# DATASCI207-005/007 Applied Machine Learning

Vilena Livinsky, PhD(c)

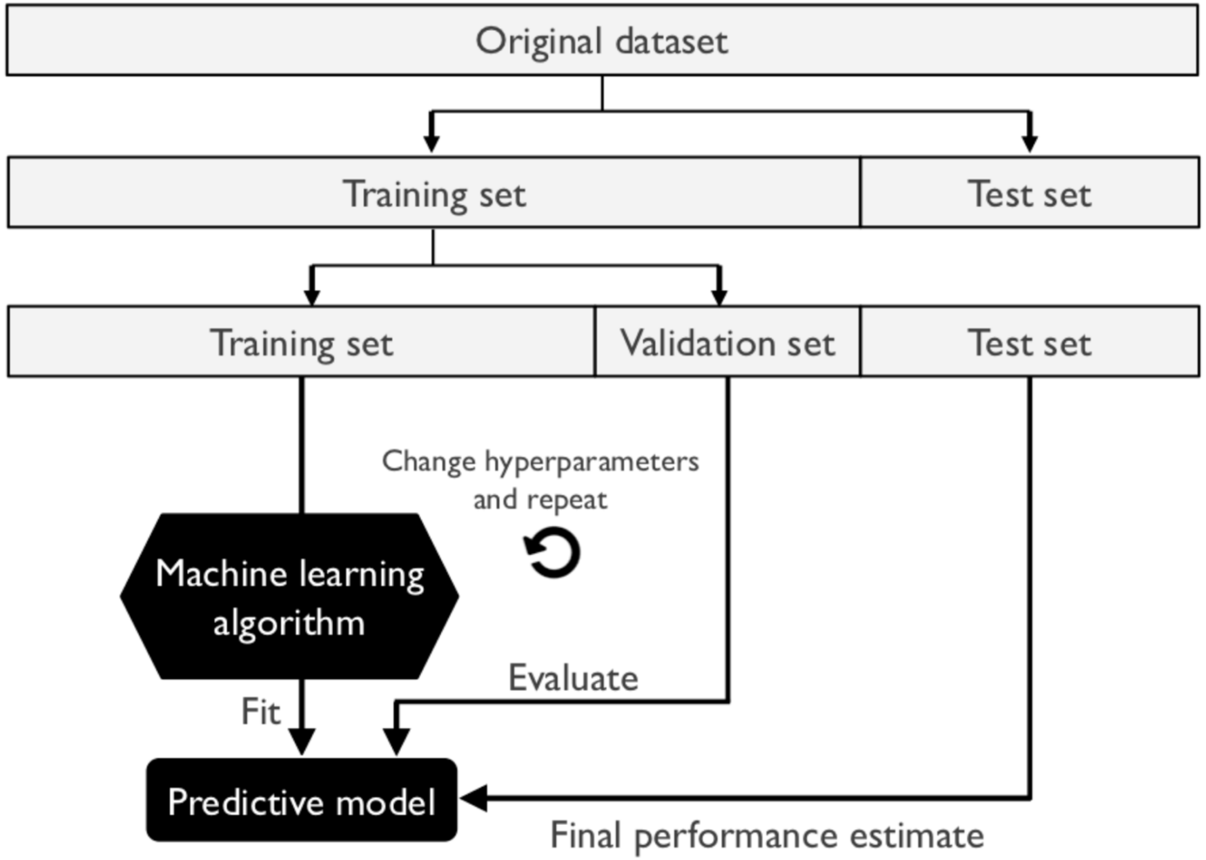
School of Information, UC Berkeley

Week 4: 09/29/2024 - 09/30/2024

# Today's Agenda

- Feature Engineering, Cont.
- Logistic Regression
- Walkthroughs:
  - Feature Engineering, Cont.
  - Logistic Regression w/Gradient Descent

# Model Workflow: Data




- Train dataset:
  - to train and optimize our machine learning model
- Test dataset:
  - keep until the very end to evaluate the final model
- Common splits:
  - 60:40, 70:30, or 80:20
    - depends on size of dataset
  - large datasets:
    - Ex.: 90:10 or 99:1

Image Ref.: Raschka, S., & Mirjalili, V. (2019). Python Machine Learning, Third Edit.

