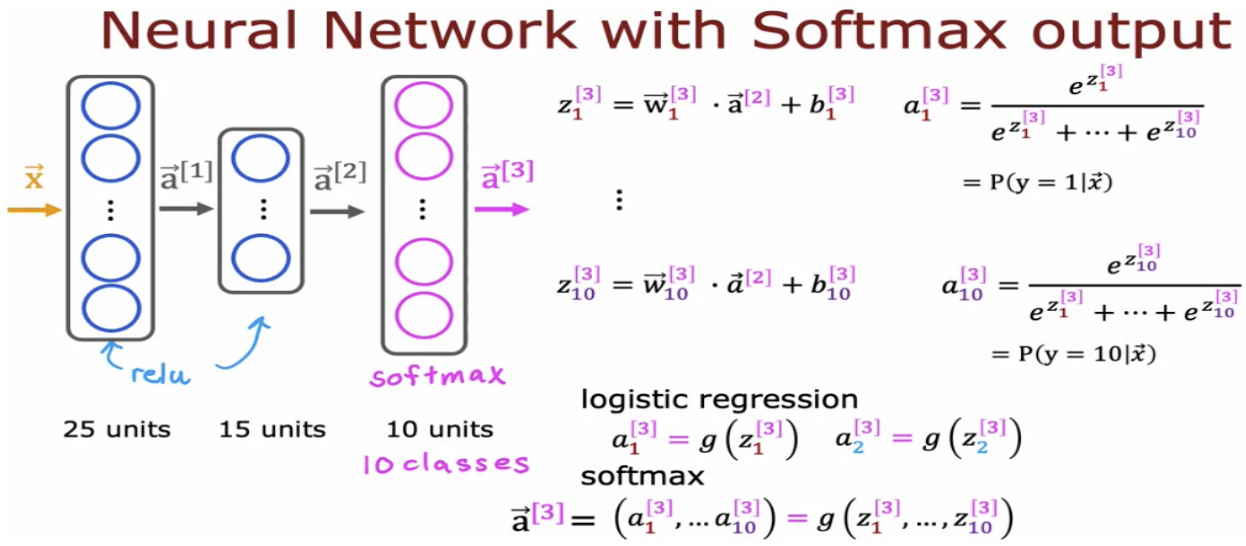
You can improve the performance of the model learning by carrying out matrix multiplication instead of for loop.

```
def dense(A_in,W,b):
    Z = np.matmul(A_in,W) + B
    A_out = g(Z)
    return A_out
```



6.2. Activation Functions

	Sigmoid	**Linear (No activation)**	**ReLU (Rectified Linear Unit)**	**Softmax**
Function	g(z) = 1/1+e**-z where z = wx+b	g(z) = z where z = wx+b	g(z) = max(0,z)	$g(z) = \frac{e^{z_j}}{\sum_{k=1}^{N} e^{z_k}}$
Use	Use for binary classification	Use for regression where **y** can be negative and for output neuron only	Use for hidden layers / use for regression where output neuron **y** can't be negative	Use for output layer of multiclass regression
Comments			z can be either 0 or positive number	Probability of jth



6.3. TensorFlow

6.3.1. Round-off errors

It turns out that while the way we have been computing the cost function for softmax is correct, there's a different way of formulating it that reduces these numerical round-off errors, leading to more accurate computations within TensorFlow.
Add (from_logits=True)
If one of the z's really small than e to negative small number becomes very, very small or if one of the z's is a very large number, then e to the z can become a very large number and by rearranging terms, TensorFlow can avoid some of these very small or very large numbers and therefore come up with more accurate computation for the loss function.

6.3.2. Multi-label Classification

A classification problem with multiple outputs is called a multi-label classification problem, which is where each input can have multiple labels. E.g. image X are three different labels corresponding to whether or not there are any cars, buses, or pedestrians in the image. Image might contain all of them or some of the options. You can implement this model with an output layer with sigmoid activation for each neuron.

6.3.3. Advanced Optimization

Depending on how gradient descent is proceeding, sometimes you wish you had a bigger learning rate Alpha, and sometimes you wish you had a smaller learning rate Alpha. The Adam algorithm can adjust the learning rate automatically. Adam stands for Adaptive Moment Estimation or A-D-A-M. Adam has different rates of learning for each of the activation functions.

6.3.4. Convolutional Layer

Dense layer neuron gets its inputs from all the activations of the previous layer. Convolutional neurons can only look at a certain part of the data. It speeds up computation, less training data and less prone to overfitting.

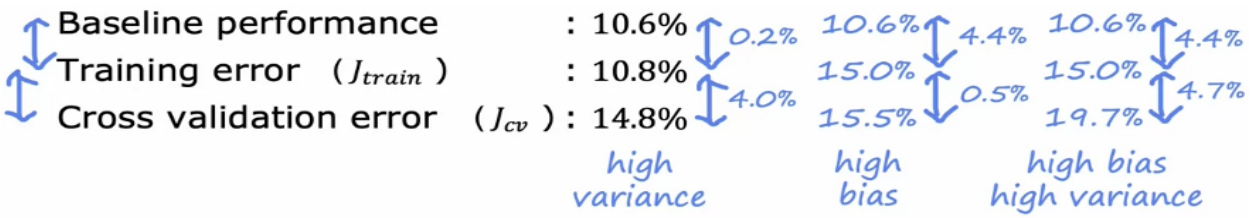
7. Tips on model building

You have implemented regularized linear regression but it makes unacceptably large error predictions.
Next steps to fix the issue:

- Get more training data (fixes high variance)
- Try smaller set of features (fixes high variance)
- Try getting additional features (fixes high bias)
- Try adding polynomial features (fixes high bias)
- Try decreasing lambda (fixes high bias)
- Try increasing lambda (fixes high variance)

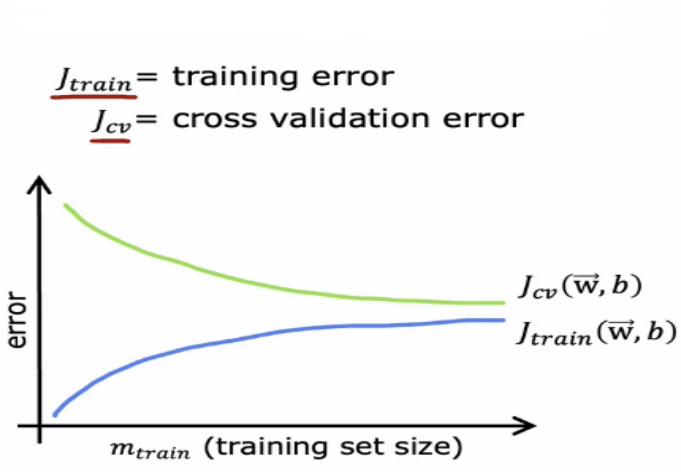
Establishing a baseline level of performance (error)

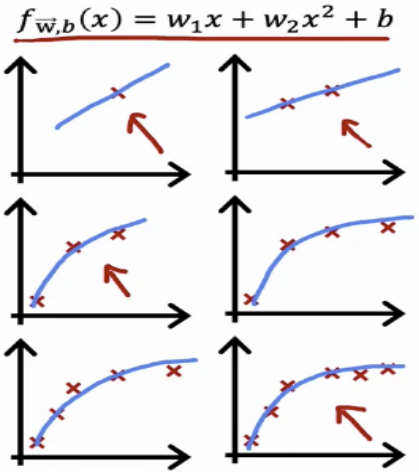
- Check if the model error higher than human error (speech recognition)
- Competing algorithms performance
- Guess based on prior experience
- High difference between baseline and training = high bias / training and dev = high variance



Learning curves

- Train set size increases | Training error (J train) increases and Dev error (J cv) decreases
- Dev error will be always higher than training error
- If model has high bias adding data will not improve it
- If model has high variance adding more data will improve it



