Deep learning as a building block in probabilistic models
Part II

Pierre-Alexandre Mattei

`http://pamattei.github.io/`

@pamattei

Inria, Maasai team
Université Côte d'Azur

Overview of talk

Recap on (supervised) generative/discriminative models

Deep discriminative models for classification

Deep discriminative models for regression

Supervised learning with uncertainty: general goal

The goal of this lecture is to **train predictive machine learning models that can produce uncertainty assessments**.

Supervised learning with uncertainty: general goal

The goal of this lecture is to **train predictive machine learning models that can produce uncertainty assessments**.

So what do we want?

Supervised learning with uncertainty: general goal

The goal of this lecture is to **train predictive machine learning models that can produce uncertainty assessments**.

So what do we want?

We want to be able to make probabilistic preditions, like

- "the probability that the temperature in Nice tomorrow is between 20 and 25 degrees in 17%",
- "the probability that this patient has this kind of cancer is 56%".

Supervised learning with uncertainty: general goal

The goal of this lecture is to **train predictive machine learning models that can produce uncertainty assessments**.

So what do we want?

We want to be able to make probabilistic preditions, like

- "the probability that the temperature in Nice tomorrow is between 20 and 25 degrees in 17%",
- "the probability that this patient has this kind of cancer is 56%".

To do that, we need to have a **probabilistic model of our data**, hence the need for **generative models**, that can either be **fully generative** or **discriminative**.

What's a generative model?

Let's start with some data $\mathcal{D}$. For example, in the **regression** case with p-dimensional continuous features,

$$\mathcal{D} = ((\mathbf{x}_1, y_1), ..., (\mathbf{x}_n, y_n)) \in (\mathbb{R}^p \times \mathbb{R})^n.$$

In the **binary classification** case,

$$\mathcal{D} = ((\mathbf{x}_1, y_1), ..., (\mathbf{x}_n, y_n)) \in (\mathbb{R}^p \times \{0, 1\})^n.$$

What's a generative model?

Let's start with some data $\mathcal{D}$. For example, in the **regression** case with p-dimensional continuous features,

$$\mathcal{D} = ((\mathbf{x}_1, y_1), ..., (\mathbf{x}_n, y_n)) \in (\mathbb{R}^p \times \mathbb{R})^n.$$

In the **binary classification** case,

$$\mathcal{D} = ((\mathbf{x}_1, y_1), ..., (\mathbf{x}_n, y_n)) \in (\mathbb{R}^p \times \{0, 1\})^n.$$

In the **unsupervised** case, the data usually looks like $\mathcal{D} = (\mathbf{x}_1, ..., \mathbf{x}_n) \in (\mathbb{R}^p)^n$.

What's a generative model?

Let's start with some data $\mathcal{D}$. For example, in the **regression** case with p-dimensional continuous features,

$$\mathcal{D} = ((\mathbf{x}_1, y_1), ..., (\mathbf{x}_n, y_n)) \in (\mathbb{R}^p \times \mathbb{R})^n.$$

In the **binary classification** case,

$$\mathcal{D} = ((\mathbf{x}_1, y_1), ..., (\mathbf{x}_n, y_n)) \in (\mathbb{R}^p \times \{0, 1\})^n.$$

In the **unsupervised** case, the data usually looks like $\mathcal{D} = (\mathbf{x}_1, ..., \mathbf{x}_n) \in (\mathbb{R}^p)^n$.

We call $(\mathbf{x}_1, ..., \mathbf{x}_n)$ the **features** and $(y_1, ..., y_n)$ the **labels.** The features are usually stored in a $n \times p$ **matrix called the design matrix.**

What's a generative model?

Let's start with some data $\mathcal{D}$. For example, in the **regression** case with p-dimensional continuous features,

$$\mathcal{D} = ((\mathbf{x}_1, y_1), ..., (\mathbf{x}_n, y_n)) \in (\mathbb{R}^p \times \mathbb{R})^n.$$

In the **binary classification** case,

$$\mathcal{D} = ((\mathbf{x}_1, y_1), ..., (\mathbf{x}_n, y_n)) \in (\mathbb{R}^p \times \{0, 1\})^n.$$

In the **unsupervised** case, the data usually looks like $\mathcal{D} = (\mathbf{x}_1, ..., \mathbf{x}_n) \in (\mathbb{R}^p)^n$.

We call $(\mathbf{x}_1, ..., \mathbf{x}_n)$ the **features** and $(y_1, ..., y_n)$ the **labels.** The features are usually stored in a $n \times p$ **matrix called the design matrix.**

A generative model "describes a process that is assumed to give rise to some data"

David MacKay, in his book *Information Theory, Inference, and Learning Algorithms* (2003).

Formally, **a generative model will just be a probability density $p(\mathcal{D})$.**

Generative models for supervised learning: *General assumptions*

Although **we'll mostly focus on the unsupervised case my lectures**, let us begin with the (arguably simpler) supervised case $\mathcal{D} = ((\mathbf{x}_1, y_1), ..., (\mathbf{x}_n, y_n))$. It could be either a regression or a classification task, for example.

Generative models for supervised learning: *General assumptions*

Although **we'll mostly focus on the unsupervised case my lectures**, let us begin with the (arguably simpler) supervised case $\mathcal{D} = ((\mathbf{x}_1, y_1), ..., (\mathbf{x}_n, y_n))$. It could be either a regression or a classification task, for example.

Most of the time, it makes sense to build generative models that assume that **the observations are independent.** This leads to

$$p(\mathcal{D}) = p((\mathbf{x}_1, y_1), ..., (\mathbf{x}_n, y_n)) = \prod_{i=1}^{n} p(\mathbf{x}_i, y_i).$$

Usually, we also further assume that the data are **identically distributed.** This means that all the $(\mathbf{x}_i, y_i)$ will follow the same distribution that we may denote $p(\mathbf{x}, y)$

Generative models for supervised learning: *General assumptions*

Although **we'll mostly focus on the unsupervised case my lectures**, let us begin with the (arguably simpler) supervised case $\mathcal{D} = ((\mathbf{x}_1, y_1), ..., (\mathbf{x}_n, y_n))$. It could be either a regression or a classification task, for example.

