Linear Regression (Contd), Linear Classification

CS771: Introduction to Machine Learning

Piyush Rai

Gradient Descent for Linear/Ridge Regression

- Just use the GD algorithm with the gradient expressions we derived
- Iterative updates for linear regression will be of the form

Also, we usually work with average gradient so the gradient term is divided by $\boldsymbol{N}$

$$\boldsymbol{w}^{(t+1)} = \boldsymbol{w}^{(t)} - \eta_t \boldsymbol{g}^{(t)}$$

Note the form of each term in the gradient expression update: Amount of current $\boldsymbol{w}$'s error on the $\boldsymbol{n}^{th}$ training example multiplied by the input $\boldsymbol{x}_n$

Unlike the closed form solution $(\boldsymbol{X}^\top \boldsymbol{X})^{-1}\boldsymbol{X}^\top \boldsymbol{y}$ of least squares regression, here we have iterative updates but do not require the expensive matrix inversion of the $D \times D$ matrix $\boldsymbol{X}^\top \boldsymbol{X}$

$$= \boldsymbol{w}^{(t)} - \eta_t \sum_{n=1}^{N} \left(y_n - \boldsymbol{w}^{(t)^\top} \boldsymbol{x}_n \right) \boldsymbol{x}_n$$

- Similar updates for ridge regression as well (with the gradient expression being slightly different; left as an exercise)

- More on iterative optimization methods later

ℓ_2 regularization and "Smoothness"

- The regularized objective we minimized is

$$L_{reg}(\boldsymbol{w}) = \sum_{n=1}^{N} (y_n - \boldsymbol{w}^\top \boldsymbol{x}_n)^2 + \lambda \boldsymbol{w}^\top \boldsymbol{w}$$

Remember – in general, weights with large magnitude are bad since they can cause overfitting on training data and may not work well on test data

- Minimizing $L_{reg}(\boldsymbol{w})$ w.r.t. $\boldsymbol{w}$ gives a solution for $\boldsymbol{w}$ that
  - Keeps the training error small
  - Has a small ℓ_2 squared norm $\boldsymbol{w}^\top \boldsymbol{w} = \sum_{d=1}^{D} w_d^2$

Good because, consequently, the individual entries of the weight vector $\boldsymbol{w}$ are also prevented from becoming too large

Not a "smooth" model since its test data predictions may change drastically even with small changes in some feature's value

- Small entries in $\boldsymbol{w}$ are good since they lead to "smooth" models

$\boldsymbol{x}_n =$	1.2	0.5	2.4	0.3	0.8	0.1	0.9	2.1
$\boldsymbol{x}_m =$	1.2	0.5	2.4	0.3	$0.8+\epsilon$	0.1	0.9	2.1

Exact same feature vectors only differing in just one feature by a small amount

$y_n = 0.8$

$y_m = 100$

Very different outputs though (maybe one of these two training ex. is an outlier)

A typical $\boldsymbol{w}$ learned without ℓ_2 reg.

3.2	1.8	1.3	2.1	10000	2.5	3.1	0.1

Just to fit the training data where one of the inputs was possibly an outlier, this weight became too big. Such a weight vector will possibly do poorly on normal test inputs

Other Ways to Control Overfitting

Note that optimizing loss functions with such regularizers is usually harder than ridge reg. but several advanced techniques exist (we will see some of those later)

- Use a regularizer $R(\boldsymbol{w})$ defined by other norms, e.g.,

ℓ_1 norm regularizer

$$\|\boldsymbol{w}\|_1 = \sum_{d=1}^{D} |w_d|$$

$$\|\boldsymbol{w}\|_0 = \#\text{nnz}(\boldsymbol{w})$$

ℓ_0 norm regularizer (counts number of nonzeros in $\boldsymbol{w}$

When should I used these regularizers instead of the ℓ_2 regularizer?

Automatic feature selection? Wow, cool!!! But how exactly?

Use them if you have a very large number of features but many irrelevant features. These regularizers can help in automatic feature selection

Using such regularizers gives a sparse weight vector $\boldsymbol{w}$ as solution (will see the reason in detail later)

sparse means many entries in $\boldsymbol{w}$ will be zero or near zero. Thus those features will be considered irrelevant by the model and will not influence prediction

- Use non-regularization based approaches
  - Early-stopping (stopping training just when we have a decent val. set accuracy)
  - Dropout (in each iteration, don't update some of the weights)
  - Injecting noise in the inputs

All of these are very popular ways to control overfitting in deep learning models. More on these later when we talk about deep learning

Linear Regression as Solving System of Linear Eqs

- The form of the lin. reg. model $\boldsymbol{y} \approx \boldsymbol{X}\boldsymbol{w}$ is akin to a system of linear equation
- Assuming N training examples with D features each, we have

First training example: $y_1 = x_{11}w_1 + x_{12}w_2 + \ldots + x_{1D}w_D$

Second training example: $y_2 = x_{21}w_1 + x_{22}w_2 + \ldots + x_{2D}w_D$

$\vdots$

N-th training example: $y_N = x_{N1}w_1 + x_{N2}w_2 + \ldots + x_{ND}w_D$

Note: Here x_{nd} denotes the d^{th} feature of the n^{th} training example

N equations and D unknowns here $(w_1, w_2, \ldots, w_D)$

- Usually we will either have $N > D$ or $N < D$
  - Thus we have an underdetermined $(N < D)$ or overdetermined $(N > D)$ system
  - Methods to solve over/underdetermined systems can be used for lin-reg as well
  - Many of these methods don't require expensive matrix inversion

Solving lin-reg as system of lin eq.

