Linear Models for Classification

CS771: Introduction to Machine Learning

Plan today

- Wrapping up linear models for regession
- Linear models for classification
  - Logistic and softmax classification

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 N

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

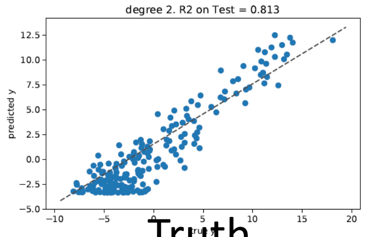
$$\boldsymbol{w}^{(t+1)} = \boldsymbol{w}^{(t)} - \eta_t \boldsymbol{g}^{(t)}$$

$$= \boldsymbol{w}^{(t)} + \eta_t \frac{2}{N} \sum_{n=1}^{N} \left(y_n - \boldsymbol{w}^{(t)\top} \boldsymbol{x}_n \right) \boldsymbol{x}_n$$

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 $\boldsymbol{X}^\top \boldsymbol{X}$ (thus faster)

- Similar updates for ridge regression as well (with the gradient expression being slightly different; left as an exercise)
- More on iterative optimization methods later

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

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

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

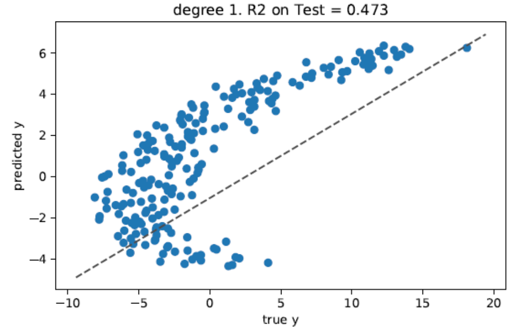
$$R^2 = 1 - \frac{\sum_{n=1}^{N}(y_n - \hat{y}_n)^2}{\sum_{n=1}^{N}(y_n - \bar{y})^2}$$

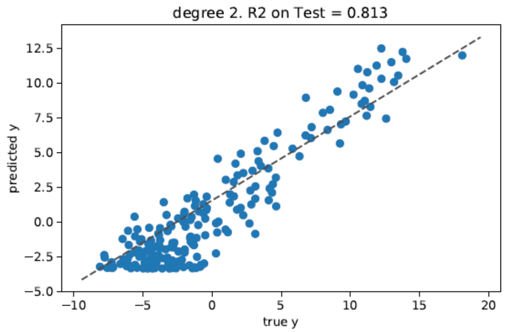
"relative" error w.r.t. a model that makes a constant prediction $\bar{y}$ for all inputs

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., $\frac{1}{N}\sum_{n=1}^{N} y_n$

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





Pic from MLAPP (Murphy)

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.

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

Now solve this!

System of lin. Eqns with D equations and D unknowns

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

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

  - Convert the score into a probability

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

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

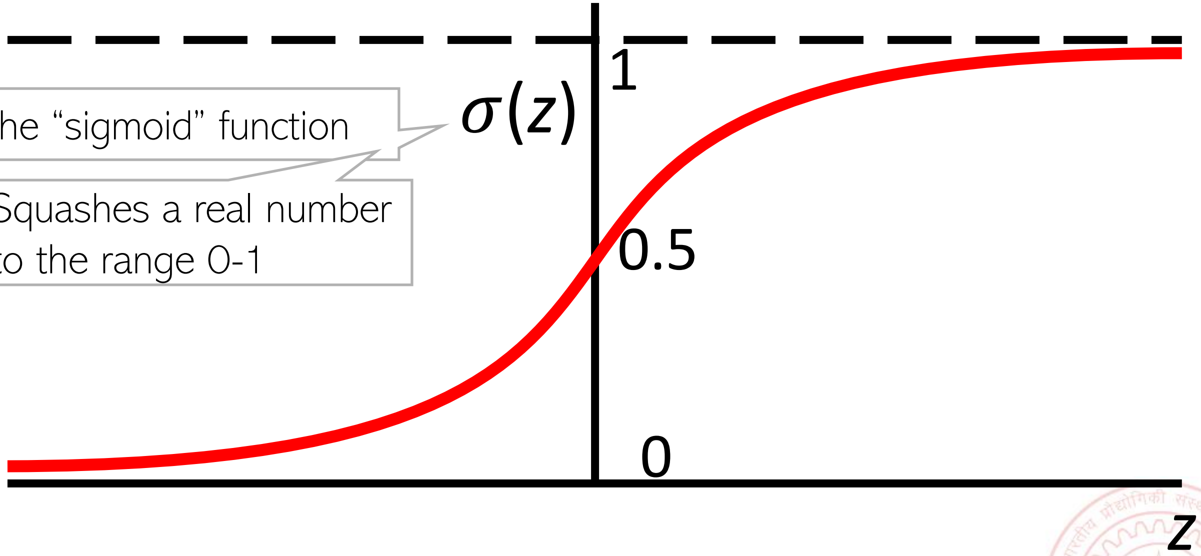
$$= \frac{1}{1 + \exp(-\boldsymbol{w}^\top \boldsymbol{x}_n)}$$

