Parameter Estimation and Probabilistic Estimators

Machine Learning

Overview

Overview

1. Discriminative and generative models.

Overview

1. Discriminative and generative models.
2. Parametric distributions.

Overview

1. Discriminative and generative models.
2. Parametric distributions.
3. Parametric hypothesis classes.

Overview

1. Discriminative and generative models.
2. Parametric distributions.
3. Parametric hypothesis classes.
4. Bayes rule.

Overview

1. Discriminative and generative models.
2. Parametric distributions.
3. Parametric hypothesis classes.
4. Bayes rule.
5. Bayesian parameter estimation.

Overview

1. Discriminative and generative models.
2. Parametric distributions.
3. Parametric hypothesis classes.
4. Bayes rule.
5. Bayesian parameter estimation.
6. Naive Bayes.

Discriminative and Generative Models

Discriminative and Generative Models

Two main types of models in supervised learning:

Discriminative and Generative Models

Two main types of models in supervised learning:

Definition (Discriminative Model)

A **discriminative model** is one that learns and models $p_{Y|X}(y \mid x)$. In other words, it learns the *conditional* distribution and is useful for classification/regression.

Two main types of models in supervised learning:

Definition (Discriminative Model)

A **discriminative model** is one that learns and models $p_{Y|X}(y \mid x)$. In other words, it learns the *conditional* distribution and is useful for classification/regression.

Definition (Generative Model)

A **generative model** is one that learns and models $p_{X,Y}(x, y)$. In other words, it learns the *joint* distribution and, on top of classification/regression, can be used for other tasks such as generating plausible data.

Discriminative and Generative Models

Two main types of models in supervised learning:

Definition (Discriminative Model)

A **discriminative model** is one that learns and models $p_{Y|X}(y \mid x)$. In other words, it learns the *conditional* distribution and is useful for classification/regression.

Definition (Generative Model)

A **generative model** is one that learns and models $p_{X,Y}(x, y)$. In other words, it learns the *joint* distribution and, on top of classification/regression, can be used for other tasks such as generating plausible data.

Generative models are more versatile; discriminative models are more effective (Ng, Jordan).

Parametric Distributions

Parametric Distributions

Sometimes we want to model our data as from a distribution μ_λ in a family of distributions

Parametric Distributions

Sometimes we want to model our data as from a distribution μ_λ in a family of distributions

$$\mathcal{M} = \{\mu_\lambda : \lambda \in \Lambda\}.$$

Parametric Distributions

Sometimes we want to model our data as from a distribution μ_λ in a family of distributions

$$\mathcal{M} = \{\mu_\lambda \colon \lambda \in \Lambda\}\,.$$

We could know this by prior knowledge,

Parametric Distributions

Sometimes we want to model our data as from a distribution μ_λ in a family of distributions

$$\mathcal{M} = \{\mu_\lambda : \lambda \in \Lambda\}.$$

We could know this by prior knowledge, *or* we could just be guessing and hoping the true distribution is close enough to one of these distributions.

Parametric Distributions

Sometimes we want to model our data as from a distribution μ_λ in a family of distributions

$$\mathcal{M} = \{\mu_\lambda : \lambda \in \Lambda\}.$$

We could know this by prior knowledge, *or* we could just be guessing and hoping the true distribution is close enough to one of these distributions. Here λ is the **parameter**.

Parametric Distributions

Sometimes we want to model our data as from a distribution μ_λ in a family of distributions

$$\mathcal{M} = \{\mu_\lambda : \lambda \in \Lambda\} .$$

We could know this by prior knowledge, *or* we could just be guessing and hoping the true distribution is close enough to one of these distributions. Here λ is the **parameter**.
Correspondingly, for unsupervised problems we can consider $\mu_{\lambda;X}$ coming from the family

Parametric Distributions

Sometimes we want to model our data as from a distribution μ_λ in a family of distributions

$$\mathcal{M} = \{\mu_\lambda : \lambda \in \Lambda\}.$$

We could know this by prior knowledge, *or* we could just be guessing and hoping the true distribution is close enough to one of these distributions. Here λ is the **parameter**.
Correspondingly, for unsupervised problems we can consider $\mu_{\lambda;X}$ coming from the family

$$\mathcal{M}_X = \left\{\mu_{\lambda;X} : \lambda \in \Lambda\right\}.$$

Parametric Distributions

Sometimes we want to model our data as from a distribution μ_λ in a family of distributions

$$\mathcal{M} = \{\mu_\lambda : \lambda \in \Lambda\}.$$

We could know this by prior knowledge, *or* we could just be guessing and hoping the true distribution is close enough to one of these distributions. Here λ is the **parameter**.