Most of the time, it makes sense to build generative models that assume that **the observations are independent.** This leads to

$$p(\mathcal{D}) = p((\mathbf{x}_1, y_1), ..., (\mathbf{x}_n, y_n)) = \prod_{i=1}^{n} p(\mathbf{x}_i, y_i).$$

Usually, we also further assume that the data are **identically distributed.** This means that all the $(\mathbf{x}_i, y_i)$ will follow the same distribution that we may denote $p(\mathbf{x}, y)$

When these two assumptions are met, we say that the data are **independent and identically distributed (i.i.d.).** This is **super useful** is practice because, rather than having to find a distribution $p((\mathbf{x}_1, y_1), ..., (\mathbf{x}_n, y_n))$ over a very large space (whose dimension grows linearly with n), **we'll just have to find a much lower dimensional distribution** $p(\mathbf{x}, y)$**.**

Generative models for supervised learning: *Do we really have to be fully generative?*

Using the product rule, we may rewrite our $p(\mathbf{x}, y)$ as

$$p(\mathbf{x}, y) = p(\mathbf{x})p(y|\mathbf{x}) = p(y)p(\mathbf{x}|y).$$

Generative models for supervised learning: *Do we really have to be fully generative?*

Using the product rule, we may rewrite our $p(\mathbf{x}, y)$ as

$$p(\mathbf{x}, y) = p(\mathbf{x})p(y|\mathbf{x}) = p(y)p(\mathbf{x}|y).$$

But if we mainly want to do (probailisitic) preditions, knowing $p(y|\mathbf{x})$ is enough. It's exactly this **conditional distribution** that will give us statements like "the probability that this patient $\mathbf{x}$ has this kind of cancer is 56%".

Generative models for supervised learning: *Do we really have to be fully generative?*

Using the product rule, we may rewrite our $p(\mathbf{x}, y)$ as

$$p(\mathbf{x}, y) = p(\mathbf{x})p(y|\mathbf{x}) = p(y)p(\mathbf{x}|y).$$

But if we mainly want to do (probailisitic) preditions, knowing $p(y|\mathbf{x})$ is enough. It's exactly this **conditional distribution** that will give us statements like "the probability that this patient $\mathbf{x}$ has this kind of cancer is 56%".

Based on these insights, there are **two main approaches** for building $p(\mathbf{x}, y)$:

- The **fully generative (or model-based) approach posits a joint distribution** $p(\mathbf{x}, y)$ (often by specifying both $p(y)$ and $p(\mathbf{x}|y)$).
- The **discriminative (or conditional) approach** just specifies $p(y|\mathbf{x})$ and completely ignores $p(\mathbf{x})$.

Generative models for supervised learning: *Do we really have to be fully generative?*

Using the product rule, we may rewrite our $p(\mathbf{x}, y)$ as

$$p(\mathbf{x}, y) = p(\mathbf{x})p(y|\mathbf{x}) = p(y)p(\mathbf{x}|y).$$

But if we mainly want to do (probailisitic) preditions, knowing $p(y|\mathbf{x})$ is enough. It's exactly this **conditional distribution** that will give us statements like "the probability that this patient $\mathbf{x}$ has this kind of cancer is 56%".

Based on these insights, there are **two main approaches** for building $p(\mathbf{x}, y)$:

- The **fully generative (or model-based) approach posits a joint distribution** $p(\mathbf{x}, y)$ (often by specifying both $p(y)$ and $p(\mathbf{x}|y)$).
- The **discriminative (or conditional) approach** just specifies $p(y|\mathbf{x})$ and completely ignores $p(\mathbf{x})$.

What do you think are the benefits of the two approaches?

Generative models for supervised learning: *Discriminative vs fully generative*

A few examples of the two approaches:

- **Discriminative:** linear and logistic regression, Neural nets for regression/classification, Gaussian process regression/classification
- **Generative:** Linear/quadratic discriminant analysis, Mixture discriminant analysis, Supervised variational autoencoders, most of the models Charles Bouveyron will talk about in his course[1]

[1] cf. his book with G. Celeux, B. Murphy et A. Raftery.

Generative models for supervised learning: *Discriminative vs fully generative*

A few examples of the two approaches:

- **Discriminative:** linear and logistic regression, Neural nets for regression/classification, Gaussian process regression/classification
- **Generative:** Linear/quadratic discriminant analysis, Mixture discriminant analysis, Supervised variational autoencoders, most of the models Charles Bouveyron will talk about in his course[1]

Some of the advantages/drawbacks:

- **Discriminative:** much easier to design (and usually train) because we don't have to model $p(\mathbf{x})$. Usually more accurate where we have a lot of data. **Cannot acommodate to missing features or do semi-supervised learning (missing labels) easily.**
- **Generative:** Can deal with missing features/labels. Usually more accurate when we do not have a lot of data. Usually more robust to adversarial examples. **Requires to specify $p(\mathbf{x})$ which is often hard because $\mathbf{x}$ may be high-dimensional/complex.**

[1] cf. his book with G. Celeux, B. Murphy et A. Raftery.

Generative vs Discriminative: a concrete example

Decanter

Police uncover Italian wine fraud

Maggie Rosen
August 23, 2007

3 shares

Police have broken up an international counterfeit wine racket involving top Italian wines.

German and Italian police forces have uncovered a cross-border scam involving table wine from Puglia and Piedmont sold as Barolo, Brunello di Montalcino, Amarone and Chianti.

Unlabelled wine was brought into Germany, where it was given fake DOC and DOCG seals and phoney labels from well-known producers as well as non-existent wineries.

Ten people have been charged. It is thought the con could be worth €750,000.

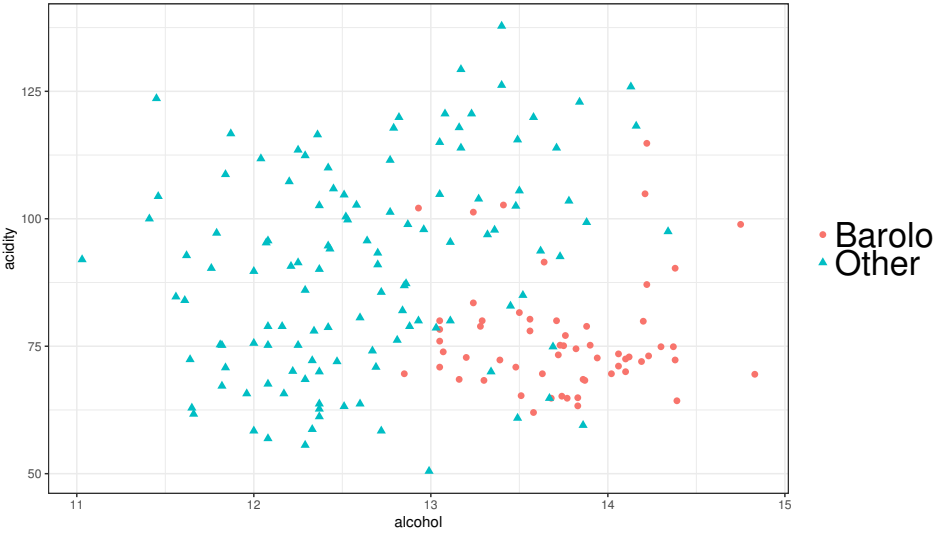
Article from `decanter.com/wine-news/police-uncover-italian-wine-fraud-88060/`

