Cost functions: Classical Machine Learning

We have thus far presented cost functions for Regression and Classification

- Mean Squared Error (Regression)
- Cross Entropy (Classification)

with little more than "intuitive justification.

We now explain them more mathematically.

In Classical Machine Learning, a technique called *Maximum Likelihood* is the basis for many algorithms.

We explain this idea and show how the cost functions encountered thus far can be explained in terms of likelihood maximization.

Supervised prediction as likelihood

In the Classification task, we are predicting a distribution of values rather than a single value.

The Regression task, at first glance, appears to predict a single value rather than a distribution of values.

There is an interpretaton of the Regression task that views it as also predicting a distribution of values.

This will be very useful in explaining where the Cost function comes from.

It's possible to have two training examples i, i' with identical features but different targets

- $\mathbf{x^{(i)}} = \mathbf{x^{(i')}}$ but
- $\mathbf{y^{(i)}} \neq \mathbf{y^{(i')}}$

This means that the estimator is not a function.

A simple explanation is one of measurement error

- Imprecision in measuring the targets $\mathbf{y^{(i)}}$, $\mathbf{y}^{(i')}$
    - There is a "true" (in terms of a function mapping) $\tilde{\mathbf{y}}^{\mathbf{(i)}}$ such that
        - $\mathbf{y^{(i)}} = \tilde{\mathbf{y}}^{\mathbf{(i)}} + \epsilon^{\mathbf{(i)}}$
        - $\mathbf{y}^{(i')} = \tilde{\mathbf{y}}^{\mathbf{(i)}} + \epsilon^{(i')}$
    - i.e., the two targets are really the same $\tilde{\mathbf{y}}^{\mathbf{(i)}}$, but have been mis-measured
- Imprecision in measuring features $\mathbf{x^{(i)}}$, $\mathbf{x}^{(i')}$
    - the "true" feature values are *different* $\tilde{\mathbf{x}}^{\mathbf{(i)}} \neq \tilde{\mathbf{x}}^{(i')}$ but mis-measured as equal
        - $\mathbf{x^{(i)}} = \tilde{\mathbf{x}}^{\mathbf{(i)}} + \epsilon^{\mathbf{(i)}}$
        - $\mathbf{x}^{(i')} = \tilde{\mathbf{x}}^{(i')} + \epsilon^{(i')}$

So rather than our model (estimator) predicting a single value, it predicts a distribution of values.

Can frame the Supervised Learning task as being creating a model to predict

$$p(\mathbf{y^{(i)}}|x^{(\mathbf{i})})$$

the *conditional probability* of $\mathbf{y^{(i)}}$ given input $\mathbf{x^{(i)}}$.

Log likelihood

Deep Learning Book 5.5 (https://www.deeplearningbook.org/contents/ml.html)

The training data $\{x^{(\mathbf{i})}, \mathbf{y}^{(\mathbf{i})} | i = 1, \ldots, m\}$ is a sample from an unknown joint distribution $p_{\text{data}}(\mathbf{x}, \mathbf{y})$.

So the training data is an empirical distribution of some true but unknown underlying $p_{\text{data}}(\mathbf{x}, \mathbf{y})$

A model (parameterized by Θ) creates an *approximation* $p_{\text{model}}(\mathbf{x}, \mathbf{y}; \Theta)$ of $p_{\text{data}}(\mathbf{x}, \mathbf{y})$.

Note that $p_{\text{data}}(\mathbf{x}, \mathbf{y})$ is not parameterized by Θ.

We can motivate the choice of Θ by the principle of Likelihood Maximization.

Given the training set, and the true joint distribution $p_{\text{data}}(\mathbf{x}, \mathbf{y})$, we can compute the likelihood (i.e, probability) of drawing the m samples in the training set as

$$\mathbb{L}_{\text{data}} = \prod_{i=1}^{m} p_{\text{data}}(\mathbf{x^{(i)}}, \mathbf{y^{(i)}})$$

Similarly, we can compute the same likelihood, using the probabilities from the model

$$\mathbb{L}_{\text{model}} = \prod_{i=1}^{m} p_{\text{model}}(\mathbf{x^{(i)}}, \mathbf{y^{(i)}}; \Theta)$$

We can turn this product into a sum by taking the log of both sides

$$\log(\mathbb{L}_{\text{model}}) = \sum_{i=1}^{m} \log(p_{\text{model}}(\mathbf{x^{(i)}}, \mathbf{y^{(i)}}; \Theta))$$

This is called the Log Likelihood.

