outraggeously
AMBITIOUS

Modeling Process & Algorithms

Week 6

Duke
PRATT SCHOOL *of* ENGINEERING

R-squared and MSE

$$R^2 = \frac{SSR}{SST} = 1 - \frac{SSE}{SST} = 1 - \frac{\sum_{i=1}^{n}(y_i - \hat{y}_i)^2}{\sum_{i=1}^{n}(y_i - \mu_y)^2} = 1 - \frac{\frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2}{\frac{1}{n}\sum_{i=1}^{n}(y_i - \mu_y)^2} = 1 - \frac{MSE}{Var(y)}$$

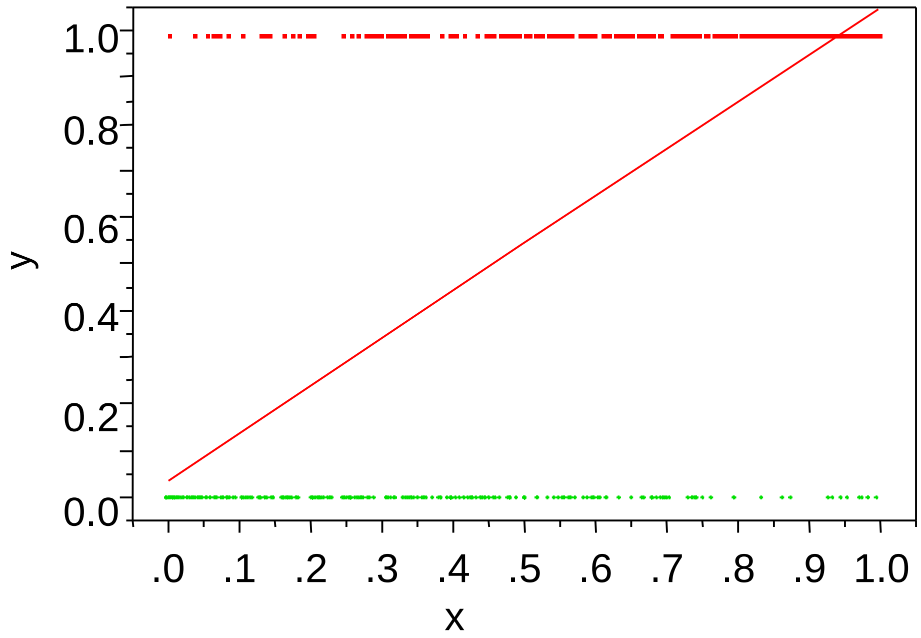
Classification

- Classification models predict a categorical target:
  - Handwritten characters
  - Benign vs. malignant tumors
  - Type of lung disease
  - Whether user will like a product
- We can have two (binary) or more classes
- Classes can also be numerical in nature through binning:
  - Income in bins of $10k
  - Sales lead scoring 1-5
- Unlike regression, classification models cannot extrapolate

Duke
PRATT SCHOOL of ENGINEERING

Let's Tackle a Classification Problem

- We now want to predict a class (e.g. 0 or 1) rather than a numerical target
- We could use linear regression to do so



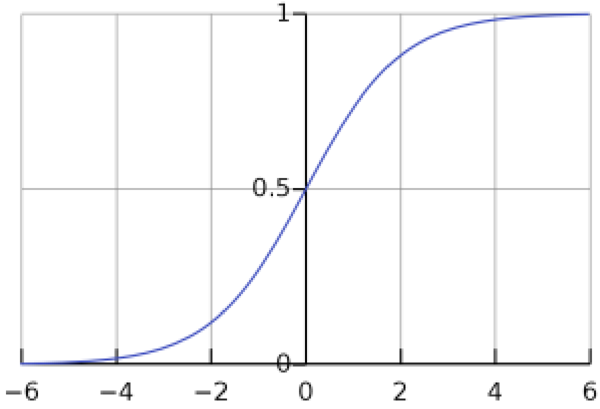
Problems:

- How do we interpret predictions between 0 and 1?
- What about predictions greater than 1?
- The linear regression will almost always predict the wrong value

Solution: Predict the Probability y=1

- Rather than predicting y, let's predict the probability P(y=1), which falls between 0 and 1
- We could use linear regression, but what do we do with predicted values <0 and >1?
- A better option would be a function that predicts outputs between 0 and 1
- One alternative: the **sigmoid (logistic)** function

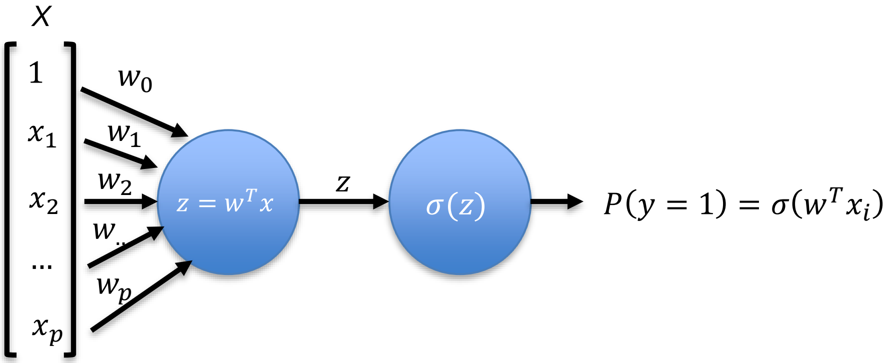
$$\sigma(z) = \frac{1}{1 + e^{-z}}$$



Source: Wikipedia

Solution: Predict the Probability y=1

- We want the output of our model to be P(y=1)
- We can use the sigmoid function to get outputs between 0 and 1
- As input to the sigmoid function we provide the output of our linear regression (w_0+w_1x)



$$P(y_i = 1) = \sigma(z_i)$$

$$= \sigma(w^T x_i)$$

$$= \frac{1}{1 + e^{-(w^T x_i)}}$$

$$= \frac{e^{w^T x_i}}{1 + e^{w^T x_i}}$$

Duke PRATT SCHOOL of ENGINEERING

Estimating the parameters

How do we estimate the parameters $w_1, w_2 \ldots w_p$?

1. Define our cost function *J(w)*
2. Find the weight/coefficient values that minimize the cost function
   - Calculate the gradient (derivative)
   - Set the gradient equal to 0
   - Solve for the coefficients

But this time it's a bit trickier:

- SSE is not convex for logistic regression
- We can use a different cost function that is, but it has no closed form solution

Duke

PRATT SCHOOL of ENGINEERING

Maximum likelihood estimation

Instead of SSE, we use **maximum likelihood estimation** to establish our cost function

Our logistic regression model gives us:

$$P(y_i = 1|x_i) = \sigma(w^T x_i)$$

$$P(y_i = 0|x_i) = 1 - \sigma(w^T x_i)$$

- We now have an x-y relationship characterized by the parameters ***w***
- Our objective is to find the parameter set ***w*** which is the **maximum likelihood estimator**
  - Results in the highest **likelihood** - probability of observing the correct y_1, y_2... y_N given inputs x_1, x_2... x_N

Duke
PRATT SCHOOL of ENGINEERING

Maximum likelihood estimation

The likelihood of our model for a single observation of x_i, y_i is:

$$P(y_i|x_i) = \begin{cases} P(y_i = 1|x_i) \text{ if } y_i = 1 \\ P(y_i = 0|x_i) \text{ if } y_i = 0 \end{cases} \xrightarrow{\text{combine}} P(y_i|x_i) = P(y_i = 1|x_i)^{y_i} * P(y_i = 0|x_i)^{1-y_i}$$