Generative vs Discriminative: a concrete example

One of the wines the bad guys counterfeited was from the Barolo region. According to Wikipedia, those wines have "pronounced tannins and acidity", and "moderate to high alcohol levels (Minimum 13%)". This would help a trained human recognise them, but **could we train an algorithm to learn those characteristics?**

Picture from Wikipedia

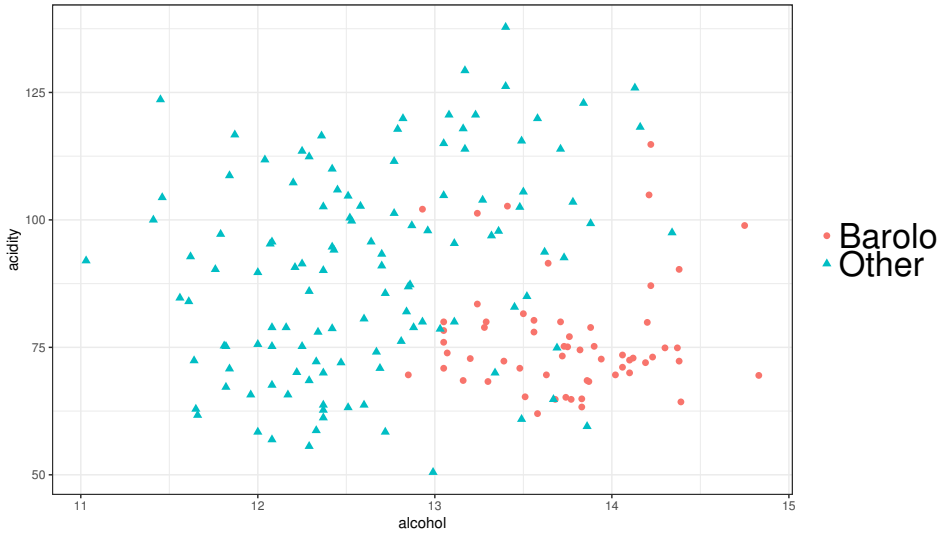
Generative vs Discriminative: a concrete example



Data from Forina, Armanino, Castino, and Ubigli, (Vitis, 1986).

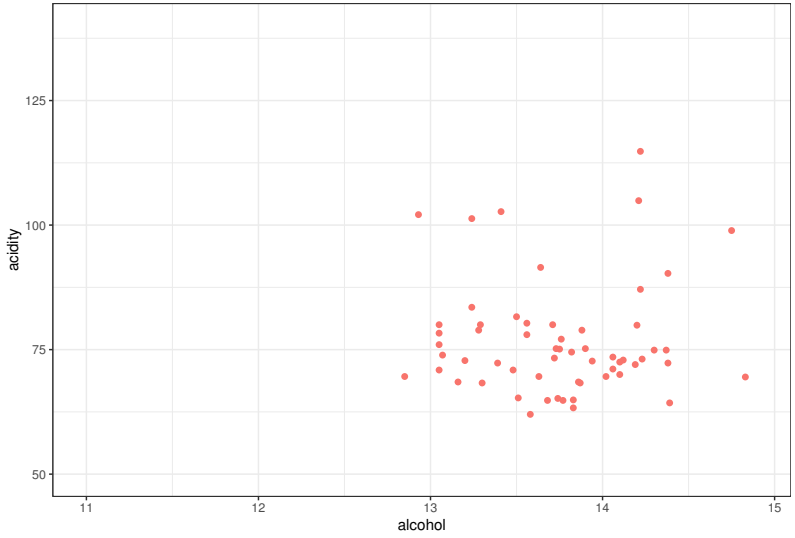
Generative vs Discriminative: a concrete example

The **generative** way would use the formula $p(\mathbf{x}, y) = p(y)p(\mathbf{x}|y)$ and model the class conditional distributions $p(\mathbf{x}|y)$ using a continuous bivariate ditribution (e.g. 2D Gaussians).



Generative vs Discriminative: a concrete example

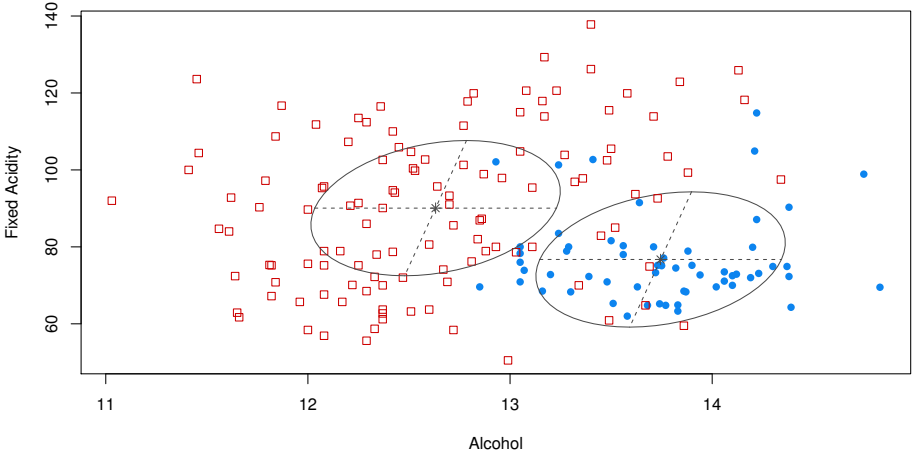
The **generative** way would use the formula $p(\mathbf{x}, y) = p(y)p(\mathbf{x}|y)$ and model the class conditional distributions $p(\mathbf{x}|y)$ using a continuous bivariate ditribution (e.g. 2D Gaussians).



