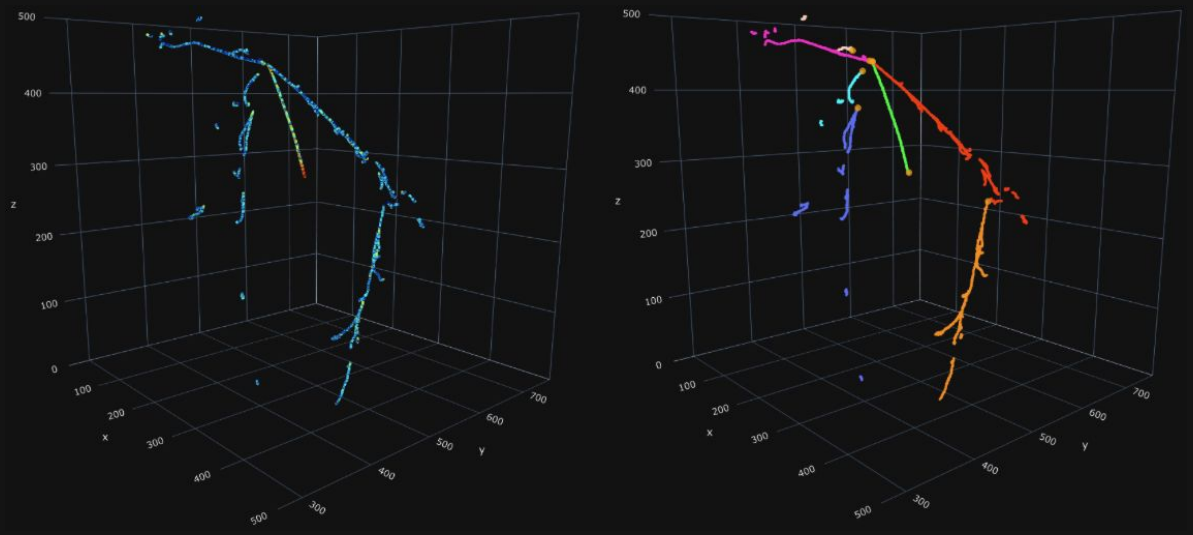
Linear Models
KMI 2020 @ Nagoya University

Original image credit: xkcd



Kazu (SLAC)
on behalf of the core team
(me + Aashwin Ananda Mishra + Francois Drielsma)

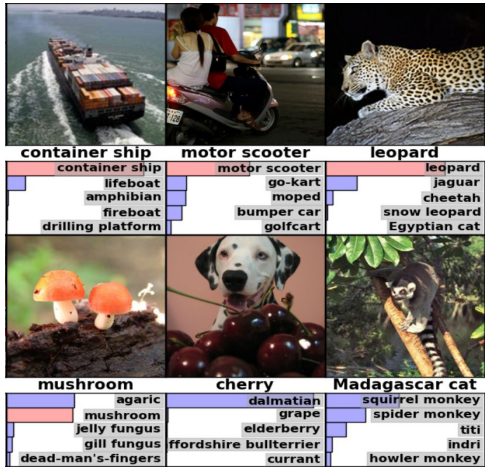
Useful references

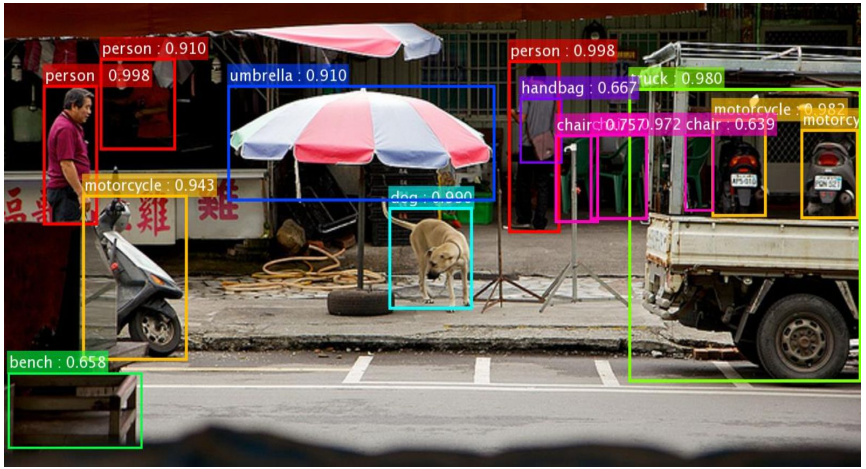
- Online textbook: deep learning (I. Goodfellow and Y. Bengio and A. Courville)
- Online textbook: pattern recognition and machine learning (C. M. Bishop)

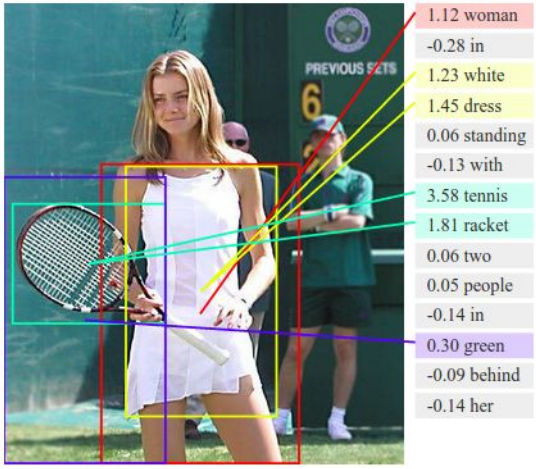
... sure, you can log out from this session and start reading those books ;) ...

Machine Learning

- Study of an algorithm that is able to *learn* from data.
  - Given some tasks T and performance measure P, the algorithm is said to learn from data, or experience E, if its performance at T, measured by P, improves with E.
- A cross-road of statistics (probability) and computer science (algorithms) where learning is casted to an optimization problem



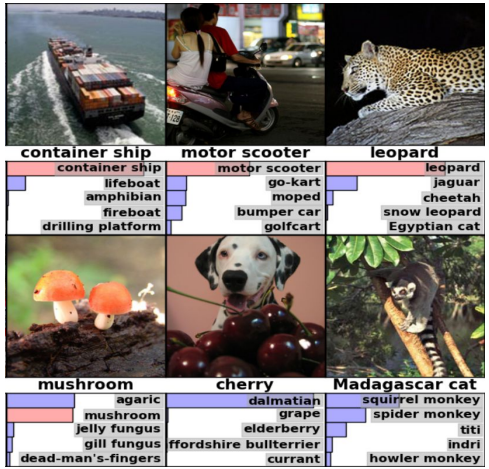


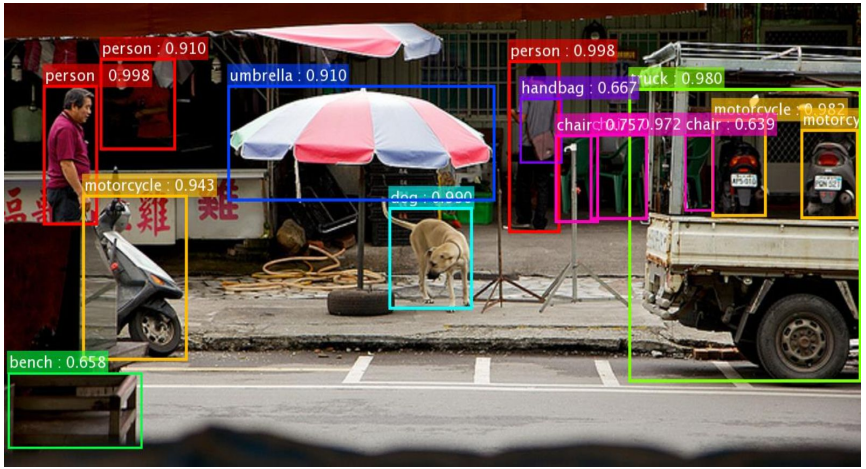


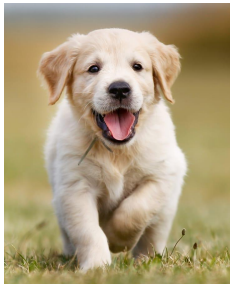
What/How to learn from data?