# Model Workflow: Data

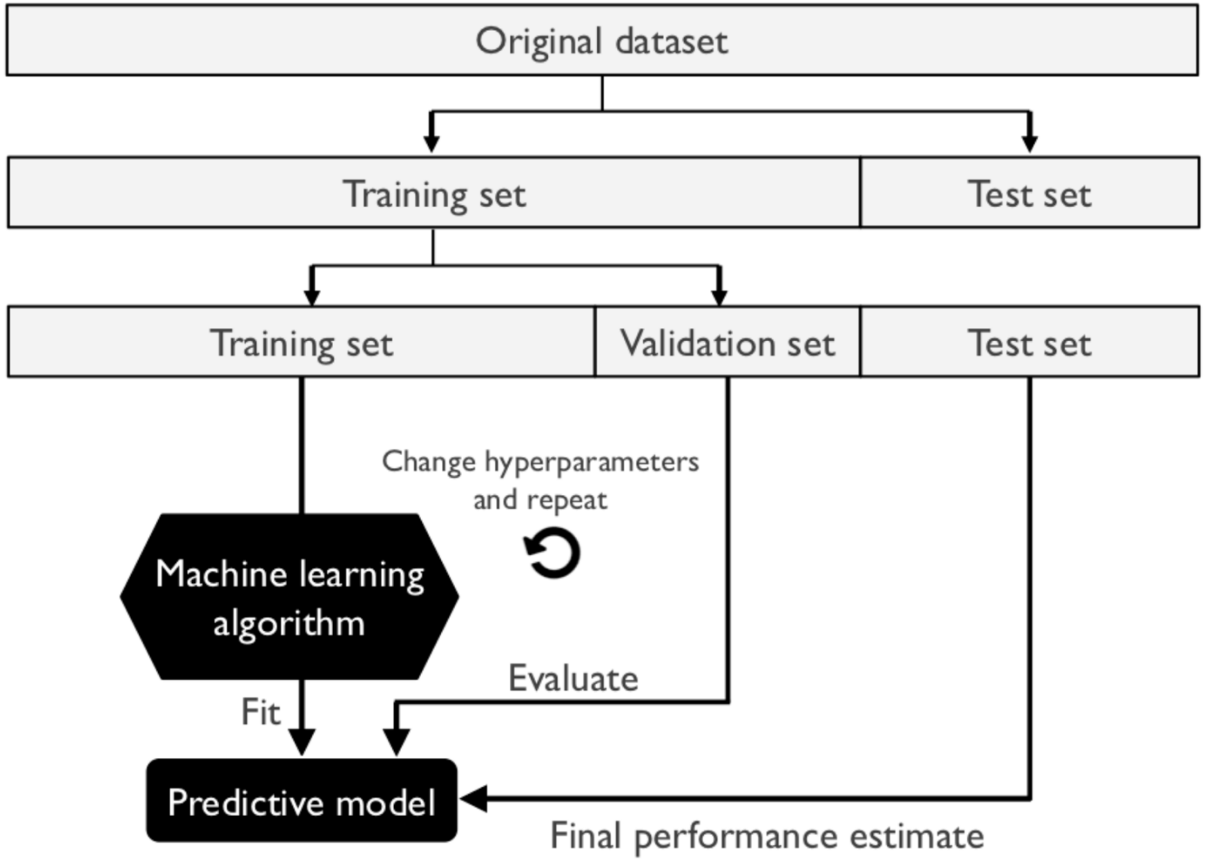
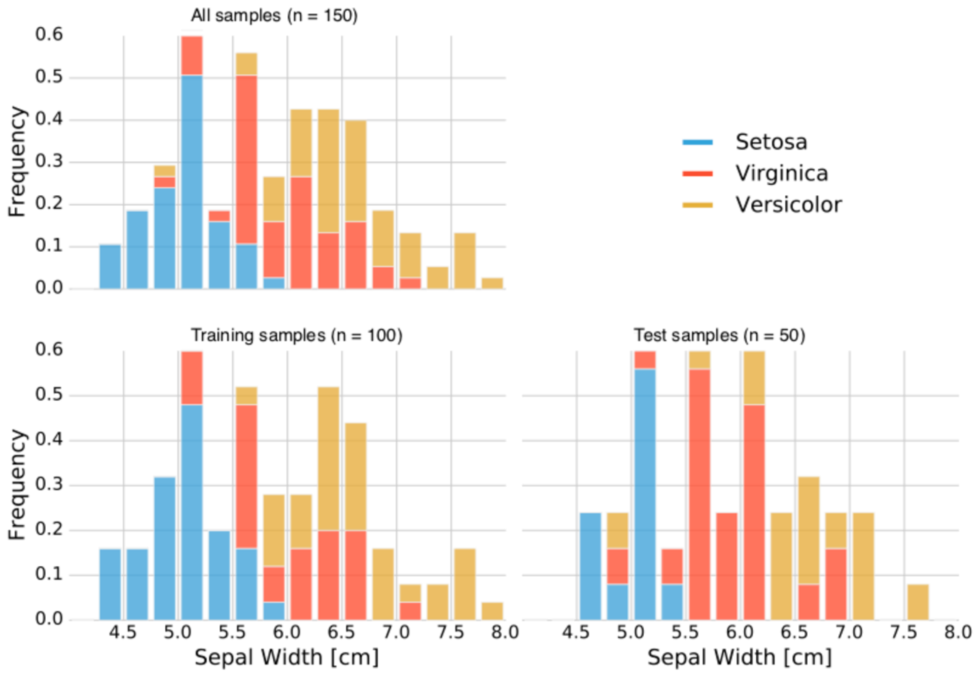



