Machine Learning

Contents

Things:

- Every type of data in an element is called a *feature*.
- Part of data that represents features is X, and the part which represents expected output is y.
- Number of elements in a dataset : m
- Number of features in an element: n
- $x_j^{(i)}$ is j^{th} feature in i^{th} element.

1 Linear Regression

1.1 Univariate Regression

- Process of fitting a straight line through 2D data, i.e. $n = 1$.
- Proposed equation is of form $\theta_0 + \theta_1 x$. This is called a hypothesis. It is represented as

$$\begin{aligned} h_\theta(x) &= \theta_0 + \theta_1 x \\ &= \begin{bmatrix} \theta_0 & \theta_1 \end{bmatrix} \begin{bmatrix} 1 \\ x \end{bmatrix} \\ &= \theta^T X \end{aligned}$$

  Here, a 1 is added to as a feature to make it easy to represent the constant term in matrix form.
- Ideally, $h_\theta(x_i) = y_i$ for $i = 0$ to m. To represent how far the current hypothesis is from the best fit, a cost function is defined.

$$J(\theta) = \frac{1}{2m} \sum_{i=1}^{m} (h_\theta(x^{(i)}) - y^{(i)})^2$$

  This function, being quadratic, will be a parabola with a single global minima.
- To find the value of θ for which $J(\theta)$ is minimum, θ_j has to be chosen such that $\frac{\partial J(\theta)}{\partial \theta_j} = 0$.
- For linear regression,

$$\frac{\partial J(\theta)}{\partial \theta_j} = \frac{1}{m} \sum_{i=1}^{m} (h_\theta(x^{(i)}) - y^{(i)}) x_j^{(i)}$$

- So what is done is, these steps are repeated,

$$\theta_0 = \theta_0 - \alpha * \frac{\partial J(\theta)}{\partial \theta_0}$$

  and

$$\theta_1 = \theta_1 - \alpha * \frac{\partial J(\theta)}{\partial \theta_1}$$

  till the partial derivatives are below some thresholds.
- This technique is called as *Gradient Descent*. This is a technique to find a *local* minima, and not a global minima.

1.2 Multivariate Regression

- In case of multivariate regression, data is n dimensional. So the output parameter y may be dependant on more than 1 $feature$.
- In this case,

$$\begin{aligned} h_\theta(X) &= \theta_0 + \theta_1 x_1 + \theta_2 x_2 \cdots + \theta_n x_n \\ &= \begin{bmatrix} \theta_0 & \theta_1 & \theta_2 & \dots & \theta_n \end{bmatrix} \begin{bmatrix} 1 \\ x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix} \\ &= \theta^T X \end{aligned}$$

  Where both X and θ are vectors.

- Rest everything, $J(\theta)$, $\frac{\partial J(\theta)}{\partial \theta_j}$, gradient descent is same as linear regression.

1.3 Polynomial Regression

- This is *same* as multivariate regression, where the additional features are just higher degree versions of previous features.
  E.g. if $x_1 = x$, then $x_2 = x^2$, $x_3 = x^3$ etc.
- In this case, hypothesis might look like $h_\theta(X) = \theta_0 + \theta_1 x + \theta_2 x^2 \cdots + \theta_n x^n$.
- This is useful if the data cannot be represented by a simple straight line.

2 Logistic Regression

Used for classifying the data into 2 or more classes.

2.1 Classifying into 2 classes

- Here, y is a vector of 0s and 1s representing 2 classes. So, the hypothesis function should output a value between 0 and 1, which might be thought as probability of that element belonging to a class.
- If $h_\theta(X) > 0.5$, predict $y = 1$, and otherwise $y = 0$.
- To limit the hypothesis between 0 and 1, a sigmoid function is used.