And from our logistic regression model we have:

$$P(y_i = 1|x_i) = \sigma(w^T x_i)$$

$$P(y_i = 0|x_i) = 1 - \sigma(w^T x_i)$$

Combining the above we get the likelihood of one observation as:

$$P(y_i|x_i) = (\sigma(w^T x_i))^{y_i} * (1 - \sigma(w^T x_i))^{1-y_i}$$

Maximum likelihood estimation

Now that we have the likelihood for one observation, we can get the likelihood for all observations:

$$L(w) = P(y_1, y_2 \dots y_N | \, x_1, x_2 \dots x_N) = \prod_{i=1}^{N} P(y_i | x_i)$$

$$L(w) = \prod_{i=1}^{N} (\sigma(w^T x_i))^{y_i} * (1 - \sigma(w^T x_i))^{1-y_i}$$

We want to find the parameter set ***w*** that maximizes this. However, it's difficult to calculate the gradient (derivative) with respect to *w* so instead we maximize the log of it, called the **log likelihood**

$$\log L(w) = \sum_{i=1}^{N} y_i \log\big(\sigma(w^T x_i)\big) + (1 - y_i)\log(1 - \sigma(w^T x_i))$$

Maximum likelihood estimation

- We can now **maximize the log likelihood**
- Or alternatively we can **minimize the negative log likelihood (NLL)**
- To be consistent with what we did in linear regression, we will use NLL as our cost function and find the ***w*** that minimizes it

$$NLL(w) = -\sum_{i=1}^{N} y_i \log\left(\sigma(w^T x_i)\right) + (1 - y_i)\log(1 - \sigma(w^T x_i))$$

- However, there is no closed form solution so we need to use an iterative solving method for *w*

Duke PRATT SCHOOL of ENGINEERING

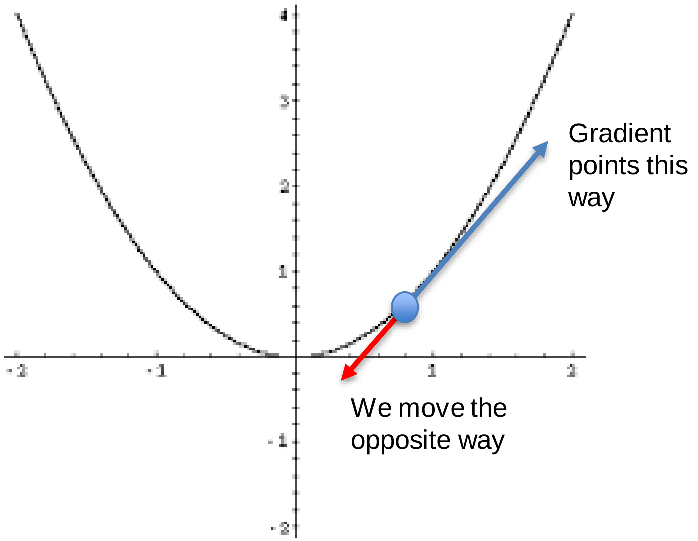
Gradient Descent

outrageously AMBITIOUS

Duke PRATT SCHOOL of ENGINEERING

Gradient descent

- Suppose we want to minimize a function such as $y = x^2$
- We start at some point on the curve and move iteratively towards the minimum
  - Move in the direction opposite the gradient
  - Move by some small value (called the "learning rate" or η) multiplied by the gradient
- We continue until we find the minimum or reach a set number of iterations



Gradient descent example

Let's try to minimize $f(x) = x^2$

Gradient: $\nabla f(x) = 2x$

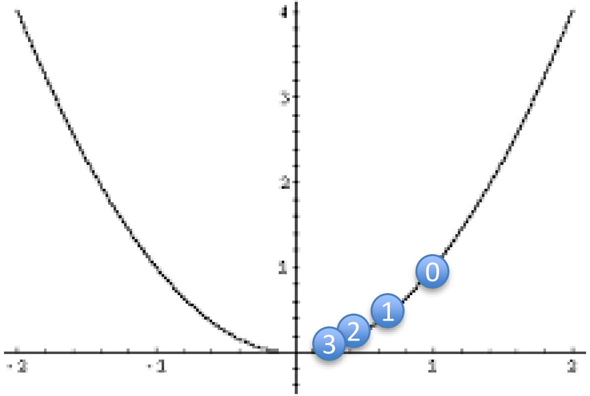
Each iteration we move opposite the gradient by: learning rate(η) * gradient

