# Notes for CS 473: Fundamentals of Machine Learning

Chris Kong

November 8, 2023

## Contents

# 1 Feasibility of Learning

## 1.1 Hoeffding Inequality

To describe the boundaries of the outcome of a sample drawn from the population, we have the following inequality:

Hoeffding Inequality

$\mathbb{P}[\mid \nu - \mu \mid > \epsilon] \leq 2e^{-2\epsilon^2 N}$

This is called **Hoeffding Inequality**. Note that in the inequality, $N$ represents the sample size; $\epsilon$ represents the closeness of approximation; $\nu$ represents the probability in the sample; $\mu$ represents the real probability in the population.

## 1.2 Boundaries in Learning

How is the above inequality connected to the process of learning? Here is the answer. First let's re-examine the process:

- $\mu$: Suppose for $\mu$, it means $h(x) \neq f(x)$ in the scale of population.
- $\nu$: Suppose for $\nu$, it means $h(x) \neq f(x)$ in the scale of sample.

Then we have it in a math form:

$$\mu = \mathbb{P}[h(x) \neq f(x)] \equiv E_{out}(h) \qquad \text{(Out-Sample Error)}$$

$$\nu = \frac{1}{N}\sum_{i=1}^{N}[h(x_i) \neq f(x_i)] \equiv E_{in}(h) \qquad \text{(In-Sample Error)}$$

Then we can combine them with the **Hoeffding Inequality** and get the following:

Expanded Hoeffding Inequality 1

$\mathbb{P}[\mid E_{in}(h) - E_{out}(h) \mid > \epsilon] \leq 2e^{-2\epsilon^2 N}$

However, this is not ideal for "Learning" and here is why:

- This is only for a single hypothesis.
- What we have is a hypothesis space $\mathcal{H}$.
- With a single hypothesis, this is more like "testing" than "learning".
- We need to come up on $g \in \mathcal{H}$ which by definition is obtained after seeing the data.

Thus we want to move a bit further by estimating the function $g \in \{h_1, h_2, ..., h_m\} = \mathcal{H}$:

$$\begin{aligned}\mathbb{P}[\mid E_{in}(g) - E_{out}(g) \mid > \epsilon] &\leq \sum_{m=1}^{M} \mathbb{P}[\mid E_{in}(h_m) - E_{out}(h_m) \mid > \epsilon] \\ &\leq \sum_{m=1}^{M} 2e^{-2\epsilon^2 N} \\ \mathbb{P}[\mid E_{in}(g) - E_{out}(g) \mid > \epsilon] &\leq 2Me^{-2\epsilon^2 N}\end{aligned}$$

In the above inequality, $M$(the number of hypothesis in $\mathcal{H}$) can be seen as the complexity of hypothesis space. Under certain assumptions (unseen samples are drawn with the same probability distribution P as the seen examples) we can bound the difference between in-sample and out-of-sample errors.

Expanded Hoeffding Inequality 2

$\mathbb{P}[\mid E_{in}(g) - E_{out}(g) \mid > \epsilon] \leq 2Me^{-2\epsilon^2 N}$

## 1.3 Summary and Thoughts

### 1.3.1 Summary

- The goal of learning is to approximate the behavior of the unknown function $f$.
- We are usually provided with a finite set of examples which respect the behavior of $f$ (which we usually call it Training Set).
- The assumption is that each example is independently drawn from drawn from the entire sample set with some probability distribution $\mathbf{P}$.
- We identify a hypothesis set $\mathcal{H}$ from which our goal is to pick a function $g$.
- $g$ should be such that it approximates the function f on unseen data.
- The inequality implies that if we can bring the in-sample error towards zero we can probabilistically guarantee that out-of-sample error will not be far either. This essentially means that the hypothesis $g$ will probabilistically approximate function $f$.

Here are two key questions:

1. Can we make sure that $E_{out}(g)$ is close to $E_{in}(g)$?
2. Can we make $E_{in}(g)$ small enough?

Hoeffding Inequality only addresses the first question. The second question is addressed by actually choosing a hypothesis set, running the learning algorithm and seeing how low can we make $E_{in}(g)$.

### 1.3.2 Complexity of $\mathcal{H}$

- There is a trade-off in deciding how complex should the hypothesis space $(M)$ be.
- More complex hypothesis enables us to choose a complex $g$ which increases the chances of making $E_{in}(g) \approx 0$.
- At the same time a complex hypothesis space makes the above bound loose making the gap between $E_{in}(g)$ and $E_{out}(g)$ large.

### 1.3.3 Complexity of $f$

- Complexity of $f$ impacts the second question above.
- A more complex $f$ is harder to learn and hence will require a more complex $g$ which in-turn implies a more complex hypothesis set $\mathcal{H}$.

## 2 Linear Models for Regression

### 2.1 Loss Function as Error Measure

As the hypothesis $h$ approaches the unknown function $f$, we would like to have a function to measure its error, such as $E(h, f)$. Even through the sample space $\mathcal{X}$ could be continuous in practical scenario, we define a point-wise error measure as below:

Basic Error Measure Function

$E(h, f) = \sum_{i=1}^{N} e(h(x_i), f(x_i))^2$

The function $e(h, f)$ here can have multiple connotations when measuring different kind of errors, and here are some examples:

- Binary classification error: $e(h, f) = [h(x) \neq f(x)]$
- Squared error: $e(h, f) = (h(x) - f(x))^2$
- Cross-entropy loss: $-[y \log p + (1 - y) \log(1 - p)]$

### 2.2 Linear Regression

The basic linear regression takes form:

$$g(X) = \beta_0 + \sum_{i=1}^{p} \beta_i x_i$$

The learnable parameters are $\beta = (\beta_1, \beta_2, ..., \beta_p)$. The model assumes that either the unknown function $f$ is linear or it can be approximated well as a linear function.
Note that even though it is a linear function, the input variables $\mathbf{x}$ could take various forms. For example:

- Original inputs: $x_1, ..., x_p$
- Transformations of inputs: $\log(x_1), ... \log(x_i)$
- Basis expansions: $x_{i+1} = x_1^2; x_{i+2} = x_1^3$ etc
- Composition of different variables: $x_{i+1} = x_1 \cdot x_2$ etc

### 2.3 Linear Regression Model: Training

We have a set of examples as training data: $D = (x^1, y^1), (x^1, y^1), ..., (x^N, y^N)$, where $x^i = [x_1^i, x_2^i, ..., x_p^i]$.