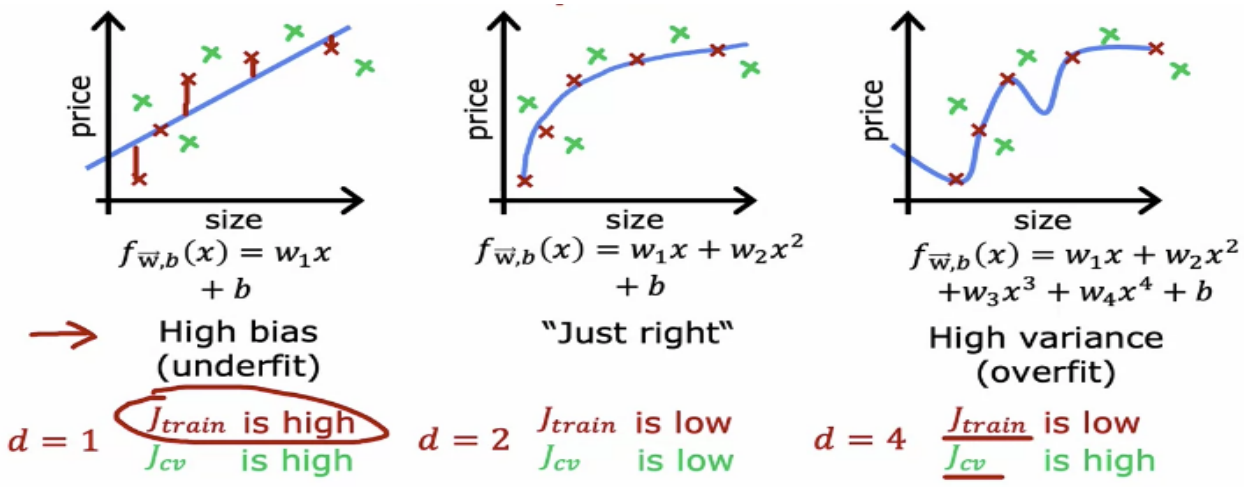


7.1. Model selection

- Split data in 3 train 60% / dev set 20% / test set 20%
    - The name cross-validation refers to that this is an extra dataset that we're going to use to check the accuracy of different models. You may also hear people call this extra dataset or validation set or development set.
- For regression models try adding polynomials (1 to 10) and see the prediction for cross-validation data set
- For neural networks try adding layers and increase number of neurons and see prediction of dev set
- Estimate generalization error (error when model sees new data) using test dataset

7.2. Bias and Variance

7.2.1. Polynomial and Bias/Variance



Fix overfitting:

- Collect more data
- Feature selection
- Regularization

7.2.2. Regularization and Bias/Variance

- Large Lambda = high bias (underfit)
- Small Lambda = high variance (overfit)
- Intermediate Lambda = just right!
- Try different lambdas (10) with cross validation

7.2.3. Neural Network and Bias/Variance

- Large neural networks are low bias machines
    - Requires big gpus
    - Slow training
    - Large neural network will usually do as well or better than a smaller one so long as regularization is chosen appropriately
- Adding data fixes variance
    - Not always possible

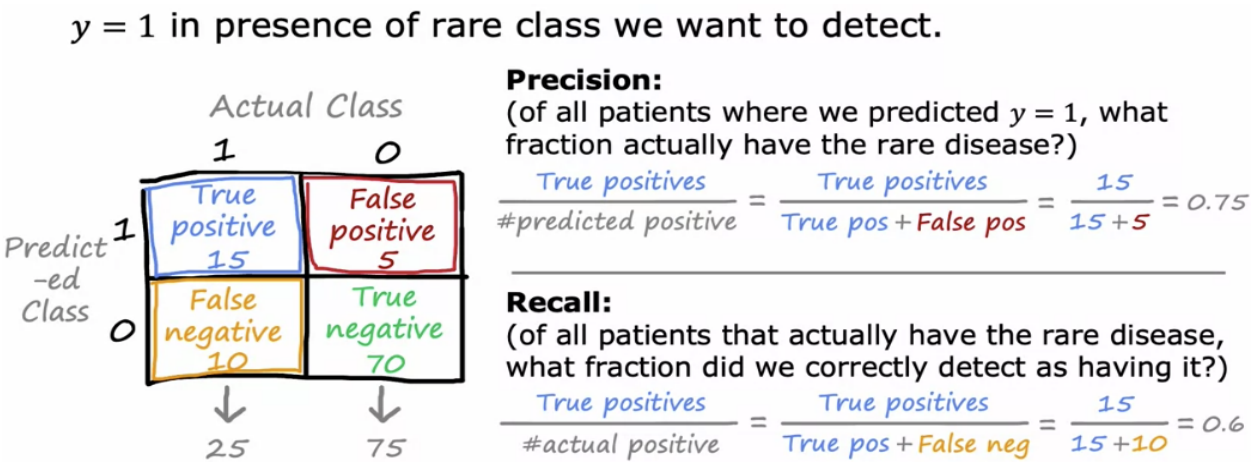
7.3. Error metrics for skewed datasets

Skewed dataset is where the ratio of positive to negative examples are high, very far from 50-50, then it turns out that the usual error metrics like accuracy don't work that well.

Precision (positive predictive value) is the fraction of relevant instances among the retrieved instances.

Recall (sensitivity) is the fraction of relevant instances that were retrieved.

You can adjust the condition of predicting 0 or 1 by moving the threshold from 0.5 to another value and compare precision and recall values. When predict 1 is > 0.5 result in higher precision, lower recall and vice versa.

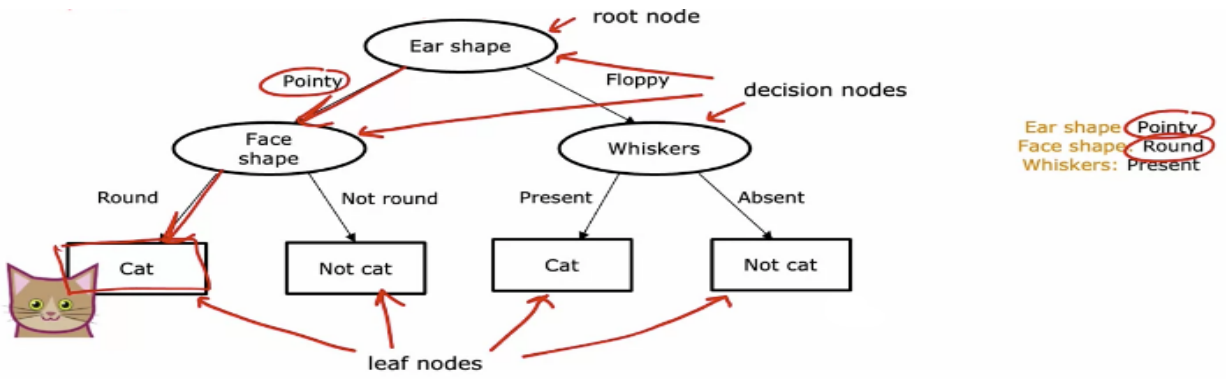


F1 score calculates average of precision and recall but in a way so that if any of the values are really small it will change accordingly.

	Precision (P)	Recall (R)	F_1
Model 1	88.3	79.1	83.4% ←
Model 2	97.0	7.3	13.6%

$$F_1 = \frac{2}{\frac{1}{P} + \frac{1}{R}}$$

8. Decision Tree Model



8.1. Decisions

Decision 1: How to choose what feature to split on at each node?

- Maximize purity (or minimize impurity)

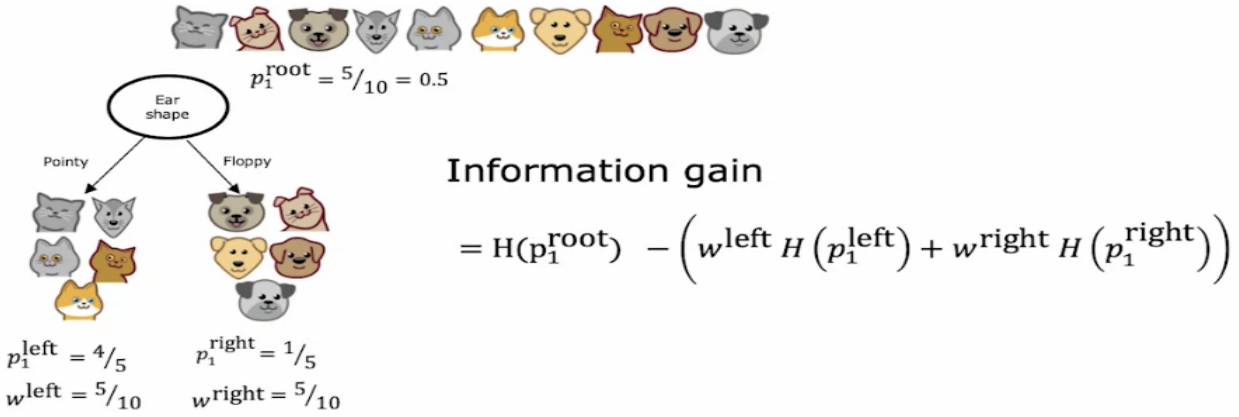
Decision 2: When do you stop splitting?

