Machine Learning

|Softmax Regression

Prof. Dr. Mostafa Elhosseini

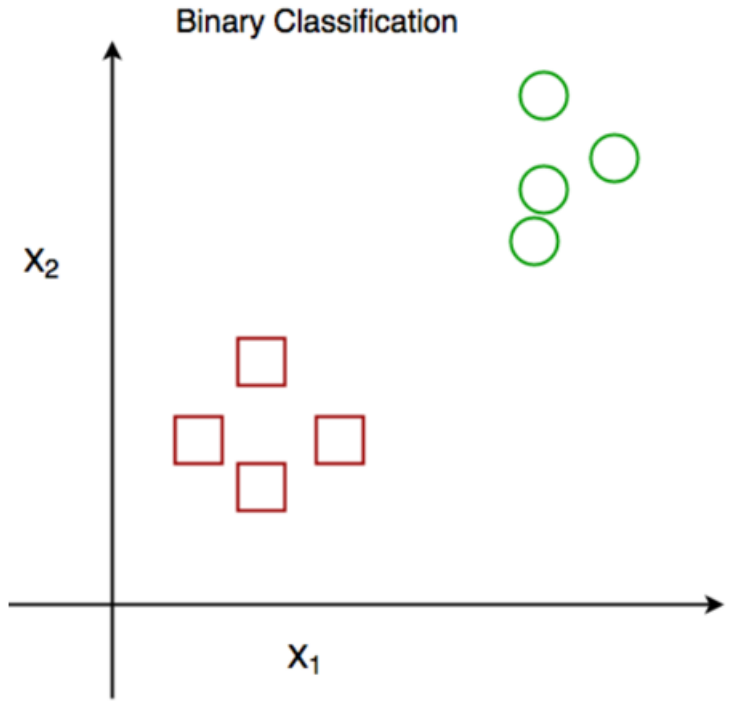
https://youtube.com/ElhosseiniAcademy

Agenda

- Multiclass Classification
- Softmax Regression
- Intuition Behind Softmax Activation function
- Numerical Example
- Error Function
- Cross Entropy
- Intuition Behind Cross Entropy
- Model Training
- Gradient of Cross Entropy

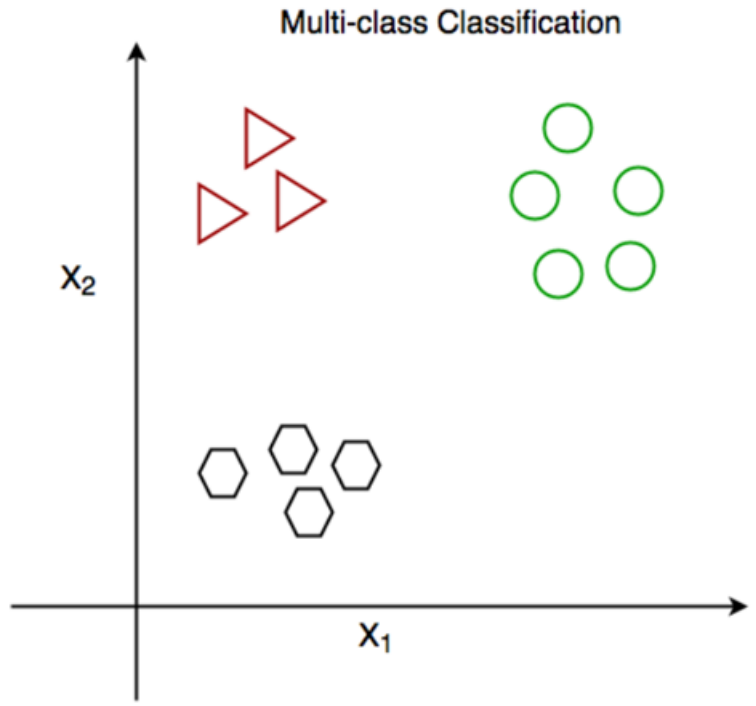
Binary Classification

- Classifying email into the two classes
  - Spam / or not spam
  - Binary classifier problem
  - Multiclass // Multinomial