Supervised learning

- Given data X and label Y + assume an underlying function P(X)=Y, learn an approximate function Q that mimics P.
- Classification (i.e. discrete labels like dog v.s. cat), regression (i.e. continuous variable like energy of a particle), etc.







What/How to learn from data?

Supervised learning

- Given data X and label Y + assume an underlying function P(X)=Y, learn an approximate function Q that mimics P.
- Classification (i.e. discrete labels like dog v.s. cat), regression (i.e. continuous variable like energy of a particle), etc.

Unsupervised learning

- Only given data, learn underlying structure
- Partitioning (clustering), principal component analysis, generative models

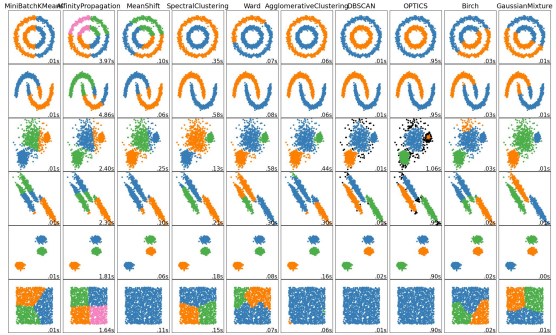
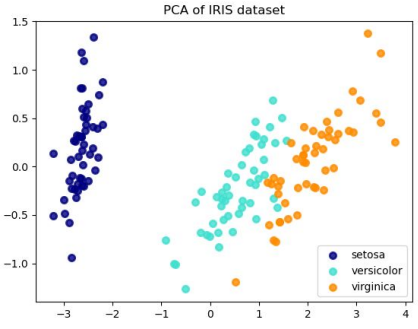


Image credit: Scikit



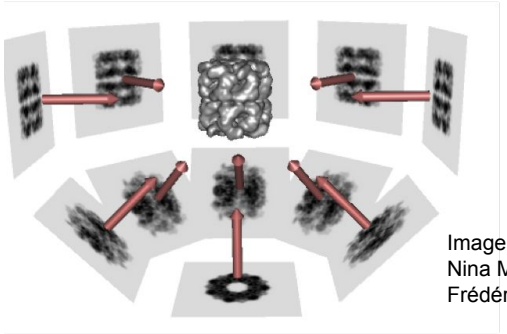


Image credit: Nina Miolane, Frédéric Poitevin*

What/How to learn from data?

Supervised learning

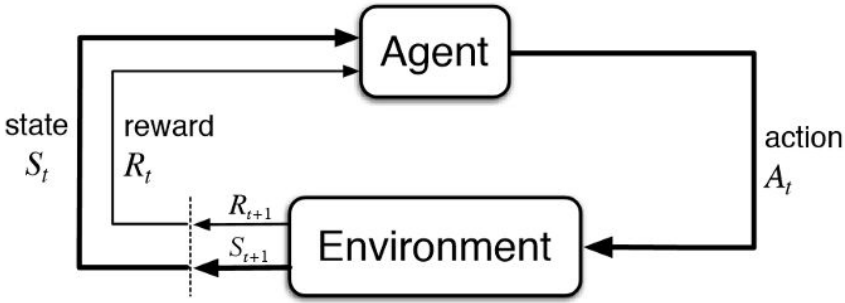
- Given data X and label Y + assume an underlying function P(X)=Y, learn an approximate function Q that mimics P.
- Classification (i.e. discrete labels like dog v.s. cat), regression (i.e. continuous variable like energy of a particle), etc.

Unsupervised learning

- Only given data, learn underlying structure
- Partitioning (clustering), principal component analysis, generative models

Reinforcement learning

- May not even have static data! Learn to gain most cumulative reward by interacting with the environment



What/How to learn from data?

Supervised learning

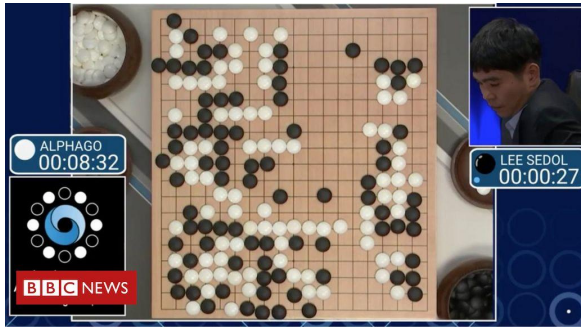
- Given data X and label Y + assume an underlying function P(X)=Y, learn an approximate function Q that mimics P.
- Classification (i.e. discrete labels like dog v.s. cat), regression (i.e. continuous variable like energy of a particle), etc.

Unsupervised learning

- Only given data, learn underlying structure
- Partitioning (clustering), principal component analysis, generative models

Reinforcement learning

- May not even have static data! Learn to gain most cumulative reward by interacting with the environment
- Playing a game, facility control, etc.



What/How to learn from data?

Supervised learning

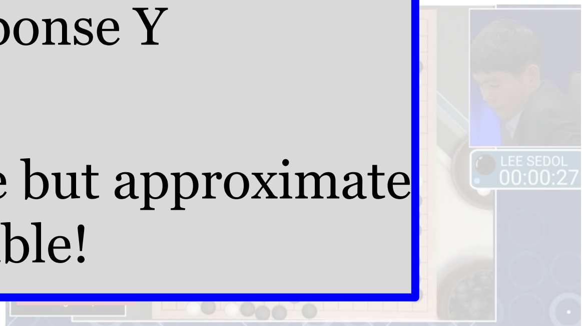
- Given data X and label Y + assume an underlying function P(X)=Y, learn an approximate function Q that mimics P.
- Classification (i.e. discrete labels like dog v.s. cat), regression (i.e. continuous variable like energy of a particle), etc.

Unsuperv

- Only giv
- Partition ... ve models

Reinforce

- May not ... gain mo ... interacti
- Playing a game, facility control, etc.



Machine learning as a function approximation

- A function G data is sampled from
- A function P(X) that provide response Y
- …

These functions may not be accessible but approximate functions may be learnable!

What/How to learn from data?

Supervised learning

- Given data X and label Y + assume an underlying function P(X)=Y, learn an approximate function Q that mimics P.
- Classification (i.e. discrete labels like dog v.s. cat), regression (i.e. continuous variable like energy of a particle), etc.

Unsuperv

- Only giv
- Partition

ve models

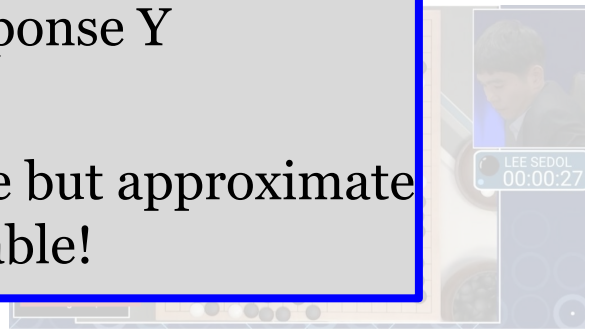
Reinforce

- May not
  gain mo
  interacti
- Playing a game, facility control, etc.