Image Ref.: Raschka, S., & Mirjalili, V. (2019). Python Machine Learning, Third Edit.

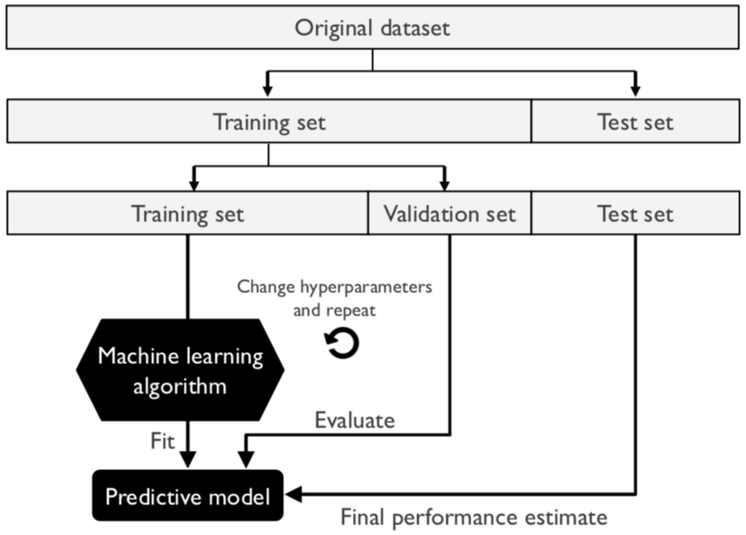
- To shuffle or not to shuffle?
- Stratify?




- [Notebook: https://github.com/rasbt/stat479-machine-learning-fs19/blob/master/05_preprocessing-and-sklearn/code/05-preprocessing-and-sklearn__notes.ipynb](https://github.com/rasbt/stat479-machine-learning-fs19/blob/master/05_preprocessing-and-sklearn/code/05-preprocessing-and-sklearn__notes.ipynb)

# Model Workflow: Data

- Holdout Cross-Validation
  - performance estimate may be <u>sensitive</u> to how we partition the training dataset into the training and validation data subsets
    - **k-fold cross-validation**
      - randomly split training dataset into k folds (without replacement)
      - k – 1 folds are used for model training
      - 1 fold is used for performance evaluation