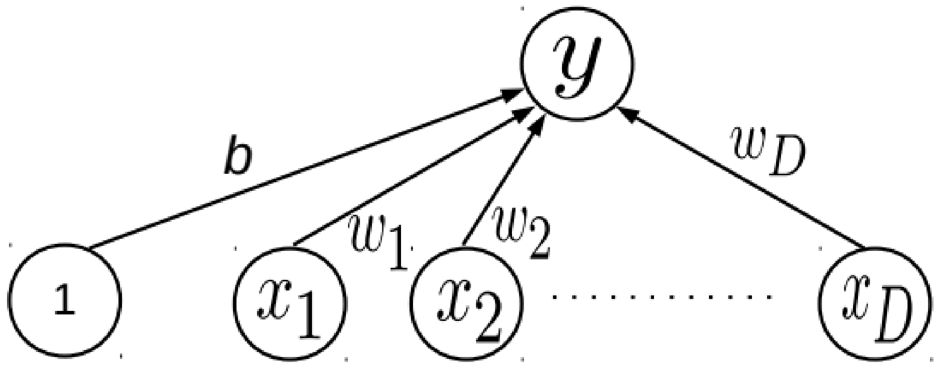
$$\boldsymbol{w} = (\boldsymbol{X}^\top \boldsymbol{X})^{-1}\, \boldsymbol{X}^\top \boldsymbol{y} \implies \boxed{\boldsymbol{A}\boldsymbol{w} = \boldsymbol{b}} \text{ where } \boldsymbol{A} = \boldsymbol{X}^\top \boldsymbol{X}, \text{ and } \boldsymbol{b} = \boldsymbol{X}^\top \boldsymbol{y}$$

Now solve this!

System of lin. Eqns with D equations and D unknowns

The bias term

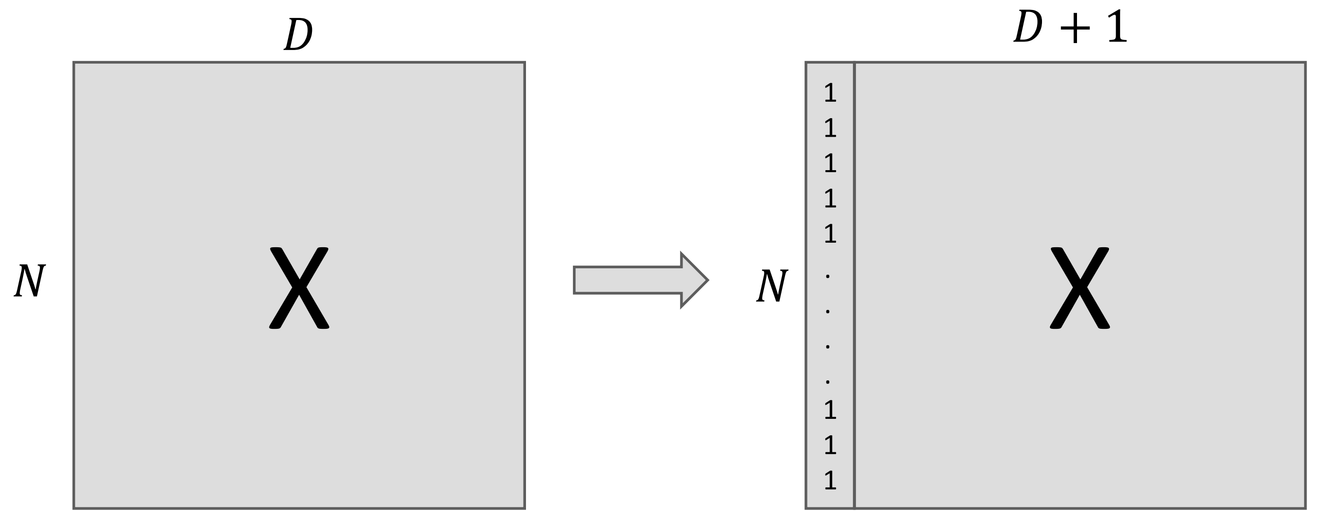
- Linear models usually also have a bias term $\boldsymbol{b}$ in addition to the weights



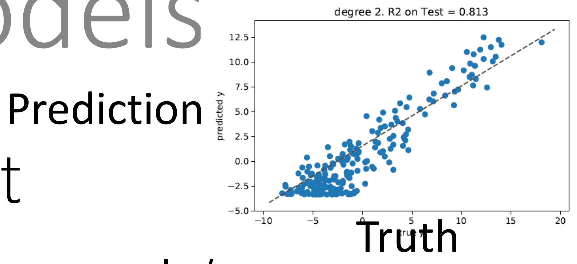
$$y = \sum_{d=1}^{D} w_d x_d + b = \boldsymbol{w}^\top \boldsymbol{x} + b$$