Machine learning as a function approximation

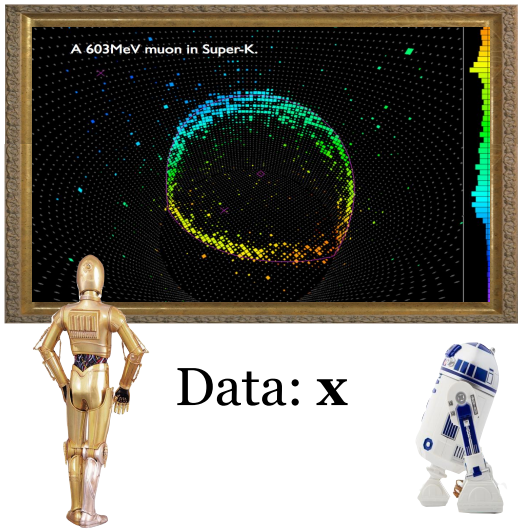
- A function G data is sampled from
- A function P(X) that provide response Y
- …

These functions may not be accessible but approximate functions may be learnable!



How it works: supervised learning

Unknown function P produce the response (y_0, y_1, …) from data (x_0, x_1, …)



Data: **x**

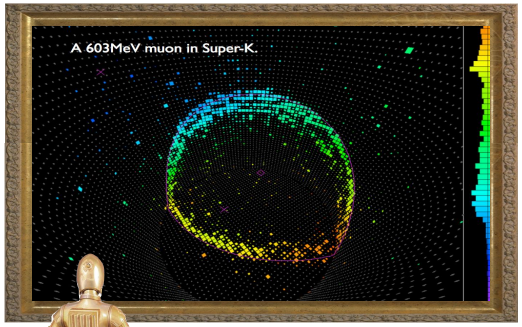
True model: P
(inaccessible)

P(**x**) = {y_0,y_1,…}
(true response)

How it works: supervised learning

Unknown function P produce the response (y_0, y_1, …) from data (x_0, x_1, …)

1. Define a parameterized model Q to mimic the behavior of P.



Data: **x**

True model: P
(inaccessible)

$P(\mathbf{x}) = \{y_0, y_1, \ldots\}$
(true response)

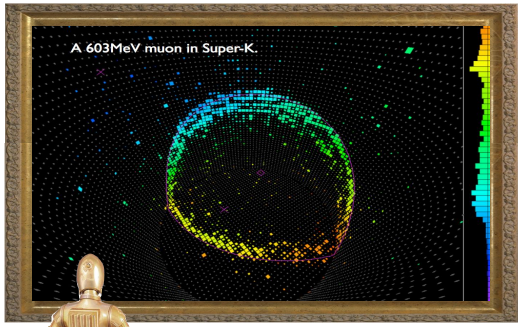
Our model: Q_{θ}

$Q_{\theta}(X)$

How it works: supervised learning

Unknown function P produce the response (y_0, y_1, …) from data (x_0, x_1, …)

1. Define a parameterized model Q to mimic the behavior of P.
2. Tune the parameters of Q to minimize the divergence between P and Q.



Data: **x**

True model: P

$P(\mathbf{x}) = \{y_0, y_1, \ldots\}$
(true response)

min.(div(P,Q))

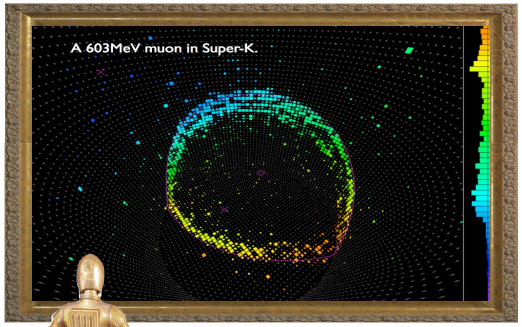
Our model: Q_{θ}

$Q_{\theta}(X)$

How it works: supervised learning

Unknown function P produce the response (y_0, y_1, …) from data (x_0, x_1, …)

1. Define a parameterized model Q to mimic the behavior of P.
2. Tune the parameters of Q to minimize the divergence between P and Q.
   - Define a loss function



Data: **x**

True model: P

Our model: Q_{θ}

L (P(**x**), Q(**x**))

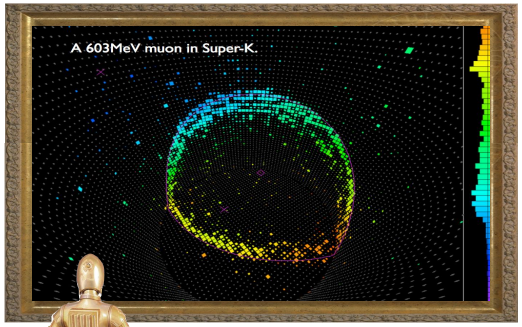
P(**x**) = {y_0,y_1,…}
(true response)

Q_{θ}(X)

How it works: supervised learning

Unknown function P produce the response (y_0, y_1, …) from data (x_0, x_1, …)

1. Define a parameterized model Q to mimic the behavior of P.
2. Tune the parameters of Q to minimize the divergence between P and Q.
   - Define a loss function
   - Minimize the loss



Data: **x**

True model: P

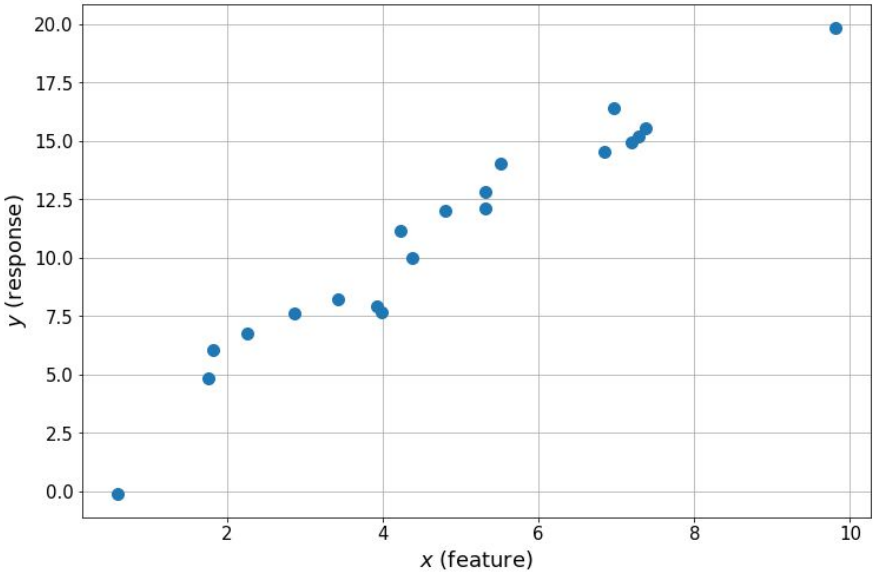
$P(\mathbf{x}) = \{y_0, y_1, \ldots\}$
(true response)