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

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

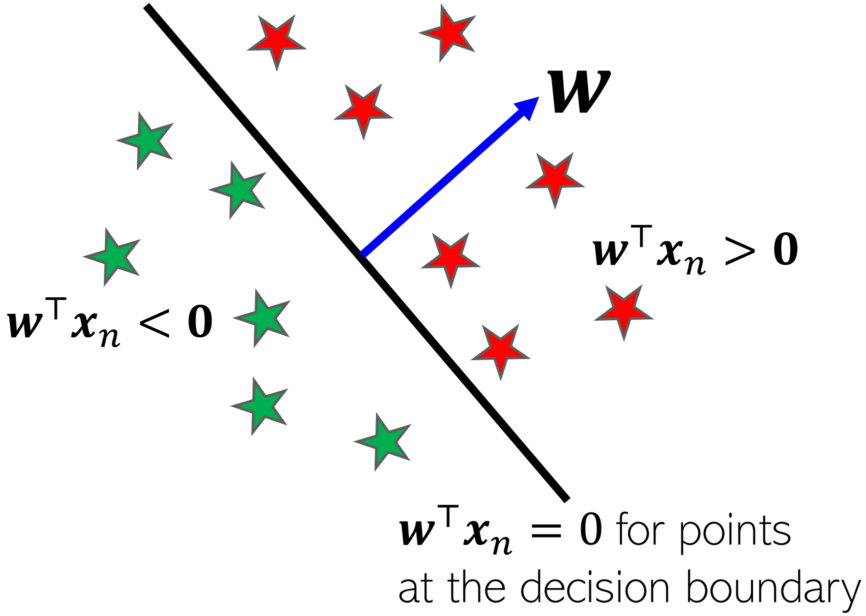


- Note: In LR, if we assume the label y_n as -1/+1 (not 0/1) then we can write

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

Linear Models: The Decision Boundary

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



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

- For logistic reg, at decision boundary

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

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

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

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

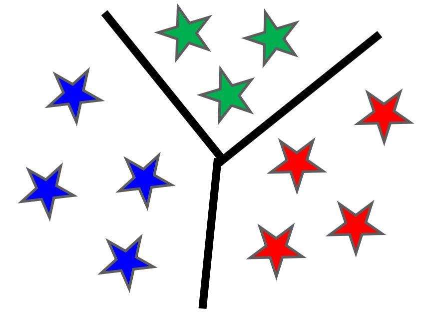
- Therefore, both views are equivalent

Linear Models for (Multi-class) Classification

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

$D \times C$ weight matrix

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



- The prediction rule is as follows

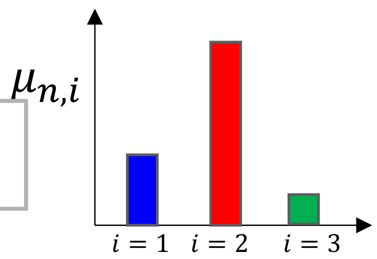
$$y_n = \operatorname{argmax}_{i \in \{1,2,\dots,C\}} \boldsymbol{w}_i^\top \boldsymbol{x}_n$$