Multiclass (Multinomial) Classification

- Distinguish between more than two classes.
  - Email: Spam – Personal email – Work-related email – family email
  - Cancer: benign – Type 0 – Type 1 – Type 2
  - Weather: Sunny – Cloudy – Overcast – Snow
  - Classifying handwritten digits.
  - Classifying news wires by topic
  - Movie descriptions

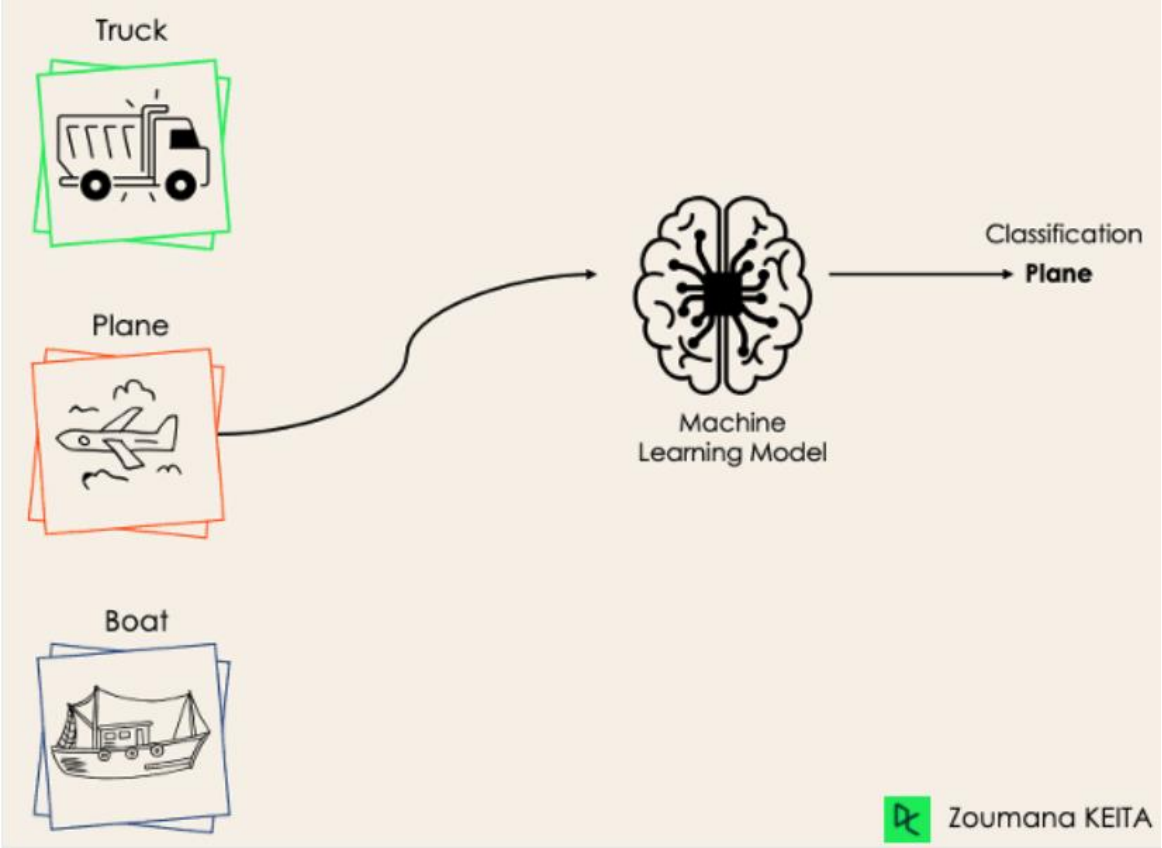


Multiclass Classification

- Multiclassification uses more than two classes, such as the **10 classes**, 0 through 9, in the **Digits** dataset.

Multiclass Classification

- The goal is to predict to which class a given input example belongs to.



Multiclass Classification

- Some algorithms are capable of handling multiple classes **directly**
  - LogisticRegression,
  - RandomForestClassifier, and
  - GaussianNB.
- Others are **strictly binary** classifiers (such as
  - SGDClassifier and
  - SVC.