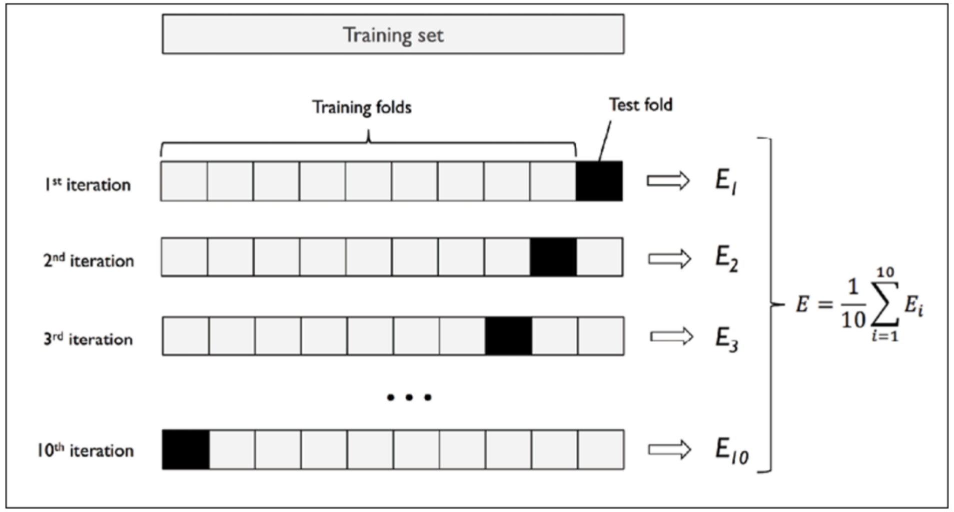




Image Ref.: Raschka, S., & Mirjalili, V. (2019). Python Machine Learning, Third Edit.

Advantages?
- lower variance in the performance estimate
- Consider data set size...

Disadvantages?
- Consider date/time of features (temporal aspect)...
- Efficiency...

# TensorFlow: General Modeling Steps

## Create

Create a model

- Create your model architecture
- *Functional* or *Sequential API*

```python
# Use Keras Sequential API to build a linear regression model.
model = keras.Sequential()
model.add(keras.layers.Dense(
    input_shape=[num_features], # each input has num_features features
    units=1,                    # there is a single output
    use_bias=True               # include a learned bias parameter
    ))
```

## Compile

Compile a model

- Decide on **loss**
- Decide on model performance metrics
- Decide on how to improve the process; choose an **optimizer**

```python
model.compile(loss='mean_squared_error', optimizer='adam',
              metrics =['mean_absolute_error'])
```

## Fit

Fit a model

- Learn a function

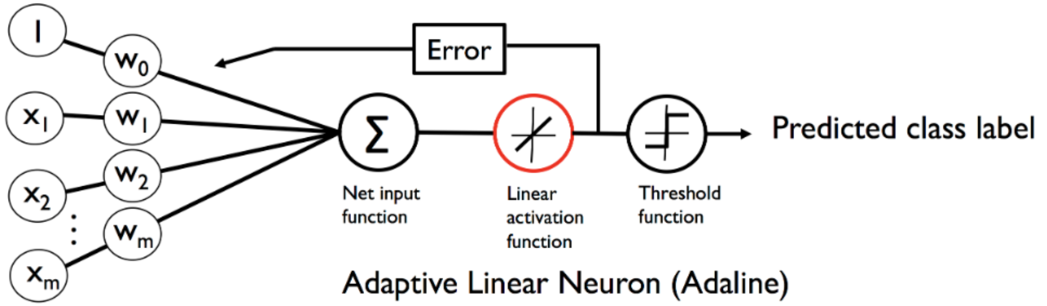
```python
# Build a model and train it. Hold out 10% of data for validation.
model = build_model(num_features=len(features))
model.fit(x=X, y=Y,
          validation_split=0.1, batch_size=16, epochs=5)

# Use the model to predict the training labels.
Y_pred = model.predict(x=X).flatten()
```