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

"softmax" classification

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

Multi-class extension of logistic regression



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

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

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

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

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

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

Probabilities must sum to 1

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

Linear Classification: Interpreting weight vectors

- Recall that multi-class classification prediction rule is

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

- Can think of $\boldsymbol{w}_i^\top \boldsymbol{x}_n$ as the score of the input for the $\boldsymbol{i}^{th}$ class (or similarity of $\boldsymbol{x}_n$ with $\boldsymbol{w}_i$)

- Once learned (we will see the methods later), these $\boldsymbol{C}$ 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

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

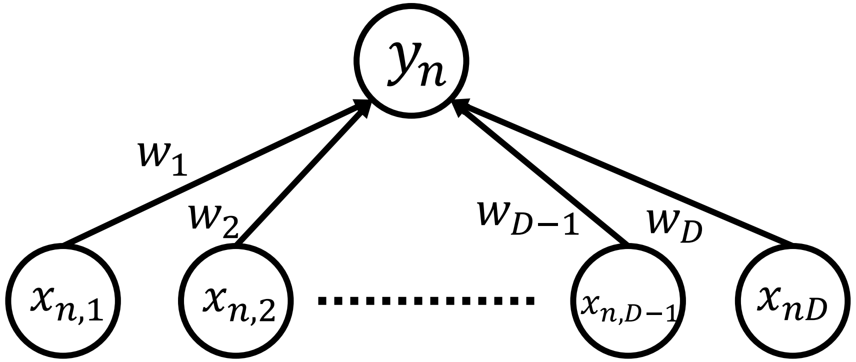
That's why the dot product of each of these weight vectors with an image from the correct class will be expected to be the largest

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

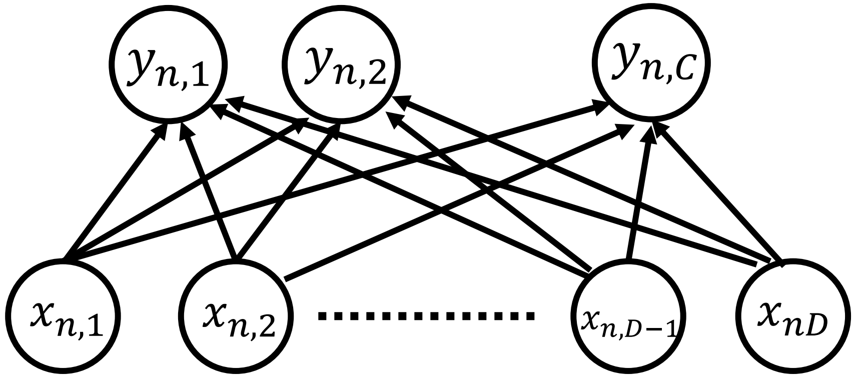
Yeah, "sort of". ☺
No wonder why LwP (with Euclidean distances) acts like a linear model. ☺

Logistic and Softmax classification: Pictorially

- Logistic regression is a linear model with single weight vector with D weights



- Softmax classification is a linear model with C weight vectors with $D \times C$ weights



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

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

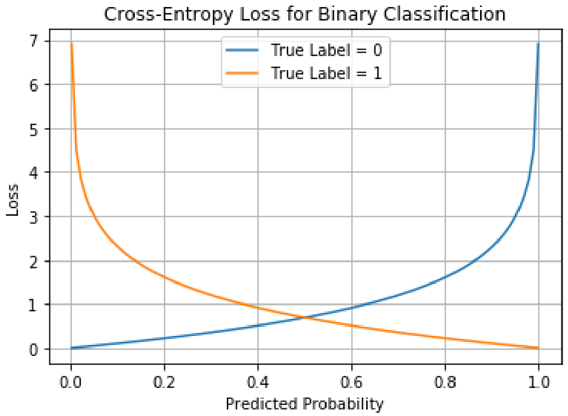
- Will be easy to optimize (same solution as the regression case)
- Can also consider other loss functions used in regression
  - Basically, pretend that the binary label is actually a continuous value and treat the problem as regression where the output can only be one of two possible values
- However, 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

- Binary cross-entropy (CE) is a popular loss function for binary classifn. 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 1 and μ_n close to 0, or y_n is 0 and μ_n close to 1

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

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

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

$\mu_{n,i}$ is the predicted probability of $\boldsymbol{x}_n$ belonging to class $\boldsymbol{i}$

$y_{n,i} = 1$ if true label of $\boldsymbol{x}_n$ is class $\boldsymbol{i}$ and 0 otherwise.

