CMPE 297, INSTRUCTOR: JORJETA JETCHEVA

LOGISTIC REGRESSION

PROBABILISTIC MACHINE LEARNING ELEMENTS

- Feature representation of the inputs
  - For each observation, we have a vector with a value for each feature
- Classification function
  - Function we would use to perform the classification
- Objective function for learning
  - A formulation that allows us to find the best parameters of the classification (focused on minimizing error on training examples)
- An algorithm for optimizing the objective function
  - An efficient way of finding the minimum value of the objective function
- Modeling phases: training & testing

LOGISTIC REGRESSION

- Uses the training set to learn a vector of weights w and a bias term b
  - Each weight is used to moderate the impact of a particular feature

$$z = (\sum_{i=1}^{n} w_i x_i) + b$$

This is an example for a single observation x, which is a vector with a value x_i for each feature i

- Or in vector form:

$$z = w \cdot x + b$$

BINARY LOGISTIC REGRESSION

$$z = w \cdot x + b$$

- Classification requires the output of the model to be a probability
  - Must range between 0 and 1, and
  - $p(y = 1) + p(y = 0) = 1$
- To convert z to a probability format, we pass it through the sigmoid (logistic) function σ:

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

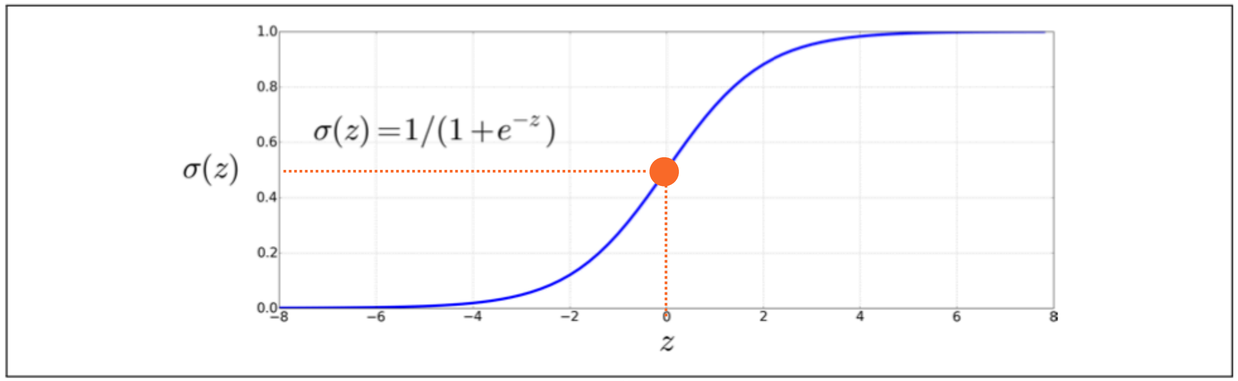


Figure 5.1 The sigmoid function $\sigma(z) = \frac{1}{1+e^{-z}}$ takes a real value and maps it to the range $[0, 1]$. It is nearly linear around 0 but outlier values get squashed toward 0 or 1.

- Using a decision boundary of 0.5, we get: $\text{decision}(x) = \begin{cases} 1 & \text{if } P(y = 1|x) > 0.5 \\ 0 & \text{otherwise} \end{cases}$

CLASSIFICATION OF SINGLE EXAMPLE IN THE CONTEXT OF SENTIMENT ANALYSIS

Var	Definition	Value in Fig. 5.2
x_1	count(positive lexicon words $\in$ doc)	3
x_2	count(negative lexicon words $\in$ doc)	2
x_3	$\begin{cases} 1 & \text{if "no"} \in \text{doc} \\ 0 & \text{otherwise} \end{cases}$	1
x_4	count(1st and 2nd pronouns $\in$ doc)	3
x_5	$\begin{cases} 1 & \text{if "!"} \in \text{doc} \\ 0 & \text{otherwise} \end{cases}$	0
x_6	log(word count of doc)	$\ln(66) = 4.19$