Correspondingly, for unsupervised problems we can consider $\mu_{\lambda;X}$ coming from the family

$$\mathcal{M}_X = \left\{\mu_{\lambda;X} : \lambda \in \Lambda\right\}.$$

Example (Normal Family)

If we suppose our data is normally distributed, we can let $\lambda = (\theta, \Sigma)$, so

$$\mathcal{M}_X = \left\{E \mapsto \mathsf{P}[\mathcal{N}(\theta, \Sigma) \in E] : \theta \in \mathbb{R}^d, \Sigma \in \mathbb{R}^{d \times d}\right\}.$$

Parametric Hypotheses

Parametric Hypotheses

Sometimes our algorithm gives a hypothesis class of the form

Parametric Hypotheses

Sometimes our algorithm gives a hypothesis class of the form

$$\mathscr{H} = \{h_\theta \colon \theta \in \Theta\}\,.$$

Parametric Hypotheses

Sometimes our algorithm gives a hypothesis class of the form

$$\mathscr{H} = \{h_\theta : \theta \in \Theta\}.$$

This is entirely dependent on the algorithm we choose.

Parametric Hypotheses

Sometimes our algorithm gives a hypothesis class of the form

$$\mathcal{H} = \{h_\theta : \theta \in \Theta\}.$$

This is entirely dependent on the algorithm we choose.

Example (Support Vector Classification)

The popular support vector machine algorithm has hypotheses of the form

$$\mathcal{H} = \left\{x \mapsto \operatorname{sign}(\langle \theta, x \rangle) : \theta \in \mathbb{R}^d\right\}.$$

Bayes' Rule

Bayes' Rule

Theorem (Bayes' Rule)

Bayes' Rule

Theorem (Bayes' Rule)

If A and B are events, then

$$\mathsf{P}[B \mid A] = \frac{\mathsf{P}[A \mid B]\,\mathsf{P}[B]}{\mathsf{P}[A]}.$$

Bayes' Rule

Theorem (Bayes' Rule)

If A and B are events, then

$$\mathsf{P}[B \mid A] = \frac{\mathsf{P}[A \mid B]\,\mathsf{P}[B]}{\mathsf{P}[A]}.$$

If x and y are random variables with density p_X and p_Y, then

$$p_{X|Y}(u \mid v) = \frac{p_{Y|X}(v \mid u)\,p_X(u)}{p_Y(v)}.$$

Bayes' Rule

Theorem (Bayes' Rule)

If A and B are events, then

$$\mathsf{P}[B \mid A] = \frac{\mathsf{P}[A \mid B]\,\mathsf{P}[B]}{\mathsf{P}[A]}.$$

If x and y are random variables with density p_X and p_Y, then

$$p_{X|Y}(u \mid v) = \frac{p_{Y|X}(v \mid u)\,p_X(u)}{p_Y(v)}.$$

Proof is not very complicated:

Theorem (Bayes' Rule)

If A and B are events, then

$$\mathsf{P}[B \mid A] = \frac{\mathsf{P}[A \mid B]\,\mathsf{P}[B]}{\mathsf{P}[A]}.$$

If x and y are random variables with density p_X and p_Y, then

$$p_{X|Y}(u \mid v) = \frac{p_{Y|X}(v \mid u)\, p_X(u)}{p_Y(v)}.$$

Proof is not very complicated:

$$\mathsf{P}[B \mid A]\,\mathsf{P}[A] = \mathsf{P}[A \cap B] = \mathsf{P}[A \mid B]\,\mathsf{P}[B]\,.$$

Bayesian Parameter Estimation

Let's say we have a distribution class

Bayesian Parameter Estimation

Let's say we have a distribution class

$$\mathcal{M} = \{\mu_\lambda : \lambda \in \Lambda\} .$$

Bayesian Parameter Estimation

Let's say we have a distribution class

$$\mathcal{M} = \{\mu_\lambda : \lambda \in \Lambda\}\,.$$

Data distributed according to $\mu_{\lambda_\star}$

Bayesian Parameter Estimation

Let's say we have a distribution class

$$\mathcal{M} = \{\mu_\lambda : \lambda \in \Lambda\} .$$

Data distributed according to $\mu_{\lambda_\star}$; want to estimate $\lambda_\star$ using the **estimator** $\widehat{\lambda} = \widehat{\lambda}(s)$.