To learn the parameter, we define an error metric (a loss function) which measures the discrepancy between the model prediction and the actual value on the training set. One reasonable metric is the squared error between the two quantities:

$$\begin{aligned} Loss(\beta) = RSS(\beta) &= \sum_{i=1}^{N} (y^i - g_\beta(x^i))^2 \\ &= \sum_{i=1}^{N} (y^i - (\beta_0 + \sum_{j=1}^{p} \beta_j x_j^i))^2 \\ &= \sum_{i=1}^{N} (y^i - \beta_0 - \sum_{j=1}^{p} \beta_j x_j^i)^2 \end{aligned}$$

Note that the "RSS" here means "Residual Sum of Squares" and the meaning of "Learning": to adjust the parameters $\beta = [\beta_0, \beta_1, ..., \beta_p]$. Now let us re-examine the "complex" equation and try to rewrite it using linear algebra. In this sense, we will have $\boldsymbol{\beta}^T = [\beta_0, \beta_1, ..., \beta p]$ and $\mathbf{X} = [1, x_1, x_2, ..., x_p]$. Thus we have:

$$y = g_{\boldsymbol{\beta}}(\mathbf{X}) = \mathbf{X} \cdot \boldsymbol{\beta}$$

Then we can have the Loss Function:

Loss Function

$Loss(\beta) = RSS(\beta) = \sum_{i=1}^{N}(y^i - \beta_0 - \sum_{j=1}^{p} \beta_j x_j^i)^2 = (\boldsymbol{y} - \boldsymbol{X}\beta)^T \cdot (\boldsymbol{y} - \boldsymbol{X}\beta)$

Remember this is a "loss" function, meaning that it measures the loss of our prediction against the actual values. Thus our goal now is to reduce or to minimize the loss to optimize our function. In other words, to minimize the loss is to find hypothesis function $g$ that approximates the true function $f$. To realize this, we need to find the derivative of RSS($\beta$) and set it to zero:

$$\frac{\partial RSS}{\partial \beta} = -2\boldsymbol{X}^T(\boldsymbol{y} - \boldsymbol{X}\beta) = 0$$
$$\hat{\beta} = (\boldsymbol{X}^T\boldsymbol{X})^{-1}\boldsymbol{X}^T\boldsymbol{y}$$

And now we have a closed form solution! Now let's carry on to the next part: Prediction.

## 2.4 Linear Regression Model: Prediction

Simplify the equation by writing it in matrix form and augmenting the vector of inputs $\boldsymbol{x}$ by 1 and subsuming the intercept/bias $\beta$ into the parameter vector $\boldsymbol{\beta}$: $\boldsymbol{\beta} = [\beta_0, \beta_1, ..., \beta_p]$ and $\boldsymbol{X} = [1, x_1, ..., x_p]$. Then we will have the following form:

$$y = g_{\beta}(\boldsymbol{X}) = \boldsymbol{X} \cdot \boldsymbol{\beta}$$
$$\hat{\boldsymbol{\beta}} = (\boldsymbol{X}^T\boldsymbol{X})^{-1}\boldsymbol{X}^T\boldsymbol{y}$$

Now we can do the prediction. For a new observation $x_0$, its prediction can be computed as:

$$\begin{aligned} \hat{y}_0 &= [1, x_0] \cdot \hat{\beta} \\ &= [1, x_0] \cdot (\boldsymbol{X}^T\boldsymbol{X})^{-1}\boldsymbol{X}^T\boldsymbol{y} \end{aligned}$$

## 2.5 Shrinkage Methods (Regularization)

Linear models as-is, particularly those with a large number of variables (and their compositions), trained using least squares estimates, have two flaws:

1. They can tune well on the provided snapshot of the training data but may not have a low out-of-sample error. This is the Bias-Variance trade-off which we will talk about in detail later.
2. A model with large number of variables is less suited for model interpretation. Typically we would like to know which (handful) variables have the most predictive power.

In order to deal with the above flaws, we come up with several methods and "Regularization" is one of them. This is accomplished by augmenting the training of the models in a way so that minimizing the loss function leads to "shrinking"/"deleting" the effect of unimportant variables, which would be effective in preventing the problem of "over-fitting". Here are several ways of **subset selection**:

- Forward and Backward-Stepwise Selection
- Forward-Stagewise Regression
- Skrinkage methods - Ridge and Lasso - Regularization

Here we will discuss Regularization: Ridge and Lasso Regression in particular.

### 2.5.1 Ridge Regression

For Ridge Regression, we impose a penalty on the size of the coefficients being learned:

$$\hat{\beta}^{ridge} = argmin_{\beta} \sum_{i=1}^{N} \left( y^i - \beta_0 - \sum_{j=1}^{p} x_j^i \cdot \beta_j \right)^2$$
$$\text{subject to} \sum_{j=1}^{p} \beta_j^2 \leq s$$

Note that this is a $\boldsymbol{L}_2$ penalty or Weight Decay. The above can also be re-written as:

$$\hat{\beta}^{ridge} = argmin_{\beta} \left[ \sum_{i=1}^{N} \left( y^i - \beta_0 - \sum_{j=1}^{p} x_j^i \cdot \beta_j \right)^2 + \lambda \sum_{j=1}^{p} \beta_j^2 \right]$$

This formula represents the objective of finding a set of parameters $\beta$ (including $\beta_0, \beta_1, \ldots, \beta_p$) that minimizes the overall value of the expression.

**First Part: Residual Sum of Squares**

$$\sum_{i=1}^{N} \left( y^i - \beta_0 - \sum_{j=1}^{p} x_j^i \cdot \beta_j \right)^2$$

This part calculates the squared difference between the model's predictions and the actual values, known as residuals. For each observation $i$ (from 1 to $N$), the difference between the predicted and actual value $y^i$ is squared. All these squared differences are then summed up. This is the part we aim to minimize as we want the model's predictions to be as close to the actual values as possible.

**Second Part: L2 Regularization Term**

$$\lambda \sum_{j=1}^{p} \beta_j^2$$

This part is the essence of Ridge Regression and what differentiates it from ordinary linear regression. It's a regularization term that penalizes the complexity of the model. Specifically, it penalizes the magnitude of the model parameters $\beta_j$. The parameter $\lambda$ is a hyper-parameter that adjusts the strength of regularization: as $\lambda$ increases, the regularization strength increases, and the magnitude of the model parameters decreases. When $\lambda = 0$, Ridge Regression reduces to ordinary linear regression.

**Summary**

Ridge Regression

$$\hat{\beta}^{ridge} = argmin_{\beta} \left[ \sum_{i=1}^{N} \left( y^i - \beta_0 - \sum_{j=1}^{p} x_j^i \cdot \beta_j \right)^2 + \lambda \sum_{j=1}^{p} \beta_j^2 \right]$$

The objective of this formula is to find a set of parameters $\beta$ that minimizes the squared difference between the model's predictions and the actual values while taking into account the complexity of the model. This is achieved by minimizing a combination of the residual sum of squares and the regularization term. In addition, we can find its matrix expression are written as follow:

$$RSS(\lambda) = (\boldsymbol{y} - \boldsymbol{X}\boldsymbol{\beta})^T (\boldsymbol{y} - \boldsymbol{X}\boldsymbol{\beta}) + \lambda \boldsymbol{\beta}^T \boldsymbol{\beta}$$
$$\hat{\beta}^{ridge} = \left( \boldsymbol{X}^T \boldsymbol{X} + \lambda \boldsymbol{I} \right)^{-1} \boldsymbol{X}^T \boldsymbol{y}$$

### 2.5.2 Lasso Regression

Lasso is similar to Ridge Regression but the penalty is different:

$$\hat{\beta}^{lasso} = argmin_{\beta} \sum_{i=1}^{N} \left( y^i - \beta_0 - \sum_{j=1}^{p} x_j^i \cdot \beta_j \right)^2$$

$$\text{subject to} \sum_{j=1}^{p} \mid \beta_j \mid \leq t$$

Lasso Regression

$$\hat{\beta}^{lasso} = argmin_{\beta} \left[ \textstyle\sum_{i=1}^{N} \left( y^i - \beta_0 - \sum_{j=1}^{p} x_j^i \cdot \beta_j \right)^2 + \lambda \sum_{j=1}^{p} \beta_j^q \right]$$

### 2.5.3 Ridge Regression Review

Ridge Regression adds an L2 regularization term to the loss function of linear regression. Its mathematical form is:

$$J(\theta) = \frac{1}{2m} \sum_{i=1}^{m} (h_\theta(x^{(i)}) - y^{(i)})^2 + \lambda \sum_{j=1}^{n} \theta_j^2$$

Where:

- $J(\theta)$ is the loss function.
- $h_\theta(x^{(i)})$ is the hypothesis function, typically $h_\theta(x) = \theta^T x$.
- $m$ is the number of training samples.
- $\theta$ are the model parameters.
- $\lambda$ is the regularization parameter.
- $n$ is the number of features.