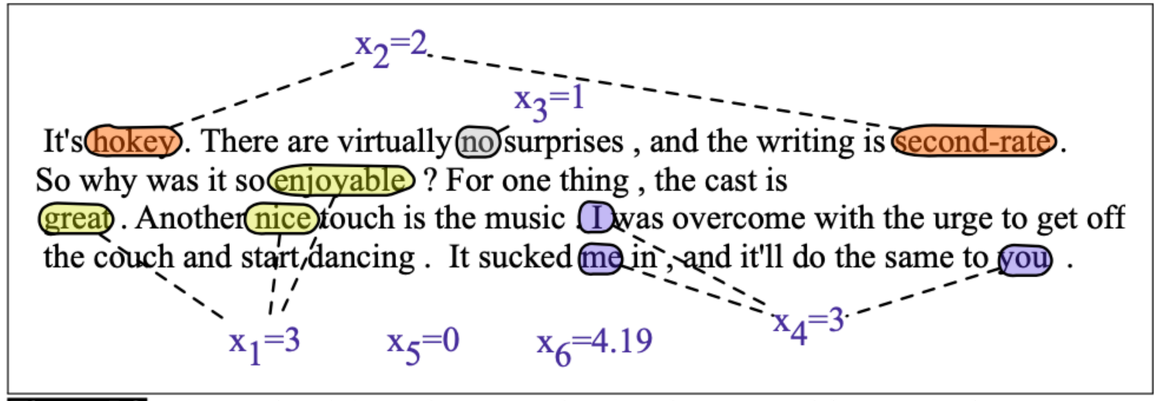


Figure 5.2 A sample mini test document showing the extracted features in the vector *x*.

Given $w = [2.5, -5.0, -1.2, 0.5, 2.0, 0.7]$, and $b = 0.1$, and the feature vector for x in the table above, x = [3,2,1,3,0,4.19], the classifier will compute the following:

$$\begin{aligned} p(+|x) = P(y=1|x) &= \sigma(\mathbf{w}\cdot\mathbf{x}+b) \\ &= \sigma([2.5,-5.0,-1.2,0.5,2.0,0.7]\cdot[3,2,1,3,0,4.19]+0.1) \\ &= \sigma(.833) \\ &= 0.70 \qquad (5.7) \\ p(-|x) = P(y=0|x) &= 1-\sigma(\mathbf{w}\cdot\mathbf{x}+b) \\ &= 0.30 \end{aligned}$$

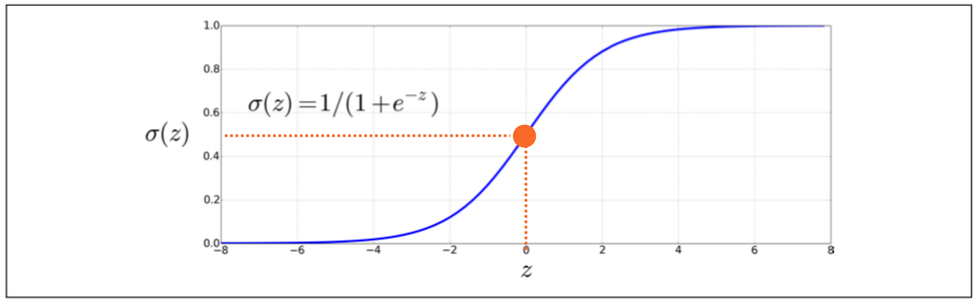


Figure 5.1 The sigmoid function $\sigma(z) = \frac{1}{1+e^{-z}}$ takes a real value and maps it to the range $[0,1]$. It is nearly linear around 0 but outlier values get squashed toward 0 or 1.

Note that the weight values indicate that x_1 is the most important feature for the + class, whereas x_2 is the most important feature for the negative class

PROCESSING OF DATA IS TYPICALLY DONE IN MATRIX FORMAT

To express processing of many examples at once, we use a matrix format.

Modern machines have optimizations for matrix operations and can perform them very efficiently.

m examples $x^{(1)} \ldots x^{(m)}$ & f features & f weights

$$\mathbf{X} = \begin{bmatrix} \mathbf{x}_1^{(1)} & \mathbf{x}_2^{(1)} & \ldots & \mathbf{x}_f^{(1)} \\ \mathbf{x}_1^{(2)} & \mathbf{x}_2^{(2)} & \ldots & \mathbf{x}_f^{(2)} \\ \mathbf{x}_1^{(3)} & \mathbf{x}_2^{(3)} & \ldots & \mathbf{x}_f^{(3)} \\ \ldots & & & \end{bmatrix}$$

The bias terms is a vector of length m with the same value for each dimension: $b = [b, b, \ldots, b]$

$$\begin{array}{lcccc} & \mathbf{y} & = & \mathbf{X} \quad \mathbf{w} & + \quad \mathbf{b} \\ \text{Dimensions} & (m \times 1) & & (m \times f)(f \times 1) & (m \times 1) \end{array}$$

Then we pass the result as input into the logistic function σ.