How do we choose Θ ?

Let us choose Θ such that the choice *maximizes* the likelihood of seeing the training set.

$$\hat{\Theta} = \underset{\Theta}{\operatorname{argmax}} \sum_{i=1}^{m} \log(p_{\text{model}}(\mathbf{x^{(i)}}, \mathbf{y^{(i)}}; \Theta))$$

That is, the choice of Θ that results in a model which best approximates the empirical (training) data.

Finding the best Θ means maximizing the log likelihood of the model.

KL divergence

We can now motivated the KL divergence:

- the difference between the log likelihood of the empirical and model distributions.

Bayes Theorem relates joint and conditional probabilities

$$
\begin{aligned}
p(\mathbf{y}|\mathbf{x}) &= \frac{p(\mathbf{x},\mathbf{y})}{p(\mathbf{x})} \\
p(\mathbf{x}, \mathbf{y}) &= p(\mathbf{y}|\mathbf{x})\ p(\mathbf{x}) \quad \text{re-arrrange the terms}
\end{aligned}
$$

So we can re-write

$$
\begin{aligned}
\log(\mathbb{L}_{\text{model}}) &= \sum_{i=1}^{m} \log(p_{\text{model}}(\mathbf{x^{(i)}}, \mathbf{y^{(i)}}; \Theta)) \\
&= \sum_{i=1}^{m} \log(p_{\text{model}}(\mathbf{y^{(i)}}|\mathbf{x^{(i)}}; \Theta))\ p(\mathbf{x^{(i)}}) \\
&= \mathbb{E}_{\mathbf{x}\sim p_{\text{data}}} \log(p_{\text{model}}(\mathbf{y^{(i)}}|\mathbf{x^{(i)}}; \Theta))
\end{aligned}
$$

and similarly for $\log(\mathbb{L}_{\text{data}})$

The difference between the log likelihoods of the two distributions

$$\log(\mathbb{L}_{\text{data}}) - \log(\mathbb{L}_{\text{model}}) \quad = \quad \mathbb{E}_{\mathbf{x}\sim p_{\text{data}}}(\log(p_{\text{data}}(\mathbf{y}|\mathbf{x}))) - \mathbb{E}_{\mathbf{x}\sim p_{\text{data}}}(\log(p_{\text{model}}(\mathbf{y}|\mathbf{x};\Theta)))$$

You hopefully recognize the difference as being equal to the definition of KL Divergence.

Thus, the KL divergence is explained as the difference between the log likelihoods of the empirical and model distributions.

Cost functions for Classical Machine Learning

We now show that our choice of cost functions

- MSE for Regression
- Binary Cross Entropy for (binary) Classification

can be justified in terms of maximization of the log likelihood.

Log likelihood of Binary classifiction

For binary classification (where $\hat{y}^{(\mathbf{i})} \in \{0, 1\}$) we compute a score as a linear function of $\mathbf{x}$

$$s(\mathbf{x}^{(\mathbf{i})}) = \Theta^T \cdot \mathbf{x}^{(\mathbf{i})}$$

and we convert the linear score into a probability via the logistic function

$$\hat{p}^{(\mathbf{i})} = \sigma(s(\hat{\mathbf{x}}^{(\mathbf{i})}))$$

A Positive prediction (i.e., prediction of value 1) is the conditional probability

$$p(\hat{y}^{(\mathbf{i})} = 1 \mid \mathbf{x}^{(\mathbf{i})}) = \hat{p}^{(\mathbf{i})}$$

And a Negative prediction (i.e., prediction of value 0) is

$$p(\hat{y}^{(\mathbf{i})} = 0 \mid \mathbf{x}^{(\mathbf{i})}) = 1 - \hat{p}^{(\mathbf{i})}$$

We can combine the equations for the two cases into the single equation

$$p(\hat{y}^{(\mathbf{i})}|\mathbf{x}^{(\mathbf{i})}) = p(\hat{y}^{(\mathbf{i})} = 1|\mathbf{x}^{(\mathbf{i})})^{\mathbf{y}^{(\mathbf{i})}} * p(\hat{y}^{(\mathbf{i})} = 0|\mathbf{x}^{(\mathbf{i})})^{(1-\mathbf{y}^{(\mathbf{i})})}$$