Can append a constant feature "1" for each input and again rewrite as $\boldsymbol{y} = \tilde{\boldsymbol{w}}^\top \tilde{\boldsymbol{x}}$ where now both $\tilde{\boldsymbol{x}} = [1, \boldsymbol{x}]$ and $\tilde{\boldsymbol{w}} = [b, \boldsymbol{w}]$ are in $\mathbb{R}^{D+1}$

We will assume the same and omit the explicit bias for simplicity of notation



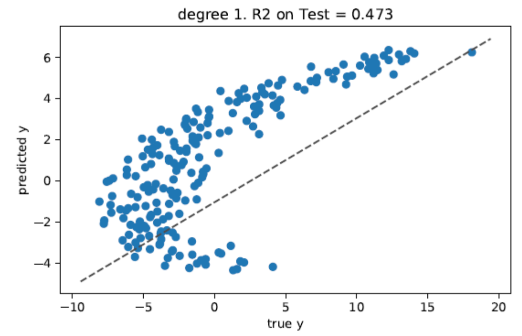
Evaluation Measures for Regression Models



- Plotting the prediction $\hat{y}_n$ vs truth y_n for the validation/test set
- Mean Squared Error (MSE) and Mean Absolute Error (MAE) on val./test set

$$MSE = \frac{1}{N}\sum_{n=1}^{N}(y_n - \hat{y}_n)^2 \qquad MAE = \frac{1}{N}\sum_{n=1}^{N}|y_n - \hat{y}_n|$$

Plots of true vs predicted outputs and R^2 for two regression models



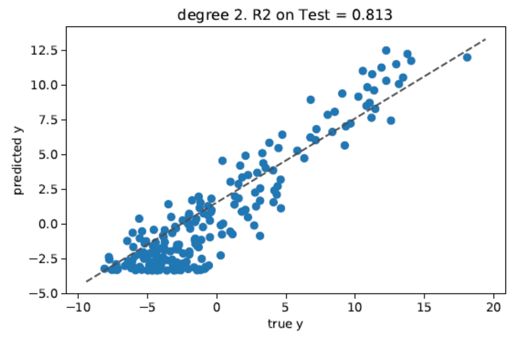
- RMSE (Root Mean Squared Error) $\triangleq \sqrt{MSE}$
- Coefficient of determination or R^2

$$R^2 = 1 - \frac{\sum_{n=1}^{N}(y_n - \hat{y}_n)^2}{\sum_{n=1}^{N}(y_n - \bar{y})^2}$$

"relative" error w.r.t. a model that makes a constant prediction $\bar{y}$ for all inputs

A "base" model that always predicts the mean $\bar{y}$ will have $R^2 = 0$ and the perfect model will have $R^2 = 1$. Worse than base models can even have negative R^2

$\bar{y}$ is empirical mean of true responses, i.e., $\frac{1}{N}\sum_{n=1}^{N} y_n$



Pic from MLAPP (Murphy)

Linear Models for Classification

Linear Models for Classification

- A linear model $\boldsymbol{y} = \boldsymbol{w}^\top \boldsymbol{x}$ can also be used in classification
- For binary classification, can treat $\boldsymbol{w}^\top \boldsymbol{x}_n$ as the "score" of input $\boldsymbol{x}_n$ and either
  - Threshold the score to get a binary label

$$y_n = \text{sign}(\boldsymbol{w}^\top \boldsymbol{x}_n)$$

Large positive score means positive label, otherwise negative label

Note that $\log \frac{\mu_n}{1-\mu_n} = \boldsymbol{w}^\top \boldsymbol{x}_n$ (the score) is also called the log-odds ratio, and often also logits

  - Convert the score into a probability

$$\mu_n = p(y = 1|\boldsymbol{x}_n, \boldsymbol{w}) = \sigma(\boldsymbol{w}^\top \boldsymbol{x}_n)$$

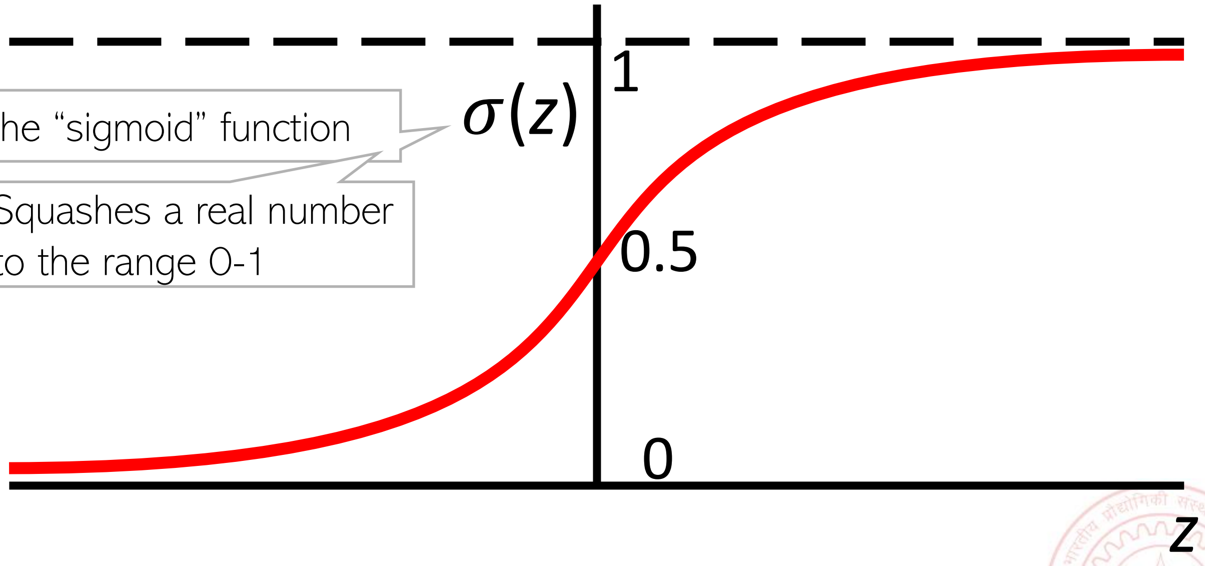
$$= \frac{1}{1 + \exp(-\boldsymbol{w}^\top \boldsymbol{x}_n)}$$