MLE

MLE

Simplest, most intuitive parameter estimation algorithm.

MLE

Simplest, most intuitive parameter estimation algorithm.

MLE Algorithm

- **Input:** a distribution class $\mathcal{M}$, a sample s.
- **Output:** an estimate $\widehat{\lambda}_{\text{MLE}}(s)$ for $\lambda_\star$.
- **Algorithm:** *Take the λ making your data most likely.*

$$\widehat{\lambda}_{\text{MLE}}(s) \in \underset{\lambda \in \Lambda}{\operatorname{argmax}}\, p_S(s \mid \lambda).$$

MLE

Logarithm is monotonic, so we have some equivalent forms:

MLE

Logarithm is monotonic, so we have some equivalent forms:

$$\widehat{\lambda}_{\text{MLE}}(s)$$

MLE

Logarithm is monotonic, so we have some equivalent forms:

$$\widehat{\lambda}_{\mathrm{MLE}}(s) \in \underset{\lambda \in \Lambda}{\operatorname{argmax}}\, p_S(s \mid \lambda)$$

MLE

Logarithm is monotonic, so we have some equivalent forms:

$$\begin{aligned}\widehat{\lambda}_{\mathrm{MLE}}(s) &\in \operatorname*{argmax}_{\lambda\in\Lambda} p_S(s \mid \lambda) \\ &= \operatorname*{argmax}_{\lambda\in\Lambda} \prod_{i=1}^{n} p_Z(z_i \mid \lambda)\end{aligned}$$

MLE

Logarithm is monotonic, so we have some equivalent forms:

$$\begin{aligned}
\widehat{\lambda}_{\mathrm{MLE}}(s) &\in \operatorname*{argmax}_{\lambda \in \Lambda} p_S(s \mid \lambda) \\
&= \operatorname*{argmax}_{\lambda \in \Lambda} \prod_{i=1}^{n} p_Z(z_i \mid \lambda) \\
&= \operatorname*{argmax}_{\lambda \in \Lambda} \sum_{i=1}^{n} \log(p_Z(z_i \mid \lambda)) \, .
\end{aligned}$$

MLE

Logarithm is monotonic, so we have some equivalent forms:

$$\begin{aligned}\widehat{\lambda}_{\mathrm{MLE}}(s) &\in \operatorname*{argmax}_{\lambda \in \Lambda} p_S(s \mid \lambda) \\ &= \operatorname*{argmax}_{\lambda \in \Lambda} \prod_{i=1}^{n} p_Z(z_i \mid \lambda) \\ &= \operatorname*{argmax}_{\lambda \in \Lambda} \sum_{i=1}^{n} \log(p_Z(z_i \mid \lambda)) \,.\end{aligned}$$

In supervised learning, we should factor $p_Z = p_{X,Y}$ *into* $p_{X|Y}\, p_Y$ *or* $p_{Y|X}\, p_X$*, whichever is easier or more efficient to compute.*

MLE

Logarithm is monotonic, so we have some equivalent forms:

$$
\begin{aligned}
\widehat{\lambda}_{\text{MLE}}(s) \in \operatorname*{argmax}_{\lambda \in \Lambda} p_S(s \mid \lambda) \\
= \operatorname*{argmax}_{\lambda \in \Lambda} \prod_{i=1}^{n} p_Z(z_i \mid \lambda) \\
= \operatorname*{argmax}_{\lambda \in \Lambda} \sum_{i=1}^{n} \log(p_Z(z_i \mid \lambda)) \,.
\end{aligned}
$$

In supervised learning, we should factor $p_Z = p_{X,Y}$ into $p_{X|Y}\, p_Y$ or $p_{Y|X}\, p_X$, whichever is easier or more efficient to compute.
In unsupervised learning, we can just treat $p_Z = p_X$ and compute the density directly.

MAP

MAP

Above solution for MLE was straightforward and simple, yet it works pretty well!