(because $\mathbf{y} \in \{0, 1\}$, one term in the product always has exponent 0)

Again, the likelihood is the product (over i) of these terms and the log likelihood is

$$\begin{aligned} l &= \sum_{i=1}^{m} \mathbf{y^{(i)}} \log(p(\hat{y}^{\mathbf{(i)}} = 1|\mathbf{x^{(i)}})) + (1 - \mathbf{y^{(i)}}) \log(p(\hat{y}^{\mathbf{(i)}} = 0|\mathbf{x^{(i)}})) \\ &= \sum_{i=1}^{m} \mathbf{y^{(i)}} \log(p(\hat{y}^{\mathbf{(i)}} = 1|\mathbf{x^{(i)}})) + (1 - \mathbf{y^{(i)}}) \log(1 - p(\hat{y}^{\mathbf{(i)}} = 1|\mathbf{x^{(i)}})) \end{aligned}$$

You should recognize the (negative of) the Log Likelihood as the Binary Cross Entropy loss.

So maximizing the log likelihood minimizes the Binary Cross Entropy Loss.

Log Likelihood of Linear models with normal errors

Our Linear models are of the form

$$\hat{\mathbf{y}}^{(\mathbf{i})} = \Theta^T \cdot \mathbf{x}^{(\mathbf{i})} + \epsilon$$

where $\epsilon \in \mathcal{N}(0, \sigma)$.

The ϵ can be interpretted in either of two ways

- as an approximation error (inability to fit data exactly)
- measurement error, as explained above

So our prediction $\hat{\mathbf{y}}^{(\mathbf{i})} = p(\hat{\mathbf{y}}^{(\mathbf{i})}|\mathbf{x}^{(\mathbf{i})})$

$$\hat{\mathbf{y}}^{(\mathbf{i})} = \Theta^T \cdot \mathbf{x}^{(\mathbf{i})} + \epsilon$$

becomes a Normal random variable with mean $\mu = \Theta^T \cdot \mathbf{x}^{(\mathbf{i})}$ and standard deviation σ.

Substituting the formula for Normal distribution, the conditional probability of $\hat{\mathbf{y}}^{(\mathbf{i})}$ given $\mathbf{x}^{(\mathbf{i})}$ is

$$
\begin{aligned}
p(\hat{\mathbf{y}}^{(\mathbf{i})}|\mathbf{x}^{(\mathbf{i})}) &= \frac{1}{\sigma\sqrt{(2\pi)}}\exp(-\frac{(\hat{\mathbf{y}}^{(\mathbf{i})}-\mu)^2}{2\sigma}) && \text{def. of Normal} \\
&\propto \exp(-\frac{(\hat{\mathbf{y}}^{(\mathbf{i})}-\Theta^T\cdot\mathbf{x}^{(\mathbf{i})})^2}{2\sigma}) && \text{def. of } \mu
\end{aligned}
$$

The Likelihood of the training set, given this model of the conditional probability, is just the product over the training set of $p(\hat{\mathbf{y}}^{(\mathbf{i})}|\mathbf{x}^{(\mathbf{i})})$:

$$\mathbb{L}_{\text{model}} = \prod_{i=1}^{m} p(\hat{\mathbf{y}}^{(\mathbf{i})}|\mathbf{x}^{(\mathbf{i})})$$

and the Log Likelihood is

$$\begin{aligned}
\mathbb{l}_{\text{model}} &= \log(\textstyle\prod_{i=1}^{m} p(\hat{\mathbf{y}}^{(\mathbf{i})}|\mathbf{x}^{(\mathbf{i})})) \\
&= \textstyle\sum_{i=1}^{m} \log(p(\hat{\mathbf{y}}^{(\mathbf{i})}|\mathbf{x}^{(\mathbf{i})})) \\
&= \textstyle\sum_{i=1}^{m} -\frac{(\hat{\mathbf{y}}^{(\mathbf{i})} - \Theta^T \cdot \mathbf{x}^{(\mathbf{i})})^2}{2\sigma} \\
&= -\frac{1}{2\sigma} \textstyle\sum_{i=1}^{m} (\hat{\mathbf{y}}^{(\mathbf{i})} - \Theta^T \cdot \mathbf{x}^{(\mathbf{i})})^2
\end{aligned}$$

You should recognize the (negative of) the Log Likelihood as the Mean Squared Error (MSE).