$$= \frac{\exp(\boldsymbol{w}^\top \boldsymbol{x}_n)}{1 + \exp(\boldsymbol{w}^\top \boldsymbol{x}_n)}$$

Popularly known as "logistic regression" (LR) model (misnomer: it is not a regression model but a classification model), a probabilistic model for binary classification

The "sigmoid" function

Squashes a real number to the range 0-1

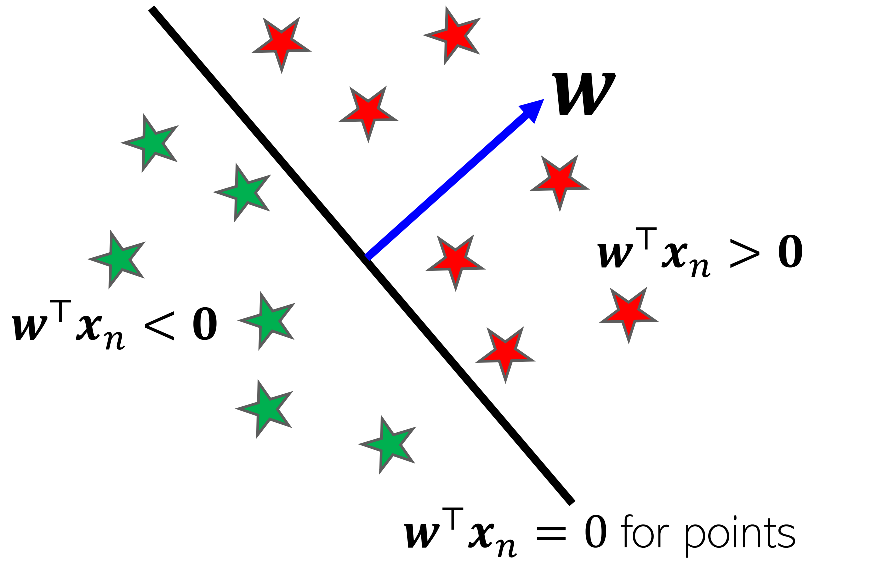


- Note: In LR, if we assume the label y_n as -1/+1 (not 0/1) then we can write

$$p(y_n|\boldsymbol{w}, \boldsymbol{x}_n) = \frac{1}{1 + \exp(-y_n \boldsymbol{w}^\top \boldsymbol{x}_n)} = \sigma(y_n \boldsymbol{w}^\top \boldsymbol{x}_n)$$

Linear Models: The Decision Boundary

- Decision boundary is where the score $\boldsymbol{w}^\top \boldsymbol{x}_n$ changes its sign


$\boldsymbol{w}$
$\boldsymbol{w}^\top \boldsymbol{x}_n > \mathbf{0}$
$\boldsymbol{w}^\top \boldsymbol{x}_n < \mathbf{0}$

$\boldsymbol{w}^\top \boldsymbol{x}_n = \mathbf{0}$ for points at the decision boundary

- Decision boundary is where both classes have equal probability for the input $\boldsymbol{x}_n$

- For logistic reg, at decision boundary

$$p(y_n = 1|\boldsymbol{w}, \boldsymbol{x}_n) = p(y_n = 0|\boldsymbol{w}, \boldsymbol{x}_n)$$

$$\frac{\exp(\boldsymbol{w}^\top \boldsymbol{x}_n)}{1 + \exp(\boldsymbol{w}^\top \boldsymbol{x}_n)} = \frac{1}{1 + \exp(\boldsymbol{w}^\top \boldsymbol{x}_n)}$$

$$\exp(\boldsymbol{w}^\top \boldsymbol{x}_n) = 1$$

$$\boldsymbol{w}^\top \boldsymbol{x}_n = 0$$

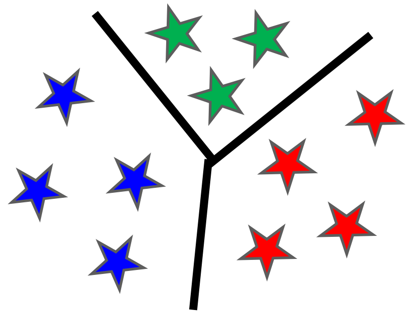
- Therefore, both views are equivalent

Linear Models for (Multi-class) Classification

- If there are $K > 2$ classes, we use K weight vectors $\{\boldsymbol{w}_i\}_{i=1}^K$ to define the model