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

Note: Unlike least squares loss for regression, for the cross-entropy loss, we can't get a closed form solution for $\boldsymbol{w}$ by applying first order optimality. Try this as exercise for binary CE loss

We can however optimize the CE loss using iterative optimization such as gradient descent

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 n^{th} training example multiplied by the input $\boldsymbol{x}_n$

- The expression for the gradient of multi-class cross-entropy loss w.r.t. weight vec of i^{th} class

Need to calculate the gradient for each of the C 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 $\boldsymbol{W} = [\boldsymbol{w}_1, \boldsymbol{w}_2, \dots, \boldsymbol{w}_C]$
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 n^{th} training example multiplied by the input $\boldsymbol{x}_n$

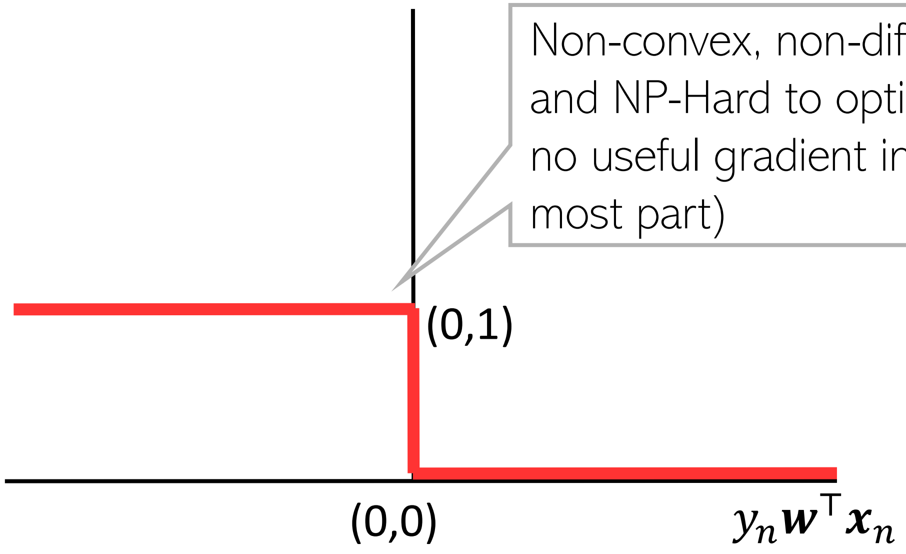
Some Other Loss Functions for Binary Classification

- Assume true label as y_n and prediction as $\hat{y}_n = \mathbf{sign}[\boldsymbol{w}^\top \boldsymbol{x}_n]$

- The zero-one loss is the most natural loss function for classification

$$\ell(y_n, \hat{y}_n) = \begin{cases} 1 & \text{if } y_n \neq \hat{y}_n \\ 0 & \text{if } y_n = \hat{y}_n \end{cases}$$