### 2.5.4 Lasso Regression Review

Lasso Regression adds an L1 regularization term to the loss function of linear regression. Its mathematical form is:

$$J(\theta) = \frac{1}{2m} \sum_{i=1}^{m} (h_\theta(x^{(i)}) - y^{(i)})^2 + \lambda \sum_{j=1}^{n} |\theta_j|$$

Similar to Ridge Regression, but the regularization term is the sum of the absolute values of the parameters.

### 2.5.5 Comparison and Thoughts

The main difference between the two lies in the regularization term:

- Ridge Regression uses L2 regularization, which shrinks the magnitude of the coefficients but doesn't make them exactly zero.
- Lasso Regression uses L1 regularization, which can make some coefficients exactly zero, thus performing feature selection.

The choice between the two regularization methods depends on the specific application and data.

# 3 Over-fitting and Cross Validation

## 3.1 Over-fitting

Over-fitting is fitting the data more than what we want. It is a phenomenon where fitting the data well no longer indicates that we will get a decent out-of-sample error. In fact, it may enlarge the out-of-sample error.

## 3.2 Regularization

Remember what we just learned about regularization and we have the following expression:

$$min \left[ E_{in}(\boldsymbol{w}) + \frac{\lambda}{N} \boldsymbol{w}^T \boldsymbol{w} \right]$$

The $\lambda$ here penalizes the parameters. The larger $\lambda$ is, the weaker the power of the fitting is. Intuitively enlarging the $\lambda$ will loosen the fitting line. A $\lambda$ that is too large will ultimately cause under-fitting.

## 3.3 Validation and Cross Validation

Before we dive deeper, we should know some basics, three components of data:

- **Training Data:** the data you use to tune the parameters of the model (search for the hypothesis);
- **Validation Data:** part of the data you use to test the goodness of the running hypothesis. Performance on the validation set is a surrogate of the performance on unseen examples — so long as the validation set is drawn from the same distribution as the test set;
- **Test Data:** once you are happy with your chosen hypothesis, test data is used to measure the final performance of the model. There is no going back after this! Otherwise you will be cheating and all bets are off whether your model will generalize to unseen data.

The big assumption here is that the dataset is representative of the real world. Throughout the learning we track the performance on the validation set. Validation Data is also used to avoid over-fitting as the $E_{out}$ will be large given by the validation data.

### 3.3.1 Validation

For any hypothesis $h$, we have:

$$E_{out}(h) = E_{in}(h) + \text{Penalty for over-fitting}$$

Before the math for validation, let us first formalize it:

- Dataset $\boldsymbol{D} = \{(x_1, y_1), (x_2, y_2), ..., (x_N, y_N)\}$
- Training set $\boldsymbol{D}_{train} \in D \leftarrow \boldsymbol{N} - \boldsymbol{K}$ examples
- Validation set $\boldsymbol{D}_{val} \in D \leftarrow \boldsymbol{K}$ examples
- Hypothesis $g^-$: the hypothesis selected after training the model on $\boldsymbol{D}_{train}$

Now we can write out the error function for the validation set:

$$E_{val}(g^-) = \frac{1}{K} \sum_{x_n \in D_{val}} e\left(g^-(x_n), y_n\right)$$

But how is the Error of Validation of $g^-$ reasonable for estimating Out-of-sample Error of $g^-$? Let's have some maths here:

$$
\begin{aligned}
\mathbb{E}\left[E_{val}(g^-)\right] &= \mathbb{E}_{D_{val}}\left[\frac{1}{K}\sum_{x_n\in D_{val}} e\left(g^-(x_n), y_n\right)\right] \\
&= \frac{1}{K}\sum_{x_n\in D_{val}} \mathbb{E}_{D_{val}}\left[e\left(g^-(x_n), y_n\right)\right] \\
&= \frac{1}{K}\sum_{x_n\in D_{val}} E_{out}(g^-) \\
&= E_{out}(g^-)
\end{aligned}
$$

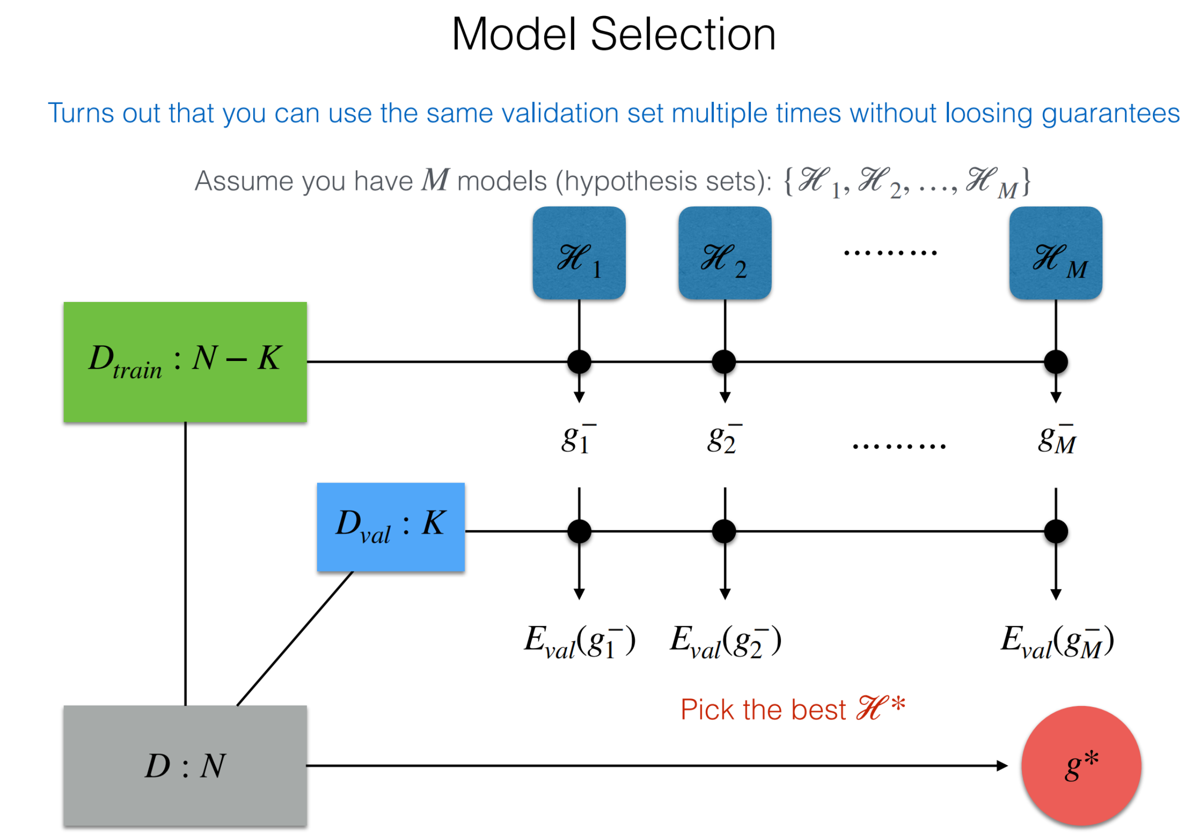
Thus we have:

$$
\begin{aligned}
\mathbb{E}\left[E_{val}(g^-)\right] &= \frac{1}{K}\sum_{x_n\in D_{val}} \mathbb{E}_{D_{val}}\left[e\left(g^-(x_n), y_n\right)\right] = E_{out}(g^-) \\
\mathbb{V}\left[E_{val}(g^-)\right] &= \frac{1}{K^2}\sum_{x_n\in D_{val}} \mathbb{V}_{D_{val}}\left[e\left(g^-(x_n), y_n\right)\right] = \frac{\sigma^2}{K}
\end{aligned}
$$

And we also have a key inequality:

$$
E_{out}(g^-) \leq E_{val}(g^-) + O\left(\frac{1}{\sqrt{K}}\right)
$$

Then we can take a look at Model Selection process as follow:



Model Selection
Turns out that you can use the same validation set multiple times without loosing guarantees
Assume you have $M$ models (hypothesis sets): $\{\mathcal{H}_1, \mathcal{H}_2, \ldots, \mathcal{H}_M\}$
$\mathcal{H}_1$
$\mathcal{H}_2$
$\mathcal{H}_M$
$D_{train} : N - K$
$g_1^-$
$g_2^-$
$g_M^-$
$D_{val} : K$
$E_{val}(g_1^-)$
$E_{val}(g_2^-)$
$E_{val}(g_M^-)$
Pick the best $\mathcal{H}^*$
$D : N$
$g^*$

Figure adapted from the book Learning from Data

### 3.3.2 Cross Validation

Cross Validation (CV) is a statistical method used to estimate the performance of machine learning models. It helps in understanding how the results of a statistical analysis will generalize to an independent dataset. Instead of using the entire dataset to both train and validate the model, it's split multiple times for both purposes. Here's a brief overview:

1. **K-Fold CV**:
    - The most commonly used method of cross-validation.
    - The dataset is randomly partitioned into $k$ equal-sized subsamples.
    - Of the $k$ subsamples, a single subsample is retained as the validation data for testing the model, and the remaining $k-1$ subsamples are used as training data.
    - The cross-validation process is then repeated $k$ times, with each of the $k$ subsamples used exactly once as the validation data.
    - The $k$ results from the folds can then be averaged (or otherwise combined) to produce a single estimation.
2. **Stratified K-Fold CV**:
    - Variation of K-Fold CV where each fold is made by preserving the percentage of samples for each class.
    - Particularly useful for imbalanced datasets.
3. **Leave-One-Out CV (LOOCV)**:
    - A special case of k-fold CV where $k$ is set to the number of observations in the dataset.
    - For a dataset with $N$ observations, $N$ separate times the learning algorithm is trained, each time leaving out one observation.
4. **Leave-P-Out CV**:
    - Similar to LOOCV, but instead of leaving one observation out, we leave $P$ observations out.
5. **Time-Series CV**:
    - Variation for time-ordered data. Instead of randomly partitioning, it takes into account the order of the data.

**Advantages**:

- Provides a more accurate estimate of out-of-sample accuracy.
- More "efficient" use of data as every observation is used for both training and validation.

**Disadvantages**:

- Computationally expensive, especially for large datasets or complex models.

In essence, cross-validation provides a tool to predict the performance of a model on unseen data and helps in preventing overfitting.

**Model Selection with Cross Validation:**

1. Define $M$ models by choosing different values of $\lambda : (\mathcal{H}, \lambda_1), (\mathcal{H}, \lambda_2), ..., (\mathcal{H}, \lambda_M)$;
2. Run the cross validation module to get an estimate of the cross validation error $E_{CV}(g^m)$;
3. Pick the model $(\mathcal{H}, \lambda^*)$ with the smallest error;
4. Train the model $(\mathcal{H}, \lambda^*)$ on the entire training set $D$ to obtain the final hypothesis $g^{m^*}$.

## 3.4 Bias and Variance

Before we go further, let us have a brief review:

- $E_{out}$: out-of-sample error (unknown);
- $E_{in}$: in-sample error (training error - known);
- Complex $\mathcal{H}$: better chance of approximating $f$ in-sample (low training error);
- Simple $\mathcal{H}$: better chance of generalizing out-of-sample (low generalization error);
- Understand: the goal of Machine Learning is to lowing $E_{out}$ as much as possible given the observed dataset.

Bias-variance allows us to express $E_{out}$ in a way that provides some cues to what hypothesis space to start with. It tells you:

1. How well $\mathcal{H}$ can approximate $f$;
2. How well we can zoom in on a good hypothesis $h \in \mathcal{H}$.

Note that BV analysis applies to only real-valued targets and uses squared error metric. Now let us start with some maths:

$$\begin{aligned}
\mathbb{E}_D\left[E_{out}(g^D)\right] &= \mathbb{E}_D\left[\mathbb{E}_x\left[(g^D(x) - f(x))^2\right]\right] \\
&= \mathbb{E}_x\left[\mathbb{E}_D\left[(g^D(x) - f(x))^2\right]\right] \\
&= \mathbb{E}_x\left[\mathbb{E}_D\left[(g^D(x))^2\right] - 2\mathbb{E}_D[g^D(x)]f(x) + f(x)^2\right]
\end{aligned}$$

Assume that we will have multiple training datasets: $\{D_1, D_2, ..., D_K\}$, then:

$$\mathbb{E}_D\left[(g^D(x))^2\right] \approx \frac{1}{K}\sum_{i=1}^{K} g^{D_i}(x) = \bar{g}(x)$$

Then we can rewrite our equation into the form that contains Variance and Bias:

$$\begin{aligned}
\mathbb{E}_D\left[E_{out}(g^D)\right] &= \mathbb{E}_x\left[\mathbb{E}_D\left[(g^D(x))^2\right] - 2\bar{g}(x)f(x) + f(x)^2\right] \\
&= \mathbb{E}_x\left[\mathbb{E}_D\left[(g^D(x))^2\right] - \bar{y}(g)^2 + \bar{y}(g)^2 - 2\bar{g}(x)f(x) + f(x)^2\right] \\
&= \mathbb{E}_x\left[\mathbb{E}_D\left[(g^D(x))^2 - \bar{y}(g)^2\right] + \left[\bar{y}(g) - f(x)\right]^2\right]
\end{aligned}$$

From the above equation, we now have:

$$\begin{aligned}
\textbf{var} &= \mathbb{E}_D\left[(g^D(x))^2 - \bar{y}(g)^2\right] \\
\textbf{bias} &= \left[\bar{y}(g) - f(x)\right]^2
\end{aligned}$$

In conclusion:

$$\begin{aligned}
\mathbb{E}_D\left[E_{out}(g^D)\right] &= \mathbb{E}_x\left[var(x) + bias(x)\right] \\
&= \textbf{var} + \textbf{bias}
\end{aligned}$$

- **var:** Variance measures the variation of the final hypothesis depending on the dataset $D$ used for training;
- **bias:** Bias measures by how much the average function that we will learn using different instances of the dataset will deviate from the true function. This in turn is a measure of how much our learning model is biased away from the true function.