$D \times K$ weight matrix

$$\boldsymbol{W} = [\boldsymbol{w}_1, \boldsymbol{w}_2, \dots, \boldsymbol{w}_K]$$



- The prediction rule is as follows

$$y_n = \text{argmax}_{i \in \{1,2,\dots,K\}} \boldsymbol{w}_i^\top \boldsymbol{x}_n$$

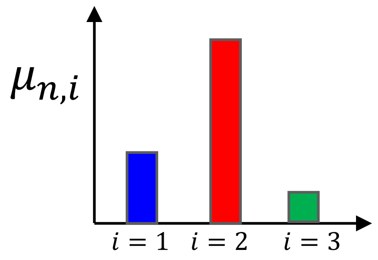
- Can think of $\boldsymbol{w}_i^\top \boldsymbol{x}_n$ as the score/similarity of the input w.r.t. the $\boldsymbol{i}^{th}$ class
- Can also use these scores to compute probability of belonging to each class

"softmax" classification

Multi-class extension of logistic regression

$$\mu_{n,i} = p(y_n = i | \boldsymbol{W}, \boldsymbol{x}_n) = \frac{\exp(\boldsymbol{w}_i^\top \boldsymbol{x}_n)}{\sum_{j=1}^K \exp(\boldsymbol{w}_j^\top \boldsymbol{x}_n)}$$

Probability of $\boldsymbol{x}_n$ belonging to class i



Note: Just like logistic regression, the scores $\boldsymbol{w}_i^\top \boldsymbol{x}_n$ are called logits (K logits in this case)

$$\boldsymbol{\mu}_n = [\mu_{n,1}, \mu_{n,2}, \dots, \mu_{n,K}]$$

$$\sum_{i=1}^K \mu_{n,i} = 1$$

Note: We actually need only $K-1$ weight vectors in softmax classification. Think why?

Vector of probabilities of $\boldsymbol{x}_n$ belonging to each of the K classes

Class $\boldsymbol{i}$ with largest $\boldsymbol{w}_i^\top \boldsymbol{x}_n$ has the largest probability

Probabilities must sum to 1

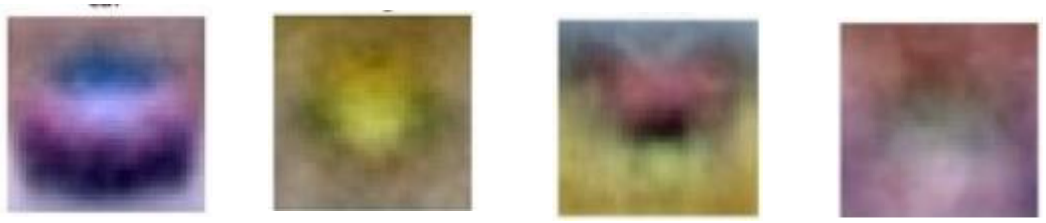
Linear Classification: Interpreting weight vectors

- Recall that multi-class classification prediction rule is

$$y_n = \text{argmax}_{i \in \{1,2,\ldots,K\}} \boldsymbol{w}_i^\top \boldsymbol{x}_n$$

- Can think of $\boldsymbol{w}_i^\top \boldsymbol{x}_n$ as the score of the input for the i^{th} class (or similarity of $\boldsymbol{x}_n$ with $\boldsymbol{w}_i$)
- Once learned (we will see the methods later), these K weight vectors (one for each class) can sometimes have nice interpretations, especially when the inputs are images

The learned weight vectors of each of the 4 classes "unflattened" and visualized as images – they kind of look like a "average" of what the images from that class should look like


$\boldsymbol{w}_{car}$ $\boldsymbol{w}_{frog}$ $\boldsymbol{w}_{horse}$ $\boldsymbol{w}_{cat}$

That's why the dot product of each of these weight vectors with an image from the correct class will be expected to be the largest

These images sort of look like class prototypes if I were using LwP ☺

Yeah, "sort of". ☺
No wonder why LwP (with Euclidean distances) acts like a linear model. ☺

Loss Functions for Classification

- Assume true label to be $y_n \in \{0,1\}$ and the score of a linear model to be $\boldsymbol{w}^\top \boldsymbol{x}_n$
- One possibility is to use squared loss just like we used in regression

$$l(y_n, \boldsymbol{w}^\top \boldsymbol{x}_n) = (y_n - \boldsymbol{w}^\top \boldsymbol{x}_n)^2$$

- Will be easy to optimize (same solution as the regression case)
- Can also consider other loss functions used in regression
  - Basically, pretend that the binary label is actually a continuous value and treat the problem as regression where the output can only be one of two possible values
- However, regression loss functions aren't ideal since $\boldsymbol{y}_n$ is discrete (binary/categorical)
- Using the score $\boldsymbol{w}^\top \boldsymbol{x}_n$ or the probability $\mu_n = \sigma(\boldsymbol{w}^\top \boldsymbol{x}_n)$ of belonging to the positive class, we have specialized loss function for binary classification