# Logistic Regression

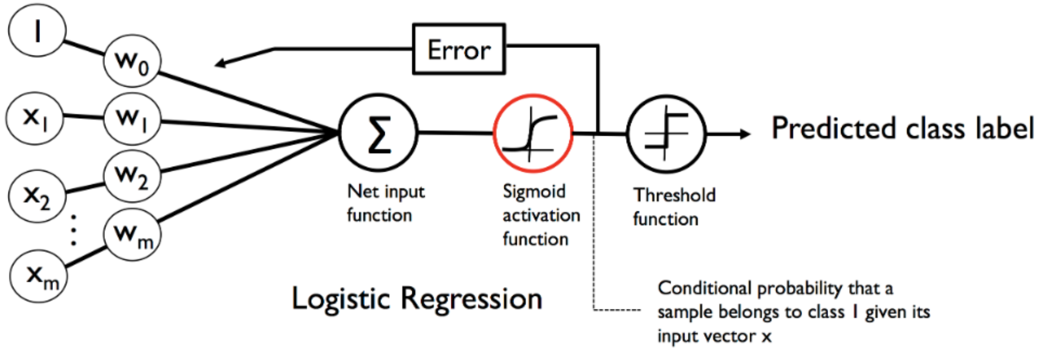
Gradient Descent

# Activation Functions: Sigmoid




linear activation function, $\phi(z)$,

$$\phi(\boldsymbol{w}^T\boldsymbol{x}) = \boldsymbol{w}^T\boldsymbol{x}$$




Sigmoid,

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

$$\hat{y} := \begin{cases} 1 & \text{if } \sigma(z) > 0.5 \\ 0 & \text{otherwise} \end{cases}$$

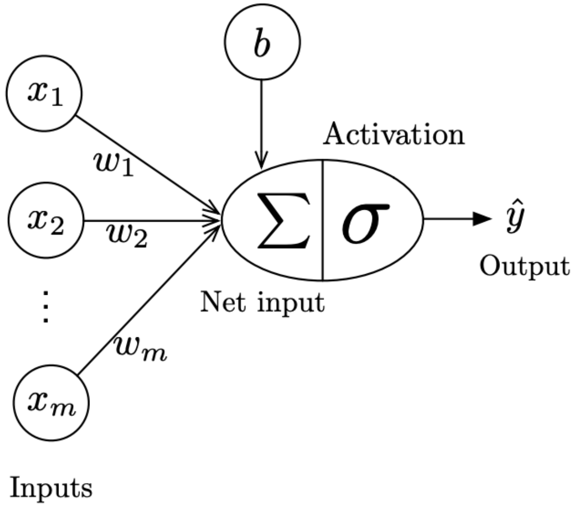



Image Ref., edited: Raschka, S., & Mirjalili, V. (2019). Python Machine Learning, Third Edit.

# Logistic Regression: Binary Classification

- Probabilistic model for binary classification
- **Odds**

$$\frac{p}{(1-p)}$$

Where p = the probability of a positive event

- **Logit**
  - inputs in range [0, 1] to values over the entire real number range $R = (-\infty, \infty)$ or $R = \mathbb{R}$
  - can help us express a linear relationship between feature values and the log-odds

$$logit(p) = \log \frac{p}{(1-p)}$$

$$logit\big(p(y=1|\boldsymbol{x})\big) = w_0x_0 + w_1x_1 + \cdots + w_mx_m = \sum_{i=0}^{m} w_ix_i = \boldsymbol{w}^T\boldsymbol{x}$$

conditional probability that a particular example belongs to class 1 given its features, x

- **Logistic sigmoid function**
  - input values over the entire real number range to range [0, 1] with intercept at 0.5
  - probability that a certain example belongs to a particular class

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

$$z = w_1x_1 + \ldots + w_mx_m + b = \sum_{l=1}^{m} w_lx_l + b = \mathbf{w}^T\mathbf{x} + b$$