Min {L (P(**x**), Q(**x**))}

Our model: Q_{θ}

$Q_{\theta}(X)$

Example A: linear regression



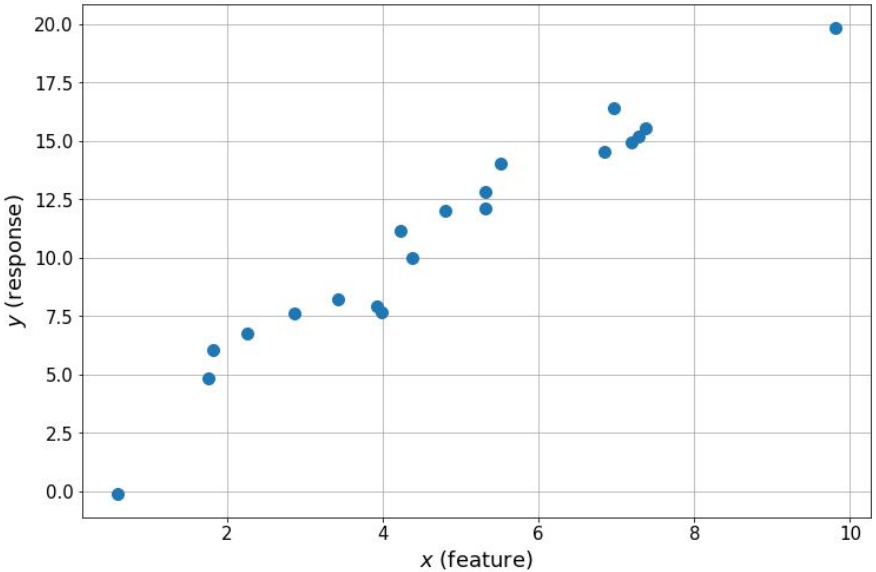
Full disclosure about data (left):

- x sampled from flat distribution [0,10]
- y = a*x + b + **ε**
  - a=2, b=1.5, **ε** ~ Normal (μ=0, σ=1.0)

In general, linear regression employs

$$Q(\mathbf{x}, \mathbf{w}) = w_0 + \sum_i^N w_i \psi_i(\mathbf{x}) = \mathbf{w}^{\mathrm{T}} \cdot \mathbf{\Psi}$$

Example A: linear regression



Full disclosure about data (left):

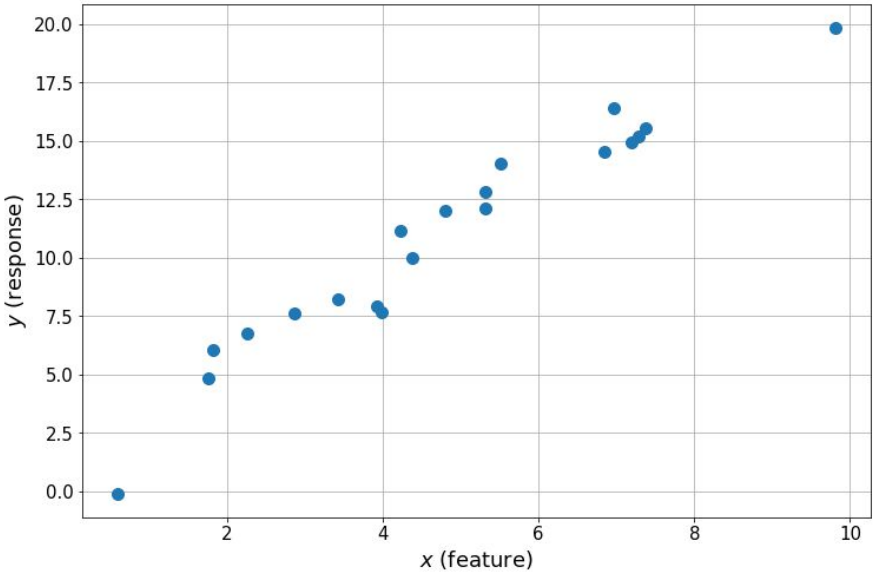
- x sampled from flat distribution [0,10]
- y = a*x + b + **ε**
  - a=2, b=1.5, **ε** ~ Normal (μ=0, σ=1.0)

In general, linear regression employs

$$Q(\mathbf{x}, \mathbf{w}) = w_0 + \sum_i^N w_i \psi_i(\mathbf{x}) = \mathbf{w}^{\mathrm{T}} \cdot \mathbf{\Psi}$$

A simple, intuitive model: $Q(x) = w_0 + w_1 x$

Example A: linear regression



Full disclosure about data (left):

- x sampled from flat distribution [0,10]
- y = a*x + b + **ε**
  - a=2, b=1.5, **ε** ~ Normal (μ=0, σ=1.0)

In general, linear regression employs

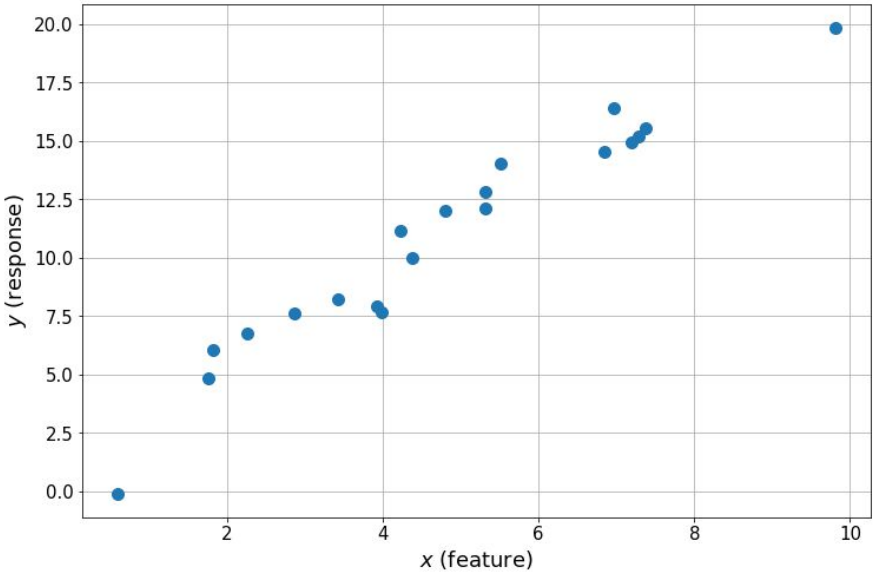
$$Q(\mathbf{x}, \mathbf{w}) = w_0 + \sum_i^N w_i \psi_i(\mathbf{x}) = \mathbf{w}^{\mathrm{T}} \cdot \mathbf{\Psi}$$

A simple, intuitive model: $Q(x) = w_0 + w_1 x$

Popular one: Mean Square Error (MSE) $\mathcal{L} = \frac{1}{m} \sum_{i=1}^{m} (y_i - Q(x_i))^2$

Tune **w** for: $\nabla_{\mathbf{w}} \mathcal{L} = 0$

Example A: linear regression



Full disclosure about data (left):

- x sampled from flat distribution [0,10]
- y = a*x + b + **ε**
  - a=2, b=1.5, **ε** ~ Normal (μ=0, σ=1.0)

In general, linear regression employs

$$Q(\mathbf{x}, \mathbf{w}) = w_0 + \sum_i^N w_i \psi_i(\mathbf{x}) = \mathbf{w}^{\mathrm{T}} \cdot \mathbf{\Psi}$$

A simple, intuitive model: $Q(x) = w_0 + w_1 x$

Popular one: Mean Square Error (MSE) $\mathcal{L} = \frac{1}{m} \sum_{i=1}^{m} (y_i - Q(x_i))^2$

Tune **w** for: $\nabla_{\mathbf{w}} \mathcal{L} = 0$

In this case, the exact solution : $w_0 = \langle y \rangle + w_1 \langle x \rangle$ and $w_1 = \frac{\sum (x_i - \langle x \rangle)(y_i - \langle y \rangle)}{\sum (x_i - \langle x \rangle)^2}$

Example A: linear regression + GD

Goal: tune the parameters **w** and achieve $\nabla_{\mathbf{w}} \mathcal{L} = 0$

Example A: linear regression + GD

Goal: tune the parameters $\mathbf{w}$ and achieve $\nabla_{\mathbf{w}}\mathcal{L} = 0$

Minimize the loss iteratively: $\mathbf{w}_{i+1} = \mathbf{w}_i - \lambda \nabla_{\mathbf{w}} \mathcal{L}(\mathbf{w}, \mathbf{x})$

called **Gradient Descent** (GD) where λ controls the rate of learning process.