- When node is 100% one class
- When splitting a node will result in the tree exceeding a maximum depth
- When improvements in purity score are below a threshold
- When number of examples in a node is below a threshold

Entropy is a measure of impurity. Formula $H(p_1) = -p_1 log_2(p_1) - p_0 log_1(p_0)$

In decision tree learning, the reduction of entropy is called information gain.

Choosing split by calculating information gain



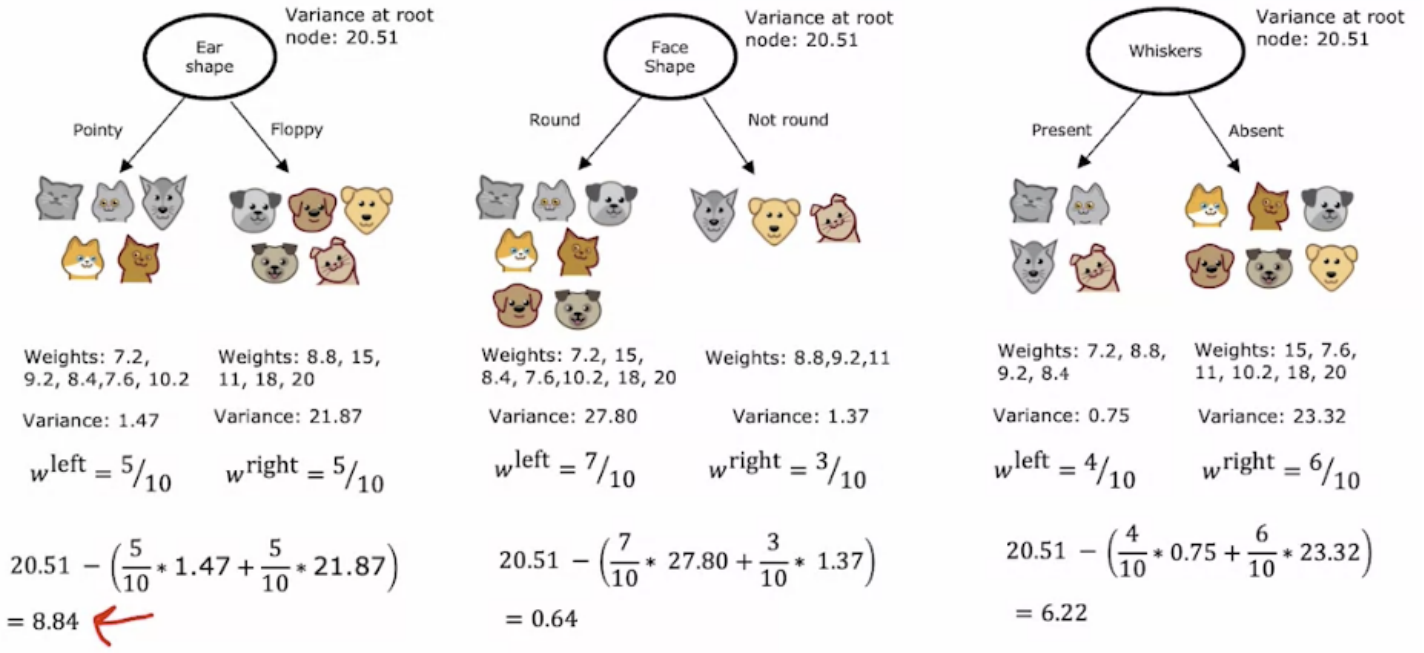
8.2. Decision Tree Learning Process

1. Start with all examples at the root node
2. Calculate information gain for all possible features, and pick the one with the highest information gain
3. Split dataset according to selected feature, and create left and right branches of the tree
4. Keep repeating splitting process until stopping criteria is met:
    a. When node is 100% one class
    b. When splitting a node will result in the tree exceeding a maximum depth
    c. When improvements in purity score are below a threshold
    d. When number of examples in a node is below a threshold

Use one hot encoding for categorical values.

For splitting continuous variables you can plot variables on x axis vs prediction value on y. When consuming splits, you would just consider different values to split on, carry out the usual information gain calculation and decide to split on that continuous value feature if it gives the highest possible information gain.

8.3. Regression Tree



The reason we use an **ensemble of trees** is by having lots of decision trees and having them vote, it makes your overall algorithm less sensitive to what any single tree may be doing because it gets only one vote out of three or one vote out of many, many different votes and it makes your overall algorithm more robust.

The process of **sampling with replacement** lets you construct a new training set that's a little bit similar to, but also pretty different from your original training set. It turns out that this would be the key building block for building an ensemble of trees. Let's take a look in the next video and how you could do that.

Randomizing Forest Algorithm

At each node, when choosing a feature to use to split, if n features are available, pick a random subset of k < n feature and allow the algorithm to only choose from that subset of features. A typical choice for the value of K would be to choose it to be the square root of n.

XGBoost

Algorithm focus more attention on the subset of examples that are not yet doing well on and get the new decision tree, the next decision tree reporting ensemble to try to do well on them.

8.4. Decision Trees vs Neural Networks

Decision trees

- Works well on tabular (structured) data
- Not recommended for unstructured data (images, audio, text)
- Fast
- Small decision trees maybe human interpretable

Neural Networks

- Works well on all types of data
- Maybe slower than a decision tree
- Works with transfer learning
- When building a system of multiple models working together, it might be easier to string together multiple neural network

9. Unsupervised Learning

9.1. Clustering

Clustering algorithm looks at the number of data points and automatically finds data points that are related or similar to each other.

9.1.1. K-means Algorithm

Randomly initialize K cluster centroids $\mu_1, \mu_2, \ldots, \mu_K$

Repeat {

 # Assign points to cluster centroids

for i = 1 to m

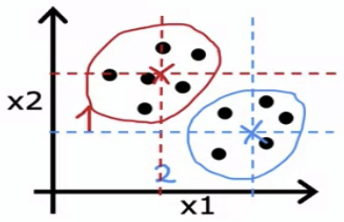
$c^{(i)}$:= index (from 1 to K) of cluster centroid closest to $x^{(i)}$

 # Move cluster centroids

for k = 1 to K

μ_k := average (mean) of points assigned to cluster k

}



$$\mu_1 = \frac{1}{4}\left[x^{(1)} + x^{(5)} + x^{(6)} + x^{(10)}\right]$$

9.1.2. Cost Function

$c^{(i)}$ - index of cluster (1,2,...,K) to which example $x^{(i)}$ is currently assigned

μ_k - cluster centroid k

$\mu_{c^{(i)}}$ - cluster centroid of cluster to which example $x^{(i)}$ has been assigned

$$J(c^{(1)},\ldots, c^{(m)}, \mu_1,\ldots, \mu_k) = \frac{1}{m}\sum_{i=1}^{m} ||x^{(i)} - \mu_{c^{(i)}}||^2$$

Randomly initialization K cluster centroids

Choose K < m

Randomly pick initial K from training examples. Repeat to find the best local optima. To find the best local optima, compute the cost function J for all of these solutions. And then to pick one with the lowest value for the cost function J.