MAP

Above solution for MLE was straightforward and simple, yet it works pretty well!
One thing it's missing is that it doesn't let us add prior information about what we *think* $\lambda_\star$ is.

MAP

Above solution for MLE was straightforward and simple, yet it works pretty well!
One thing it's missing is that it doesn't let us add prior information about what we *think* $\lambda_\star$ is.
Solution: treat λ like a *random variable*, and give it a distribution μ_Λ and density p_Λ which reflects prior information.

MAP

MAP

More advanced parameter estimation algorithm.

MAP

More advanced parameter estimation algorithm.

MAP Algorithm

- **Input:** a distribution class $\mathcal{M}$, a sample s, a prior p_Λ on the value of $\lambda_\star$.
- **Output:** an estimate $\widehat{\lambda}_{\mathrm{MAP}}(s)$ for $\lambda_\star$.
- **Algorithm:** *Take the λ which is most likely, given the data and your prior.*

$$\widehat{\lambda}_{\mathrm{MAP}}(s) \in \operatorname*{argmax}_{\lambda \in \Lambda} p_{\Lambda|S}(\lambda \mid s).$$

MAP

MAP

We can use monotonicity of the logarithm again:

MAP

We can use monotonicity of the logarithm again:

$$\widehat{\lambda}_{\mathrm{MAP}}(s)$$

MAP

We can use monotonicity of the logarithm again:

$$\widehat{\lambda}_{\mathrm{MAP}}(s) = \operatorname*{argmax}_{\lambda \in \Lambda} p_{\Lambda|S}(\lambda \mid s)$$

MAP

We can use monotonicity of the logarithm again:

$$\begin{aligned}\widehat{\lambda}_{\mathrm{MAP}}(s) &= \operatorname*{argmax}_{\lambda \in \Lambda} p_{\Lambda|S}(\lambda \mid s) \\ &= \operatorname*{argmax}_{\lambda \in \Lambda} \frac{p_{S|\Lambda}(s \mid \lambda)\, p_{\Lambda}(\lambda)}{p_S(s)}\end{aligned}$$

MAP

We can use monotonicity of the logarithm again:

$$\begin{aligned}
\widehat{\lambda}_{\mathrm{MAP}}(s) &= \operatorname*{argmax}_{\lambda \in \Lambda} p_{\Lambda|S}(\lambda \mid s) \\
&= \operatorname*{argmax}_{\lambda \in \Lambda} \frac{p_{S|\Lambda}(s \mid \lambda)\, p_{\Lambda}(\lambda)}{p_S(s)} \\
&= \operatorname*{argmax}_{\lambda \in \Lambda} p_{S|\Lambda}(s \mid \lambda)\, p_{\Lambda}(\lambda)
\end{aligned}$$

MAP

We can use monotonicity of the logarithm again:

$$\begin{aligned}
\widehat{\lambda}_{\mathrm{MAP}}(s) &= \operatorname*{argmax}_{\lambda \in \Lambda} p_{\Lambda|S}(\lambda \mid s) \\
&= \operatorname*{argmax}_{\lambda \in \Lambda} \frac{p_{S|\Lambda}(s \mid \lambda)\, p_{\Lambda}(\lambda)}{p_S(s)} \\
&= \operatorname*{argmax}_{\lambda \in \Lambda} p_{S|\Lambda}(s \mid \lambda)\, p_{\Lambda}(\lambda) \\
&= \operatorname*{argmax}_{\lambda \in \Lambda} p_{\Lambda}(\lambda) \prod_{i=1}^{n} p_{Z|\Lambda}(z_i \mid \lambda)
\end{aligned}$$

MAP

We can use monotonicity of the logarithm again:

$$\begin{aligned}
\widehat{\lambda}_{\mathrm{MAP}}(s) &= \operatorname*{argmax}_{\lambda \in \Lambda} p_{\Lambda|S}(\lambda \mid s) \\
&= \operatorname*{argmax}_{\lambda \in \Lambda} \frac{p_{S|\Lambda}(s \mid \lambda) p_{\Lambda}(\lambda)}{p_S(s)} \\
&= \operatorname*{argmax}_{\lambda \in \Lambda} p_{S|\Lambda}(s \mid \lambda) p_{\Lambda}(\lambda) \\
&= \operatorname*{argmax}_{\lambda \in \Lambda} p_{\Lambda}(\lambda) \prod_{i=1}^{n} p_{Z|\Lambda}(z_i \mid \lambda) \\
&= \operatorname*{argmax}_{\lambda \in \Lambda} \left(\log(p_{\Lambda}(\lambda)) + \sum_{i=1}^{n} \log\big(p_{Z|\Lambda}(z_i \mid \lambda)\big) \right).
\end{aligned}$$