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

  As $z \to -\infty$, $g(z) \to 0$,
  as $z \to \infty$, $g(z) \to 1$,
  and $g(z) = 0.5$ at $z = 0$.
- Here, the hypothesis is represented as

$$\begin{aligned} h_\theta(X) &= g(\theta^T X) \\ &= \frac{1}{1 + e^{-\theta^T X}} \end{aligned}$$

- $y = 1$ is predicted if $h_\theta(X) > 0.5$, i.e. $\theta^T X > 0$, and $y = 0$ if $\theta^T X \leq 0$.
- Assume there are 2 features in the data, x_1 and x_2. $\theta^T X = \theta_0 + \theta_1 x_1 + \theta_2 x_2$. $y = 1$ will be predicted if the point lies on e.g. left side of the line $\theta_0 + \theta_1 x_1 + \theta_2 x_2 = 0$, and $y = 0$ if it lies on the other side.
- This line/curve $\theta^T X = 0$ is called *Decision Boundary*.
- If higher order features like x_1^2, x_2^2, $x_1 x_2$ etc. are included, decision boundary can be parabolic/ circular/ elliptical etc.
- The cost function in this case, if wrote similar to that of linear regression, may not be convex i.e. it may have multiple local minima. This can be a problem since gradient descent is being used to optimize the parameters.

- So, the cost function for a single element is defined as

$$Cost(h_\theta(X), y) = \begin{cases} -log(h_\theta(X)) & \text{if } y = 1 \\ -log(1 - h_\theta(X)) & \text{if } y = 0 \end{cases}$$

  Here, if $y = 1$, cost is $-log(h_\theta(X))$, which is 0 if $h_\theta(X) = 1$, and tends to infinity if $h_\theta(X) = 0$. Similarly for $y = 0$.

- Cost function considering all elements is

$$\begin{aligned} J(\theta) &= \frac{1}{m}\sum_{i=0}^{m} Cost(h_\theta(X^{(i)}), y^{(i)}) \\ &= -\frac{1}{m}\sum_{i=0}^{m} y^{(i)} log(h_\theta(X^{(i)})) + (1 - y^{(i)}) log(1 - h_\theta(X^{(i)})) \end{aligned}$$

- For logistic regression as well, gradient of $J(\theta)$ comes out to be

$$\frac{\partial J(\theta)}{\partial \theta_j} = \frac{1}{m}\sum_{i=1}^{m} (h_\theta(x^{(i)}) - y^{(i)}) x_j^{(i)}$$

- All θ_j parameters can be adjusted by using gradient descent in a similar way as linear regression.

2.2 Multiclass classification

- If data is to be classified into more than 2 classes, a technique called one-vs-all is used.
- If there are K classes, K logistic regression models, $h_\theta^{(k)}(X)$ are trained.
- Value of $h_\theta^{(k)}(X)$ represents the probability that an element belongs to class k.
- Class corresponding to maximum value of $h_\theta^{(k)}(X)$ is picked.

3 Regularization

- If there are too many features, and not many elements in the dataset, model may be too biased for the given dataset. It will fit the given points perfectly, but it'll fail to generalize, or it'll perform poorly on the test dataset. This problem is called as overfitting.
- To avoid this, some extra features can be manually removed, but it is hard to choose what features to keep and what to remove.
- Other method to remove this, is called regularization. For example, let's say the available dataset is approximately in a shape of parabola, but not exactly a parabola. So, we include quadratic, cubic, and 4^{th} degree terms.
  This causes the model to overfit to the data.

3.1 L2 Regularization

- So, let's say we *penalize* θ_3 and θ_4 by adding the terms $1000\theta_3 + 1000\theta_4$ to the cost function. This will result in bigger cost when either of θ_3 or θ_4 are big. So the values of coefficients of higher degree terms will automatically be less, which will result in less dramatic curve. This is less prone to overfitting.

- It's not always possible to choose the features to regularize, so as a convention, all parameters except θ_0 are penalized.
- Cost function with regularization looks like :

$$J(\theta) = \frac{1}{2m}\left[\sum_{i=1}^{m}(h_\theta(x^{(i)}) - y^{(i)})^2 + \lambda\sum_{j=1}^{n}\theta_j^2\right]$$

- For both linear and logistic regression,

$$\frac{\partial J(\theta)}{\partial \theta_j} = \frac{1}{m}\sum_{i=1}^{m}(h_\theta(x^{(i)}) - y^{(i)})x_j^{(i)} + \frac{\lambda}{m}\theta_j$$

- If value of λ is set too large, $\theta_j \approx 0$ for $j = 1$ to n. So, effective hypothesis will be $h_\theta(X) = \theta_0$, which won't represent the data accurately either. This problem is called underfitting. Underfitting can also occur if there aren't enough features to fit the data properly.