Sigmoid

*Inverse*

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

$$\text{logit}(p(y = 1 \mid \mathbf{x})) = z$$

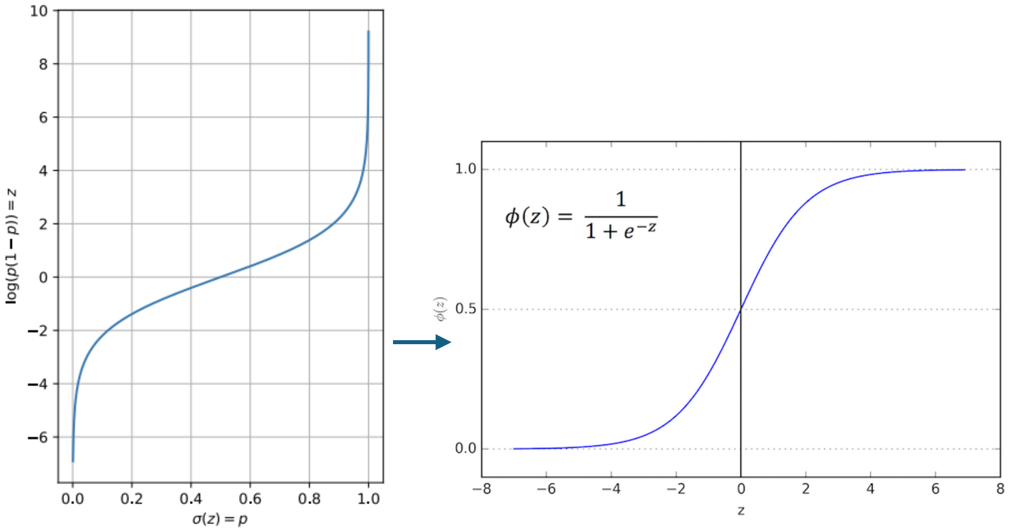
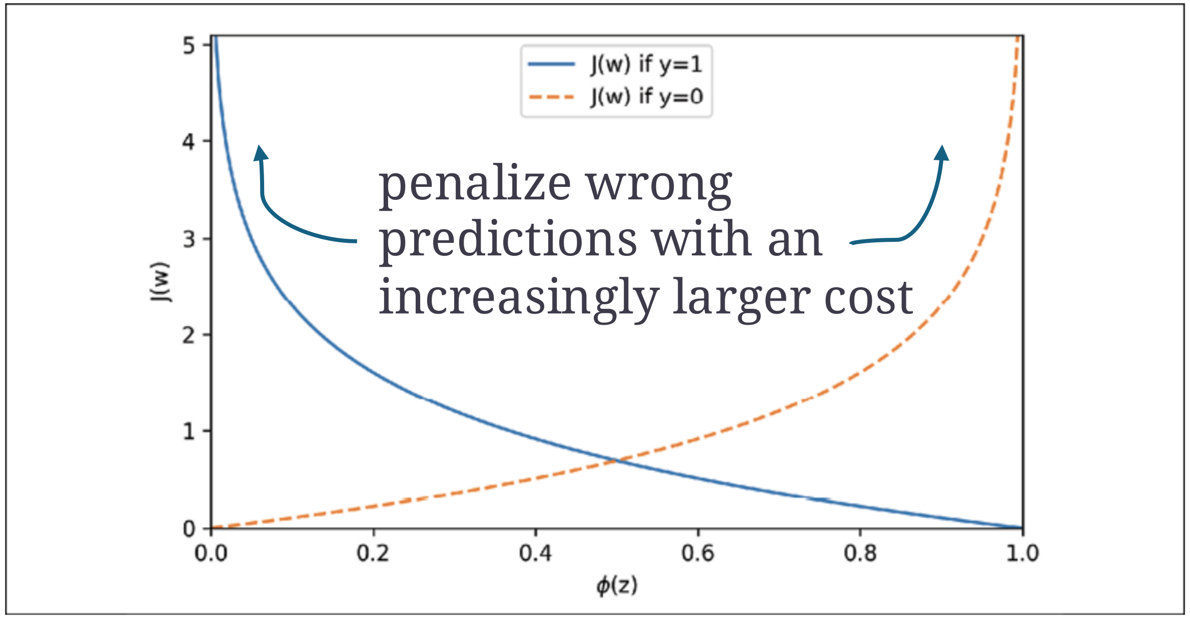