$$x_{i+1} = x_i - \eta \nabla f(x_i)$$

$$y_{i+1} = x_{i+1}^2$$

Assume $\eta = 0.1$ and let's start at x_0=1

$$x_{i+1} = x_i - 0.1 * 2x_i = x_i - 0.2x_i = 0.8x_i$$



i	x_i	y_i
0	1	1
1	0.8	0.64
2	0.64	0.410
3	0.512	0.262

Duke
PRATT SCHOOL of ENGINEERING

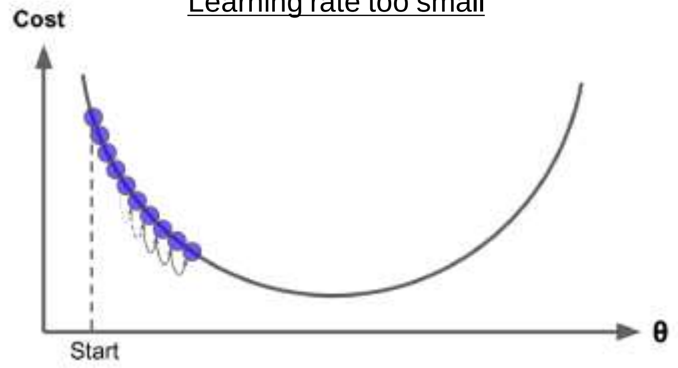
Learning rate

- The learning rate η is a hyperparameter we must set

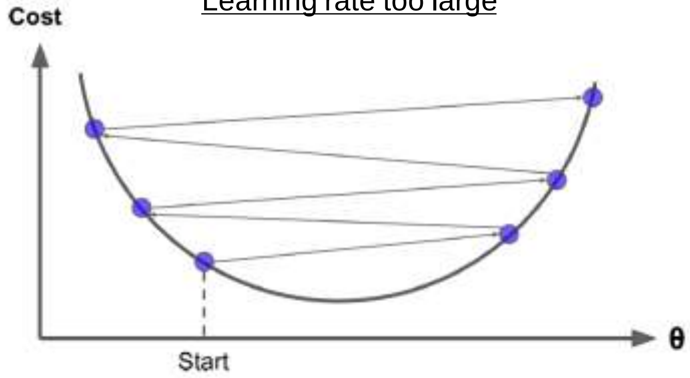
$$x_{i+1} = x_i - \eta \nabla f(x_i)$$

- If the learning rate is too small, algorithm may take a very long time to converge
- If the learning rate is too large, the gradient will bounce around and may even diverge
- Some cost functions have several local minima. Fortunately our NLL cost function is convex, meaning it only has one minimum that is the global minimum

Learning rate too small



Learning rate too large



Source: "Hands—On Machine Learning with Scikit Learn & Tensorflow"

Duke PRATT SCHOOL of ENGINEERING

Batch gradient descent on NLL

Our cost function is the **negative log likelihood (NLL)**:

$$NLL(w) = -\sum_{i=1}^{N} y_i \log\left(\sigma(w^T x_i)\right) + (1 - y_i)\log(1 - \sigma(w^T x_i))$$