## 4 Linear Models for Classification

Let us have a brief review on linear model. In a linear system, we assume the following relationship:

$$\begin{aligned} f(x) &\approx w_0 + \sum_{i=1}^{p} w_i x_i \\ &= \boldsymbol{w}^T \boldsymbol{x} \end{aligned}$$

Now what if $y$ is binary? Let us assume that $y \in \{-1, +1\}$. We will apply some thing called **Simple Linear Classifier**, which is also known as **Perceptron**. Then we have the **Perceptron Learning Algorithm(PLA)**:

$$h(x) = sign(\boldsymbol{w}^T \boldsymbol{x}^j)$$

Perceptron Learning Algorithm(PLA)

$h(x) = sign(\boldsymbol{w}^T \boldsymbol{x}^j)$

### 4.1 Logistic Regression

How about a loss function? Since the output is not linear, we will leave that aside for a moment and we will look at something more fun: Can we tune the output to make it between 0 and 1? You will find that this question is very realistic and meaningful if we take it as for certain probability problem. For example: How likely will it rain tomorrow? Typically the answer is "yes" or "no". But what if we can give some number between 0 and 1 as a probability of rain? We can express it into the following form:

$$f(x) = \mathbb{P}[y = 1 \mid x]$$

Thus we will need a new function for doing this. Let us first re-define the input function:

$$s = \sum_{i=0}^{N} w_i x_i = \boldsymbol{w}^T \boldsymbol{x}$$

Then we need a function with proper range (meaning that for any input, the output should be between 0 and 1):

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

And here we go, we have a new relation:

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

Then we solved the question we just raised. The function is given:

General Logistic Regression

$P(y \mid x) = h(x) = \sigma(y \cdot \boldsymbol{w}^T \boldsymbol{x})$

Using the above function, we can compute the likelihood of the training data D :

$$D = \{\{x^1, y^1\}, \{x^2, y^2\}, ..., \{x^N, y^N\}\}$$
$$P(\mathbf{Y} \mid \mathbf{X}) = \prod_{i=1}^{N} P(y^i \mid x^i) = \prod_{i=1}^{N} \sigma(y^i \cdot \boldsymbol{w}^T \boldsymbol{x}^i)$$

The process of learning reduces to adjusting the parameters to maximize the above likelihood. This is called **Maximum Likelihood Estimation**.

### 4.1.1 Loss Function for Logistic Regression

In practise, a variant of the logistic (loss) function is adopted to be minimized. We took the $-\frac{1}{N}\log(.)$ of the likelihood and minimize it. The two are equivalent since the function is monotonically decreasing. The reasons are grounded in computational stability. Thus we have:

$$E_{in}(w) = -\frac{1}{N}\ln\left(\prod_{i=1}^{N} P(y^i \mid x^i)\right) = \frac{1}{N}\sum_{i=1}^{N}\ln\left(\frac{1}{P(y^i \mid x^i)}\right) = \frac{1}{N}\sum_{i=1}^{N}\ln\left(\frac{1}{\sigma\left(y^i \boldsymbol{w}^T \boldsymbol{x}^i\right)}\right)$$

Thus we have:

Cross-Entropy Loss

$E_{in}(w) = \frac{1}{N}\sum_{i=1}^{N}\ln\left(1 + e^{-y^i \boldsymbol{w}^T \boldsymbol{x}^i}\right)$

This error measure is small when $y^i \boldsymbol{w}^T \boldsymbol{x}^i$ is large and positive. Thus minimizing the error measure pushes $\boldsymbol{w}$ to classify each $\boldsymbol{x}^i$ correctly.

Transformation of Cross-Entropy Loss

$E_{in}(w) = -\sum_{i=1}^{N}\left[y^i \cdot \log\sigma\left(\boldsymbol{w}^T\boldsymbol{x}^i\right) + (1-y^i)\cdot\log(1-\sigma\left(\boldsymbol{w}^T\boldsymbol{x}^i\right))\right]$

### 4.1.2 Gradient Descent

As we already have the loss function for the Logistic Regression, we aim to find its minimum to completed the regression. Now normally, for a linear function, we only need to find out the point where the slope is zero ($df = 0$). If we expand that to a multi-variable function, we should utilize the concept of gradient. However, it is note worthy that things in really is not that simple! For a model that have a lot of entries, it is hard or sometimes impossible to find the minimum. Thus we here introduce a vastly used algorithm: **Gradient Descent Algorithm**.

**Gradient Descent** The goal of gradient descent is to minimize the cost function $J(\theta)$. It does so by iteratively adjusting the parameters $\theta$ in the direction of steepest decrease of $J$.The update rule for gradient descent is:

$$\theta := \theta - \alpha\nabla J(\theta)$$

Where:

- $\alpha$ is the learning rate, a hyperparameter that determines the step size at each iteration.
- $\nabla J(\theta)$ is the gradient of the cost function with respect to $\theta$.

For logistic regression, the gradient is given by:

$$\nabla J(\theta) = \frac{1}{m}\sum_{i=1}^{m}(h_\theta(\mathbf{x}^{(i)}) - y^{(i)})\mathbf{x}^{(i)}$$

By repeatedly applying the update rule, gradient descent converges to the optimal values of $\theta$ that minimize the cost function. In summary, gradient descent in logistic regression involves:

1. Initializing the parameters $\theta$ (often to zero or small random values).
2. Computing the gradient of the cost function with respect to $\theta$.
3. Updating the parameters in the direction of the negative gradient.
4. Repeating steps 2 and 3 until convergence or for a fixed number of iterations.

It's worth noting that there are various forms of gradient descent, such as stochastic gradient descent (SGD) and mini-batch gradient descent, which differ in how they use the training data to compute the gradient.

### 4.1.3 Other Gradient Descent Methods

**Stochastic Gradient Descent (SGD)**

Stochastic Gradient Descent, often abbreviated as SGD, is a variant of gradient descent where the gradient of the cost function is estimated based on a single randomly chosen training example at each iteration. This introduces a lot of variance in the updates, which can help escape local minima but can also lead to a less stable convergence. The update rule for SGD is:

$$\theta := \theta - \alpha \nabla J_i(\theta)$$

Where $\nabla J_i(\theta)$ is the gradient of the cost function with respect to $\theta$ for the $i^{th}$ training example.