• Barolo

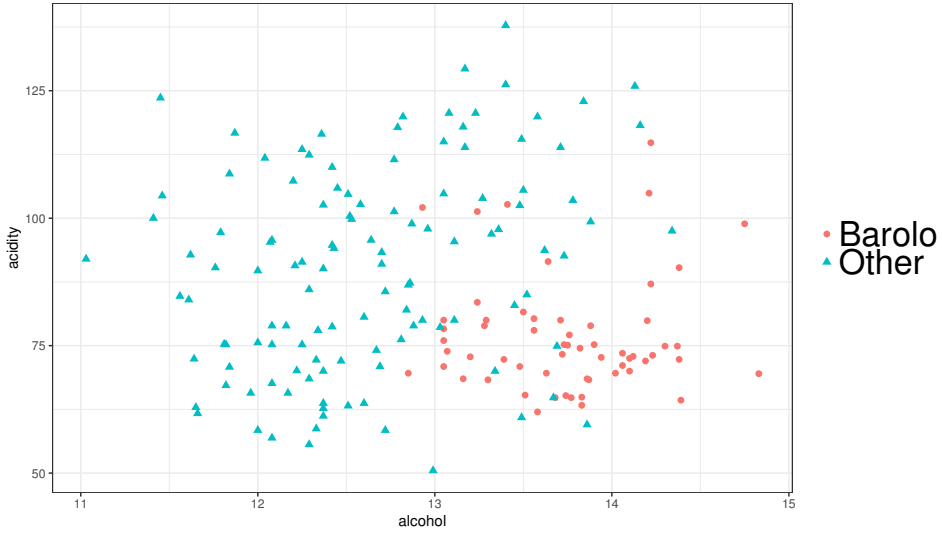
Generative vs Discriminative: a concrete example

The **generative** way would use the formula $p(\mathbf{x}, y) = p(y)p(\mathbf{x}|y)$ and model the class conditional distributions $p(\mathbf{x}|y)$ using a continuous bivariate ditribution (e.g. 2D Gaussians). Here is what we obtain using the R package `Mclust` (Scrucca, Fop, Murphy, and Raftery, R Journal, 2016).



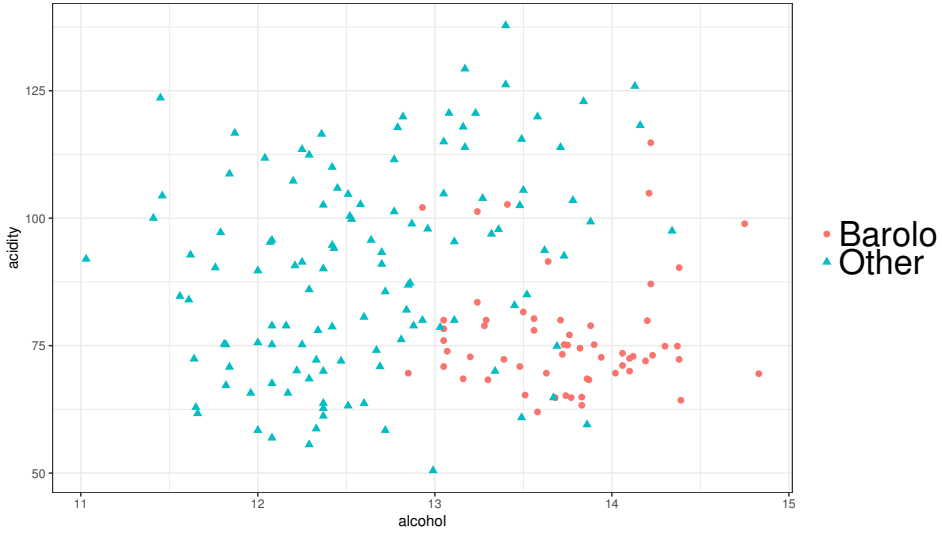
Generative vs Discriminative: a concrete example

The **discriminative** way would only model $p(y|\mathbf{x})$. Since there are only 2 classes, this means that $p(y|\mathbf{x})$ will be a **Bernoulli random variable whose parameter $\pi(\mathbf{x}) \in [0, 1]$ is a function of the features.**



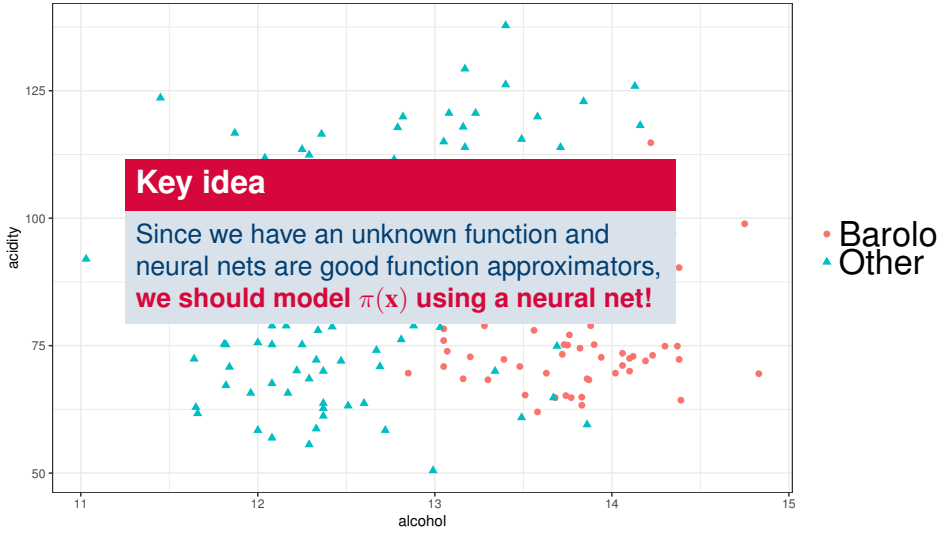
Generative vs Discriminative: a concrete example

The **discriminative** way would only model $p(y|\mathbf{x})$. Since there are only 2 classes, this means that $p(y|\mathbf{x})$ will be a **Bernoulli random variable whose parameter $\pi(\mathbf{x}) \in [0, 1]$ is a function of the features.**



Generative vs Discriminative: a concrete example

The **discriminative** way would only model $p(y|\mathbf{x})$. Since there are only 2 classes, this means that $p(y|\mathbf{x})$ will be a **Bernoulli random variable whose parameter $\pi(\mathbf{x}) \in [0, 1]$ is a function of the features.**



Overview of talk

Recap on (supervised) generative/discriminative models

Deep discriminative models for classification

Deep discriminative models for regression

How to create a deep discriminative model?