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

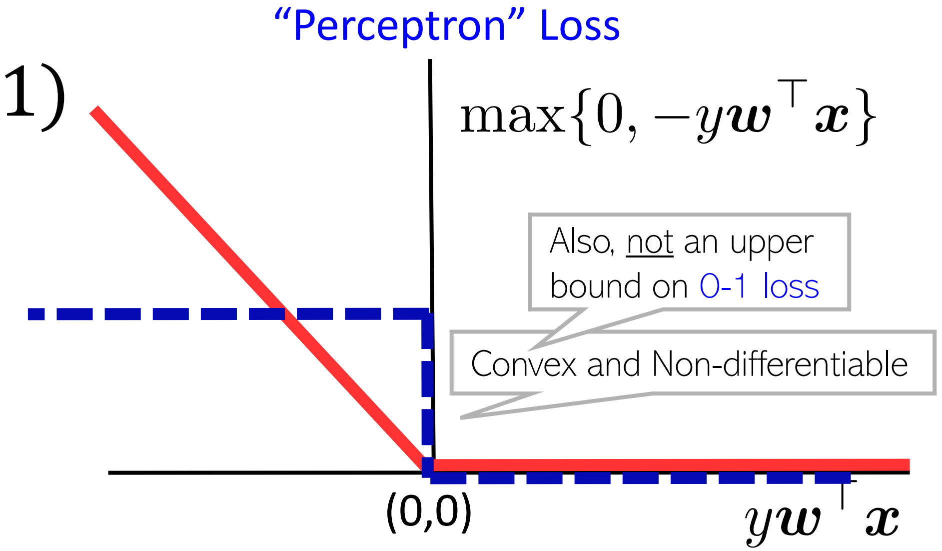


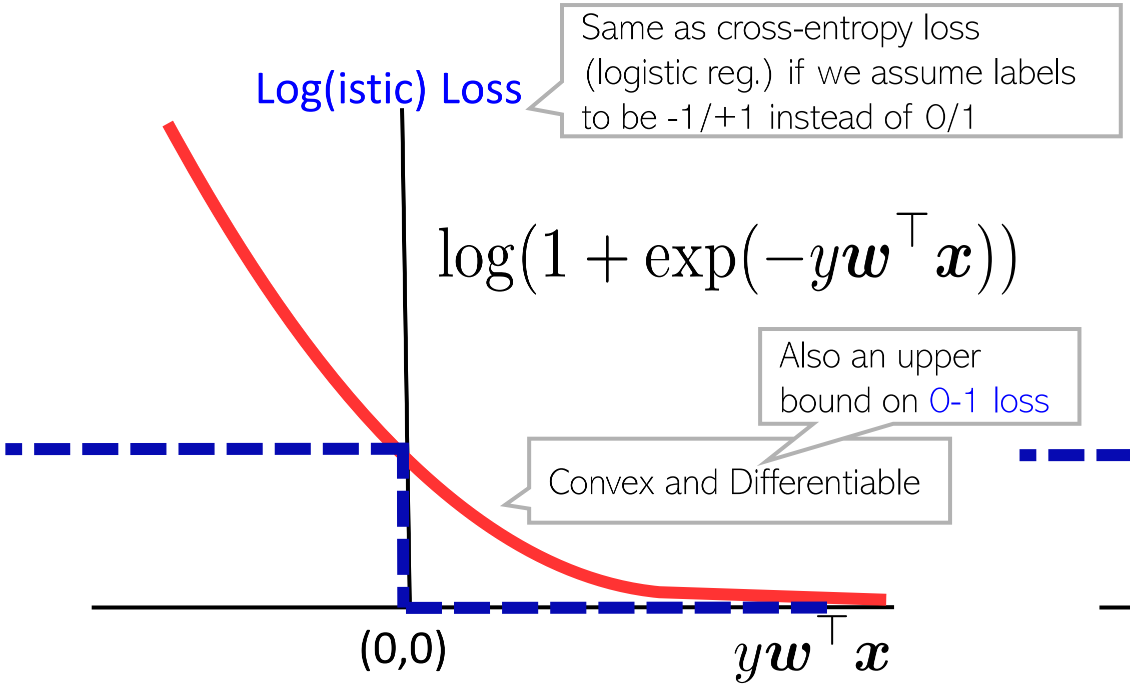
Non-convex, non-differentiable, and NP-Hard to optimize (also no useful gradient info for the most part)