MULTINOMIAL LOGISTIC REGRESSION

MULTINOMIAL LOGISTIC REGRESSION

- Hard multi-class classification
  - We have K classes and one of them is the correct one (hard classification)
  - The output of the classifier is a 1-hot vector with 1 for the predicted class, and 0s for all other K-1 classes
- Alternative: Multinomial logistic regression, also known as Softmax regression

SOFTMAX FUNCTION

- The softmax function (a generalization of the sigmoid) takes a vector of K values and maps them to a probability distribution, resulting in K values that are between 0 and 1, that sum up to 1

Example: z = [0.6, 1.1, −1.5, 1.2, 3.2, −1.1] becomes softmax(z) = [0.055, 0.090, 0.006, 0.099, 0.74, 0.010]

$$\text{softmax}(\mathbf{z}) = \left[\frac{\exp(\mathbf{z}_1)}{\sum_{i=1}^{K}\exp(\mathbf{z}_i)}, \frac{\exp(\mathbf{z}_2)}{\sum_{i=1}^{K}\exp(\mathbf{z}_i)}, \ldots, \frac{\exp(\mathbf{z}_K)}{\sum_{i=1}^{K}\exp(\mathbf{z}_i)}\right]$$

- Note that when we expand z (recall that $z = w \cdot x + b$) we have a weight vector w_k and bias $b_{k'}$ for each class

$$p(\mathbf{y}_k = 1|\mathbf{x}) = \frac{\exp(\mathbf{w}_k \cdot \mathbf{x} + b_k)}{\sum_{j=1}^{K}\exp(\mathbf{w}_j \cdot \mathbf{x} + b_j)}$$

- We can represent the weights as a $K \times f$ matrix W, b is a $K \times 1$ -dimensional vector (x is a single example with dimensions $f \times 1$)

$$\hat{\mathbf{y}} = \text{softmax}(\mathbf{W}\mathbf{x} + \mathbf{b})$$

FEATURES IN BINARY VS. MULTINOMIAL LOGISTIC REGRESSION

- A feature in binary logistic regression influences the classifier towards one of the classes (and away from the other)
- In multinomial logistic regression, each feature can be thought of as providing "evidence" for each class expressed by the weight (can be positive or negative for more than 1 class)
  - E.g., the weights for the 3 classes (+, -, and neutral (0))

Feature	Definition	$\mathbf{w}_{5,+}$	$\mathbf{w}_{5,-}$	$\mathbf{w}_{5,0}$
$f_5(x)$	$\begin{cases} 1 & \text{if } \text{“!”} \in \text{doc} \\ 0 & \text{otherwise} \end{cases}$	3.5	3.1	−5.3

LOGISTIC REGRESSION VS. NAIVE BAYES

- Naive Bayes
  - Strong conditional independence assumptions which leads it to overestimating the evidence (of the input features), i.e., giving it a higher probability
    - Treats each feature as a separate feature with its own probability which gets multiplied into the formula (when there is inter-dependence, there is overlap in the probabilities)
  - Fast to train, and works well for smaller datasets
- Logistic Regression is more robust to correlated features
  - It will assign part of the weight to one of the overlapping/correlated features, and the rest to the other correlated feature
  - Serves as default approach for larger datasets

NORMALIZATION

FEATURE SCALING/NORMALIZATION

- Attribute transformation is when a function maps the entire set of values of a given attribute to a new set of replacement values such that each old value can be identified with one of the new values
  - E.g., x can be transformed as follows: x^k, $\log(x)$, or e^x

- It is common to normalize features so that they have comparable ranges
  - Used to adjust differences in scale of different attributes and thus ensure they attribute values that are larger do not affect the modeling process disproportionately

COMMON NORMALIZATION TECHNIQUES

- Standardization
  - Subtract the dataset mean and divide by the dataset standard deviation from each data point
  - Results in feature values with 0 mean & standard deviation of 1
  - Less susceptible to outliers but assumes data is normally distributed

$$norm\_x = \frac{x - mean(D)}{std(D)}$$

norm_x - value of the attribute after normalization
x - original value of the attribute
D - all data values in the dataset