We'll **focus now on the discriminative approach using neural nets**, because it is simpler. For more on the differences and links between the generative and discriminative schools, a wonderful reference is Tom Minka's short note on the subject: *Discriminative models, not discriminative training* [2].

[2] https://tminka.github.io/papers/minka-discriminative.pdf

How to create a deep discriminative model?

We'll **focus now on the discriminative approach using neural nets**, because it is simpler. For more on the differences and links between the generative and discriminative schools, a wonderful reference is Tom Minka's short note on the subject: *Discriminative models, not discriminative training* [2].

Let us go back to our discriminative model: we can write it

$$p(y|\mathbf{x}) = \mathcal{B}(y|\pi(\mathbf{x})) = \pi(\mathbf{x})^y(1-\pi(\mathbf{x}))^{1-y},$$

where $\mathcal{B}(\cdot|\theta)$ denotes the density of a Bernoulli distribution with parameter $\theta \in [0,1]$. The key idea is then to **model the function $\mathbf{x} \mapsto \pi(\mathbf{x})$ using a neural net.**

[2] https://tminka.github.io/papers/minka-discriminative.pdf

How to create a deep discriminative model?

We'll **focus now on the discriminative approach using neural nets**, because it is simpler. For more on the differences and links between the generative and discriminative schools, a wonderful reference is Tom Minka's short note on the subject: *Discriminative models, not discriminative training* [2].

Let us go back to our discriminative model: we can write it

$$p(y|\mathbf{x}) = \mathcal{B}(y|\pi(\mathbf{x})) = \pi(\mathbf{x})^y(1-\pi(\mathbf{x}))^{1-y},$$

where $\mathcal{B}(\cdot|\theta)$ denotes the density of a Bernoulli distribution with parameter $\theta \in [0,1]$. The key idea is then to **model the function $\mathbf{x} \mapsto \pi(\mathbf{x})$ using a neural net.**

This key idea goes way beyond the discriminative context

This general strategy of **using outputs of neural nets as parameters of simple probability distributions is the main recipe for building deep generative models.** It has been used extensively, for example in deep latent variable models such as variational autoencoders (VAEs) or generative adversarial networks (GANs).

[2] https://tminka.github.io/papers/minka-discriminative.pdf

How to model π

Our discriminative model for binary classification is

$$p(y|\mathbf{x}) = \mathcal{B}(y|\pi(\mathbf{x})) = \pi(\mathbf{x})^y(1-\pi(\mathbf{x}))^{1-y},$$

and we wish to model π using a neural net. **But what kind of neural net?**

How to model π

Our discriminative model for binary classification is

$$p(y|\mathbf{x}) = \mathcal{B}(y|\pi(\mathbf{x})) = \pi(\mathbf{x})^y(1-\pi(\mathbf{x}))^{1-y},$$

and we wish to model π using a neural net. **But what kind of neural net?**

The only really important constraint of the problem is that we need to have

$$\forall \mathbf{x} \in \mathbb{R}^p,\ \pi(\mathbf{x}) \in [0,1].$$

Is it possible to enforce that easily in a neural net?

How to model π

Our discriminative model for binary classification is

$$p(y|\mathbf{x}) = \mathcal{B}(y|\pi(\mathbf{x})) = \pi(\mathbf{x})^y(1-\pi(\mathbf{x}))^{1-y},$$

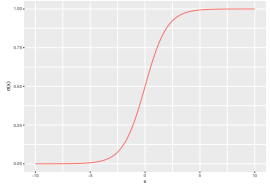
and we wish to model π using a neural net. **But what kind of neural net?**

The only really important constraint of the problem is that we need to have

$$\forall \mathbf{x} \in \mathbb{R}^p,\ \pi(\mathbf{x}) \in [0,1].$$

Is it possible to enforce that easily in a neural net?

Yes! By using a function that only output stuff in $[0,1]$ as the output layer. For example the **logistic sigmoid function** $\sigma : a \mapsto \frac{1}{1+\exp(-a)}$.



How to model π

So at the end, we'll model π using the formula

$$\pi(\mathbf{x}) = \sigma(f_{\boldsymbol{\theta}}(\mathbf{x})),$$

where σ is the sigmoid function and $f_\theta : \mathbb{R}^p \longrightarrow \mathbb{R}$ **is any neural network** (whose weights are stored in a vector $\boldsymbol{\theta}$) that takes the features as input and returns an unconstrained real number.

How to model π

So at the end, we'll model π using the formula

$$\pi(\mathbf{x}) = \sigma(f_{\boldsymbol{\theta}}(\mathbf{x})),$$

where σ is the sigmoid function and $f_\theta : \mathbb{R}^p \longrightarrow \mathbb{R}$ **is any neural network** (whose weights are stored in a vector $\boldsymbol{\theta}$) that takes the features as input and returns an unconstrained real number.

We have a lot of flexibility to choose f_θ. In particular, if the features $\mathbf{x}_1, ..., \mathbf{x}_n$ are images, we could use a CNN. In the case of time-series, we could use a recurrent neural net. In the case of sets, we could use a deepsets architecture (Zaheer et al., NeurIPS 2017).

How to model π

So at the end, we'll model π using the formula

$$\pi(\mathbf{x}) = \sigma(f_{\boldsymbol{\theta}}(\mathbf{x})),$$

where σ is the sigmoid function and $f_\theta : \mathbb{R}^p \longrightarrow \mathbb{R}$ **is any neural network** (whose weights are stored in a vector $\boldsymbol{\theta}$) that takes the features as input and returns an unconstrained real number.

We have a lot of flexibility to choose f_θ. In particular, if the features $\mathbf{x}_1, ..., \mathbf{x}_n$ are images, we could use a CNN. In the case of time-series, we could use a recurrent neural net. In the case of sets, we could use a deepsets architecture (Zaheer et al., NeurIPS 2017).