For i = 1 to 100 { (50-1000)

Randomly initialize K-means. (k random examples)

Run K-means. Get $c^{(1)}, \ldots, c^{(m)}, \mu_1, \mu_1, \ldots, \mu_k$

Computer cost function (distortion)

$J(c^{(1)}, \ldots, c^{(m)}, \mu_1, \mu_1, \ldots, \mu_k)$

}

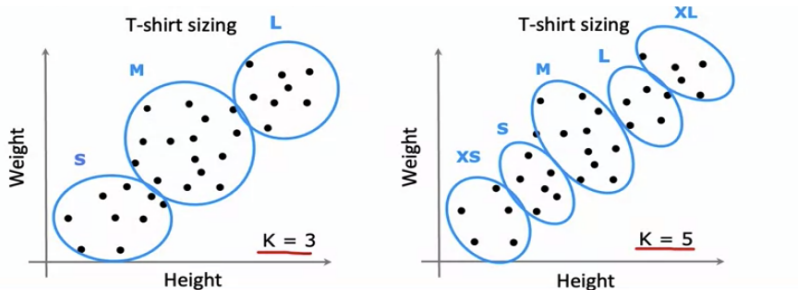
Pick set of clusters that gave lowest cost J

9.1.3. Choosing number of clusters

Elbow method (not recommended): run K-means with a variety of values of K and plot the cost function or the distortion function J as a function of the number of clusters.

Better Option: Evaluate K-means based on a metric for how well it performs for downstream/later purpose.

E.g.

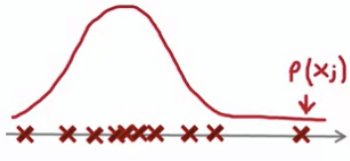


9.2. Anomaly Detection

9.2.1. Anomaly Detection Algorithm with Gaussian (Normal) Distribution

1. Choose n features x_i that you think might be indicative of anomalous examples.
2. Fit parameters $\mu_1, \dots, \mu_n, \sigma_1^2, \dots, \sigma_n^2$

$$\mu_j = \frac{1}{m}\sum_{i=1}^{m} x_j^{(i)} \qquad \sigma_j^2 = \frac{1}{m}\sum_{i=1}^{m}(x_j^{(i)} - \mu_j)^2$$



3. Given new example x, compute $p(x)$:

$$p(x) = \prod_{j=1}^{n} p(x_j; \mu_j, \sigma_j^2) = \prod_{j=1}^{n} \frac{1}{\sqrt{2\pi}\sigma_j} exp(-\frac{(x_j - \mu_j)^2}{2\sigma_j^2})$$

Anomaly if $p(x) < \varepsilon$

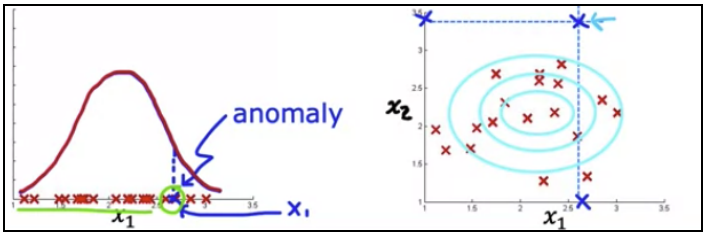
Aircraft engines dataset example
10000 good engines
20 flawed engines (anomalous, y = 1)
Training: 6000 good engines (y=0)
CV: 2000 good engines and 10 anomalous
Test: 2000 good engines and 10 anomalous

9.2.2. Anomaly Detection vs. Supervised Learning

Anomaly Detection	Supervised Learning
Very small number of positive examples (y=1) and large number of negative examples (y=0) examples. Many different types of anomalies. Hard for any algorithm to learn from positive examples what the anomalies look like. Future anomalies may look nothing like any of the anomalous examples we've seen so far. E.g. Fraud	Large number of positive and negative examples. Enough to get a sense of what positive examples are like. Future positive examples likely to be similar to ones in the training set. E.g. Spam

Choosing Features
It is important to use the right features in Unsupervised learning as there is no y.

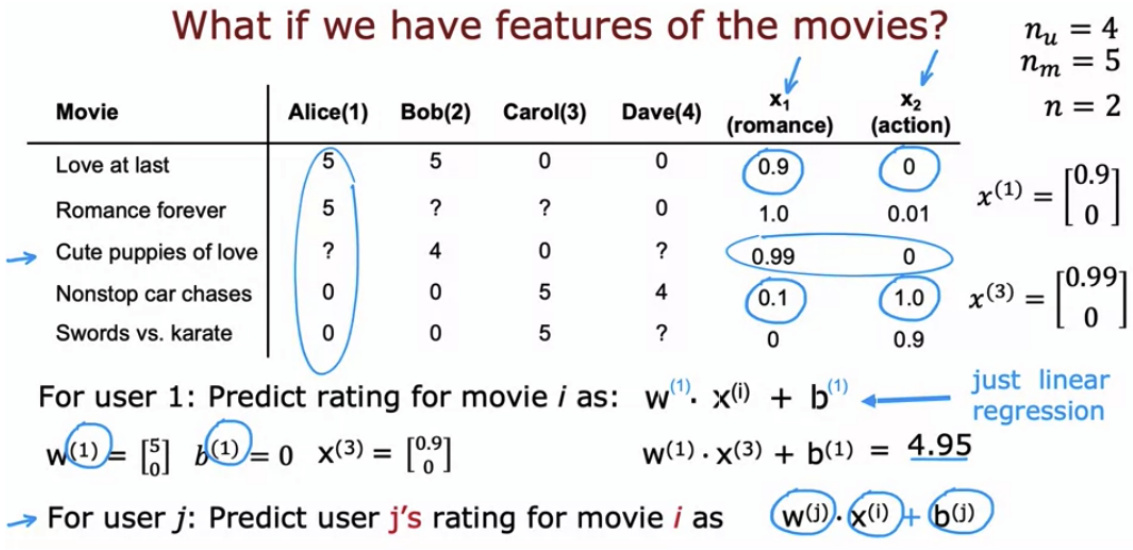
- Make sure the feature has gaussian distribution
- If not gaussian then compute log(x) or log(x+c) or x**2



9.2.3. Error Analysis for Anomaly Detection

Want: $p(x) \geq \varepsilon$ large for normal examples x. $p(x) < \varepsilon$ small for anomalous examples x.
Most common problem: $p(x)$ is comparable (say, both large) for normal and anomalous examples x.

9.3. Recommender Systems



Notation

$r(i,j) = 1\ if\ the\ user\ j\ has\ rated\ movie\ i\ (0\ otherwise)$

$y^{(i,j)} = rating\ given\ by\ user\ j\ on\ movie\ i\ (if\ defined)$

$w^{(j)},\ b^{(j)} = parameters\ for\ user\ j$

$x^{(i)} = feature\ vector\ for\ movie\ i$

$m^{(j)} = no.\ of\ movies\ rated\ by\ user\ j$

9.3.1. Cost Function

To learn parameters $w^{(j)}, b^{(j)}$ for user j :

$$J(w^{(j)}, b^{(j)}) = \frac{1}{2} \sum_{i:r(i,j)} \left(w^{(j)} \cdot x^{(i)} + b^{(j)} - y^{(i,j)}\right)^2 + \frac{\lambda}{2} \sum_{k=1}^{n} \left(w_k^{(j)}\right)^2$$

To learn parameters $w^{(1)}, b^{(1)},\ w^{(2)}, b^{(2)}, \cdots w^{(n_u)}, b^{(n_u)}$ for all users :

$$J\begin{pmatrix} w^{(1)}, \ \ldots, w^{(n_u)} \\ b^{(1)}, \ \ldots, b^{(n_u)} \end{pmatrix} = \frac{1}{2} \sum_{j=1}^{n_u} \sum_{i:r(i,j)=1} \underbrace{\left(w^{(j)} \cdot x^{(i)} + b^{(j)} - y^{(i,j)}\right)^2}_{f(x)} + \frac{\lambda}{2} \sum_{j=1}^{n_u} \sum_{k=1}^{n} \left(w_k^{(j)}\right)^2$$

Example without x1 and x2 features

Movie	Alice (1)	Bob (2)	Carol (3)	Dave (4)	x_1 (romance)	x_2 (action)
Love at last	5	5	0	0	?	?
Romance forever	5	?	?	0 ←	?	? $x^{(2)}$
Cute puppies of love	?	4	0	? ←	?	? $x^{(3)}$
Nonstop car chases	0	0	5	4 ←	?	?
Swords vs. karate	0	0	5	? ←	?	?