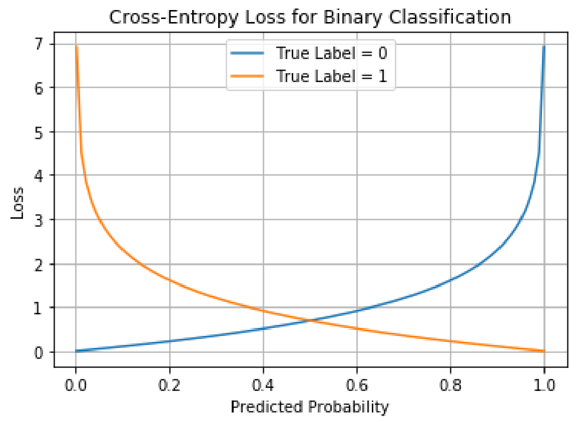
Loss Functions for Classification: Cross-Entropy

- Cross-entropy (CE) is a popular loss function for binary classification. Used in logistic reg.
- Assuming true $y_n \in \{0,1\}$ and $\mu_n = \sigma(\boldsymbol{w}^\top \boldsymbol{x}_n)$ as predicted prob of $y_n = 1$, CE loss is

$$L(\boldsymbol{w}) = -\left[\sum_{n=1}^{N} y_n \log \mu_n + (1 - y_n)\log(1 - \mu_n)\right]$$

Very large loss if y_n is 1and μ_n close to 0, or y_n is 0and μ_n close to 1

This is precisely what we want from a good loss function for binary classification



- For multi-class classification, the CE loss is defined as

$$L(\boldsymbol{W}) = -\sum_{n=1}^{N}\sum_{i=1}^{K} y_{n,i} \log \mu_{n,i}$$

CE loss is also convex in $\boldsymbol{w}$ (can prove easily using definition of convexity; will see later). Therefore unique solution is obtained when we minimize it

$y_{n,i} = 1$ if true label of $\boldsymbol{x}_n$ is class i and 0 otherwise. $\mu_{n,i}$ is the predicted probability of $\boldsymbol{x}_n$ belonging to class i

Note: Sometimes we divide the loss function (not just CE but others too like squared loss) by the number of training examples N (doesn't make a difference to the solution; just a scaling factor. All relevant quantities, such as gradients will also get divided by N

Cross-Entropy Loss: The Gradient

- The expression for the gradient of binary cross-entropy loss

Note the μ_n is a function of $\boldsymbol{w}$

$$\boldsymbol{g} = \nabla_{\boldsymbol{w}} L(\boldsymbol{w}) = -\sum_{n=1}^{N} (y_n - \mu_n)\, \boldsymbol{x}_n$$

Using this, we can now do gradient descent to learn the optimal $\boldsymbol{w}$ for logistic regression:

$$\boldsymbol{w}^{(t+1)} = \boldsymbol{w}^{(t)} - \eta_t \boldsymbol{g}^{(t)}$$

Note the form of each term in the gradient expression: Amount of current $\boldsymbol{w}$'s error in predicting the label of the $\boldsymbol{n}^{th}$ training example multiplied by the input $\boldsymbol{x}_n$

- The expression for the gradient of multi-class cross-entropy loss

Need to calculate the gradient for each of the K weight vectors

$$\boldsymbol{g}_i = \nabla_{\boldsymbol{w}_i} L(\boldsymbol{W}) = -\sum_{n=1}^{N} (y_{n,i} - \mu_{n,i})\, \boldsymbol{x}_n$$

Using these gradients, we can now do gradient descent to learn the optimal $\boldsymbol{W} = [\boldsymbol{w}_1, \boldsymbol{w}_2, \dots, \boldsymbol{w}_K]$ For the softmax classification model

Note the form of each term in the gradient expression: Amount of current $\boldsymbol{W}$'s error in predicting the label of the $\boldsymbol{n}^{th}$ training example multiplied by the input $\boldsymbol{x}_n$

Some Other Loss Functions for Binary Classification

- Assuming true label as y_n and prediction as $\hat{y}_n = \text{sign}[\boldsymbol{w}^\top \boldsymbol{x}_n]$, the most natural classification loss function would be a "0-1 Loss"
  - Loss = 1 if $\hat{y}_n \neq y_n$ and Loss = 0 if $\hat{y}_n = y_n$
  - Assuming labels as +1/-1, it means

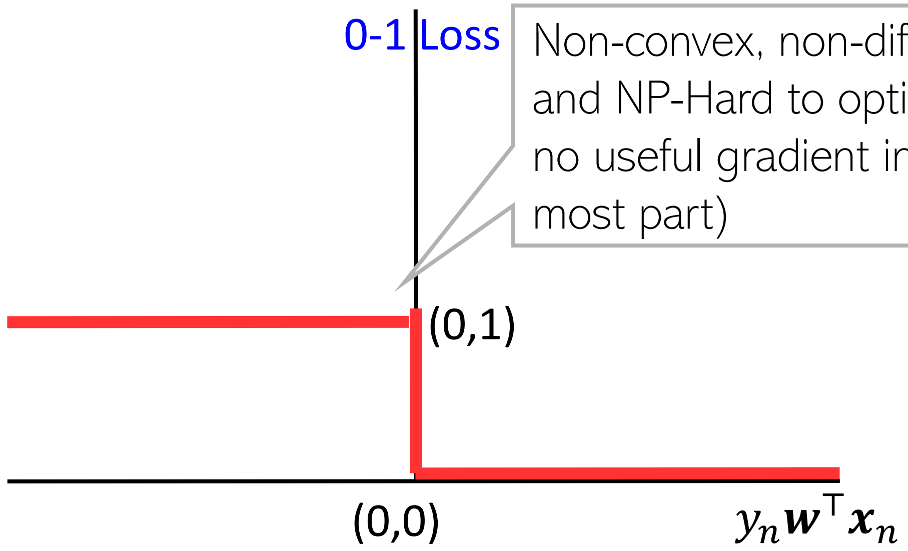
$$\ell(y_n, \hat{y}_n) = \begin{cases} 1 & \text{if } y_n \boldsymbol{w}^\top \boldsymbol{x}_n < 0 \\ 0 & \text{if } y_n \boldsymbol{w}^\top \boldsymbol{x}_n \geq 0 \end{cases}$$