For the wine example, we could just take a small MLP

$$f_{\boldsymbol{\theta}}(\mathbf{x}) = \mathbf{W}_1 \tanh(\mathbf{W}_0 \mathbf{x}_i + \mathbf{b}_0) + b_1.$$

How to model π

So at the end, we'll model π using the formula

$$\pi(\mathbf{x}) = \sigma(f_{\boldsymbol{\theta}}(\mathbf{x})),$$

where σ is the sigmoid function and $f_\theta : \mathbb{R}^p \longrightarrow \mathbb{R}$ **is any neural network** (whose weights are stored in a vector $\boldsymbol{\theta}$) that takes the features as input and returns an unconstrained real number.

We have a lot of flexibility to choose f_θ**.** In particular, if the features $\mathbf{x}_1, ..., \mathbf{x}_n$ are images, we could use a CNN. In the case of time-series, we could use a recurrent neural net. In the case of sets, we could use a deepsets architecture (Zaheer et al., NeurIPS 2017).

For the wine example, we could just take a small MLP

$$f_{\boldsymbol{\theta}}(\mathbf{x}) = \mathbf{W}_1 \tanh(\mathbf{W}_0 \mathbf{x}_i + \mathbf{b}_0) + b_1.$$

Since the function π and the model $p(y|\mathbf{x})$ now depend on some parameters $\boldsymbol{\theta}$, **we'll denote them by** $\pi_{\boldsymbol{\theta}}$ **and** $p_{\boldsymbol{\theta}}(y|\mathbf{x})$ **from now on.**

How to find θ

There are many ways to find good parameter values for a generative model. One could use Bayesian inference, score matching, the method of moments, adversarial training... Let us focus on one of the most traditional ways: **maximum likelihood.** The idea is to find a $\hat{\boldsymbol{\theta}}$ that maximises the **log-likelihood function** $\log p_{\boldsymbol{\theta}}(\mathcal{D})$**.**

How to find θ

There are many ways to find good parameter values for a generative model. One could use Bayesian inference, score matching, the method of moments, adversarial training... Let us focus on one of the most traditional ways: **maximum likelihood.** The idea is to find a $\hat{\theta}$ that maximises the **log-likelihood function** $\log p_{\boldsymbol{\theta}}(\mathcal{D})$**.**

In the discriminative case, the likelihood is:

$$\log p_{\boldsymbol{\theta}}(\mathcal{D}) = \sum_{i=1}^{n} \log p_{\boldsymbol{\theta}}(y_i, \mathbf{x}_i) = \sum_{i=1}^{N} \log p_{\boldsymbol{\theta}}(y_i|\mathbf{x}_i) + \sum_{i=1}^{N} \log p(\mathbf{x}_i),$$

but, since we don't model $p(\mathbf{x})$, $\sum_{i=1}^{n} \log p(\mathbf{x}_i)$ is constant, and maximising $\ell(\boldsymbol{\theta})$ is equivalent to maximising

$$\ell(\boldsymbol{\theta}) = \sum_{i=1}^{n} \log p_{\boldsymbol{\theta}}(y_i|\mathbf{x}_i).$$

We'll also call $\ell(\boldsymbol{\theta})$ the likelihood (in fact, we'll call any function that is equal to $\log p_{\boldsymbol{\theta}}(\mathcal{D})$ up to a constant the likelihood).

How to find θ: from ML to XENT

We have

$$\ell(\boldsymbol{\theta}) = \sum_{i=1}^{n} \log p_{\boldsymbol{\theta}}(y_i|\mathbf{x}_i) = \sum_{i=1}^{N} \log \left(\pi(\mathbf{x})^{y_i}(1-\pi(\mathbf{x}))^{1-y_i}\right),$$

which leads to

$$\ell(\boldsymbol{\theta}) = \sum_{i=1}^{n} \left[y_i \pi(\mathbf{x}_i) + (1-y_i)\log(1-\pi(\mathbf{x}_i))\right].$$

How to find θ: from ML to XENT

We have

$$\ell(\boldsymbol{\theta}) = \sum_{i=1}^{n} \log p_{\boldsymbol{\theta}}(y_i|\mathbf{x}_i) = \sum_{i=1}^{N} \log \left(\pi(\mathbf{x})^{y_i}(1-\pi(\mathbf{x}))^{1-y_i}\right),$$

which leads to

$$\ell(\boldsymbol{\theta}) = \sum_{i=1}^{n} \left[y_i \pi(\mathbf{x}_i) + (1-y_i)\log(1-\pi(\mathbf{x}_i))\right].$$

We will want to maximise this function, which is equivalent to minimising its opposite, which is called the **cross-entropy loss**.
The cross-entropy loss is the most commonly used loss for neural networks, and is **a way of doing maximum likelihood without necessarily saying it.**

Multiclass classification

If we have a multiclass problem (with K classes), we need to replace the Bernoulli distribution with a **categorical distribution**[3]. The model becomes

$$p(y|\mathbf{x}) = \text{Cat}(y|\boldsymbol{\pi}(\mathbf{x})).$$

The output of the neural net $\boldsymbol{\pi}(\mathbf{x})$ is no longer a single probability but a **vector of proportions** of dimension K:

$$\boldsymbol{\pi}(\mathbf{x}) = (\boldsymbol{\pi}(\mathbf{x})_1, ..., \boldsymbol{\pi}(\mathbf{x})_K).$$

Of course, the proportions must be in $[0, 1]$ and sum to one:

$$\sum_{k=1}^{K} \boldsymbol{\pi}(\mathbf{x})_k = 1.$$

[3] Bishop (see e.g. Section 2.2) calls this a multinomial distribution

Multiclass classification

If we have a multiclass problem (with K classes), we need to replace the Bernoulli distribution with a **categorical distribution**[3]. The model becomes

$$p(y|\mathbf{x}) = \text{Cat}(y|\boldsymbol{\pi}(\mathbf{x})).$$

The output of the neural net $\boldsymbol{\pi}(\mathbf{x})$ is no longer a single probability but a **vector of proportions** of dimension K:

$$\boldsymbol{\pi}(\mathbf{x}) = (\boldsymbol{\pi}(\mathbf{x})_1, ..., \boldsymbol{\pi}(\mathbf{x})_K).$$

Of course, the proportions must be in $[0, 1]$ and sum to one:

$$\sum_{k=1}^{K} \boldsymbol{\pi}(\mathbf{x})_k = 1.$$

How can we enforce that the outputs of our neural net sum to one?

[3] Bishop (see e.g. Section 2.2) calls this a multinomial distribution

Multiclass classification with the softmax

A simple way to make sure that the outputs of a neural net are indeed in $[0, 1]$ and sum to one is to use a **softmax as a last layer**.