So maximizing the log likelihood minimizes the MSE.

Complex loss functions: multiple objectives

Regularization objectives

Cost functions for Deep Learning: Preview

The Cost functions for Classical Machine Learning were perhaps motivated by the desire for closed form solutions.

In Deep Learning, the optimization is typically solved via search.

This opens the possibilities of complex cost functions that don't require closed form solution.

As we will see in the Deep Learning part of this course, the key part of solving a task is in *defining* a cost function that mirrors the task's objective.

Thus, many cost functions are problem specific and often quite creative.

Cool cost functions: Neural Style Transfer

Neural Style Transfer

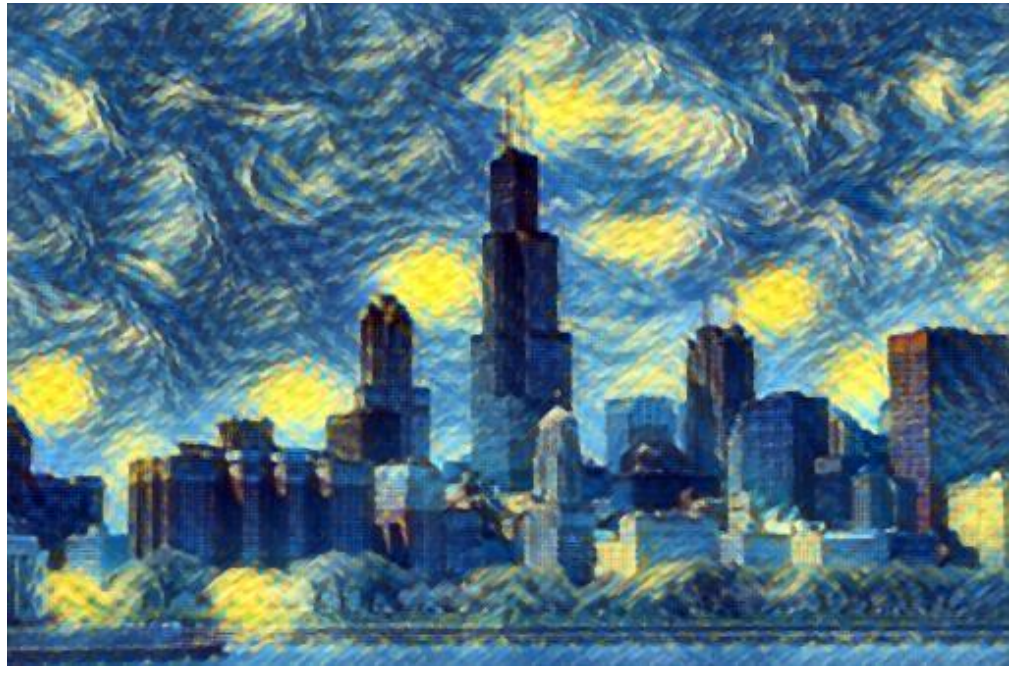
Given

- a "Content" Image that you want to transform
- a "Style" Image (e.g., Van Gogh "Starry Night")
- Generate a New image that is the Content image redrawn in the style of the Style Image
    - Gatys: A Neural Algorithm for Style (https://arxiv.org/abs/1508.06576)
    - Fast Neural Style Transfer (https://github.com/jcjohnson/fast-neural-style)

Content image

Style image

Generated image



Cost function

Definitions:

- Style image, represented as a vector of pixels $\vec{a}$
- Content image, represented as a vector of pixels $\vec{p}$
- Generated image, represented as a vector of pixels $\vec{x}$

The Loss function (which we want to minimize by varying $\vec{x}$) has two parts

$$\mathrm{L} = \mathrm{L}_{\text{content}}(\vec{p}, \vec{x}) + \mathrm{L}_{\text{style}}(\vec{a}, \vec{x})$$

- a Content Loss
    - measure of how different the generated image $\vec{x}$ is from Content image $\vec{p}$
- a Style Loss
    - measure of how different the "style" of generated $\vec{x}$ is from style of Style image $\vec{a}$

Key: defining what is "style" and similarity of style

In [3]:
```
print("Done")
```

```
Done
```