MAP

We can use monotonicity of the logarithm again:

$$
\begin{aligned}
\widehat{\lambda}_{\text{MAP}}(s) &= \operatorname*{argmax}_{\lambda \in \Lambda} p_{\Lambda|S}(\lambda \mid s) \\
&= \operatorname*{argmax}_{\lambda \in \Lambda} \frac{p_{S|\Lambda}(s \mid \lambda)\, p_\Lambda(\lambda)}{p_S(s)} \\
&= \operatorname*{argmax}_{\lambda \in \Lambda} p_{S|\Lambda}(s \mid \lambda)\, p_\Lambda(\lambda) \\
&= \operatorname*{argmax}_{\lambda \in \Lambda} p_\Lambda(\lambda) \prod_{i=1}^{n} p_{Z|\Lambda}(z_i \mid \lambda) \\
&= \operatorname*{argmax}_{\lambda \in \Lambda} \left(\log(p_\Lambda(\lambda)) + \sum_{i=1}^{n} \log\big(p_{Z|\Lambda}(z_i \mid \lambda)\big) \right).
\end{aligned}
$$

We can do the same factorization of $p_{Z|\Lambda}$ *as MLE.*

Naive Bayes

Naive Bayes

Generative supervised learning algorithm for classification using parameter estimation.

Naive Bayes

Generative supervised learning algorithm for classification using parameter estimation.

Naive Bayes Algorithm

- **Input:** A known or learned parameter λ, a new data point $x \in \mathcal{X}$.
- **Output:** A predicted output $\widehat{y}(x)$ corresponding to x.
- **Algorithm:** *Maximize the posterior probability:*

$$\widehat{y}(x) \in \underset{y \in \mathcal{Y}}{\operatorname{argmax}}\ p_{Y \mid X}(y \mid x, \lambda).$$

Naive Bayes

Naive Bayes

Use the same tricks as MAP to compute.

Naive Bayes

Use the same tricks as MAP to compute.

$$\widehat{y}(x)$$

Naive Bayes

Use the same tricks as MAP to compute.

$$\widehat{y}(x) \in \operatorname*{argmax}_{y \in \mathcal{Y}} p_{Y|X}(y \mid x, \lambda)$$

Naive Bayes

Use the same tricks as MAP to compute.

$$\begin{aligned}\widehat{y}(x) &\in \operatorname*{argmax}_{y \in \mathcal{Y}} p_{Y|X}(y \mid x, \lambda) \\ &= \operatorname*{argmax}_{y \in \mathcal{Y}} \frac{p_{X,Y}(x, y \mid \lambda)}{p_X(x)}\end{aligned}$$

Naive Bayes

Use the same tricks as MAP to compute.

$$
\begin{aligned}
\widehat{y}(x) &\in \operatorname*{argmax}_{y \in \mathcal{Y}} p_{Y\mid X}(y \mid x, \lambda) \\
&= \operatorname*{argmax}_{y \in \mathcal{Y}} \frac{p_{X,Y}(x, y \mid \lambda)}{p_X(x)} \\
&= \operatorname*{argmax}_{y \in \mathcal{Y}} p_{X,Y}(x, y \mid \lambda)
\end{aligned}
$$

Naive Bayes

Use the same tricks as MAP to compute.

$$
\begin{aligned}
\widehat{y}(x) &\in \operatorname*{argmax}_{y \in \mathcal{Y}} p_{Y \mid X}(y \mid x, \lambda) \\
&= \operatorname*{argmax}_{y \in \mathcal{Y}} \frac{p_{X,Y}(x, y \mid \lambda)}{p_X(x)} \\
&= \operatorname*{argmax}_{y \in \mathcal{Y}} p_{X,Y}(x, y \mid \lambda) \\
&= \operatorname*{argmax}_{y \in \mathcal{Y}} p_{X \mid Y}(x \mid y, \lambda)\, p_Y(y \mid \lambda)
\end{aligned}
$$