Multiclass classification with the softmax

A simple way to make sure that the outputs of a neural net are indeed in $[0, 1]$ and sum to one is to use a **softmax as a last layer**.

The **softmax function** (aka normalised exponential), that's defined as:

$$\text{softmax}(a_1, ..., a_K) = \left(\frac{\exp a_1}{\sum_{j=1}^K \exp a_j}, \frac{\exp a_2}{\sum_{j=1}^K \exp a_j}, ..., \frac{\exp a_K}{\sum_{j=1}^K \exp a_j} \right)^\intercal.$$

So at the end our model will look like

$$p(y|\mathbf{x}) = \text{Cat}(y|\boldsymbol{\pi}_{\boldsymbol{\theta}}(\mathbf{x})),$$

with

$$\boldsymbol{\pi}_{\theta}(\mathbf{x}) = \text{softmax}(f_{\boldsymbol{\theta}}(\mathbf{x})),$$

The function $f_{\boldsymbol{\theta}}$ can be modelled by **any kind of neural network** with input space $\mathbb{R}^p$ (the data space) and output space $\mathbb{R}^K$. The unknown weights of the network are denoted by $\boldsymbol{\theta}$.

How to find θ: from ML to XENT (continued)

In the multiclass setting, the goal is again to maximise the likelihood:

$$\ell(\boldsymbol{\theta}) = \sum_{i=1}^{n} \log p_{\boldsymbol{\theta}}(y_i|\mathbf{x}_i) = \sum_{i=1}^{n} \log \left(\mathsf{Cat}(y_i|\boldsymbol{\pi}_{\boldsymbol{\theta}}(\mathbf{x}_i))\right).$$

One convenient way of writing the categorical density with parameter π is

$$\mathsf{Cat}(y|\boldsymbol{\pi}) = \pi_1^{y_1} ... \pi_K^{y_K},$$

where y **is a one-hot encoding** of the label. This can be used to rewrite the likelihood:

$$\ell(\boldsymbol{\theta}) = \sum_{i=1}^{n} \sum_{k=1}^{K} y_{ik} \log \boldsymbol{\pi}_{\boldsymbol{\theta}}(\mathbf{x}_i)_k.$$

The opposite of this quantity is often called the **cross-entropy loss**. So minimising the cross-entropy is equivalent to maximising the likelihood of a discriminative model.

Overview of talk

Recap on (supervised) generative/discriminative models

Deep discriminative models for classification

Deep discriminative models for regression

What is regression again?

In regression, the goal is to predict a **continuous target** $y \in \mathbb{R}$ using some features $\mathbf{x} \in \mathbb{R}^p$. As before, the discriminative approach is to model only $p(y|\mathbf{x})$. This is no longer a discrete but a continuous distribution. What distribution could we choose?

What is regression again?

In regression, the goal is to predict a **continuous target** $y \in \mathbb{R}$ using some features $\mathbf{x} \in \mathbb{R}^p$. As before, the discriminative approach is to model only $p(y|\mathbf{x})$. This is no longer a discrete but a continuous distribution. What distribution could we choose?

Often, we simply use a Gaussian! Indeed, the most famous regression model is the **Gaussian linear regression**:

$$p_{\boldsymbol{\beta},\sigma}(y|x) = \mathcal{N}(y|\mu + \mathbf{x}^T\boldsymbol{\beta}, \sigma^2),$$

which can be rewritten in a perhaps more familiar vector form

$$\mathbf{y} = \mathbf{X}\boldsymbol{\beta} + \mu\mathbf{1}_n + \boldsymbol{\varepsilon},$$

with $\boldsymbol{\varepsilon} \sim \mathcal{N}(0, \sigma^2\mathbf{I}_n)$.

What is regression again?

In regression, the goal is to predict a **continuous target** $y \in \mathbb{R}$ using some features $\mathbf{x} \in \mathbb{R}^p$. As before, the discriminative approach is to model only $p(y|\mathbf{x})$. This is no longer a discrete but a continuous distribution. What distribution could we choose?

Often, we simply use a Gaussian! Indeed, the most famous regression model is the **Gaussian linear regression**:

$$p_{\boldsymbol{\beta},\sigma}(y|x) = \mathcal{N}(y|\mu + \mathbf{x}^T\boldsymbol{\beta}, \sigma^2),$$

which can be rewritten in a perhaps more familiar vector form

$$\mathbf{y} = \mathbf{X}\boldsymbol{\beta} + \mu\mathbf{1}_n + \boldsymbol{\varepsilon},$$

with $\boldsymbol{\varepsilon} \sim \mathcal{N}(0, \sigma^2\mathbf{I}_n)$.

How do we "make this model deep"?

What is regression again?

In regression, the goal is to predict a **continuous target** $y \in \mathbb{R}$ using some features $\mathbf{x} \in \mathbb{R}^p$. As before, the discriminative approach is to model only $p(y|\mathbf{x})$. This is no longer a discrete but a continuous distribution. What distribution could we choose?

Often, we simply use a Gaussian! Indeed, the most famous regression model is the **Gaussian linear regression**:

$$p_{\boldsymbol{\beta},\sigma}(y|x) = \mathcal{N}(y|\mu + \mathbf{x}^T\boldsymbol{\beta}, \sigma^2),$$

which can be rewritten in a perhaps more familiar vector form

$$\mathbf{y} = \mathbf{X}\boldsymbol{\beta} + \mu\mathbf{1}_n + \boldsymbol{\varepsilon},$$

with $\varepsilon \sim \mathcal{N}(0, \sigma^2\mathbf{I}_n)$.

How do we "make this model deep"?

By replacing the simple linear function $\mathbf{x} \mapsto \mu + \mathbf{x}^T\beta$ by a neural network $\mu_{\boldsymbol{\theta}}(\mathbf{x})$.

Deep regression

In regression, the goal is to predict a **continuous target** $y \in \mathbb{R}$ using some features $\mathbf{x} \in \mathbb{R}^p$. As before, the discriminative approach is to model only $p(y|\mathbf{x})$. This is no longer a discrete but a continuous distribution. What distribution could we choose?

Deep regression

In regression, the goal is to predict a **continuous target** $y \in \mathbb{R}$ using some features $\mathbf{x} \in \mathbb{R}^p$. As before, the discriminative approach is to model only $p(y|\mathbf{x})$. This is no longer a discrete but a continuous distribution. What distribution could we choose?