Example A: linear regression + GD

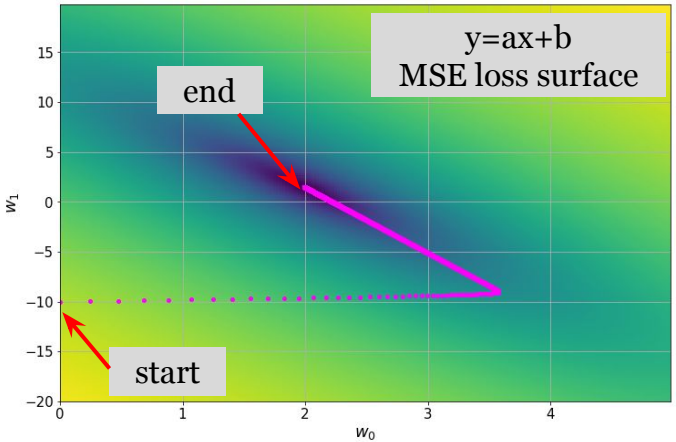
Goal: tune the parameters $\mathbf{w}$ and achieve $\nabla_{\mathbf{w}}\mathcal{L} = 0$

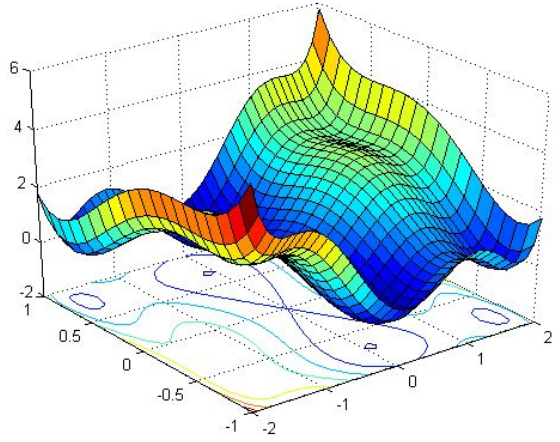
Minimize the loss iteratively: $\mathbf{w}_{i+1} = \mathbf{w}_i - \lambda \nabla_{\mathbf{w}} \mathcal{L}(\mathbf{w}, \mathbf{x})$

called **Gradient Descent** (GD) where λ controls the rate of learning process.

For a smooth convex functions, GD achieves the loss O(1/k) after k steps.

For non-convex functions, there may be multiple optimal points.





Example A: linear regression + GD

Goal: tune the parameters **w** and achieve $\nabla_{\mathbf{w}} \mathcal{L} = 0$ ... **faster**?

Mini-batch Stochastic GD (SGD) use a **subset of data** for gradient.

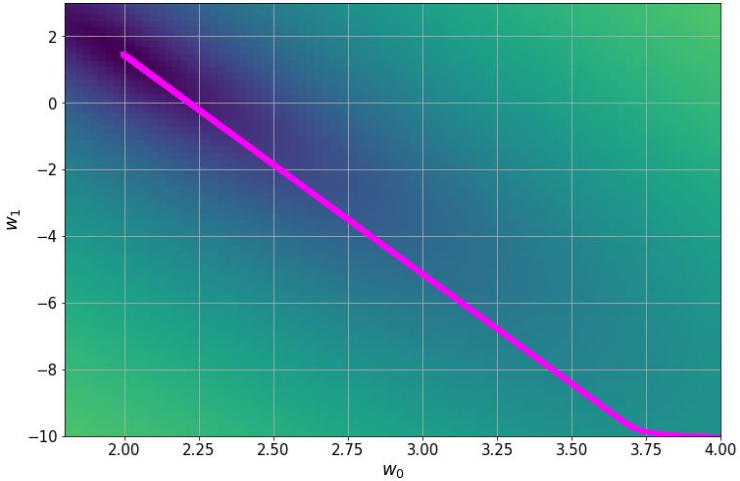
Example A: linear regression + GD

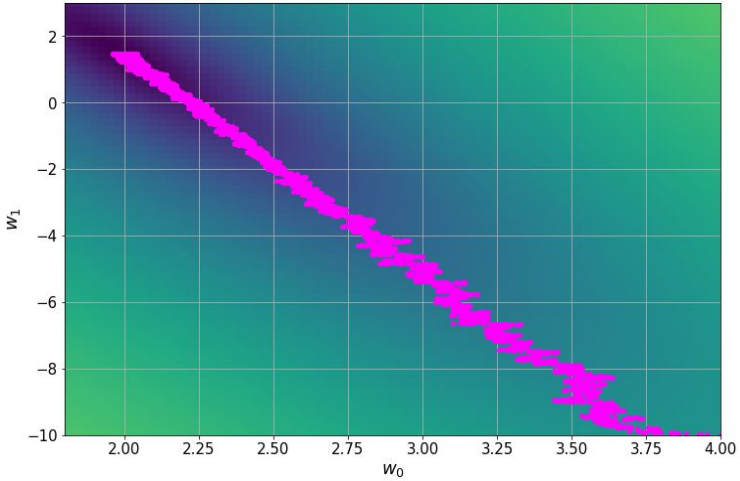
Goal: tune the parameters **w** and achieve $\nabla_{\mathbf{w}}\mathcal{L} = 0$ … **faster**?

Mini-batch Stochastic GD (SGD) use a **subset of data** for gradient.

1. Create a batch = random subset of data.
2. Compute the gradient for the batch and update the parameters.

Note: when data is always new (never seen before), called “online learning”