- Min-Max scaling
  - From each data point, subtract the dataset minimum value, and divide by the difference between the max and min value across the dataset
  - All feature values will be in the range of [0,1]
  - Susceptible to outliers but does not make assumptions about the underlying distribution of the data

$$norm\_x = \frac{x - min(D)}{max(D) - min(D)}$$

TRAINING A BINARY LOGISTIC REGRESSION CLASSIFIER

TRAINING A BINARY LOGISTIC REGRESSION CLASSIFIER

- We need to find the logistic regression function $\sigma(w \cdot x + b)$ that best fits our data
- That amount of finding the values of w and b that parameterize this best fitting function
  - We define the best fitting logistic regression function as the one that has the **lowest difference** (or loss) between the predictions $\hat{y}$ it produces & the ground truth, y, we have in our data: $L(\hat{y}, y)$
  - $L(\hat{y}, y)$ for logistic regression is chosen to predict the correct class labels as more likely, i.e., we choose parameters w and b that maximize the log probability of the true y labels given the input x
    - The resulting loss function is **cross-entropy loss**

DERIVATION OF CROSS-ENTROPY LOSS

- We need to maximize $p(y|x)$, and since y is either 1 or 0, we can use a Bernoulli distribution formulation:

$$p(y|x) = \hat{y}^y(1-\hat{y})^{1-y}$$ ($\hat{y}$ is predicted value for the class, y is the ground truth, x is the input)

- When we take the log of the above, we get the following:

$$log\ p(y|x) = log[\hat{y}^y(1-\hat{y})^{1-y}] = ylog\hat{y} + (1-y)log(1-\hat{y})$$

- We typically are looking for a loss that we can minimize. To minimize p (instead of maximize), we just negate it, which gives us the cross-entropy (CE) loss:

$$L_{CE}(\hat{y}, y) = -\ log\ p(y|x) = -\ [ylog\hat{y} + (1-y)log(1-\hat{y})]$$

- Because we represent $\hat{y}$ as $\sigma(w \cdot x + b)$, we plug it in the loss function formula to get

$$L_{CE}(\hat{y}, y) = -\ [ylog\sigma(\mathbf{w} \cdot \mathbf{x} + b) + (1-y)log(1-\sigma(\mathbf{w} \cdot \mathbf{x} + b))]$$

- Then we use gradient descent to minimize L (find w and b at which it takes on its minimum value)

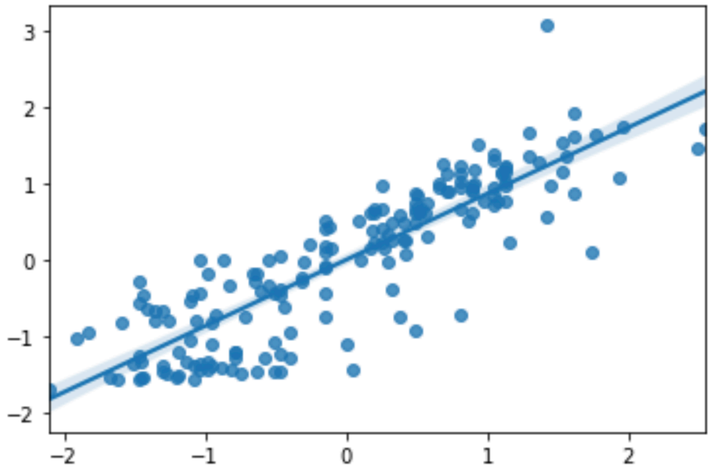
GRADIENT DESCENT

CALCULUS REVIEW: BASICS

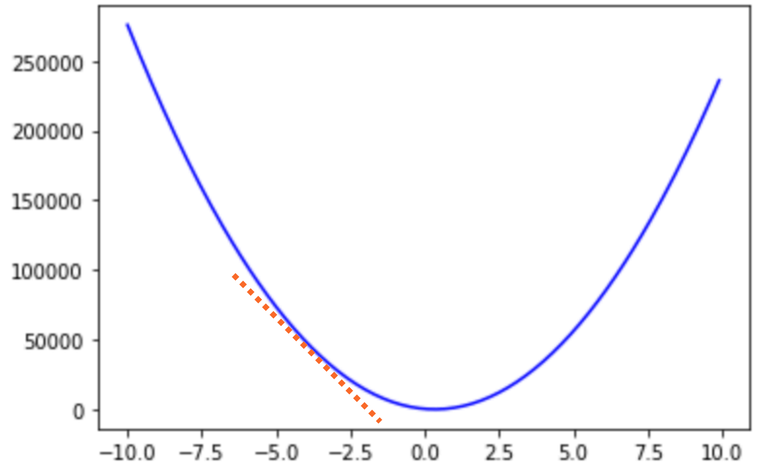
- Derivative is the "rate of change" of a function $y = f(x)$ wrt a variable (e.g., x)
  - It is the ratio of the change in the dependent (target) variable to that of the independent variable(s) (the attributes/features)

$$f(x)' = \triangle f = \frac{\partial f}{\partial x}$$

For a line, the derivative is the same as the slope.