Same as

$$\mathbb{I}[y_n \boldsymbol{w}^\top \boldsymbol{x}_n < 0]$$

or

$$\mathbb{I}[\text{sign}(\boldsymbol{w}^\top \boldsymbol{x}_n) \neq y_n]$$



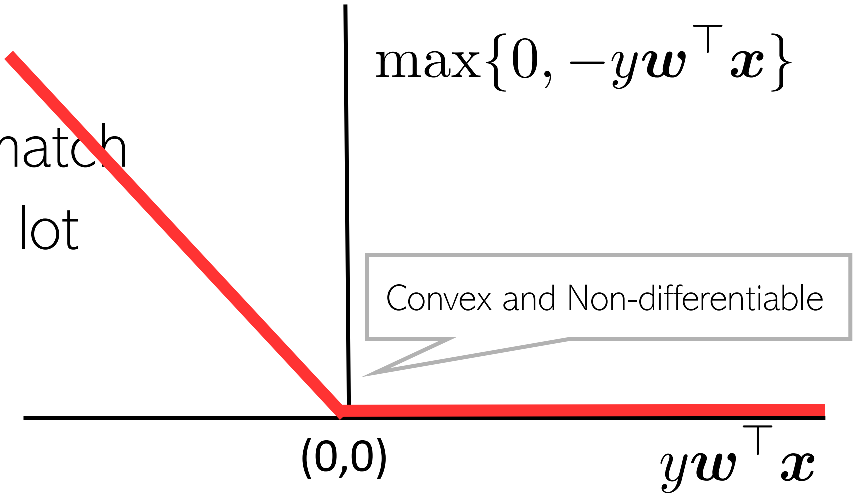
Non-convex, non-differentiable, and NP-Hard to optimize (also no useful gradient info for the most part)

Since optimizing 0-1 loss is not easy, we instead optimize some surrogate loss, such as cross-entropy/logistic loss, hinge loss, etc

Some Other Loss Func for Binary Classification

- An ideal loss function for classification should be such that
  - Loss is small/zero if true $\boldsymbol{y_n}$ and $\mathbf{sign}(\boldsymbol{w}^\top \boldsymbol{x}_n)$ match
  - Loss is large/non-zero if true $\boldsymbol{y_n}$ and $\mathbf{sign}(\boldsymbol{w}^\top \boldsymbol{x}_n)$ do not match
  - Loss is small/large if true $\boldsymbol{y_n}$ and its predicted prob. differ a lot
  - Large positive $\boldsymbol{y_n}\boldsymbol{w}^\top \boldsymbol{x}_n \Rightarrow$ small/zero loss
  - Large negative $\boldsymbol{y_n}\boldsymbol{w}^\top \boldsymbol{x}_n \Rightarrow$ large/non-zero loss