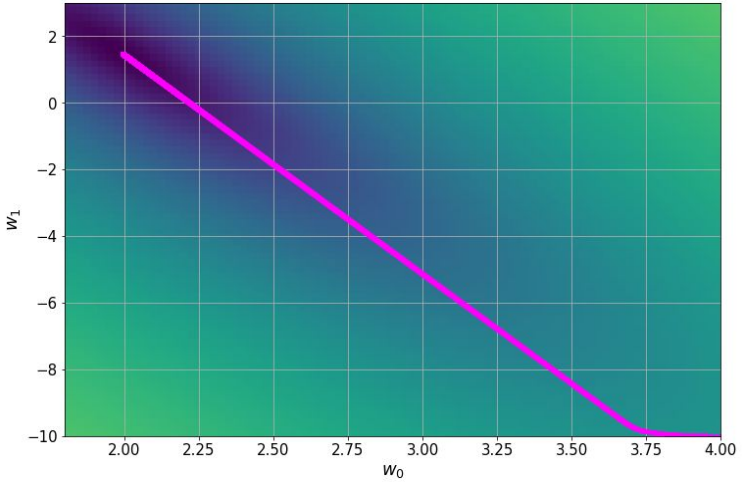
Example A: linear regression + GD

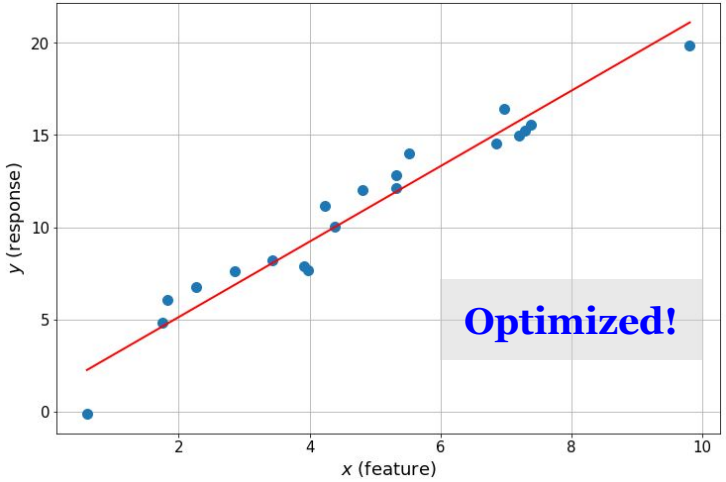
Goal: tune the parameters **w** and achieve $\nabla_{\mathbf{w}}\mathcal{L} = 0$ … **faster**?

Mini-batch Stochastic GD (SGD) use a **subset of data** for gradient.

1. Create a batch = random subset of data.
2. Compute the gradient for the batch and update the parameters.

Note: when data is always new (never seen before), called “online learning”



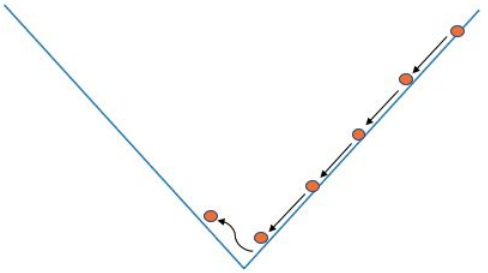


Basic loss functions for regressions

1. **Mean Absolute Error** (MAE, L1 loss)

$$\text{MAE} = \frac{1}{m}\sum_{i=1}^{m}|y_i - Q(\mathbf{x}, \mathbf{w})|$$

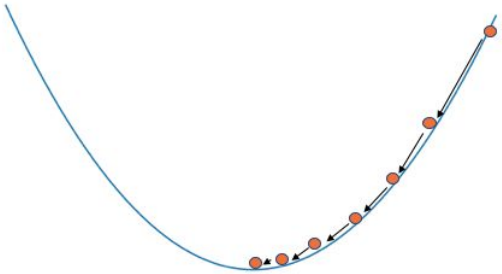
- Could benefit from an adjusted learning rate



2. **Mean Square Error** (MSE, L2 loss)

$$\text{MSE} = \frac{1}{m}\sum_{i=1}^{m}(y_i - Q(x_i))^2$$

- Loss gets small when <1 but may explode when >> 1



3. **Huber Loss** … combine them together

$$\text{Huber} = \begin{cases} \frac{1}{2}a^2 \ldots \text{for } a \le \delta \\ \delta|a| - \frac{1}{2}\delta^2 \ldots \text{otherwise} \end{cases}$$

L1 while loss is large,
L2 when it's small (***δ***: ***hyperparameter***)

(will be back on "hyperparameter" later)

Side-track: “Minimize the divergence of P & Q”

A metric to measure the divergence between P and Q

$$D_{\mathrm{KL}}(P\|Q) = \int P(\mathbf{x}, y) \log \frac{P(\mathbf{x}, y)}{Q_{\mathbf{w}}(\mathbf{x}, y)} d\mathbf{x} dy$$

Kullback-Leibler (KL) Divergence

D_{KL} >= 0 always, and 0 if and only if P and Q are identical

Side-track: “Minimize the divergence of P & Q”

A metric to measure the divergence between P and Q

$$D_{\mathrm{KL}}(P\|Q) = \int P(\mathbf{x}, y) \log \frac{P(\mathbf{x}, y)}{Q_{\mathbf{w}}(\mathbf{x}, y)} d\mathbf{x}dy$$

Kullback-Leibler (KL) Divergence

D_{KL} >= 0 always, and 0 if and only if P and Q are identical

Minimization of D_{KL} using (S)GD can be “tuning of $Q_{\mathbf{w}}$”

$$\mathbf{w}_{i+1} = \mathbf{w}_i - \lambda \nabla_{\mathbf{w}} D_{\mathrm{KL}}(P\|Q)$$

Side-track: “Minimize the divergence of P & Q”

A metric to measure the divergence between P and Q

$$D_{\mathrm{KL}}(P\|Q) = \int P(\mathbf{x}, y) \log \frac{P(\mathbf{x}, y)}{Q_{\mathbf{w}}(\mathbf{x}, y)} d\mathbf{x}dy$$

Kullback-Leibler (KL) Divergence

D_{KL} >= 0 always, and 0 if and only if P and Q are identical

Minimization of D_{KL} using (S)GD can be “tuning of $Q_{\mathbf{w}}$”

$$\mathbf{w}_{i+1} = \mathbf{w}_i - \lambda \nabla_{\mathbf{w}} D_{\mathrm{KL}}(P\|Q) \Rightarrow \int P(\mathbf{x}, y) \nabla_{\mathbf{w}} \log \frac{P(\mathbf{x}, y)}{Q_{\mathbf{w}}(\mathbf{x}, y)} d\mathbf{x}dy = \left\langle \nabla_{\mathbf{w}} \log \frac{P(\mathbf{x}, y)}{Q_{\mathbf{w}}(\mathbf{x}, y)} \right\rangle$$

Side-track: "Minimize the divergence of P & Q"

A metric to measure the divergence between P and Q

$$D_{\mathrm{KL}}(P\|Q) = \int P(\mathbf{x}, y) \log \frac{P(\mathbf{x}, y)}{Q_{\mathbf{w}}(\mathbf{x}, y)} d\mathbf{x}dy$$

Kullback-Leibler (KL) Divergence

D_{KL} >= 0 always, and 0 if and only if P and Q are identical

Minimization of D_{KL} using (S)GD can be "tuning of $Q_{\mathbf{w}}$"

$$\mathbf{w}_{i+1} = \mathbf{w}_i - \lambda \nabla_{\mathbf{w}} D_{\mathrm{KL}}(P\|Q) \Rightarrow \int P(\mathbf{x}, y) \nabla_{\mathbf{w}} \log \frac{P(\mathbf{x}, y)}{Q_{\mathbf{w}}(\mathbf{x}, y)} d\mathbf{x}dy = \left\langle \nabla_{\mathbf{w}} \log \frac{P(\mathbf{x}, y)}{Q_{\mathbf{w}}(\mathbf{x}, y)} \right\rangle$$

Can compute the expectation value using m data instances

$$\left\langle \nabla_{\mathbf{w}} \log \frac{P(\mathbf{x}, y)}{Q_{\mathbf{w}}(\mathbf{x}, y)} \right\rangle = \frac{1}{m} \sum_{\mathbf{x}, y} \nabla_{\mathbf{w}} \log \frac{P(\mathbf{x}_i, y_i)}{Q_{\mathbf{w}}(\mathbf{x}_i, y_i)}$$

... P is still an unknown function, but average probability of $P(x_i,y_i)$ might be possible to approximate from many samples

Side-track: MLE and KL-Divergence

Consider a measurement

- Total number of measurements: N
- Observed occurrence N_i for an event type i

We cannot measure underlying probability p_i but can build a model q_i to approximate p_i , and try to find a "good q_i".