For non-linear functions, the derivative measures the slope of the tangent line at any particular point

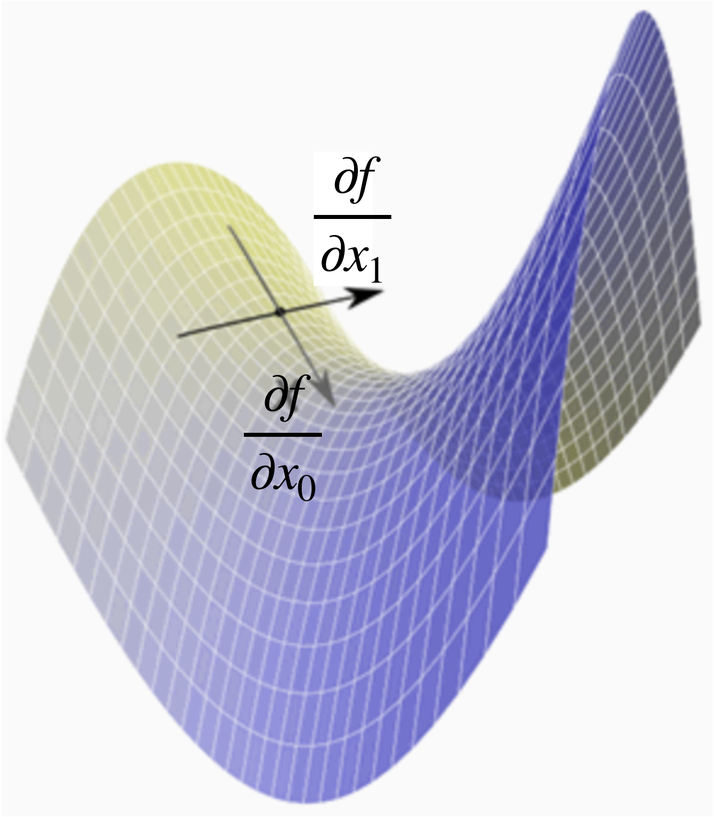


CALCULUS REVIEW (CONT).

- When we have a function that takes multiple input variables, we characterize it with a gradient vector: a vector where each entry is a partial derivative at point p wrt one of the input variables $x_1 \ldots x_n$

$$\triangle f(p) = \begin{bmatrix} \frac{\partial f}{\partial x_0}(p) \\ \frac{\partial f}{\partial x_1}(p) \\ \vdots \\ \frac{\partial f}{\partial x_n}(p) \end{bmatrix}$$