**Mini-batch Gradient Descent**

Mini-batch Gradient Descent is a compromise between full gradient descent and SGD. Instead of using the entire training set or a single training example, a mini-batch of $b$ training examples is used to compute the gradient at each step. This can lead to a more stable convergence than SGD and can also take advantage of optimization techniques and parallel processing. The update rule for mini-batch gradient descent is:

$$\theta := \theta - \alpha \nabla J_{\text{batch}}(\theta)$$

Where $\nabla J_{\text{batch}}(\theta)$ is the gradient of the cost function with respect to $\theta$ for the selected mini-batch of training examples. Both SGD and Mini-batch Gradient Descent introduce randomness in the optimization process, which can help escape local minima and saddle points, but they may require a well-tuned learning rate and other hyperparameters for effective convergence.

## 5 Confusion Matrix (07)

# 6 Bayesian Theory

## 6.1 Bayesian Decision Theory

Bayesian decision theory is a branch of decision theory that combines probability theory and utility theory to form decision rules. Here are the basic concepts and mathematical formulas of Bayesian decision theory:

### 6.1.1 Basic Components:

- **Prior Probability** $P(\theta)$: The probability of a parameter or hypothesis $\theta$ before observing any data.
- **Likelihood** $P(x|\theta)$: The probability of observing data $x$ given a parameter or hypothesis $\theta$.
- **Posterior Probability** $P(\theta|x)$: The probability of a parameter or hypothesis $\theta$ after observing data $x$. It can be calculated using Bayes' theorem:

$$P(\theta|x) = \frac{P(x|\theta) \times P(\theta)}{P(x)}$$

- **Loss Function** $L(\theta, a)$: This is the loss or cost of taking action $a$ when the true parameter or hypothesis is $\theta$.

### 6.1.2 Bayesian Decision Rule:

The goal of the Bayesian decision rule is to minimize the expected loss. The expected loss is the weighted average of the losses for all possible values of $\theta$ given action $a$. The mathematical formula for expected loss is:

$$E[L(a)] = \sum_{\theta} L(\theta, a) \times P(\theta|x)$$

The Bayesian decision rule chooses the action $a^*$ that minimizes the expected loss:

$$a^* = \arg\min_a E[L(a)]$$

### 6.1.3 Example 1: Medical Diagnosis

Suppose a doctor is diagnosing a patient. There are two possible diseases, $\theta_1$ and $\theta_2$. The doctor can choose treatment $a_1$ or $a_2$. Each treatment has a loss for each disease, for example, the patient might suffer if the wrong treatment is chosen. The doctor can use the Bayesian decision rule to choose the best treatment. First, the doctor calculates the posterior probability of each disease based on the symptoms. Then, the doctor calculates the expected loss for each treatment and chooses the treatment with the smallest expected loss.

### 6.1.4 Example 2: Apples and Oranges (Image Classifier)

1. **Settings:**
    - Let $\omega$ denote the state of nature;
    - $\omega_1$: apples in the wild;
    - $\omega_2$: oranges in the wild;
    - $P(\omega_1)$ and $P(\omega_2)$ are the **prior probabilities** of apples and oranges;
    - $\boldsymbol{x}$: features that we measured. Note that it is a vector contains features;
    - $P(\boldsymbol{x} \mid \omega_1)$ and $P(\boldsymbol{x} \mid \omega_2)$ are the **class conditional probability density** functions

2. Idea: With all we have above, suppose we are given a new image, and we observe $\boldsymbol{x}$. What can we say about the image? We know the joint probability can be written as:

$$P(\omega_j, \boldsymbol{x}) = P(\boldsymbol{x} \mid \omega_j)P(\omega_j) = P(\omega_j \mid \boldsymbol{x})P(\boldsymbol{x})$$

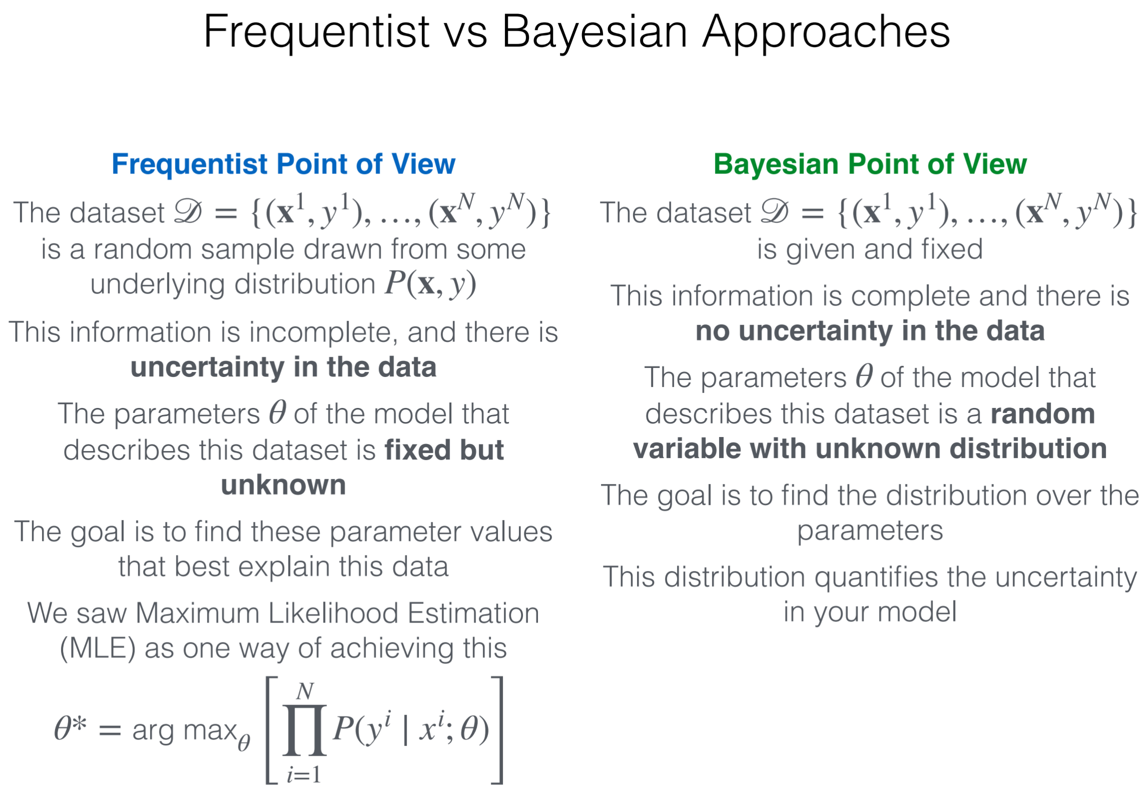
And according to Bayes Rule:

$$P(\omega_j \mid \boldsymbol{x}) = \frac{P(\boldsymbol{x} \mid \omega_j)P(\omega_j)}{P(\boldsymbol{x})}$$

In our case we have two classes:

$$P(\boldsymbol{x}) = \sum_{i=1}^{2} P(\boldsymbol{x} \mid \omega_i)P(\omega_i)$$

### 6.1.5 Comparison



## Frequentist vs Bayesian Approaches

**Frequentist Point of View**

The dataset $\mathscr{D} = \{(\mathbf{x}^1, y^1), \ldots, (\mathbf{x}^N, y^N)\}$ is a random sample drawn from some underlying distribution $P(\mathbf{x}, y)$

This information is incomplete, and there is **uncertainty in the data**

The parameters $\theta$ of the model that describes this dataset is **fixed but unknown**

The goal is to find these parameter values that best explain this data

We saw Maximum Likelihood Estimation (MLE) as one way of achieving this

$$\theta* = \text{arg max}_\theta \left[ \prod_{i=1}^{N} P(y^i \mid x^i; \theta) \right]$$

**Bayesian Point of View**

The dataset $\mathscr{D} = \{(\mathbf{x}^1, y^1), \ldots, (\mathbf{x}^N, y^N)\}$ is given and fixed

This information is complete and there is **no uncertainty in the data**

The parameters $\theta$ of the model that describes this dataset is a **random variable with unknown distribution**

The goal is to find the distribution over the parameters

This distribution quantifies the uncertainty in your model

### 6.1.6 Summary

In summary, Bayesian decision theory provides a method to make decisions by combining probability and utility. It can be applied in various fields, from medicine to finance to machine learning.