How to define + find a "good q_i?"

Side-track: MLE and KL-Divergence

Consider a measurement

- Total number of measurements: N
- Observed occurrence N_i for an event type i

We cannot measure underlying probability p_i but can build a model q_i to approximate p_i, and try to find a "good q_i".

How to define + find a "good q_i?"

Maximum Likelihood Estimation (MLE)

- Maybe: "good" if q_i has a high likelihood to yield the observed data

Side-track: MLE and KL-Divergence

Consider a measurement

- Total number of measurements: N
- Observed occurrence N_i for an event type i

We cannot measure underlying probability p_i but can build a model q_i to approximate p_i, and try to find a "good q_i".

How to define + find a "good q_i?"

Maximum Likelihood Estimation (MLE)

- Maybe: "good" if q_i has a high likelihood to yield the observed data
- Probability to observe data x: $P(\vec{x}|\vec{q}) = q_0^{N_0} \cdot q_1^{N_1} \cdots q_k^{N_k} \times \frac{N!}{N_0! \cdot N_1! \cdots N_k!}$
  - When N and N_i are large ... $N_i \approx N \cdot p_i$ and $N_i! \approx N_i^{N_i}$
  - ... which yields $P(\vec{x}|\vec{q}) \approx \exp\left[-\sum_i^{\text{all}} p_i \log \frac{p_i}{q_i}\right]$

Maximizing this likelihood means to take the limit of the exponent becoming zero, which is same as minimizing D_{KL}

Side-track: MLE and KL-Divergence

Consider a measurement

- Total number of measurements: N
- Observed occurrence N_i for an event type i

We cannot measure underlying probability p_i but can build a model q_i to approximate p_i, and try to find a "good q_i".

How to define + find a "good q_i?"

Maximum Likelihood Estimation (MLE)

- Maybe: "good" if q_i has a high likelihood to yield the observed data
- Probability to observe data x: $P(\vec{x}|\vec{q}) = q_0^{N_0} \cdot q_1^{N_1} \cdots q_k^{N_k} \times \frac{N!}{N_0! \cdot N_1! \cdots N_k!}$
  - When N and N_i are large ... $N_i \approx N \cdot p_i \quad \text{and} \quad N_i! \approx N_i^{N_i}$

This is a crude, partial credit argument (or zero credit if you got a lucky prof.)

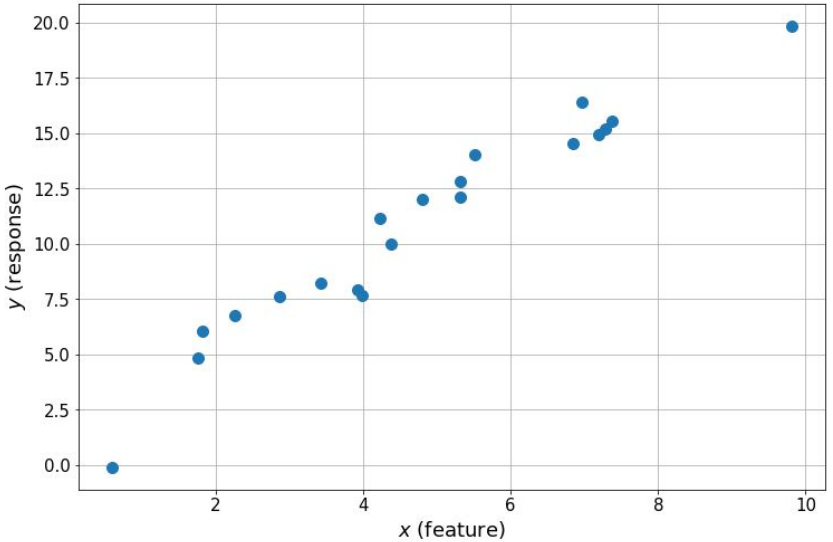
  - ... which yields $P(\vec{x}|\vec{q}) \approx \exp\left[-\sum_i^{\text{all}} p_i \log \frac{p_i}{q_i}\right]$

Maximizing this likelihood means to take the limit of the exponent becoming zero, which is same as minimizing D_{KL}

Back to example: MLE and MSE

Recall our data

- x sampled from flat distribution [0,10]
- y = a*x + b + **ε**
  - a=2, b=1.5, **ε** ~ Normal (μ=0, σ=1.0)



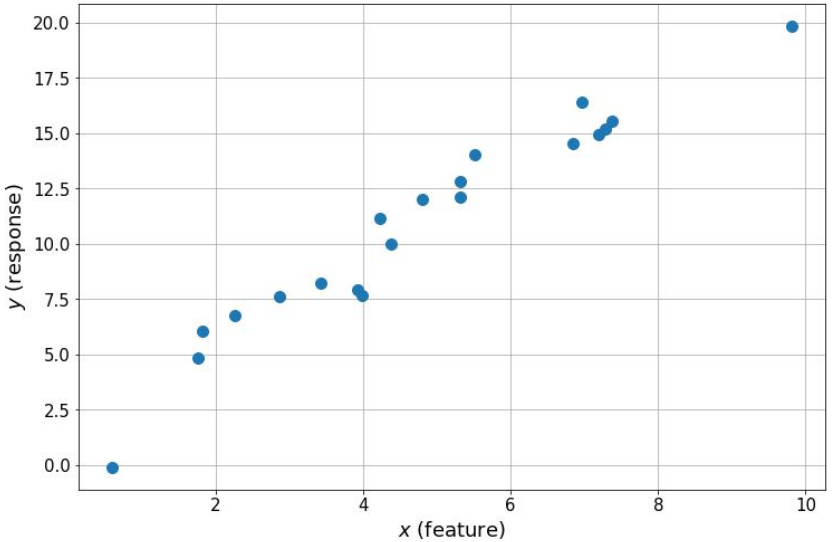
Back to example: MLE and MSE

Recall our data

- x sampled from flat distribution [0,10]
- y = a*x + b + **ε**
  - a=2, b=1.5, **ε** ~ Normal (μ=0, σ=1.0)

… so equivalently we can consider that

$$P(y_i|x_i) \propto \exp\left[-\frac{(y_i-(ax_i+b))^2}{2\sigma^2}\right]$$



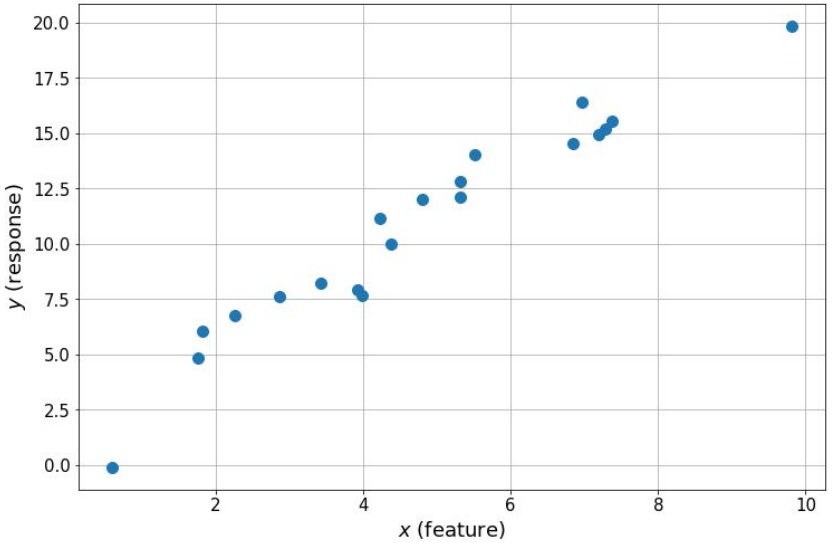
Back to example: MLE and MSE

Recall our data

- x sampled from flat distribution [0,10]
- y = a*x + b + **ε**
  - a=2, b=1.5, **ε** ~ Normal (μ=0, σ=1.0)