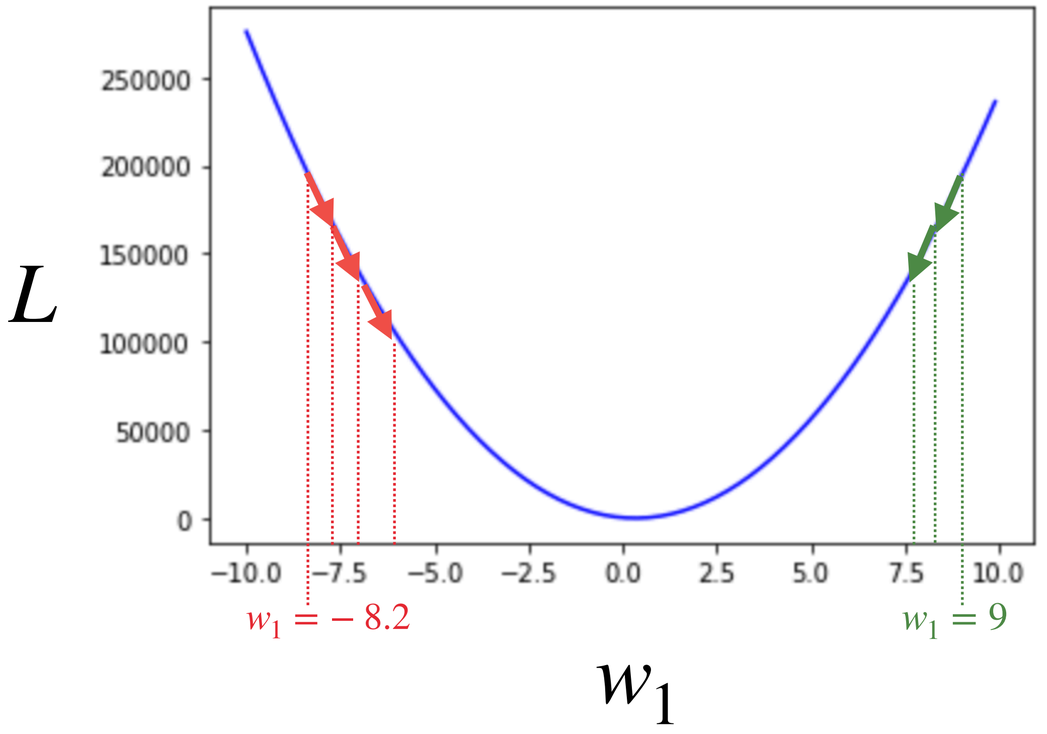
Indicates rate of change in each dimension



https://www.mathsisfun.com/calculus/derivatives-partial.html

GRADIENT DESCENT TO FIND THE MINIMUM VALUE OF A FUNCTION

- We use a simple quadratic loss function with a single parameter w1



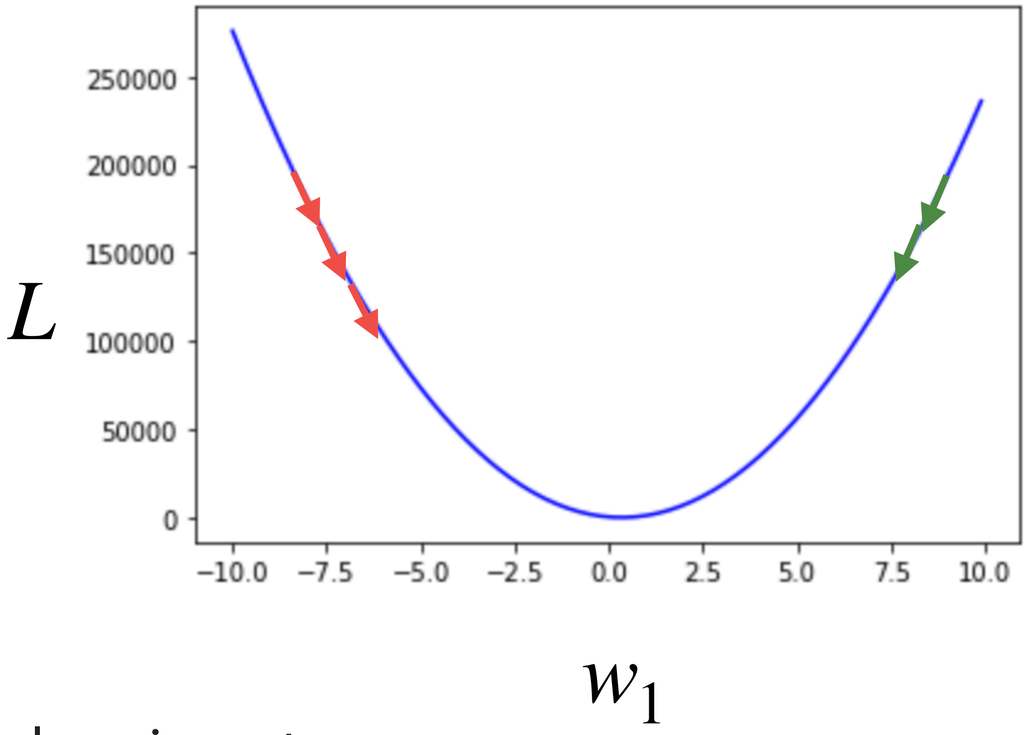
- We start with a random initial value for w_1
- We compute the gradient of L at w_1: $\triangle L = \frac{\partial L}{\partial w_1}$
  - If the gradient is negative, we increase w_1
  - If the gradient is positive, we reduce w_1
- Using the gradient to "give us directions", we take steps along the function towards its minimum