Naive Bayes

Use the same tricks as MAP to compute.

$$
\begin{aligned}
\widehat{y}(x) &\in \operatorname*{argmax}_{y \in \mathcal{Y}} p_{Y \mid X}(y \mid x, \lambda) \\
&= \operatorname*{argmax}_{y \in \mathcal{Y}} \frac{p_{X,Y}(x, y \mid \lambda)}{p_X(x)} \\
&= \operatorname*{argmax}_{y \in \mathcal{Y}} p_{X,Y}(x, y \mid \lambda) \\
&= \operatorname*{argmax}_{y \in \mathcal{Y}} p_{X \mid Y}(x \mid y, \lambda)\, p_Y(y \mid \lambda) \\
&= \operatorname*{argmax}_{y \in \mathcal{Y}} \left(\log\left(p_{X \mid Y}(x \mid y, \lambda)\right) + \log(p_Y(y \mid \lambda))\right).
\end{aligned}
$$

Naive Bayes

Use the same tricks as MAP to compute.

$$
\begin{aligned}
\widehat{y}(x) &\in \operatorname*{argmax}_{y \in \mathcal{Y}} p_{Y|X}(y \mid x, \lambda) \\
&= \operatorname*{argmax}_{y \in \mathcal{Y}} \frac{p_{X,Y}(x, y \mid \lambda)}{p_X(x)} \\
&= \operatorname*{argmax}_{y \in \mathcal{Y}} p_{X,Y}(x, y \mid \lambda) \\
&= \operatorname*{argmax}_{y \in \mathcal{Y}} p_{X|Y}(x \mid y, \lambda) p_Y(y \mid \lambda) \\
&= \operatorname*{argmax}_{y \in \mathcal{Y}} \left(\log\left(p_{X|Y}(x \mid y, \lambda)\right) + \log(p_Y(y \mid \lambda))\right).
\end{aligned}
$$

If each class's points are drawn from a separate distribution, then λ contains the parameters for each distribution, and Y gives the distribution to evaluate.

Naive Bayes

Use the same tricks as MAP to compute.

$$\begin{aligned}
\widehat{y}(x) \in \operatorname*{argmax}_{y \in \mathcal{Y}} p_{Y|X}(y \mid x, \lambda) &= \operatorname*{argmax}_{y \in \mathcal{Y}} \frac{p_{X,Y}(x, y \mid \lambda)}{p_X(x)} \\
&= \operatorname*{argmax}_{y \in \mathcal{Y}} p_{X,Y}(x, y \mid \lambda) \\
&= \operatorname*{argmax}_{y \in \mathcal{Y}} p_{X|Y}(x \mid y, \lambda)\, p_Y(y \mid \lambda) \\
&= \operatorname*{argmax}_{y \in \mathcal{Y}} \left(\log\left(p_{X|Y}(x \mid y, \lambda)\right) + \log(p_Y(y \mid \lambda))\right).
\end{aligned}$$

If each class's points are drawn from a separate distribution, then λ contains the parameters for each distribution, and Y gives the distribution to evaluate. Thus, λ lets us evaluate $p_{X|Y}$.

Naive Bayes

Use the same tricks as MAP to compute.

$$
\begin{aligned}
\widehat{y}(x) &\in \operatorname*{argmax}_{y\in\mathcal{Y}} p_{Y|X}(y \mid x, \lambda) \\
&= \operatorname*{argmax}_{y\in\mathcal{Y}} \frac{p_{X,Y}(x, y \mid \lambda)}{p_X(x)} \\
&= \operatorname*{argmax}_{y\in\mathcal{Y}} p_{X,Y}(x, y \mid \lambda) \\
&= \operatorname*{argmax}_{y\in\mathcal{Y}} p_{X|Y}(x \mid y, \lambda) p_Y(y \mid \lambda) \\
&= \operatorname*{argmax}_{y\in\mathcal{Y}} \left(\log\left(p_{X|Y}(x \mid y, \lambda)\right) + \log(p_Y(y \mid \lambda))\right).
\end{aligned}
$$

If each class's points are drawn from a separate distribution, then λ contains the parameters for each distribution, and Y gives the distribution to evaluate. Thus, λ lets us evaluate $p_{X|Y}$. We estimate p_Y using data by just counting proportion in each class.