Image Ref.: Sebastian Raschka, Intro to Deep Learning

# Log Loss

- a.k.a.:
  - “logarithmic loss” OR
  - “binary cross-entropy”
- Use:
  - classification problems
  - measures the performance of a classification model by quantifying the difference between predicted probabilities and actual values
  - evaluates how close the predicted probabilities are to the actual binary outcomes (0 or 1)
    - **Lower Log Loss**: Indicates better model performance. It means the predicted probabilities are closer to the actual outcomes.
    - **Higher Log Loss**: Indicates poorer model performance. It means the predicted probabilities are further from the actual outcomes.




$$LogLoss = \frac{1}{m}\sum_i -y_i \log(\hat{y}_i) - (1 - y_i)\log(1 - \hat{y}_i)$$

# Logistic Regression

Given the output:

$$h(\mathbf{x}) = \sigma(\mathbf{w}^\top \mathbf{x} + b)$$

For a binary class problem (0 and 1), we want these probabilities to be:

$$\begin{bmatrix} P(y=0|\mathbf{x}) \approx 1 & \text{if } y=0 \\ P(y=1|\mathbf{x}) = 1 - P(y=0|\mathbf{x}) \approx 1 & \text{if } y=1 \end{bmatrix}$$

Goal:
Maximize probability for true label

Can represent as a piece-wise function like so:

Can summarize like so:

We compute the posterior as:

$$P(y|\mathbf{x}) = \begin{cases} h(\mathbf{x}) & \text{if } y=1 \\ 1-h(\mathbf{x}) & \text{if } y=0 \end{cases} \longrightarrow P(y|\mathbf{x}) = a^y(1-a)^{(1-y)}$$

Where h(x) = a

## Maximum Likelihood Estimation

$$P(y^{[i]}, \ldots, y^{[n]}|\mathbf{x}^{[1]}, \ldots, \mathbf{x}^{[n]}) = \prod_{i=1}^{n} P(y^{[i]}|\mathbf{x}^{[i]})$$

$$\begin{aligned} L(\mathbf{w}) &= P(\mathbf{y} \mid \mathbf{x}; \mathbf{w}) \\ &= \prod_{i=1}^{n} P(y^{(i)} \mid x^{(i)}; \mathbf{w}) \\ &= \prod_{i=1}^{n} \left(\sigma\left(z^{(i)}\right)\right)^{y^{(i)}} \left(1-\sigma\left(z^{(i)}\right)\right)^{1-y^{(i)}} \end{aligned}$$

Maximize class membership probabilities for all examples in train

## Log-Likelihood "Loss"

maximize the (natural) log

$$\begin{aligned} l(\mathbf{w}) &= \log L(\mathbf{w}) \\ &= \sum_{i=1}^{n} \left[y^{(i)} \log\left(\sigma\left(z^{(i)}\right)\right) + \left(1-y^{(i)}\right) \log\left(1-\sigma\left(z^{(i)}\right)\right)\right] \end{aligned}$$

<u>minimize negative log-likelihood</u>

$$\begin{aligned} \mathcal{L}(\mathbf{w}) &= -l(\mathbf{w}) \\ &= -\sum_{i=1}^{n} \left[y^{(i)} \log\left(\sigma\left(z^{(i)}\right)\right) + \left(1-y^{(i)}\right) \log\left(1-\sigma\left(z^{(i)}\right)\right)\right] \end{aligned}$$

# Logistic Regression: Binary Classifier

## Gradient Descent

**Predictions** $\boxed{\boxed{\sigma}(XW^T)}$

**Differences** $\boxed{\boxed{\sigma}(XW^T)} - Y$

**Gradient** $\frac{1}{m}(\overbrace{\sigma(XW^T) - Y})X$