So our gradient is:

$$\nabla NLL(w) = -\sum_{i=1}^{N} [y_i - \sigma(w^T x_i)] x_i$$

Duke
PRATT SCHOOL of ENGINEERING

Batch gradient descent on NLL

Our gradient is:

$$\nabla NLL(w) = -\sum_{i=1}^{N} [y_i - \sigma(w^T x_i)] x_i$$

Therefore, our gradient descent equation is:

$$w_t = w_{t-1} - \eta \nabla NLL$$

$$w_t = w_{t-1} + \eta \sum_{i=1}^{N} [y_i - \sigma(w_{t-1}^T x_i)] x_i$$

Duke
PRATT SCHOOL of ENGINEERING

Recap so far

- In logistic regression we use the **logistic/sigmoid function** to predict the probability of the positive class (y=1)

$$P(y=1) = \sigma(w^T x) = \frac{1}{1 + e^{-(w^T x)}}$$

- We calculate the likelihood of ***w*** and our cost function we seek to minimize is the **negative log likelihood**
- We typically use **gradient descent** to find values of the parameters *w* that **minimize the NLL**
- Then we have a model that can generate probabilistic predictions
- We convert those probabilities into a 0/1 prediction by setting our **threshold** (default = 0.5)

Duke
PRATT SCHOOL of ENGINEERING

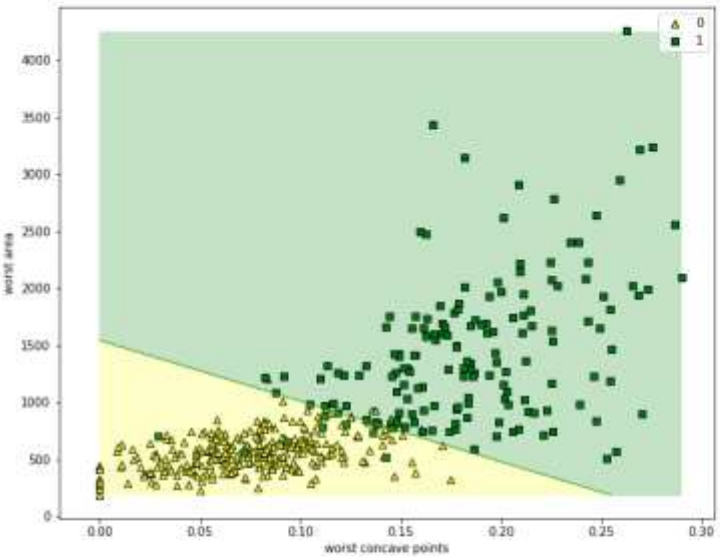
Logistic regression in SKLearn

- Scikit-learn makes it very easy for us to run a logistic regression classifier
- Behind the scenes SKLearn uses an iterative solving method to get the parameters *w*
- We can generate predictions in two ways:
  - model.predict(X) gives us 0/1 predictions
  - model.predict_proba(X) gives us the probabilities from 0 to 1
    - We then apply a threshold to convert to 0/1 predictions
- We can also apply regularization just as we do in linear regression. In fact, SKLearn applies the L2 penalty (ridge) by default

Duke
PRATT SCHOOL of ENGINEERING

Decision boundary

- Logistic regression is still considered a linear model
  - It is a linear combination of the coefficients *w* – no interactions between coefficients
- We can plot its predictions vs. features and see that it forms a linear boundary separating each predicted class



Duke PRATT SCHOOL of ENGINEERING

DEMO: LOGISTIC REGRESSION

outrageously
AMBITIOUS

What if we have more than two classes?

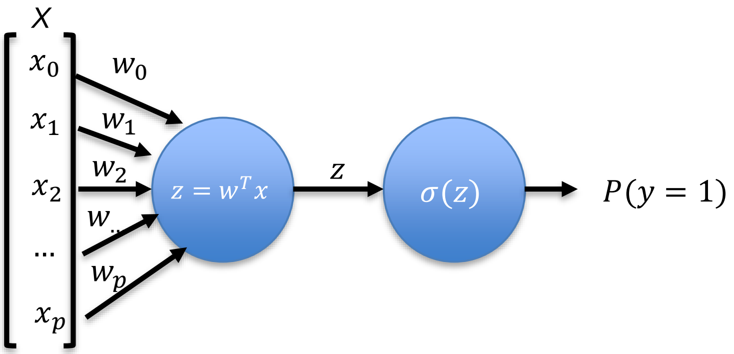
Duke
PRATT SCHOOL of ENGINEERING

Predicting multiple classes

- Logistic regression function gave us the probability of the positive class
- But what if we have several classes – which one is “positive”?
- Instead of the sigmoid function, we use another function called the **softmax function** to give us the probability of belonging to each class