$w^{(1)} = \begin{bmatrix} 5 \\ 0 \end{bmatrix}$, $w^{(2)} = \begin{bmatrix} 5 \\ 0 \end{bmatrix}$, $w^{(3)} = \begin{bmatrix} 0 \\ 5 \end{bmatrix}$, $w^{(4)} = \begin{bmatrix} 0 \\ 5 \end{bmatrix}$

$b^{(1)} = 0$, $b^{(2)} = 0$, $b^{(3)} = 0$, $b^{(4)} = 0$ ←

using $w^{(j)} \cdot x^{(i)} + b^{(j)}$

$w^{(1)} \cdot x^{(1)} \approx 5$
$w^{(2)} \cdot x^{(1)} \approx 5$
$w^{(3)} \cdot x^{(1)} \approx 0$
$w^{(4)} \cdot x^{(1)} \approx 0$

→ $x^{(1)} = \begin{bmatrix} 1 \\ 0 \end{bmatrix}$

9.3.2. Cost Function when features are unknown

Given $w^{(1)}, b^{(1)}, w^{(2)}, b^{(2)}, \cdots, w^{(n_u)}, b^{(n_u)}$

to learn $x^{(i)}$:

$$J(x^{(i)}) = \frac{1}{2} \sum_{j:r(i,j)=1} \left(w^{(j)} \cdot x^{(i)} + b^{(j)} - y^{(i,j)}\right)^2 + \frac{\lambda}{2} \sum_{k=1}^{n} \left(x_k^{(i)}\right)^2$$

→ To learn $x^{(1)}, x^{(2)}, \cdots, x^{(n_m)}$:

$$J(x^{(1)}, x^{(2)}, \ldots, x^{(n_m)}) = \frac{1}{2} \sum_{i=1}^{n_m} \sum_{j:r(i,j)=1} \left(w^{(j)} \cdot x^{(i)} + b^{(j)} - y^{(i,j)}\right)^2 + \frac{\lambda}{2} \sum_{i=1}^{n_m} \sum_{k=1}^{n} \left(x_k^{(i)}\right)^2$$

9.3.3. Collaborative Filtering

	$j=1$	$j=2$	$j=3$
	Alice	Bob	Carol
$i=1$ Movie1	5	5	?
$i=2$ Movie2	?	2	3

Cost function to learn $w^{(1)}, b^{(1)}, \cdots w^{(n_u)}, b^{(n_u)}$:

$$\min_{w^{(1)}, b^{(1)}, \cdots, w^{(n_u)}, b^{(n_u)}} \frac{1}{2} \sum_{j=1}^{n_u} \sum_{i:r(i,j)=1} \left(w^{(j)} \cdot x^{(i)} + b^{(j)} - y^{(i,j)}\right)^2 + \frac{\lambda}{2} \sum_{j=1}^{n_u} \sum_{k=1}^{n} \left(w_k^{(j)}\right)^2$$

Cost function to learn $x^{(1)}, \cdots, x^{(n_m)}$:

$$\min_{x^{(1)}, \cdots, x^{(n_m)}} \frac{1}{2} \sum_{i=1}^{n_m} \sum_{j:r(i,j)=1} \left(w^{(j)} \cdot x^{(i)} + b^{(j)} - y^{(i,j)}\right)^2 + \frac{\lambda}{2} \sum_{i=1}^{n_m} \sum_{k=1}^{n} \left(x_k^{(i)}\right)^2$$

Put them together:

$$\min_{\substack{w^{(1)}, \ldots, w^{(n_u)} \\ b^{(1)}, \ldots, b^{(n_u)} \\ x^{(1)}, \ldots, x^{(n_m)}}} J(w, b, x) = \frac{1}{2} \sum_{(i,j):r(i,j)=1} \left(w^{(j)} \cdot x^{(i)} + b^{(j)} - y^{(i,j)}\right)^2 + \frac{\lambda}{2} \sum_{j=1}^{n_u} \sum_{k=1}^{n} \left(w_k^{(j)}\right)^2 + \frac{\lambda}{2} \sum_{i=1}^{n_m} \sum_{k=1}^{n} \left(x_k^{(i)}\right)^2$$

9.3.4. Gradient Descent

repeat {

$$w_i^{(j)} = w_i^{(j)} - \alpha \frac{\partial}{\partial w_i^{(j)}} J(w, b, x)$$

$$b^{(j)} = b^{(j)} - \alpha \frac{\partial}{\partial b^{(j)}} J(w, b, x)$$

$$x_k^{(i)} = x_k^{(i)} - \alpha \frac{\partial}{\partial x_k^{(i)}} J(w, b, x) \}$$

Collaborative filtering with binary labels

1 = watched the movie till the end,

0 = did not watch till the end,

? = did not start watching the movie

Other examples:

1. Did user j purchase an item after being shown?
2. Did user j like an item?
3. Did user j spend at least 30 sec with an item?
4. Did user j click on an item?

Movie	Alice(1)	Bob(2)	Carol(3)	Dave(4)
Love at last	1	1	0	0
Romance forever	1	?	?	0
Cute puppies of love	?	1	0	?
Nonstop car chases	0	0	1	1
Swords vs. karate	0	0	1	?

Cost function for binary application

Previous cost function:

$$\frac{1}{2}\sum_{(i,j):r(i,j)=1}\left(\underbrace{w^{(j)}\cdot x^{(i)}+b^{(j)}}_{f(x)}-y^{(i,j)}\right)^2+\frac{\lambda}{2}\sum_{i=1}^{n_m}\sum_{k=1}^{n}\left(x_k^{(i)}\right)^2+\frac{\lambda}{2}\sum_{j=1}^{n_u}\sum_{k=1}^{n}\left(w_k^{(j)}\right)^2$$

Loss for binary labels $y^{(i,j)}$: $f_{(w,b,x)}(x)=g(w^{(j)}\cdot x^{(i)}+b^{(j)})$

$$L\left(f_{(w,b,x)}(x),y^{(i,j)}\right)=-y^{(i,j)}\log\left(f_{(w,b,x)}(x)\right)-\left(1-y^{(i,j)}\right)\log\left(1-f_{(w,b,x)}(x)\right)$$ ← Loss for single example

$$J(w,b,x)=\sum_{(i,j):r(i,j)=1}\underbrace{L\left(f_{(w,b,x)}(x),y^{(i,j)}\right)}_{g(w^{(j)}\cdot x^{(i)}+b^{(j)})}$$ ← cost for all examples

Mean normalization of the dataset will help to improve accuracy and performance. Impute null data with mean of the rating.

9.3.5. TensorFlow Implementation of Collaborative Filtering

Gradient descent algorithm

Repeat until convergence

$w = w-\alpha \frac{\partial}{\partial w} J(w,b,x)$

$b = b-\alpha \frac{\partial}{\partial b} J(w,b,x)$

$X = X-\alpha \frac{\partial}{\partial x} J(w,b,x)$

```
# Instantiate an optimizer.
optimizer = keras.optimizers.Adam(learning_rate=1e-1)

iterations = 200
for iter in range(iterations):
    # Use TensorFlow's GradientTape
    # to record the operations used to compute the cost
    with tf.GradientTape() as tape:

        # Compute the cost (forward pass is included in cost)
        cost_value = cofiCostFuncV(X, W, b, Ynorm, R,
            num_users, num_movies, lambda)
                       n_u        n_m
    # Use the gradient tape to automatically retrieve
    # the gradients of the trainable variables with respect to
        the loss
    grads = tape.gradient( cost_value, [X,W,b] )

    # Run one step of gradient descent by updating
    # the value of the variables to minimize the loss.
    optimizer.apply_gradients( zip(grads, [X,W,b]) )
```

9.3.6. Limitations of Collaborative Filtering

Cold start problem. How to

- rank new items that few users have rated?
- show something reasonable to new users who have rated a few items?