Often, we simply use a Gaussian! Indeed, the most famous regression model is the **Gaussian linear regression**:

$$p_{\boldsymbol{\beta},\sigma}(y|x) = \mathcal{N}(y|\mu + \mathbf{x}^T\boldsymbol{\beta}, \sigma^2),$$

which can be rewritten in a perhaps more familiar vector form

$$\mathbf{y} = \mathbf{X}\boldsymbol{\beta} + \mu\mathbf{1}_n + \boldsymbol{\varepsilon},$$

with $\boldsymbol{\varepsilon} \sim \mathcal{N}(0, \sigma^2\mathbf{I}_n)$.

Deep regression

In regression, the goal is to predict a **continuous target** $y \in \mathbb{R}$ using some features $\mathbf{x} \in \mathbb{R}^p$. As before, the discriminative approach is to model only $p(y|\mathbf{x})$. This is no longer a discrete but a continuous distribution. What distribution could we choose?

Often, we simply use a Gaussian! Indeed, the most famous regression model is the **Gaussian linear regression**:

$$p_{\boldsymbol{\beta},\sigma}(y|x) = \mathcal{N}(y|\mu + \mathbf{x}^T\boldsymbol{\beta}, \sigma^2),$$

which can be rewritten in a perhaps more familiar vector form

$$\mathbf{y} = \mathbf{X}\boldsymbol{\beta} + \mu\mathbf{1}_n + \boldsymbol{\varepsilon},$$

with $\boldsymbol{\varepsilon} \sim \mathcal{N}(0, \sigma^2\mathbf{I}_n)$.

How do we "make this model deep"?

Deep regression

In regression, the goal is to predict a **continuous target** $y \in \mathbb{R}$ using some features $\mathbf{x} \in \mathbb{R}^p$. As before, the discriminative approach is to model only $p(y|\mathbf{x})$. This is no longer a discrete but a continuous distribution. What distribution could we choose?

Often, we simply use a Gaussian! Indeed, the most famous regression model is the **Gaussian linear regression**:

$$p_{\boldsymbol{\beta},\sigma}(y|x) = \mathcal{N}(y|\mu + \mathbf{x}^T\boldsymbol{\beta}, \sigma^2),$$

which can be rewritten in a perhaps more familiar vector form

$$\mathbf{y} = \mathbf{X}\boldsymbol{\beta} + \mu\mathbf{1}_n + \boldsymbol{\varepsilon},$$

with $\boldsymbol{\varepsilon} \sim \mathcal{N}(0, \sigma^2\mathbf{I}_n)$.

How do we "make this model deep"?

By replacing the simple linear function $\mathbf{x} \mapsto \mu + \mathbf{x}^T\beta$ by a neural network $\mu_{\boldsymbol{\theta}}(\mathbf{x})$.

Deep regression

The simplest deep regression model is

$$p_{\boldsymbol{\theta}}(y|\mathbf{x}) = \mathcal{N}(y|\mu_{\boldsymbol{\theta}}(\mathbf{x}), \sigma^2),$$

which could also be written

$$\mathbf{y} = (\mu_{\boldsymbol{\theta}}(\mathbf{x}_i))_{i=1}^n + \boldsymbol{\varepsilon},$$

with $\boldsymbol{\varepsilon} \sim \mathcal{N}(0, \sigma^2 \mathbf{I}_n)$.
What are the parameters to learn?

Deep regression

The simplest deep regression model is

$$p_{\boldsymbol{\theta}}(y|\mathbf{x}) = \mathcal{N}(y|\mu_{\boldsymbol{\theta}}(\mathbf{x}), \sigma^2),$$

which could also be written

$$\mathbf{y} = (\mu_{\boldsymbol{\theta}}(\mathbf{x}_i))_{i=1}^n + \boldsymbol{\varepsilon},$$

with $\boldsymbol{\varepsilon} \sim \mathcal{N}(0, \sigma^2 \mathbf{I}_n)$.
What are the parameters to learn?

We have to learn $\boldsymbol{\theta}$ **and** σ**.** Do you see a way to generalise this model further by adding another deep learning touch?

Deep regression

The simplest deep regression model is

$$p_{\boldsymbol{\theta}}(y|\mathbf{x}) = \mathcal{N}(y|\mu_{\boldsymbol{\theta}}(\mathbf{x}), \sigma^2),$$

which could also be written

$$\mathbf{y} = (\mu_{\boldsymbol{\theta}}(\mathbf{x}_i))_{i=1}^n + \boldsymbol{\varepsilon},$$

with $\boldsymbol{\varepsilon} \sim \mathcal{N}(0, \sigma^2 \mathbf{I}_n)$.
What are the parameters to learn?

We have to learn $\boldsymbol{\theta}$ **and** σ**.** Do you see a way to generalise this model further by adding another deep learning touch?

Rather than assuming that σ is constant, we could rather **model it using a neural net** $\sigma_{\boldsymbol{\theta}(\mathbf{x})}$**!**
Why on earth would that be a good idea?

Deep heteroscedatic regression

A deep heteroscedatic regression model is

$$p_{\theta}(y|\mathbf{x}) = \mathcal{N}(y|\mu_{\theta}(\mathbf{x}), \sigma_{\theta}(\mathbf{x})^2),$$

its key strength is that it allows to model **non-constant uncertainties about the value of the target.**

Deep heteroscedatic regression

A deep heteroscedatic regression model is

$$p_{\theta}(y|\mathbf{x}) = \mathcal{N}(y|\mu_{\theta}(\mathbf{x}), \sigma_{\theta}(\mathbf{x})^2),$$

its key strength is that it allows to model **non-constant uncertainties about the value of the target.**

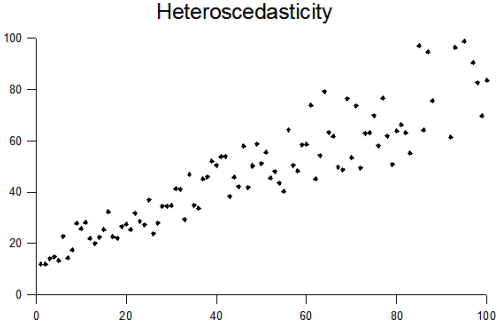


Figure from https://en.wikipedia.org/wiki/Heteroscedasticity.