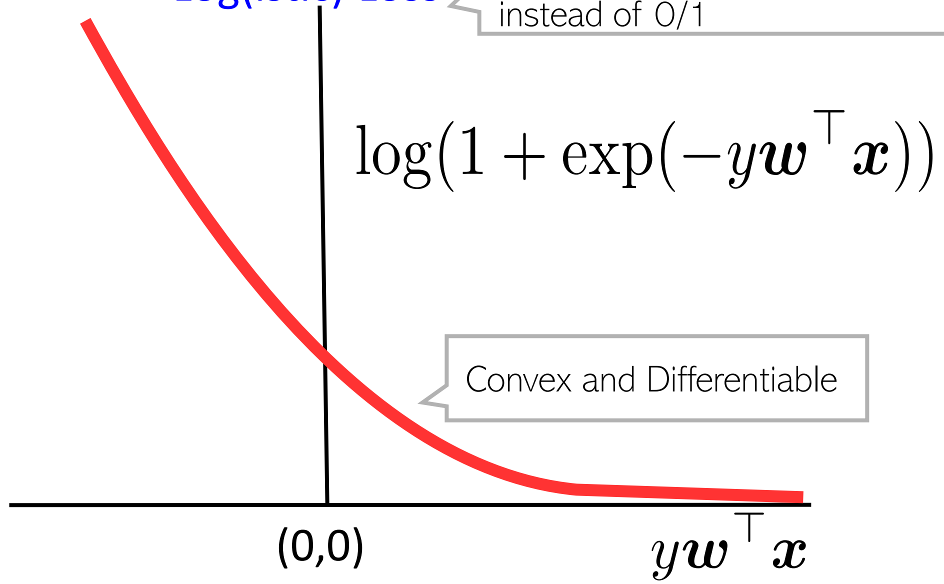
“Perceptron” Loss



Convex and Non-differentiable

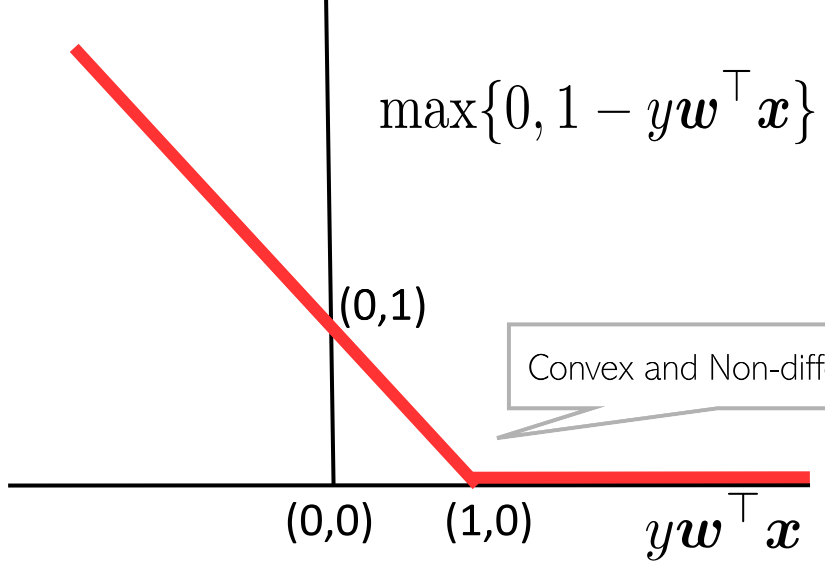
Log(istic) Loss

Same as cross-entropy loss if we assume labels to be -1/+1 instead of 0/1



Convex and Differentiable

Hinge Loss



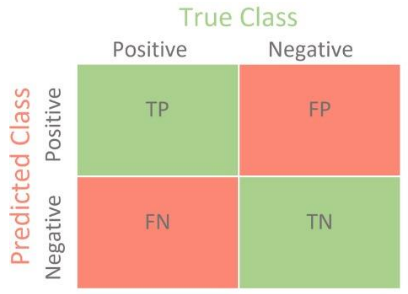
Convex and Non-differentiable

Evaluation Measures for Binary Classification

- Average classification error or average accuracy (on val./test data)

$$err(\boldsymbol{w}) = \frac{1}{N}\sum_{n=1}^{N}\mathbb{I}[y_n \neq \hat{y}_n] \qquad acc(\boldsymbol{w}) = \frac{1}{N}\sum_{n=1}^{N}\mathbb{I}[y_n = \hat{y}_n]$$

- The cross-entropy loss itself (on val./test data)
- Precision, Recall, and F1 score (preferred if labels are imbalanced)
  - Precision (P): Of positive predictions by the model, what fraction is true positive
  - Recall (R): Of all true positive examples, what fraction the model predicted as positive
  - F1 score: Harmonic mean of P and R
- Confusion matrix is also a helpful measure



		True Class	
		Positive	Negative
Predicted Class	Positive	TP	FP
	Negative	FN	TN

Various other metrics such as error/accuracy, P, R, F1, etc. can be readily calculated from the confusion matrix

Evaluation Measures for Multi-class Classification

- Average classification error or average accuracy (on val./test data)

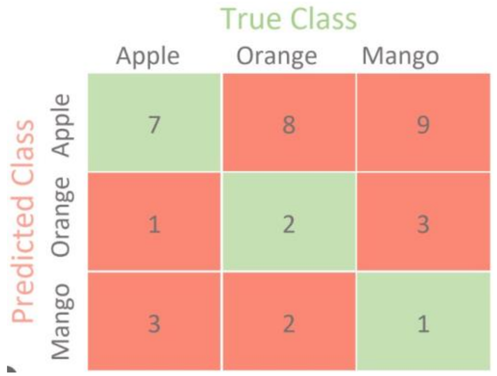
$$err(\boldsymbol{w}) = \frac{1}{N}\sum_{n=1}^{N} \mathbb{I}[y_n \neq \hat{y}_n] \qquad acc(\boldsymbol{w}) = \frac{1}{N}\sum_{n=1}^{N} \mathbb{I}[y_n = \hat{y}_n]$$

- Top-k accuracy

y_n is the true label, $\hat{S}_n$ is the set of top-k predicted classes for $\boldsymbol{x}_n$ (based on the predicted probabilities/scores of the various classes)

$$\text{Top} - \text{k Accuracy} = \frac{1}{N}\sum_{n=1}^{N} \text{is\_correct\_top\_k}[y_n, \hat{S}_n]$$

- The cross-entropy loss itself (on val./test data)
- Class-wise Precision, Recall, and F1 score (preferred if labels are imbalanced)
- Confusion matrix



		True Class		
		Apple	Orange	Mango
Predicted Class	Apple	7	8	9
	Orange	1	2	3
	Mango	3	2	1

Various other metrics such as error/accuracy, P, R, F1, etc. can be readily calculated from the confusion matrix

Coming up next

- Optimization techniques for machine learning