Use side information about items or users:

- Item: Genre, movie stars, studio, …
- User: Demographics (age, gender, location), expressed preferences, …

Collaborative filtering vs Content-based filtering

Collaborative filtering recommend items to you based on rating of users who face similar ratings as you

Content-based filtering recommend items to you based on features of user and item to find good match

9.3.7. Content-based filtering neural network

Recommending from a large catalog: Retrieval & Ranking

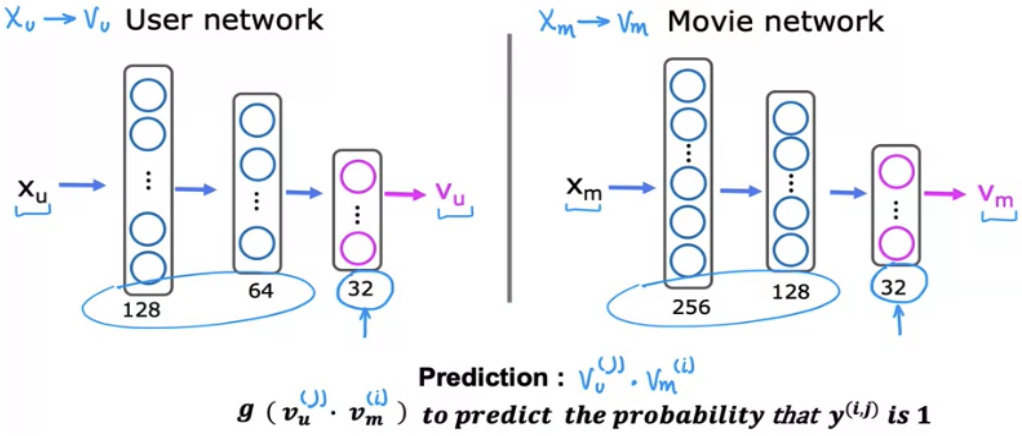
Retrieval:

- Generate large list of plausible item candidates
  - For each of the last 10 movies watched by the user, find 10 most similar movies
  - For most viewed 3 genres, find the top 10 movies
  - Top 20 movies in the country
- Combine retrieved items into list, removing duplicates and items already watched/purchased

Ranking:

- Take list retrieved and rank using learned model
- Display ranked items to user

Retrieving more items results in better performance but slower recommendations. To analyze/optimize the trade-off, carry out offline experiments to see if retrieving additional items result in more relevant recommendations



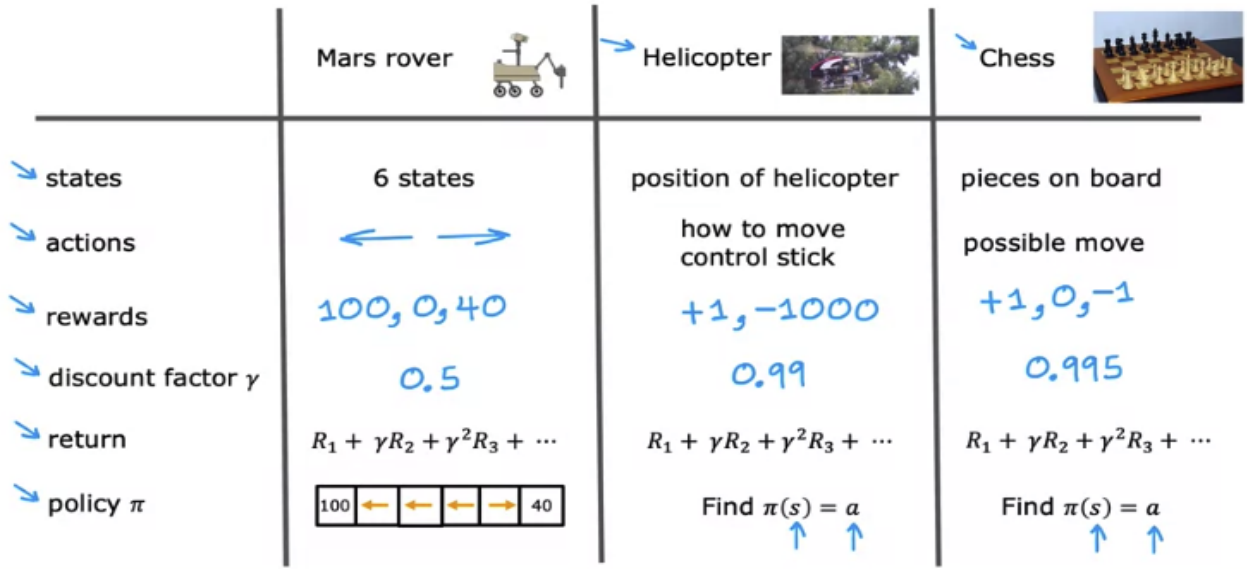
10. Reinforcement Learning

One way to think of why reinforcement learning is so powerful is you have to tell it what to do rather than how to do it. A key input to reinforcement learning is something called the reward. Specifying the reward function rather than the optimal action gives you a lot more flexibility in how you design the system. So the way to think of the reward function is a bit like training a dog. So how do you get a puppy to behave well? You let it do its thing and whenever it does something good, you go, good dog. And whenever they did something bad, you go, bad dog.

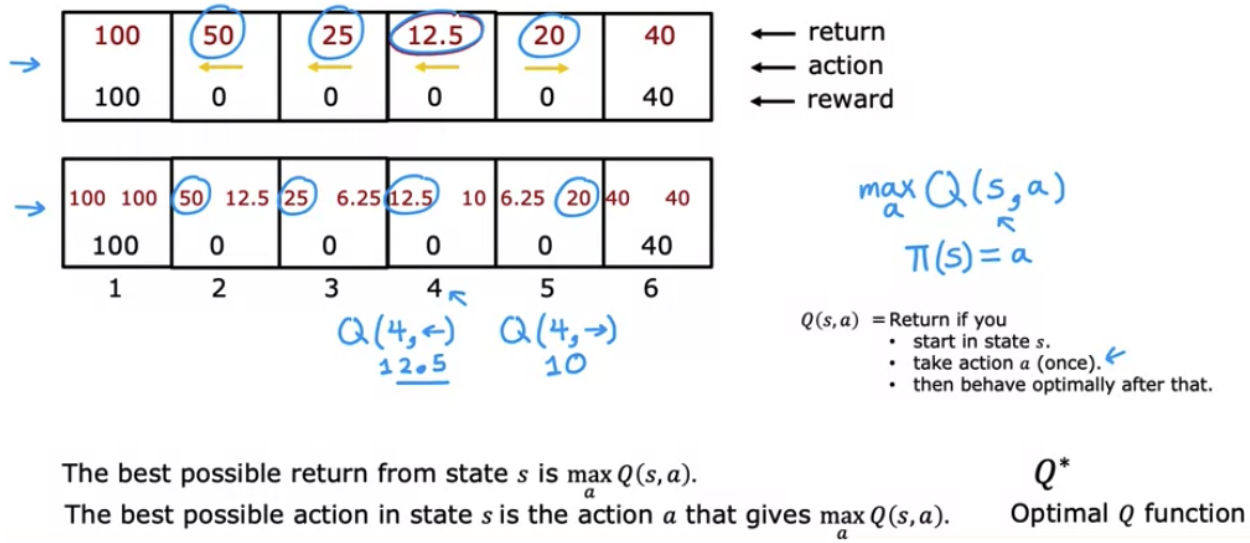
The return in reinforcement learning is the sum of the rewards that the system gets, weighted by the discount factor, where rewards in the far future are weighted by the discount factor raised to a higher power.

A policy is a function $a = \pi(s)$ mapping from states to actions, that tells you what action a to take in a given state s

10.1. Key Factors



The state action value function (Q function) is a function typically denoted by the letter uppercase Q. And it's a function of a state you might be in as well as the action you might choose to take in that state.



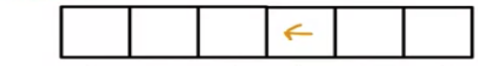
The best possible return from state s is $\max_a Q(s,a)$.

The best possible action in state s is the action a that gives $\max_a Q(s,a)$.

Q^* Optimal Q function

10.2. Bellman Equation

$Q(s,a)$ = Return if you
- start in state s.
- take action a (once).
- then behave optimally after that.



s : current state
a : current action
$R(s)$ = reward of current state
s' : state you get to after taking action a
a' : action that you take in state s'

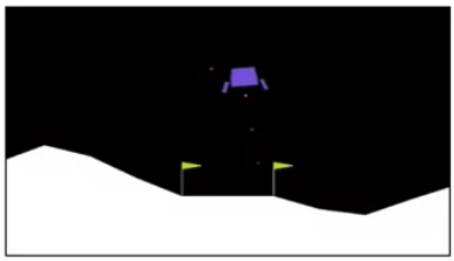
$$Q(s,a) = R(s) + \gamma \max_{a'} Q(s',a')$$

10.3. Markov Decision Process (MDP)

Markov decision process the future depends only on where you are now, not on how you got here.