3.2 Dropout

- Dropout is randomly making some nodes in the hidden layers as 0.
- This makes the network smaller and makes it less prone to overfitting.
- As nodes are *turned off* randomly, network does not rely too much on one single feature.
- A value *keep_prob* is set. A random number between 0 and 1 is generated for each node, and if it is greater than keep_prob, activation is made 0.
- z values in the next layer are divided by keep_prob before processing to keep the value within expected range.
- Difficult to calculate the cost function as nodes are turned off randomly.

4 Neural Networks

- Many logistic regression model cascaded to eachother.
- A single logistic regression model can be thought of as a NN with n units in the input layer, 0 hidden layers, and K units in the output layer.
- These allow users to fit non-linear hypothesis without going for the polynomial features.
- Activation of unit i in layer l is represented by $a_i^{(l)}$.
- Matrix containing weights that map layer l to layer $l+1$ is represented as $\Theta^{(l)}$.
- Vector of linear combinations of previous layer is denoted by $z^{(l)} = \Theta^{(l-1)}a^{(l-1)}$, and $a^{(l)} = g(z^{(l)})$. So, $a^{(2)} = g(\Theta^{(1)}X)$, $a^{(3)} = g(\Theta^{(1)}a^{(2)})$, and so on. In a 3 layer NN, $h_\Theta(X) = a^{(3)}$. Here, $g(z)$ can be any activation function.
- In this case, $h_\Theta(X)$ is a vector of K elements. $h_\Theta(X)_k$ represents k^{th} element in that vector.
- Generally, an additional unit with activation 1 is added in every layer. It is called the bias unit.
- Cost function for NN is defined in a similar way as logistic regression.

- Cost of one single (i^{th}) element:

$$cost^{(i)} = -\sum_{k=1}^{K} y_k^{(i)} log(h_\theta(X^{(i)})_k) + (1 - y_k^{(i)}) log(1 - h_\theta(X^{(i)})_k)$$

- Cost function for m elements:

$$J(\Theta) = \frac{1}{m} \sum_{i=1}^{m} cost^{(i)}$$

$$J(\Theta) = -\frac{1}{m} \sum_{i=1}^{m} \sum_{k=1}^{K} y_k^{(i)} log(h_\theta(X^{(i)})_k) + (1 - y_k^{(i)}) log(1 - h_\theta(X^{(i)})_k)$$

- Cost function with regularization:

$$J(\Theta) = -\frac{1}{m} \sum_{i=1}^{m} \sum_{k=1}^{K} y_k^{(i)} log(h_\theta(X^{(i)})_k) + (1 - y_k^{(i)}) log(1 - h_\theta(X^{(i)})_k) + \frac{\lambda}{2m} \sum_{l=1}^{L} \sum_{i=1}^{n_{l+1}} \sum_{j=1}^{n_l} (\Theta_{ij}^{(l)})^2$$