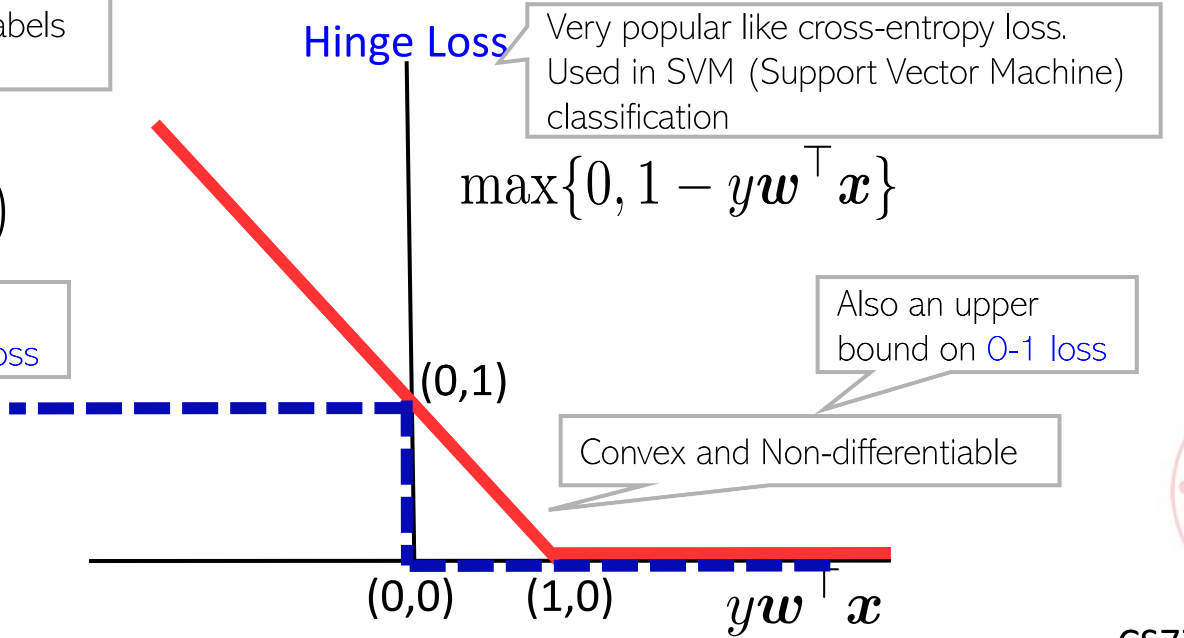
- Since zero-one loss is hard to minimize, we use some surrogate loss function
  - Popular examples: Cross-entropy (also called logistic loss), hinge loss , etc
  - Note: Ideally, surrogate loss (approximation of zero-one) must be an upper bound (must be larger than the 0-1 loss for all values of $y_n \boldsymbol{w}^\top \boldsymbol{x}_n$) since our goal is minimization

Some Other Loss Func for Binary Classification

- For an ideal loss function, assuming $y_n \in (-1, +1)$
  - Large positive $y_n \boldsymbol{w}^\top \boldsymbol{x}_n \Rightarrow$ small/zero loss
  - Large negative $y_n \boldsymbol{w}^\top \boldsymbol{x}_n \Rightarrow$ large/non-zero loss
  - Small (large) loss if predicted probability of the the true label is large (small)

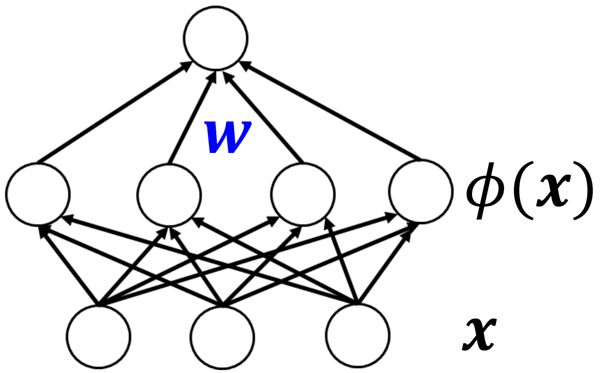


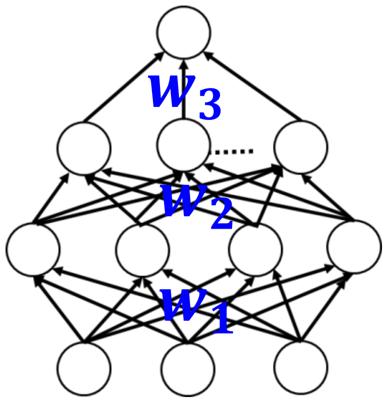




Nonlinear Classification using Linear Models?

- Yes, transform the original features and apply logistic or softmax classification model on top
- Feature transformation can be pre-defined (e.g., using kernels) or learned (using neural nets)





- Similar to how we nonlinearlize a linear model for regression
- Only the loss function $\ell(y_n, f(x_n))$ changes
  - Binary CE loss for if using logistic regression at the top
  - Multiclass CE if using softmax classification at the top
  - Or other classification loss functions if using other linear classifiers at the top

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

True Class

Predicted Class	Apple	Orange	Mango
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