Continuous state Markov decision process, continuously MTP. The state of the problem isn't just one of a small number of possible discrete values, like a number from 1-6. Instead, it's a vector of numbers, any of which could take any of a large number of values. Whereas for the Mars rover example, the state was just one of six possible numbers. It could be one, two, three, four, five or six. For the car, the state would comprise this vector of six numbers, and any of these numbers can take on any value within its valid range.

10.4. Algorithm



Initialize neural network randomly as guess of $Q(s,a)$.

Repeat {

Take actions in the lunar lander. Get $(s,a,R(s),s')$.

Store 10,000 most recent $(s,a,R(s),s')$ tuples.

Replay Buffer

Train neural network:

Create training set of 10,000 examples using

$x = (s,a)$ and $y = R(s) + \gamma \max_{a'} Q(s',a')$

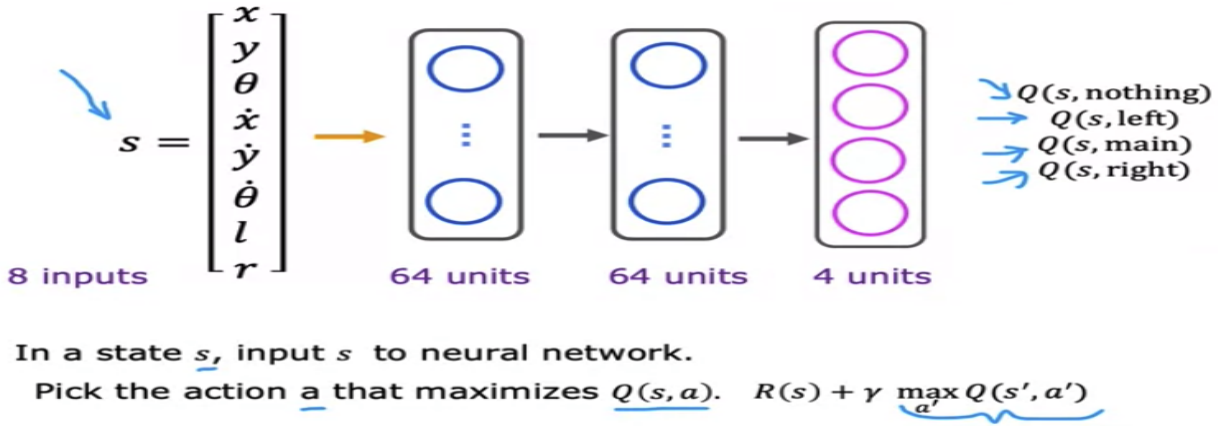
Train Q_{new} such that $Q_{new}(s,a) \approx y$.

Set $Q = Q_{new}$.

$f_{w,B}(x) \approx y$

x, y $x^{(1)}, y^{(1)}$ ... $x^{(10000)}, y^{(10000)}$

10.4.1. Deep Reinforcement Learning Architecture



In a state s, input s to neural network.
Pick the action a that maximizes $Q(s,a)$. $R(s) + \gamma \max_{a'} Q(s', a')$

10.4.2. Greedy policy

How to choose actions while still learning?
In some state s
Option 1: Pick the action a that maximizes Q(s,a)
Option 2:

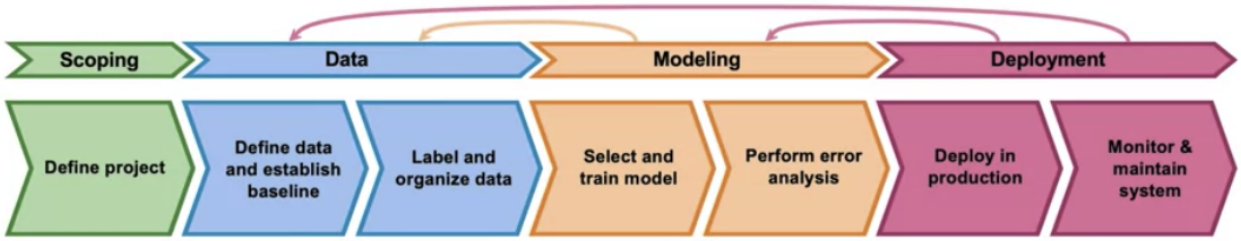
- With probability 0.95, pick the action a that maximizes Q(s,a). Greedy, 'Exploitation'
- With probability 0.05, pick action a randomly. 'Exploration'

Given a training set with a high number of examples, every step of gradient descent computes sum over all examples.
Mini-batch gradient descent looks at the random subset of the data on each iteration.
Soft Update allows you to make a more gradual change to Q or to the neural network parameters W and B that affect your current guess for the Q function Q of s, a. Soft update method causes the reinforcement learning algorithm to converge more reliably and it makes it less likely that algorithm will oscillate or divert or have other undesirable properties.

10.5. Limitations of Reinforcement Learning

- Much easier to get to work in a simulation than a real robot
- Far fewer application than supervised and unsupervised learning
- Exciting research direction with potential for future applications

11. Machine Learning Project Lifecycle



11.1. Scoping

Scoping refers to picking what project to work on and planning out the scope of the project.
Questions:

- What project should we work on?
- What are the metrics for success?
- What are the resources (data, time, people) needed?

11.1.1. Scoping process

1. Brainstorm business problems (not AI problems)
2. Brainstorm AI solution
3. Assess the feasibility and value of potential solutions
   a. To identify feasibility use external benchmark (literature, other company, competitors)
   b. Gather diligence on value from tech and business teams then have both of the teams to agree on metrics
4. Determine milestones
5. Budget for resources

11.1.2. Ethical consideration

- Is this project creating net positive societal value?
- Is this project reasonably fair and free from bias?
- Have any ethical concerns been openly aired and debated?

11.1.3. Milestones & Resourcing

- ML metrics (accuracy, precision/recall, etc.)
- Software metrics (latency, throughput)
- Business metrics (revenue, etc.)
- Resources needed (data, personnel, help from other teams)
- Timeline

If unsure, consider benchmarking to other projects, or build a POC (Proof of Concept) first.

11.2. Data

11.2.1. Define Data

Major types of data problems

Unstructured data:

- May or may not have huge collection of unlabeled examples x
- Human can label more data
- Data augmentation more likely to be helpful

Structured data:

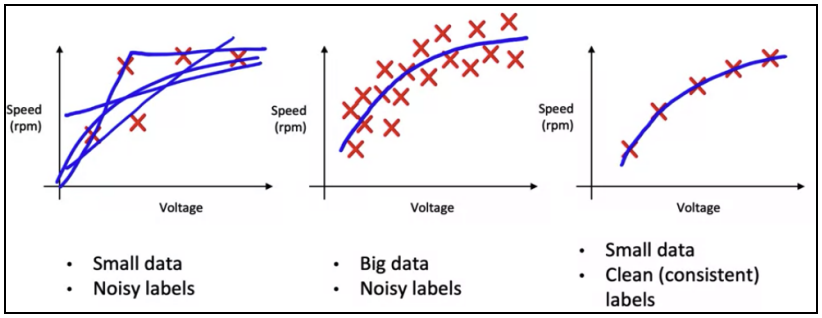
- May be more difficult to obtain more data
- Human labeling may not be possible (with some exceptions)

Small data (<10k):

- Clean labels are critical
- Can manually look through dataset and fix labels
- Can get all the labelers to talk to each other

Big data (>10k):

- Emphasis data process (create a data definition process and share that with all labelers)



Data definition questions (for labeling data)

- What is the input x?
  - Lighting? Contract? Resolution?
  - What features need to be included?
- What is the target label y?
  - How can we assure labelers give consistent labels?