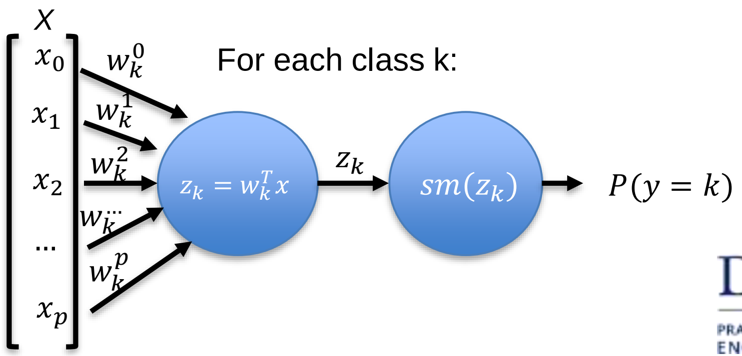
Binary (2 classes)

$$P(y_i = 1|x_i) = \sigma(w^T x)$$



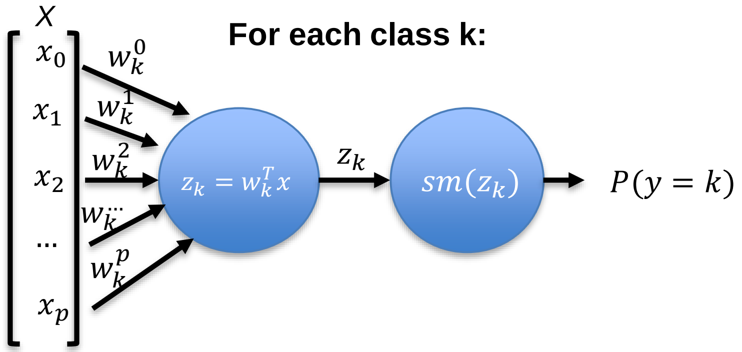
Multiclass

$$P(y_i = k|x_i) = softmax(w_k^T x)$$



Softmax regression

- In logistic regression we had a single set of weights *w*
- In softmax regression we have a set of weights for each class: w_k
- We multiple the weights by *x* to get a "score" *z* for each class
- We then apply the softmax function to normalize the scores so that they sum to 1