Naive Bayes

Use the same tricks as MAP to compute.

$$
\begin{aligned}
\widehat{y}(x) &\in \operatorname*{argmax}_{y \in \mathcal{Y}} p_{Y|X}(y \mid x, \lambda) \\
&= \operatorname*{argmax}_{y \in \mathcal{Y}} \frac{p_{X,Y}(x, y \mid \lambda)}{p_X(x)} \\
&= \operatorname*{argmax}_{y \in \mathcal{Y}} p_{X,Y}(x, y \mid \lambda) \\
&= \operatorname*{argmax}_{y \in \mathcal{Y}} p_{X|Y}(x \mid y, \lambda) p_Y(y \mid \lambda) \\
&= \operatorname*{argmax}_{y \in \mathcal{Y}} \left(\log\left(p_{X|Y}(x \mid y, \lambda)\right) + \log(p_Y(y \mid \lambda))\right).
\end{aligned}
$$

If each class's points are drawn from a separate distribution, then λ contains the parameters for each distribution, and Y gives the distribution to evaluate. Thus, λ lets us evaluate $p_{X|Y}$.
We estimate p_Y using data by just counting proportion in each class.
Quick and easy classifier!

Training and Inference

Training and Inference

Machine learning algorithms usually have a **training phase** and **inference phase**.

Training and Inference

Machine learning algorithms usually have a **training phase** and **inference phase**.

- The training phase is where the model is constructed and updated.

Training and Inference

Machine learning algorithms usually have a **training phase** and **inference phase**.

- The training phase is where the model is constructed and updated.
- The inference phase is where the model is applied to data it hasn't seen before.

Training and Inference

Machine learning algorithms usually have a **training phase** and **inference phase**.

- The training phase is where the model is constructed and updated.
- The inference phase is where the model is applied to data it hasn't seen before.

In `SKLearn` these are encapsulated by the APIs `model.fit` and `model.predict`.

Naive Bayes Algorithm

Naive Bayes Algorithm

procedure TRAIN(s_{train})

$\widehat{\lambda} \leftarrow \texttt{Estimate}(s_{\text{train}})$ ▷ Parameter estimation via MLE.

for $(x_i, y_i) = z_i \in s_{\text{train}}$ **do**

$\widehat{p}_Y(y_i \mid \lambda) \leftarrow \widehat{p}_Y(y_i \mid \lambda) + \frac{1}{|s_{\text{train}}|}$ ▷ Estimates p_Y by $\widehat{p}_Y$.

Naive Bayes Algorithm

procedure TRAIN(s_{train})
  $\widehat{\lambda} \leftarrow \texttt{Estimate}(s_{\text{train}})$ ▷ Parameter estimation via MLE.
  for $(x_i, y_i) = z_i \in s_{\text{train}}$ **do**
    $\widehat{p}_Y(y_i \mid \lambda) \leftarrow \widehat{p}_Y(y_i \mid \lambda) + \frac{1}{|s_{\text{train}}|}$ ▷ Estimates p_Y by $\widehat{p}_Y$.

procedure INFERENCE(x)
  return $\operatorname{argmax}_{y \in \mathcal{Y}} \left(\underbrace{\log\left(p_{X|Y}(x \mid y, \widehat{\lambda})\right)}_{\text{closed form}} + \underbrace{\log\left(\widehat{p}_Y(y \mid \widehat{\lambda})\right)}_{\text{estimated in training}} \right)$ ▷ Iterates through all $y \in \mathcal{Y}$.

Recap

Main concepts:

- **Generative models**: learn $p_{X,Y}(x, y)$
- **Discriminative models**: learn $p_{Y|X}(y \mid x)$
- **MLE**:
$$\widehat{\lambda}_{\mathrm{MLE}}(s) \in \underset{\lambda \in \Lambda}{\operatorname{argmax}}\, p_S(s \mid \lambda)$$
- **MAP**:
$$\widehat{\lambda}_{\mathrm{MAP}}(s) \in \underset{\lambda \in \Lambda}{\operatorname{argmax}}\, p_{\Lambda|S}(\lambda \mid s)$$
- **Naive Bayes**:
$$\widehat{y}(x) \in \underset{y \in \mathcal{Y}}{\operatorname{argmax}}\, p_{Y|X}(y \mid x, \lambda)$$