- To train this network, all $\frac{\partial J(\Theta)}{\partial \Theta_{ij}^{(l)}}$ are needed.
- To evaluate these gradients, a method called *Back Propogation* is used.
- Some required notes:
  - Find $\delta_j^{(l)}$ for every node, which may be thought of as error in activation for j^{th} node in l^{th} layer of network.
  - $\delta^{(L)} = a^{(L)} - y = h_\Theta(X) - y$.
    Delta value for output layer is simply the difference between obtained output and expected output.
  - $\delta^{(l)} = (\Theta^{(l)})^T \delta^{(l+1)} .* g'(z^{(l)})$
    Where $g'(z)$ is first derivative of the activation function, $.*$ operator means element-wise multiplication of the elements.For sigmoid, $g'(z^{(l)}) = a^{(l)} .* (1 - a^{(l)})$.
  - $\delta^{(1)}$ is not calculated, as error in input cannot be defined.
  - $\frac{\partial J(\Theta)}{\partial \Theta_{ij}^{(l)}} = a_j^{(l)} * \delta_i^{(l+1)}$ for all elements.
- These steps are followed:
  - Set $\Delta_{ij}^{(l)} = 0$ for all l, i, j.
  - For all m elements (i denotes element number):
    - set $a^{(1)} = x^{(i)}$.
    - Forward propogate to find $a^{(l)}$ for $l = 1, 2, \ldots, L$.
    - Set $\delta^{(L)} = a^{(L)} - y^{(i)}$.
    - Find $\delta^{(L-1)}, \ldots, \delta^{(2)}$, using $\delta^{(l)} = (\Theta^{(l)})^T \delta^{(l+1)} .* g'(z^{(l)})$.
    - $\Delta_{ij}^{(l)} += a_j^{(l)} * \delta_i^{(l+1)}$
  - $\frac{\partial J(\Theta)}{\partial \Theta_{ij}^{(l)}} = \frac{1}{m} \Delta_{ij}^{(l)} + \lambda \Theta_{ij}^{(l)}$
    λ term is added for non bias units only, i.e. for $j \neq 0$.

- For more info on backprop, refer this, and this.
- In NN, all values of Θ parameters cannot be initialized to 0, or anything equal, as activations of different nodes in a layer is are linear combinations of activations of previous layer. Every node will be identical to every other node in that layer if all weights are equal. So, the weights need to be randomly initialized.

4.1 Gradient Checking

- Even after implementing backprop, it's a good practice to check if it is working correctly.
- To check,
$$\frac{\partial J(\Theta)}{\partial \Theta} = \frac{J(\Theta - \epsilon) - J(\Theta - \epsilon)}{2\epsilon}$$
This is used.
- To check $\frac{\partial J(\Theta)}{\partial \Theta_{ij}^{(l)}}$, value of $\Theta_{ij}^{(l)}$ is changed by ϵ, rest are left as they were. Cost is calculated with this new Θ, and approximate gradient is calculated.
- If this approximate gradient matches the one calculated using backprop, it is implemented correctly.
- This method of calculating gradient is computationally very slow, that's why backprop is used.

4.2 Activation Functions

1. Sigmoid:
   - $g(z) = \frac{1}{1+e^{-z}}$
   - Maps the input between 0 and 1, centered around 0.5.

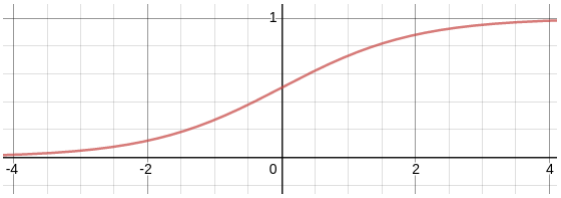


Figure 1: Sigmoid

2. Tanh:
   - $g(z) = \frac{e^z - e^{-z}}{e^z + e^{-z}}$
   - Maps the input between -1 and 1, centered around 0.
   - Almost always better than sigmoid.

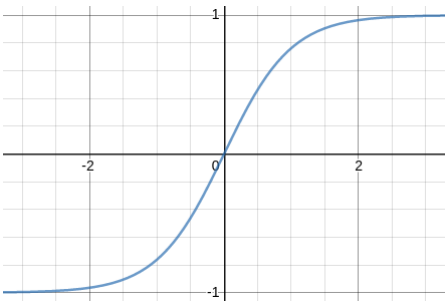


Figure 2: Tanh

3. Rectified Linear Unit (ReLU):
   - $g(z) = max(0, z)$
   - Makes all negative values 0, doesn't change the positive ones.
   - Most popular one.

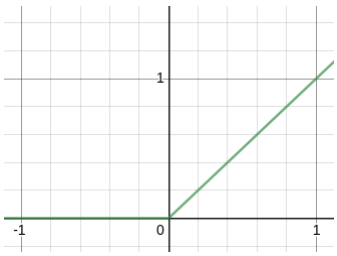


Figure 3: ReLU

4. LeakyReLU:
   - $g(z) = max(0.01 * z, z)$
   - Scales down negative values significantly, doesn't change positive values.
   - Improved version of ReLU.