## 6.2 Bayesian Parameter Estimation

Let us start with some prior distribution $P(\theta)$, which encodes our knowledge about the parameters before looking at the data. And we are given a dataset $\mathcal{D} = \{(\boldsymbol{x}^1, y^1), (\boldsymbol{x}^2, y^2), ..., (\boldsymbol{x}^N, y^N)\}$. Our goal is to find out the posterior distribution of $\theta$ given the dataset $\mathcal{D}$: $P(\theta \mid \mathcal{D})$. We will use the Bayes Theorem.

$$\begin{aligned} P(\theta \mid \mathcal{D}) &= \frac{P(\mathcal{D} \mid \theta)P(\theta)}{P(\mathcal{D})} \\ &= \frac{P(\mathcal{D} \mid \theta)P(\theta)}{\int_\theta P(\mathcal{D} \mid \theta)P(\theta)d\theta} \\ &= \frac{\left[\prod_{i=1}^N P(y^i \mid \boldsymbol{x}^i, \theta)\right] P(\theta)}{\int_\theta \left[\prod_{i=1}^N P(y^i \mid \boldsymbol{x}^i, \theta)\right] P(\theta)d\theta} \end{aligned}$$

Here is an example: Suppose we are going to estimate the parameter of a biased coin. We flipped 100 time and we get heads for 55 time and tail for 45 times. What would be the probability that it will be heap for the next flip? Flipping the coin can be seen as independent Bernoulli random variables with parameter $\theta$:

$$p(\theta \mid \mathcal{D}) \propto p(\mathcal{D} \mid \theta)p(\theta)$$

Then under the Bernoulli model with i.i.d observations, we have the following:

$$\begin{aligned} \text{Likelihood: } p(\mathcal{D} \mid \theta) &= \theta^{N_H}(1-\theta)^{N_T} \\ \text{Choose a prior: } p(\theta; a, b) &= \frac{\Gamma(a+b)}{\Gamma(a)\Gamma(b)}\theta^{a-1}(1-\theta)^{b-1} \\ &\propto \theta^{a-1}(1-\theta)^{b-1} \\ p(\theta \mid \mathcal{D}) &\propto p(\mathcal{D} \mid \theta)p(\theta) \\ &\propto \left[\theta^{N_H}(1-\theta)^{N_T}\right]\left[\theta^{a-1}(1-\theta)^{b-1}\right] \\ &= \theta^{N_H+a-1}(1-\theta)^{N_T+b-1} \end{aligned}$$

After calculation, we can also find out the expectation:

Given the posterior distribution parameters $a = 56$ and $b = 46$, the expected value of $\theta$ is given by:

$$\mathbb{E}[\theta] = \frac{a}{a+b}$$

Substituting in the values:

$$\mathbb{E}[\theta] = \frac{56}{56+46} = \frac{56}{102} = \frac{28}{51}$$

Thus, the best estimate for the probability of getting heads on the next flip, based on the observed data and the chosen prior, is $\frac{28}{51}$ or approximately 0.549.

### 6.2.1 Gamma Function and Beta Distribution*

**1. Gamma Function ($\Gamma$):**

The Gamma function is a mathematical function that generalizes the factorial function to non-integer values. It's defined as:

$$\Gamma(z) = \int_0^\infty t^{z-1}e^{-t}\,dt$$

for complex numbers $z$ with a positive real part. Some key properties and facts about the Gamma function:

- For positive integers $n$:

  $$\Gamma(n) = (n-1)!$$

  This shows how the Gamma function generalizes the factorial function.

- The Gamma function is defined for all complex numbers except non-positive integers (where it has poles).

**2. Beta Distribution:**
The Beta distribution is a continuous probability distribution defined on the interval [0, 1]. It's commonly used in Bayesian statistics to model the distribution of random variables that represent probabilities. The probability density function (pdf) of the Beta distribution is:

$$f(x; a, b) = \frac{x^{a-1}(1-x)^{b-1}}{B(a, b)}$$

where $0 \leq x \leq 1$ and $a, b > 0$. The function $B(a, b)$ is the Beta function, which serves as a normalization constant to ensure that the total probability integrates to 1. The **Beta function** is defined in terms of the Gamma function:

$$B(a, b) = \frac{\Gamma(a)\Gamma(b)}{\Gamma(a+b)}$$

Some properties of the Beta distribution:

- The **mean** of the Beta distribution is:
$$\mu = \frac{a}{a+b}$$
- The **variance** is:
$$\sigma^2 = \frac{ab}{(a+b)^2(a+b+1)}$$
- The shape of the Beta distribution is determined by the parameters $a$ and $b$:
  - If $a = b = 1$, it's the uniform distribution on [0,1].
  - If $a > b$, the distribution is skewed towards 1.
  - If $a < b$, the distribution is skewed towards 0.
  - If $a = b > 1$, the distribution is symmetric and bell-shaped.

The Beta distribution is particularly useful in Bayesian statistics because it's the conjugate prior for the Bernoulli, binomial, and negative binomial likelihoods. This means that if you start with a Beta prior and observe data that follows one of these likelihoods, the posterior distribution will also be a Beta distribution.

## 6.3 Prediction with Bayesian Models

Given the training set $\mathcal{D}$ and a new observation set $\mathcal{D}'$, posterior predictive distribution is the distribution over future observables given the past observations. We can compute it by marginalizing out the parameters. Here are the steps:

$$\begin{aligned}
p(\mathcal{D}' \mid \mathcal{D}) &= \int p(\theta \mid \mathcal{D}) p(\mathcal{D}' \mid \theta) d\theta \\
\theta_{pred} &= Pr(x' = H \mid \mathcal{D}) \\
&= \int p(\theta \mid \mathcal{D}) Pr(x' = H \mid \mathcal{D}) d\theta \\
&= \int Beta(\theta; N_H + a, N_T + b) \cdot \theta d\theta \\
&= \mathbb{E}_{Beta}(\theta; N_H + a, N_T + b) \\
&= \frac{N_H + a}{N_H + N_T + a + b}
\end{aligned}$$

Let's break down the above steps.