GRADIENT DESCENT COMPUTATION

Perform the below update for each w_i until the value of the loss function stops decreasing

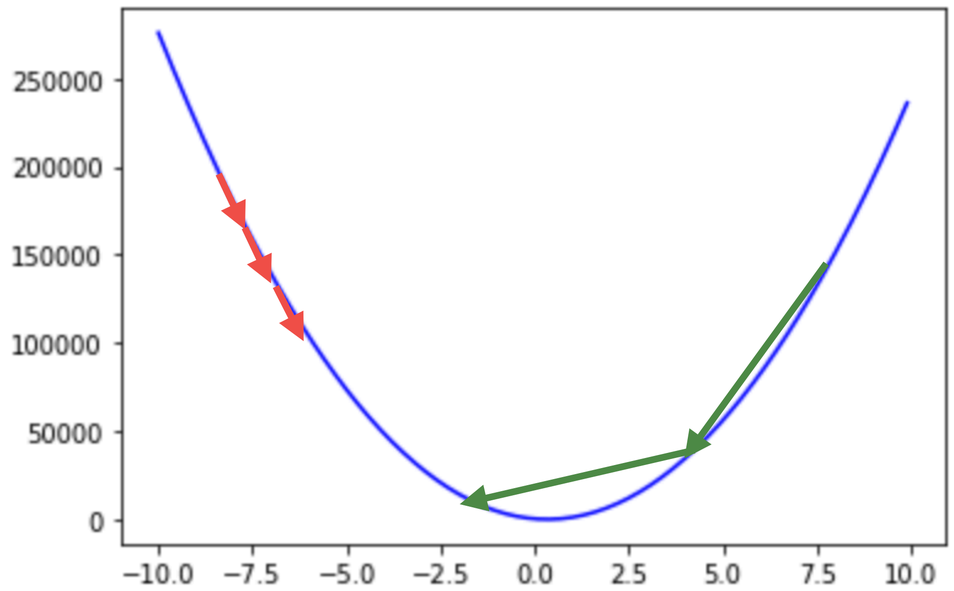
$$w_i = w_i - \alpha \frac{\partial L}{\partial w_i}$$

In our example, there is only one weight, w_1, but in others there may be multiple weights, e.g., $w_1, w_2, \ldots, w_n$



- The size of each step depends on a parameter α called the learning rate

LEARNING RATE



$$w_i = w_i - \alpha \frac{\partial L}{\partial w_i}$$

- If the learning rate is too small, convergence take a long time
- If the learning rate is too big, we can overshoot the minimum
- The learning rate can be constant or decay over time

OPTIMIZATIONS TO REDUCE COMPUTATIONAL COMPLEXITY

$$w_i = w_i - \alpha \frac{\partial L}{\partial w_i}$$

- So far we assumed batch gradient descent, where at each step
  - Gradient used to update the weight(s) is the average over the training examples
- Stochastic gradient descent, where at each step
  - Gradient used to update the weight(s) is for a single training example
  - We can stop it after some number of examples
- Mini-batch gradient descent, at each step
  - Gradient used to update the weight(s) is averaged over a batch of examples (e.g., 100)
  - Most common

LOGISTIC REGRESSION MODEL TRAINING SUMMARY

SUMMARY OF LOGISTIC REGRESSION TRAINING STEPS

- Formulate loss function
- Find w and b that minimize the loss function
- w and b are the parameters of the best fitting logistic regression model

REGULARIZATION

REDUCING MODEL COMPLEXITY: REGULARIZATION

Regularization is a family of techniques intended to reduce model complexity and thus reduce overfitting, typically by reducing the number and/or magnitude of model parameters

https://medium.com/@kylecaron/introduction-to-linear-regression-part-4-sci-kit-learn-the-bias-variance-trade-off-and-3520a0343764

REDUCING MODEL COMPLEXITY: REGULARIZATION

https://medium.com/@kvlecaron/introduction-to-linear-regression-part-4-sci-kit-learn-the-bias-variance-

- More complex models tend to have more parameters/coefficients/weights, e.g.,
  - Linear models have 2 parameters, a and b: $y = ax+b$
  - Quadratic models have 3 parameters, a, b and c: $y = ax^2 + bx + c$
  - Polynomial models have 3+, e.g., polynomial regression: $y = \beta_0 + \beta_1 x + \beta_2 x^2 + \beta_3 x^3 + \cdots + \beta_n x^n + \varepsilon$