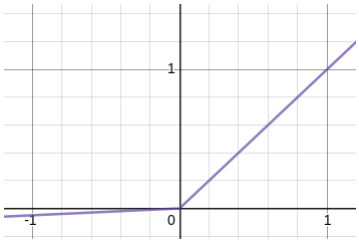


Figure 4: LeakyReLU

5. Softmax:
   - Used exclusively for output layer of a classifier.
   - Takes a vector as input and outputs a vector of probabilities.
   - $g(z) = \frac{e^z}{sum(e^z)}$
   - Sum of all the values in layer equals 1.
   - Cost for this is just the log of value corresponding to the correct class.

4.3 Vanishing Gradient

- For different activation functions like sigmoid or tanh, gradient of the function may approach 0 if magnitude of the inputs becomes too large.
- One of the solutions to this is to use ReLU for which the gradient remains 1 for positive inputs.
- If there are a lot of negative values, ReLU might be problematic as well. In that case, leakyReLU is used.

5 Diagnosing Errors

If a model is designed and trained, but it is not predicting the output as expected when new data is introduced to it, some changes need to be made. To decide the changes, there are some things that can be done.

- A metric to evaluate how good a model is performing is necessary to determine if changes are required to the model.
- Error on training set is not a good metric as a model may overfit the training data, so the training error will be very less, but it'll perform poorly on other inputs. That's why dataset is divided into 2 parts, training set and a test set. Model is evaluated on the basis of error in test set.
- Error is just the cost of the function with $\lambda = 0$. In classification problem, error can be the number of wrong predictions.

5.1 Degree vs Error, λ vs Error

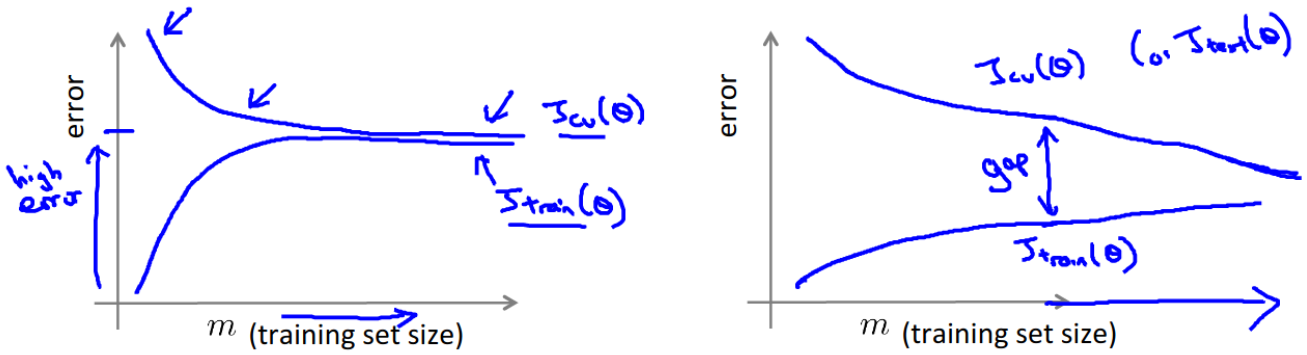
- While designing a model, degree of the hypothesis has to be chosen. If it is too less, model may underfit the data. If it is too large, model may overfit the data. So, different models of varying degrees are trained, and their test set errors are calculated.
- If both training error and test set erros are plotted w.r.t. degree of the hypothesis, training error will decrease as degree increases, as model will fit the data more and more closely. But the test set error will decrease for some time, and then it'll start increasing because of overfitting. Degree for which test set error is minimum is chosen.
- However, test set error is no longer a good metric, as the degree of hypothesis is *trained* using that dataset. So in practice, dataset is divided into 3 parts, training set, cross validation set, and test set. Degree of polynomial and the value of lambda are trained using cross validation set, and test set is purely for evaluation purpose.
- Similar thing can be done with the regularization parameter λ. For a model, if a training error and CV error vs λ plot is plotted, training error will increase with increase in λ. And CV error will start decreasing, hit a minima, and then start increasing again. Previous and later high value of error corresponds to overfitting and underfitting respectively.
- **Overfitting is called as high variance.**
  Underfitting is called as high bias.
- Whether the model has high variance or bias can be figured out by looking at their training and CV errors.
  - If $J_{CV}(\theta) \approx J_{train}(\theta)$, and both are high, that means model is performing poorly even on training data. This means the model is underfitting, i.e. has high bias.