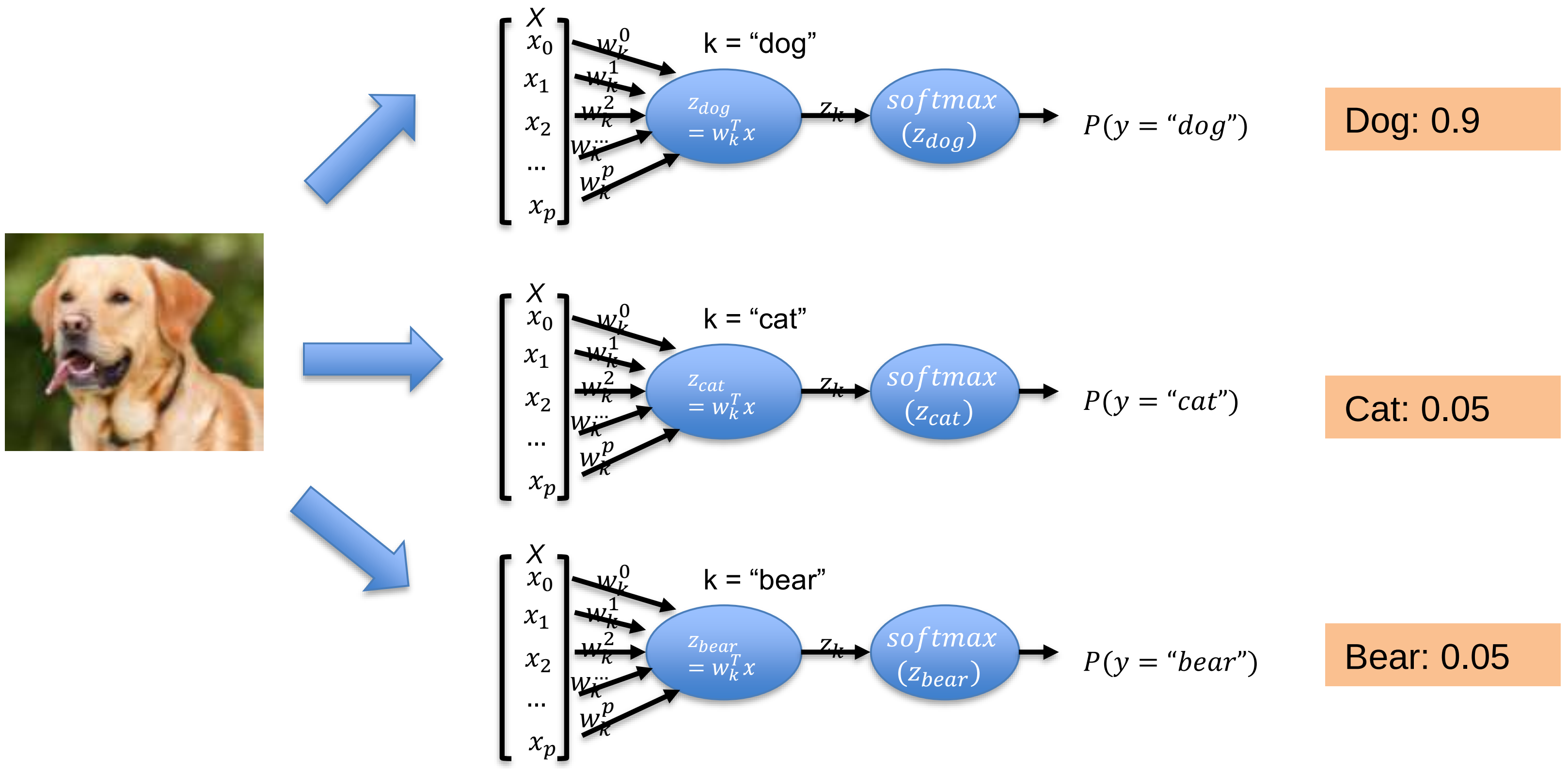


- The "prediction" is the class with the highest score, or all scores above a threshold

$$softmax(z_k) = \frac{e^{z_k}}{\sum_{j=1}^{k} e^{z_k}} \qquad P(y = k) = softmax(w_k^T x) = \frac{e^{w_k^T x}}{\sum_{j=1}^{k} e^{w_k^T x}}$$

Duke
PRATT SCHOOL of ENGINEERING

Softmax regression



Duke
PRATT SCHOOL of ENGINEERING

Softmax regression

In softmax regression we use **cross entropy** as the cost function instead of negative log likelihood as we did for logistic regression

$$H(p,q) = -\sum_{i=1}^{N} p(x)\log[q(x)]$$

$$J(w) = -\sum_{i=1}^{N}\sum_{k=1}^{K} y_k^{(i)}\log(softmax(w_k^T x_i))$$

$y_k^{(i)}$ is the probability that the ith instance belongs to class k (0 or 1)

- When there are just 2 classes, we can show that the above cost function is equivalent to negative log likelihood
- As in logistic regression, we solve for *w* that minimizes this using gradient descent. We can then apply our model and take the class with the highest probability as the predicted class

Duke
PRATT SCHOOL of ENGINEERING

DEMO: SOFTMAX REGRESSION