- Among models with a certain number of parameters, those that have parameters with smaller values tend to be simpler, and those that have parameters with larger values tend to be more complex
  - A small parameter value, de-emphasizes the effect of the corresponding feature (e.g., that the parameter is multiplied with) – conceptually this result in shallower kinks in the model curve and therefore less complexity
  - When some of the coefficients are very small, it is as if we are using a smaller degree (smoother) polynomial with those parameters being negligible (numerically very small, e.g., close to 0)

$y = ax^2 + bx + c$ becomes equivalent to $y = bx + c$ for small values of a

L1 REGULARIZATION

$$L(w) = \sum_{i=1}^{n} [y_i - (w_0 + w_1 x_i)]^2$$

L1 regularization, adds a penalty term to the loss function as follows:

Our original loss function

Regularized Loss = $L(w) + \lambda \sum_j |w_j|$

penalty term

λ: a parameter to control the level of regularization

w_j: model coefficients (or weights) of the model we are trying regularize

Among parameters that give you models with minimum/low loss, choose the one that has the lowest-valued sum of model coefficients

Effect of L1 regularization

- Causes coefficients of features that are not important to be close to 0 or equal to 0
  - thus reducing sensitivity of the model to features with low importance
  - Often eliminating features altogether (when a coeff. = 0): Fewer features => lower complexity

L2 REGULARIZATION

L2 regularization, adds a penalty term to the loss function as follows:

Our original loss function

$$\textbf{Regularized Loss} = L(w) + \lambda \sum_j |w_j|^2$$

penalty term

λ: a parameter to control the level of regularization

w_j: model coefficients (or weights) of the model we are trying regularize

Among parameters that give you models with minimum/ low loss, choose the one that has the lowest-valued sum of the *squares* of the model coefficients

Effect of L2 regularization

- Causes coefficients of features that are not important to be smaller
  - thus reducing sensitivity of the model to features with low importance

Example of why L2 does not tend to zero out weights whereas L1 does:

- Given a single weight , $w = 0.02$, we get $\lambda * 0.02$ penalty term in L1, and $\lambda * 0.0004$ in L2 (typical default $\lambda=1$)
- As weights get smaller (< 1), the penalty term in L2 is already very small and close to 0, so L2 does not need to zero out weights in order to make the penalty term small (whereas L1 does)

L1 VS. L2 REGULARIZATION

- If data is believed to to have a handful of prominent features we'd use L1 (especially in cases of high dimensional data with many irrelevant features)
  - L1 is a form of dimensionality reduction
- Otherwise we'd use L2 regularization

- Some regularized regressions have their own name
  - Lasso regression - L1 regularization
  - Ridge Regression - L2 regularization

TRAINING A MULTINOMIAL LOGISTIC REGRESSION CLASSIFIER

GENERALIZATION TO K > 2

- Binary logistic regression cross entropy (CE) loss:

$$L_{CE}(\hat{y}, y) = -\log p(y \mid x) = -\left[y \log \hat{y} + (1-y)\log(1-\hat{y})\right]$$

- Instead of 2 terms as in above, in multinomial logistic regression we have K terms and both y and $\hat{y}$ are vectors

$$L_{CE}(\hat{\mathbf{y}}, \mathbf{y}) = -\sum_{k=1}^{K} \mathbf{y_k} \log \hat{\mathbf{y}}_\mathbf{k}$$

- The values of $\hat{y}$ for the incorrect classes are set to 0, so the only remaining term is $\hat{y}_c$ where c stands for the index of the correct class, and we can express the overall loss as follows:

$$L_{CE}(\hat{\mathbf{y}}, \mathbf{y}) = -\sum_{k=1}^{K} \mathbf{y_k} \log \hat{\mathbf{y}}_\mathbf{k} = -\log \hat{\mathbf{y}}_\mathbf{c} = -\log p(\hat{\mathbf{y}}_\mathbf{c} = \mathbf{1} \mid x) = -\log \frac{exp(\mathbf{w_c} \cdot \mathbf{x} + b_c)}{\sum_{j=1}^{K} exp(\mathbf{w_j} \cdot \mathbf{x} + b_j)}$$

INTERPRETABILITY

INTERPRETABILITY

- The values of the weights in logistic regression are an indication of which features are important
- When features are human-designed, we can therefore have an intuitively meaningful analysis for how the classifier is making decisions
  - E.g., length of product review may be a strong indication of a negative review