– $J_{CV}(\theta) >> J_{train}(\theta)$ means model has fit the data properly, but it is not generalizing well. So, the model is overfitting, i.e. has high variance.

Depending on this, different parameters can be tweaked, and changes can be made.

5.2 Learning Curves

- Plot of error w.r.t. number of elements in training set is called a learning curve.
- Generally, training error will increase with increasing m, as it becomes more and more difficult to fit all the points in given dataset.
- And CV error will decrease with increase in m, as model is able to capture the pattern better and better.



(a) Underfitting (b) Overfitting

Figure 5: Learning Curves

- It can be seen from this figure that if a model is underfitting the data, getting more data is not going to help improve the performance, but in case of overfitting, using more training data may improve the model.
- Changes that can be done:

- Get more data for training set	to reduce overfitting
- Try smaller set of features	to reduce overfitting
- Get additional features	to reduce underfitting
- Try adding polynomial features	to reduce underfitting
- Decrease λ	to reduce underfitting
- Increase λ	to reduce overfitting

- In NNs, having many hidden layers/ hidden units is equivalent of having many polynomial features. It is prone to overfitting.
- Similarly, having too few hidden layers/ hidden units may result in underfitting.

6 Optimization Techniques

1. Gradient descent with momentum:
   - Let $d\theta$ be the gradient vector for all the parameters. In normal gradient descent, θ is updated proportional to $d\theta$, and prpopriotnality constant is the learning rate α.
   - In this version, a vector $v\theta$ is defined. It is initialized as 0, and in every iteration, it is updated as $v\theta = \beta * v\theta + (1 - \beta) * d\theta$ where $d\theta$ is gradient on that iteration.

- θ is updated as $\theta = \theta - \alpha * v\theta$.
- This is like applying a *low pass filter* to the gradients. Results in damping the oscillations in directions of gradients.
- Works faster than normal gradient descent, especially in mini-batch gradient descent.

2. RMSprop
   - Similar to above.
     $S_{d\theta}$ is initialized as 0 and updated as $S_{d\theta} = \beta * S_{d\theta} + (1 - \beta) * d\theta^2$ where $d\theta^2$ is element wise square.
   - θ is updated as $\theta = \theta - \alpha * \frac{d\theta}{\sqrt{S_{d\theta}} + \epsilon}$, where ϵ is a small value which is added to avoid division by a very small number.
   - Here every gradient is divided by RMS of magnitudes of the gradient.
3. Adaptive Moment Estimation (Adam)
   - Combination of both momentum and RMSprop
   - t^{th} low pass filtered output is divided by $1 - \beta^t$ to correct the initial offset in case of a large β.
   - Here, both $v\theta$ and $S_{d\theta}$ are initialized to 0. In t^{th} iteration,

$$
\begin{aligned}
v\theta &= \beta_1 * v\theta + (1 - \beta_1) * d\theta \\
S_{d\theta} &= \beta_2 * S_{d\theta} + (1 - \beta_2) * d\theta^2 \\
v\theta &= \frac{v\theta}{(1 - \beta_1^t)} \\
S_{d\theta} &= \frac{S_{d\theta}}{(1 - \beta_2^t)} \\
\theta &= \theta - \alpha * \frac{v\theta}{\sqrt{S_{d\theta}} + \epsilon}
\end{aligned}
$$

   - Generally, $\beta_1 = 0.9$ and $\beta_2 = 0.999$.

7 Convolutional Neural Network

-