Multiclass Classification

- However, there are various strategies that you can use to perform multiclass classification using multiple binary classifiers
  - One-Versus-All OvA
  - One-Versus-One OvO

Softmax Regression

- Multinomial (multiclass) logistic regression

- Given an instance represented by a feature vector x, **softmax** regression outputs the **probabilities that** x **belongs** to each of several classes.

- The model calculates these probabilities by applying the softmax function to the linear scores of each class calculated from x.

Softmax Regression

- Note: Each class has its own dedicated parameter vector $\boldsymbol{\theta}^{(k)}$

- when given an **instance** $\boldsymbol{x}$, the Softmax Regression model first **computes a score** $\mathrm{s_k}(\boldsymbol{x})$ for each class k : (**logit** for class k)

$$\mathrm{s_k}(\boldsymbol{x}) = \boldsymbol{\theta_k}^{\boldsymbol{T}}.\boldsymbol{x}$$

- Then **estimates the probability** of each class by **applying the softmax** function to the scores

Softmax Regression

- Estimate the probability $\hat{P}_k$ that the instance belongs to class k

$$\hat{p}_k = \sigma\big(s(x)\big)_k = \frac{\exp\big(s_k(x)\big)}{\sum_{j=1}^{k} \exp\left(s_j(x)\right)}$$

- Softmax is an activation function
- k is the number of classes
- $s(x)$ is the score of class for the instance x
- $\sigma\big(s(x)\big)_k$ is the estimated probability that the instance x belongs to class k given the scores of each class for that instance

Softmax Regression

- Notes:
- The term e^{z_i} transforms the raw score into a positive number.
- Exponentiation ensures that all transformed scores are positive and enhances the differences between scores.
- The exponentiation step ensures that larger logits correspond to larger probabilities, but the relative differences between logits are maintained.
- A small change in the logit can lead to a significant change in the probability due to the exponential nature of the function.

Softmax Regression

- Notes:
- The normalization step ensures that the probabilities across all classes sum to 1. This makes the classes compete against each other; increasing the probability of one class decreases the probability of others.
- Softmax regression is a **linear classifier** because the decision boundaries it produces are linear. The **nonlinearity** introduced by the **softmax** function **affects the output probabilities** but does not change the linear nature of the decision boundaries.

Softmax Regression

- Key Reasons for Using Softmax
- Interpretability: Softmax turns logits into probabilities, making model outputs easier to interpret.
- Training: It facilitates gradient-based optimization by providing probabilities needed for the cross-entropy loss function.
- Inference: It provides confidence scores for each class, useful in decision-making processes.

Softmax Regression

- Use the **argmax** function to find the class k that has the highest score.

$$\hat{y} = \arg\max_k \sigma(s(\mathbf{x}))_k$$

$$\hat{y} = \arg\max_k s_k(\mathbf{x})$$

$$\hat{y} = \arg\max_k (\theta_k^T \cdot \mathbf{x})$$

Numerical Example

- Input feature $\boldsymbol{x} = [x_1, x_2]$ Suppose $\boldsymbol{x} = [2, 3]$
- Three classes $(0, 1, 2)$
- $\boldsymbol{\theta_0} = [0.5, 1]$ $\boldsymbol{\theta_1} = [-1, 0.5]$ $\boldsymbol{\theta_2} = [1.5, -0.5]$

- $s_0(x) = {\theta_0}^T . \boldsymbol{x} = 0.5x_1 + 1.0x_2 = 4$
- $s_1(x) = {\theta_1}^T . \boldsymbol{x} = -1x_1 + 0.5x_2 = -0.5$
- $s_2(x) = {\theta_2}^T . \boldsymbol{x} = 1.5x_1 - 0.5x_2 = 1.5$

Numerical Example

- $s_0(x) = {\theta_0}^T . \boldsymbol{x} = 0.5x_1 + 1.0x_2 = \mathbf{4}$
- $s_1(x) = {\theta_1}^T . \boldsymbol{x} = -1x_1 + 0.5x_2 = -0.5$
- $s_2(x) = {\theta_2}^T . \boldsymbol{x} = 1.5x_1 - 0.5x_2 = 1.5$
- logits $z = [4, -0.5, 1.5]$
- Exponentiate the logits: $e^z = [e^4, e^{-0.5}, e^{1.5}] = [54.598, 0.606, 4.48]$
- Normalization term: $\sum_{j=1}^{3} e^{z_j} = 54.598 + 0.606 + 4.48 = 59.6862$