1. **Marginalizing Out the Parameters**

   $$p(\mathcal{D}' \mid \mathcal{D}) = \int p(\theta \mid \mathcal{D}) p(\mathcal{D}' \mid \theta) d\theta$$

   This equation represents the probability of observing the new data $\mathcal{D}'$ given the training data $\mathcal{D}$. We compute this by integrating (or averaging) over all possible values of the parameter $\theta$. The term $p(\theta \mid \mathcal{D})$ is the posterior distribution of $\theta$ given the training data, and $p(\mathcal{D}' \mid \theta)$ is the likelihood of the new data given $\theta$.

2. **Predictive Probability for a Single New Observation**

   $$\theta_{pred} = Pr(x' = H \mid \mathcal{D})$$

   This represents the probability that a single new observation $x'$ is a head (H) given the training data $\mathcal{D}$.

3. **Computing the Predictive Probability**

   $$\theta_{pred} = \int p(\theta \mid \mathcal{D}) Pr(x' = H \mid \theta) d\theta$$

   Here, we're averaging the probability of observing a head over all possible values of $\theta$, weighted by the posterior distribution of $\theta$.

4. **Substituting the Posterior Distribution**

   $$\theta_{pred} = \int Beta(\theta; N_H + a, N_T + b) \cdot \theta d\theta$$

   Given that the posterior distribution of $\theta$ is a Beta distribution with parameters $N_H + a$ and $N_T + b$ (where $N_H$ is the number of observed heads and $N_T$ is the number of observed tails), we multiply this by $\theta$ (the probability of observing a head) and integrate.

5. **Expectation of the Beta Distribution**

   $$\theta_{pred} = \mathbb{E}_{Beta}(\theta; N_H + a, N_T + b)$$

   The integral in the previous step is essentially computing the expected value (or mean) of $\theta$ under the Beta distribution.

6. **Computing the Expectation**

   $$\theta_{pred} = \frac{N_H + a}{N_H + N_T + a + b}$$

   The expected value of the Beta distribution is given by $\frac{a}{a+b}$. Substituting in the values for $a$ and $b$ from the posterior distribution, we get the above equation.

In summary, the provided equations describe how to compute the probability of observing a particular outcome (like a head in a coin flip) for new data based on previously observed data, by averaging over all possible values of the underlying parameter $\theta$ in a Bayesian framework. The result is a predictive distribution that takes into account both the observed data and our prior beliefs about $\theta$.

Now let's generalize what we just got. Suppose we are given the training set $\mathcal{D}$and a new observation $\boldsymbol{x}$, what is the value of y? What we are looking for is the distribution $P(y \mid \boldsymbol{x}, \mathcal{D})$.
1. For logistic regression:

$$P(y \mid \boldsymbol{x}, \theta) = h_\theta(\boldsymbol{x}) = \frac{1}{1 + e^{-\theta^T \boldsymbol{x}}}$$

2. We also have:

$$P(\theta \mid \mathcal{D}) = \frac{\left[\prod_{i=1}^{N} P(y^i \mid \boldsymbol{x}^i, \theta)\right] P(\theta)}{\int_\theta \left[\prod_{i=1}^{N} P(y^i \mid \boldsymbol{x}^i, \theta)\right] P(\theta) d\theta}$$

3. We combine them together:

$$P(y \mid \boldsymbol{x}, \mathcal{D}) = \int_{\theta} P(y \mid \boldsymbol{x}, \theta) P(\theta \mid \mathcal{D}) d\theta = \int_{\theta} h_{\theta}(\boldsymbol{x}) P(\theta \mid \mathcal{D}) d\theta$$

But it is a hard-to-compute integral.

## 6.4 Bayesian Model in Practise

In practice we normally have two options:

- **Option 1:** Do away with the full integral to integrate across the entire parameter distribution. Instead we follow the steps:
    1. Compute the point estimate of the parameters:
    $$\theta_{\text{MAP}} = \arg\max_{\theta} \prod_{i=1}^{N} P(y^i \mid \boldsymbol{x}^i, \theta) \cdot P(\theta)$$
    2. Use the estimate to compute $y$ for a new $\boldsymbol{x}$ (Inference)
    $$\hat{y} = \arg\max_{y} p(y \mid \boldsymbol{x}; \theta = \theta_{\text{MAP}})$$
    3. This is called **Maximum A Posterior (MAP)** estimate. It is very similar to the Maximum Likelihood Estimate (MLE) but with an additional prior term.
- **Option 2:** Choose the distribution function $P(y \mid \boldsymbol{x}, \theta)$ such that the integral can be computed analytically.

## 6.5 Bayesian Linear Regression

Bayesian linear regression combines the principles of linear regression with Bayesian inference to estimate the parameters of a linear model.

### 6.5.1 Three Key Steps in Bayesian Methods

1. **Prior Distribution**:
   Represents our initial beliefs about the parameters before observing any data.
2. **Likelihood**:
   Describes the probability of observing the data given the model parameters.
3. **Posterior Distribution**:
   Combines the prior and the likelihood to give an updated belief about the parameters after observing the data.

### 6.5.2 Prior Distribution

The prior distribution, $p(w)$, describes our beliefs about the weights $w$ before observing any data. A common choice is a Gaussian prior for $w$:

$$w \sim \mathcal{N}(m_0, S_0)$$

Where:

- $m_0$ is the prior mean, often set to a zero vector.
- $S_0$ is the prior covariance matrix, describing the uncertainty in the weights.

### 6.5.3 Likelihood

The likelihood function describes the probability of observing the data given the model parameters. In linear regression, it's commonly assumed that the errors are independent and identically distributed with Gaussian noise:

$$y_i \sim \mathcal{N}(w^T x_i, \sigma^2)$$

For the entire dataset $\mathcal{D}$:

$$p(\mathcal{D}|w) = \prod_{i=1}^{N} \mathcal{N}(y_i; w^T x_i, \sigma^2)$$

### 6.5.4 Posterior Distribution

The posterior distribution combines the prior and the likelihood to give an updated belief about the parameters:

$$p(w|\mathcal{D}) = \frac{p(\mathcal{D}|w)p(w)}{p(\mathcal{D})}$$

Where:

- $p(w|\mathcal{D})$ is the posterior distribution.
- $p(\mathcal{D}|w)$ is the likelihood.
- $p(w)$ is the prior distribution.
- $p(\mathcal{D})$ is the marginal probability of the data, also known as the normalization constant.

When using a Gaussian prior and Gaussian likelihood, the posterior distribution is also Gaussian, simplifying computation and inference.

### 6.5.5 How to Choose the Prior

Prior should reflect your knowledge about the distribution before looking at the data.

- **Objective Priors:** should maximize the impact of data on posterior. Non-Informative Priors are a type of objective priors (e.g., uniform distribution)
- **Subjective Priors:** that capture some domain/external knowledge about the problem (e.g., Gaussian priors)
- **Conjugate Priors:** a prior is considered conjugate with respect to the likelihood if the resulting posterior distribution is of the same form as the prior distribution — used primarily for simplifying the calculation of the integral

### 6.5.6 Maths for Bayesian Linear Regression

TO BE COMPLETED

# 7 Probabilistic Models

## 7.1 Linear Regression Model

## 8 Support Vector Machine

## 9 Neural Networks

## 10 Deep Learning

## 11 CONTINUED