Improving label consistency

- Have multiple labelers label same example
- When there is a disagreement, have MLE, SME and/or labelers discuss definition of y or reach agreement
- If labelers believe that x doesn’y contain enough information, consider changing x
- Iterate until it is hard to significantly increase agreement
- Standardize labels
- Merge classes of labels
- Have a class/label to capture uncertainty
- For big data consider having multiple labelers label every example and using voting or consensus labels to increase accuracy

11.2.2. Establish Baseline

Baseline helps to indicate what might be possible. In some cases it also gives a sense of what is irreducible error/Bayes error.

- Human Level Performance (HLP)
- Literature search for state-of-the-art / open source
- Quick-and-dirty implementation
- Performance of older system

HLP is the accuracy of a human making a guess on y output. You can establish a baseline based on HLP. Usually works well with unstructured data (images, audio, text). When the ground truth label is externally defined, HLP gives an estimate for Bayes error/irreductible error. However, often ground truth is just another human label.

11.2.3. Label and Organize Data

How long should you spend obtaining data?

- Get into iteration loop as quickly as possible
- Instead of asking: How long would it take to obtain m examples? Ask: How much data can we obtain in k days?
- Exception: if you have worked on the problem before and from experience you know you need m examples

When making a decision on where to get data you can have a brainstorm session where you list all your possible data sources. You have to consider data quality, privacy and regulatory constraints.
Labeling data

- Options: In-house vs. outsourced vs. crowdsourced
- Having MLEs label data is expensive. But doing this for just a few days is usually fine
- Who is qualified to label
- Don’t increase data by more than 10x at a time (if you have dataset of 1000k in first run, get 3x - 10x for next run)

11.2.4. Meta-data, Data Provenance and Lineage

Keep track of data provenance (where the data came from) and lineage (sequence of steps). Create extensive documentation. Meta-data is data about data. Gathering meta-data can be helpful at the error analysis stage and to keep track of data provenance.

11.2.5. Data Augmentation

Data Augmentation refers to modifying an existing training example to create a new training example usually used for unstructured data. How can you modify or warp or distort or make more noise in your data in a way so that what you get is similar to what you have in your test set. Goal of data augmentation is to create realistic examples that the algorithm does poorly on, but the humans (or other baseline) do well on.
Checklist:

- Does it sound realistic?
- Is the x -> y mapping clear? (e.g., can humans recognize speech?)
- Is the algorithm currently doing poorly on it?

For unstructured data: if the model is large and the mapping x->y is clear, then adding data rarely hurts accuracy. On the other hand if the model is small, then it might hurt the accuracy.
Data Synthesis refers to using artificial data inputs to create a new training example.

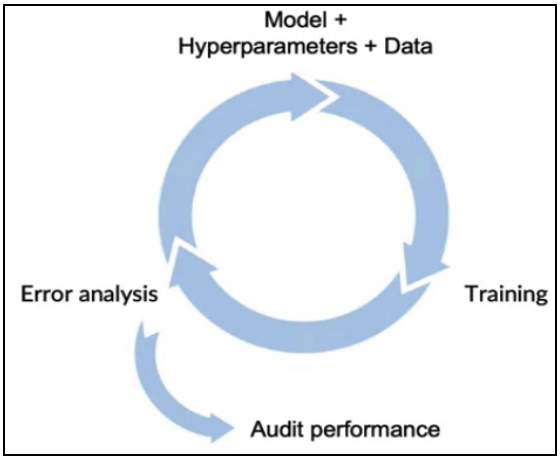
11.2.6. From Big Data to Good Data

Try to ensure consistently high-quality data in all phases of the ML project lifecycle.
Good data:

- Covers important cases (good coverage of input x)
- Is defined consistently (definition of labels y is unambiguous)
- Has timely feedback from production data (distribution covers data drift and concept dring)
- Is sized appropriately

11.3. Modeling

11.3.1. Iterative Loop of ML Development



- Choose Architecture
- Train Model
- Diagnostics
- Repeat
- Audit Performance

Note: once the decision is made on which model will go to production, good practice would be doing final performance audit before pushing the model to production.
Make sure that a model does well on business metrics / project goals, not dev/test dataset.
Model-centric approach - work on code to improve accuracy
Data-centric approach - work on data quality to improve accuracy

11.3.2. Key Challenges

- Performance on disproportionately important examples
- Performance on key slices of the dataset (E.g. loan approval model: make sure not to discriminate by ethnicity, gender, location, language etc.)
- Skewed data distribution (99% negative case and 1% positive, cancer test result or fraudulent transaction)

11.3.3. Tips For Getting Started on ML Project

- Literature search to see what's possible (courses, blogs, open-source project)
- Find open-source implementations if available
- A reasonable algorithm with good data will often outperform a great algorithm with not so good data
- Take into account deployment constraints, if you plan to deploy the project
- Sanity check for code and algorithm before running on the large dataset

11.3.4. Error Analysis

Error analysis process refers to manually looking through prediction examples and trying to gain insights into where the algorithm is going wrong. Find a set of examples that the algorithm has misclassified examples from the cross validation set and group them into common themes or common properties or common traits. Error analysis is an iterative process just like model development. Create a spreadsheet and tag each case and find patterns.
Useful metrics for each tag:

- What fraction of errors has that tag?
- Of all data with that tag, what fraction is misclassified?
- What fraction of all the data has that tag?

- How much room for improvement is there on data with that tag?

Decide on most important categories to work on based on:

- How much room for improvement there is
- How frequently that category appears
- How easy is to improve accuracy in that category
- How important it is to improve in that category

Type	Accuracy	Human level performance	Gap to HLP	% of data
Clean Speech	94%	95%	1%	60% → 0.6%
Car Noise	89%	93%	4%	4% → 0.16%
People Noise	87%	89%	2%	30% → 0.6%
Low Bandwidth	70%	70%	0%	6% → ~0%

For categories you want to prioritize:

- Collect more data
- Use data augmentation to get more data
- Improve label accuracy/data quality

11.3.5. Auditing Framework

Check for accuracy, fairness/bias, and other problems

1. Brainstorm the ways the system might go wrong
   a. Performance on subset of data (e.g., ethnicity, gender)
   b. How common are certain errors (e.g., FP, FN)
   c. Performance on rare classes
2. Establish metrics to assess performance against these issues on appropriate slices of data
3. Get business/product owner buy-in

11.3.6. Experiment Tracking

What to track?

- Algorithm/code versioning
- Dataset used
- Hyperparameters
- Results

Tracking tools

- Text files
- Spreadsheet
- Experiment tracking system (MLflow, Sage Maker Studio, Comet)

Desirable features

- Information needed to replicate results
- Experiment results, ideally with summary metrics/analysis
- Perhaps also: Resource monitoring, visualization, model error analysis

11.4. Deployment

11.4.1. Key Challenges

Concept drift occurs when the patterns the model learned no longer hold. In essence, the very meaning of what we are trying to predict evolves. Depending on the scale, this will make the model less accurate or even obsolete.

Data drift (feature drift, population, or covariate shift): the input data has changed. The distribution of the variables is meaningfully different. As a result, the trained model is not relevant for this new data.

Software engineering issues

Checklist of questions:

- Real-time or Batch
- Cloud vs Edge/browser
- Compute Resources (CPU/GPU/Memory)
- Latency, throughput (QPS)
- Logging
- Security and privacy

11.4.2. Key Ideas

- Gradual ramp up with monitoring
- Rollback

11.4.3. Common Deployment Patterns

Shadow mode where the ML system shadows the human and runs in parallel. The ML system's output is not used for any decisions drawing this phase. Helps to verify performance of the model.

Canary (ready to let model make decisions) Roll out to a small fraction, around 5%, of traffic initially. Monitor system and ramp up traffic gradually. Helps to spot problems early on.

Blue Green has both old (blue) and new (green) model versions and randomly has the router switch to predict through one of the models. Easy to roll back if something goes wrong.

11.4.4. Monitoring

The best way to monitor models in production is to build dashboards

- Brainstorm the things that could go wrong
- Brainstorm few statistics/metrics that will detect the problem

Examples of metrics

- Software metrics: Memory, compute, latency, throughput, server load
- Input metrics: Avg input length, avg input volume, num of missing values
- Output metrics: num of return null

Monitoring dashboards

- Set thresholds for alarms
- Adept metrics and thresholds over time

Model maintenance can be done through manual or automatic retraining.