Numerical Example

- Calculate the probabilities
- $p(y=0) = \frac{e^{4}}{59.6862} = \mathbf{0.9146}$ $p(y=1) = \frac{e^{-0.5}}{59.6862} = 0.0102$
- $p(y=2) = \frac{e^{1.5}}{59.6862} = 0.0752$
- The softmax function converts the logits [4, -0.5, 1.5] into probabilities [0.9146, 0.0102, 0.0752]. The class with the highest probability is class 0, with a probability of 0.9146.

Training

- The objective of training a softmax regression model is to estimate high probabilities for the correct class labels (target classes) and low probabilities for the incorrect classes.
- $J(\theta) = -\frac{1}{m}\sum_{i=1}^{m}\sum_{k=1}^{K} {y_k}^{(i)} \log({\hat{p}_k}^{(i)})$ - Cross Entropy
- m: is the number of training examples K: is the number of classes
- ${y_k}^{(i)}$: is a binary indicator (0 or 1) if class label k is the correct classification for example i.
- ${\hat{p}_k}^{(i)}$: is the predicted probability that example i belongs to class k.

Intuition behind Cross-entropy

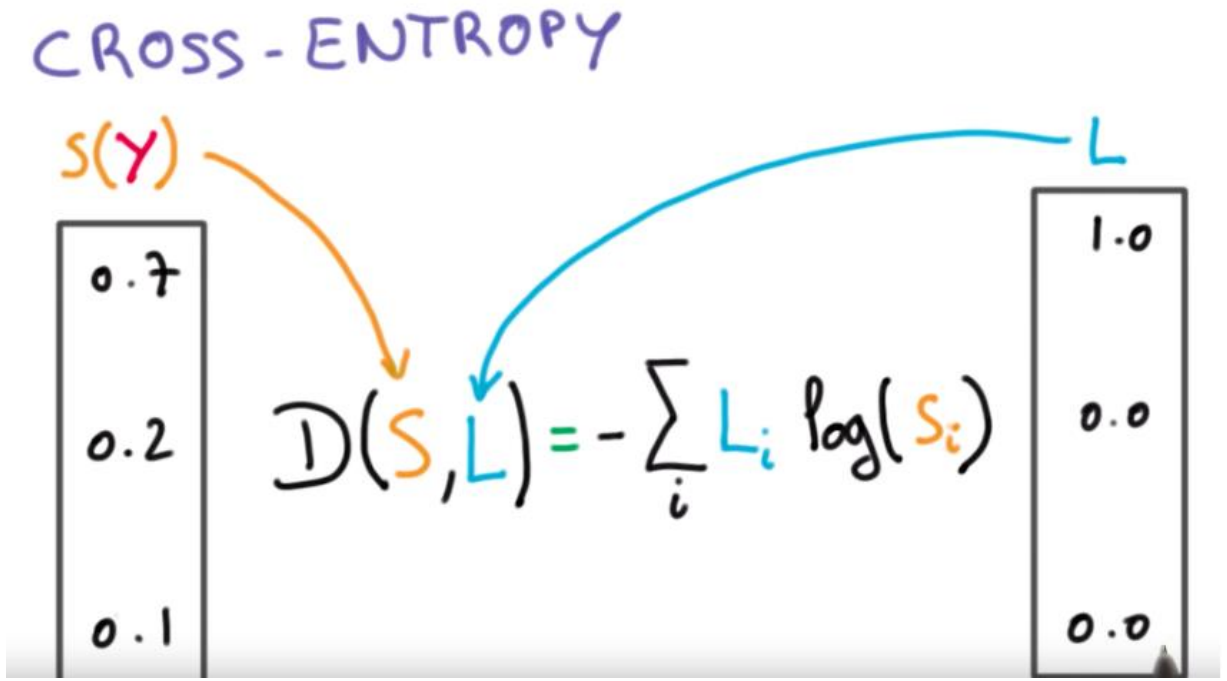
- If we hard code our label data to the vectors below, in a format typically used to turn categorical data into numbers
- [1,0,0]: #cat [0,1,0]: #dog [0,0,1]: #bird
- The output probabilities are saying 70% sure it is a cat, 20% a dog, 10% a bird
- **Cross Entropy** Loss in this case measures how similar your predictions are to the actual labels
  - The natural way to measure the distance between two probability vectors
  - It will be denoted by D for distance

Intuition behind Cross-entropy

- your model predicts a vector of probabilities [0.7,0.2,0.1]
- the first entry is the most likely. And yes your true label says [1,0,0)] - definitely a cat, not a dog at entry 2, and definitely not a bird at entry 3
- The dot product is the sum of the multiplication of two vectors entry-by-entry
- 0.7 *1 + 0.2 * 0 + 0.1 * 0= 0.7
- Note that the **incorrect entry will always be zero** because anything multiplied by zero is zero
- And the true label always only have one entry that is 1 [1,0,0], [0,1,0], [0,0,1]

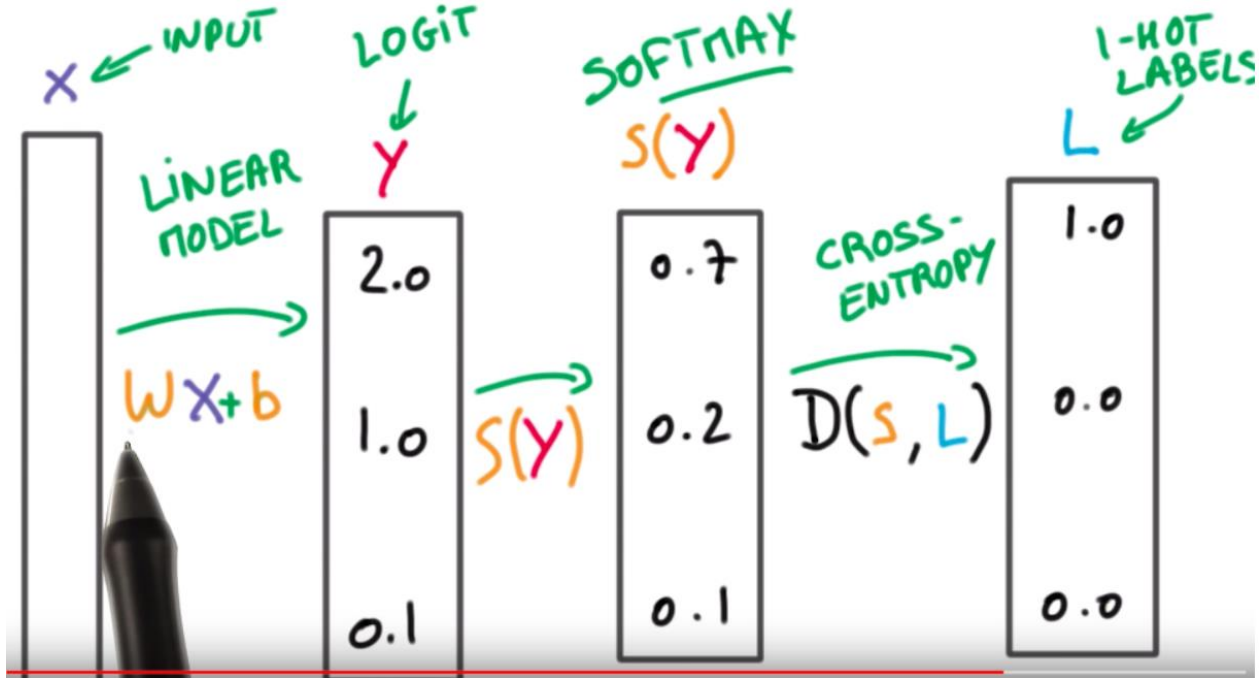
Intuition behind Cross-entropy

- Step 1 Softmax takes the log of that first entry, usually an less-than-one number
- Step 2 the entry is multiplied by the ground truth $\log(0.7) \times 1$ - we do that for each entry $\log(0.2) \times 0$ and $\log(0.1) \times 0$



Intuition behind Cross-entropy

- Data Science Bootcamp



Intuition behind Cross-entropy

- If we have three classes ($K = 3$) and the true class for a given example is class 2, the one-hot encoded label would be: $y = [0,1,0]$
- Because of the one-hot encoding, for each example, only one entry in the vector $y^{(i)}$ is 1 (the correct class), while the rest are 0. This means:
- The cross-entropy loss only considers the term in the sum where $y^{(i)} = 1$.
- For the correct class k, the loss simplifies to: $L^{(i)} = -\log(\hat{p}_k{}^{(i)})$

Intuition behind Cross-entropy

- For the correct class k, the loss simplifies to: $L^{(i)} = -\log(\hat{p}_k{}^{(i)})$
- All other terms where $y_k{}^{(i)} = 0$ do not contribute to the loss, as $0 \times \log(\hat{p}_k{}^{(i)}) = 0$.

- **High Confidence**: If the model is confident and correctly predicts the probability for the **true class** as high (close to 1), $\hat{p}_k{}^{(i)} = 0.9$
- $\log(0.9) \approx -0.105$
- **Small Loss**: Since the loss function is **negative** log-likelihood, loss:$L^{(i)} \approx 0.105$

Intuition behind Cross-entropy

- **Low Confidence**: If the model predicts a low probability for the true class, say $\hat{p}_k{}^{(i)} = 0.1$
- $\log(0.1) \approx -2.302$
- **High Loss**: Since the loss function is **negative** log-likelihood, loss:$L^{(i)} \approx 2.302$
- Intuition: The model is not confident in its prediction, so the penalty (loss) is high.

Gradient of the Loss Function

- The gradient of the loss function $J(\theta)$ with respect to the weight $\theta_{k,j}$ (the weight connecting feature j to class k) is given by:
- $\frac{\partial J(\theta)}{\partial \theta_{k,j}} = \frac{1}{m}\sum_{i=1}^{m}\left(\hat{p}_k{}^{(i)} - y_k{}^{(i)}\right) x_j{}^{(i)}$

- $\hat{p}_k{}^{(i)}$: predicted probability for class k and example i
- $y_k{}^{(i)}$: actual label for class k (1 if correct, 0 otherwise).
- $x_j{}^{(i)}$: the *j*-th feature of the *i*-th training example.

Gradient of the Loss Function

- The gradient of the loss function $J(\theta)$ with respect to the weight $\theta_{k,j}$ (the weight connecting feature j to class k) is given by:
- $\frac{\partial J(\theta)}{\partial \theta_{k,j}} = \frac{1}{m}\sum_{i=1}^{m}\left(\hat{p}_k{}^{(i)} - y_k{}^{(i)}\right) x_j{}^{(i)}$

- $\hat{p}_k{}^{(i)} - y_k{}^{(i)}$: represents the error for class k for the i-th example. It shows how much the predicted probability deviates from the actual label.
- $x_j{}^{(i)}$: scales this error by the value of the j-th feature in the i-th example. This shows how much each feature contributes to the overall error.

Gradient of the Loss Function

- Adjusting the weights in the direction opposite to the gradient
- The general update rule for gradient descent is:
- $\frac{\partial J(\theta)}{\partial \theta_{k,j}} = \frac{1}{m}\sum_{i=1}^{m}\left(\hat{p}_k{}^{(i)} - y_k{}^{(i)}\right) x_j{}^{(i)}$
- $\theta_{k,j} := \theta_{k,j} - \alpha \frac{\partial J(\theta)}{\partial \theta_{k,j}}$
- $\theta_{k,j} := \theta_{k,j} - \alpha \frac{1}{m}\sum_{i=1}^{m}\left(\hat{p}_k{}^{(i)} - y_k{}^{(i)}\right) x_j{}^{(i)}$

Next Lecture

- Numerical Examples
- Iris Dataset