… so equivalently we can consider that

$$P(y_i|x_i) \propto \exp\left[-\frac{(y_i-(ax_i+b))^2}{2\sigma^2}\right]$$



The likelihood is a product of all data instances

$$\text{Likelihood } \mathcal{L} = \prod_i P(y_i|x_i)$$

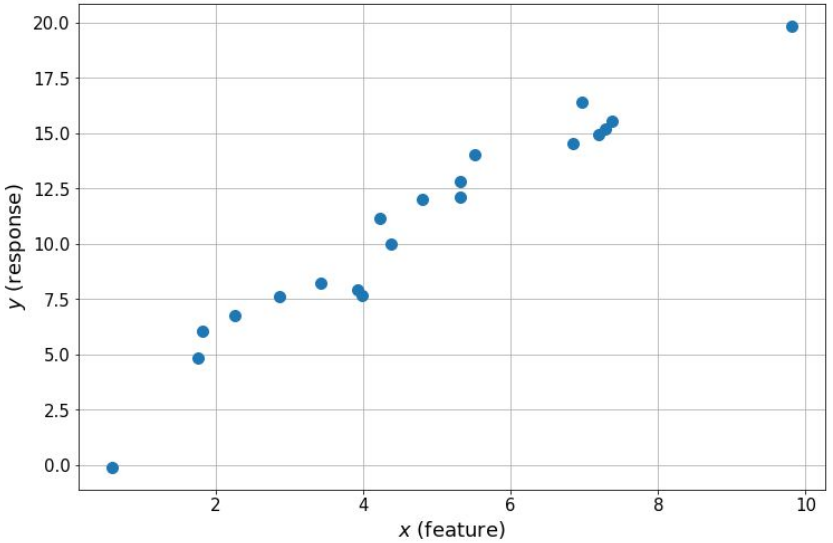
Back to example: MLE and MSE

Recall our data

- x sampled from flat distribution [0,10]
- y = a*x + b + **ε**
  - a=2, b=1.5, **ε** ~ Normal (μ=0, σ=1.0)

… so equivalently we can consider that

$$P(y_i|x_i) \propto \exp\left[-\frac{(y_i-(ax_i+b))^2}{2\sigma^2}\right]$$



The likelihood is a product of all data instances

$$\text{Likelihood } \mathcal{L} = \prod_i P(y_i|x_i) \Rightarrow -\log\mathcal{L} \propto \sum_i (y_i-(ax_i+b))^2 \ldots \text{(MSE)}$$

So minimizing MSE was same as maximum likelihood estimate **in this case**.

Example B: multivariate linear regression

Nothing special. We can just generalize for m regression targets…

$$\text{MSE} = \sum_{i}^{n} \left(y_i - \sum_{j}^{m} w_j x_{ji} \right)^2 = \|\mathbf{y} - \mathbf{X} \cdot \mathbf{w}\|^2$$

Example B: multivariate linear regression

Nothing special. We can just generalize for m regression targets...

$$\text{MSE} = \sum_{i}^{n} \left(y_i - \sum_{j}^{m} w_j x_{ji} \right)^2 = \|\mathbf{y} - \mathbf{X} \cdot \mathbf{w}\|^2$$

n x m matrix

n x 1 vector

m x 1 vector

Example B: multivariate linear regression

Nothing special. We can just generalize for m regression targets…

$$\text{MSE} = \sum_{i}^{n} \left(y_i - \sum_{j}^{m} w_j x_{ji} \right)^2 = \|\mathbf{y} - \mathbf{X} \cdot \mathbf{w}\|^2$$

n x m matrix

n x 1 vector

m x 1 vector

Taking a gradient and solve for minimization:

$$\nabla_{\mathbf{w}} \text{MSE} = 2(\mathbf{y} - \mathbf{w} \cdot \mathbf{X})\mathbf{X} \Rightarrow \boxed{\mathbf{w} = \left(\mathbf{X}^T \mathbf{X}\right)^{-1} \mathbf{X}^T \mathbf{y}}$$

Example B: multivariate linear regression

Nothing special. We can just generalize for m regression targets…

$$\mathrm{MSE} = \sum_i^n \left(y_i - \sum_j^m w_j x_{ji} \right)^2 = \|\mathbf{y} - \mathbf{X} \cdot \mathbf{w}\|^2$$

n x m matrix

n x 1 vector

m x 1 vector

Taking a gradient and solve for minimization:

$$\nabla_{\mathbf{w}} \mathrm{MSE} = 2(\mathbf{y} - \mathbf{w} \cdot \mathbf{X})\mathbf{X} \Rightarrow \boxed{\mathbf{w} = \left(\mathbf{X}^T \mathbf{X}\right)^{-1} \mathbf{X}^T \mathbf{y}}$$

Recall “linear” means “linear to features”…

My “features” could be *m*-th power of *x* …

Maybe fit 15-th degree polynomial to a sin curve?

Example B: multivariate linear regression

Nothing special. We can just generalize for m regression targets…

$$\text{MSE} = \sum_i^n \left(y_i - \sum_j^m w_j x_{ji} \right)^2 = \|\mathbf{y} - \mathbf{X} \cdot \mathbf{w}\|^2$$

n x m matrix

n x 1 vector

m x 1 vector

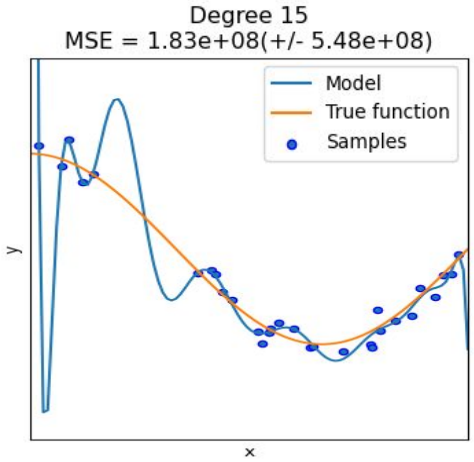
Taking a gradient and solve for minimization:

$$\nabla_{\mathbf{w}} \text{MSE} = 2(\mathbf{y} - \mathbf{w} \cdot \mathbf{X})\mathbf{X} \Rightarrow \boxed{\mathbf{w} = \left(\mathbf{X}^T\mathbf{X}\right)^{-1}\mathbf{X}^T\mathbf{y}}$$

Recall “linear” means “linear to features”…

My “features” could be *m*-th power of *x* …

Maybe fit 15-th degree polynomial to a sin curve?



From scikit webpage

Example B: multivariate linear regression

Hyperparameter: design constants of algorithms (e.g. learning rate)

Overfit: a model has "memorized data". Works well on train data but poorly on independent samples. Typical for complex model + low data statistics

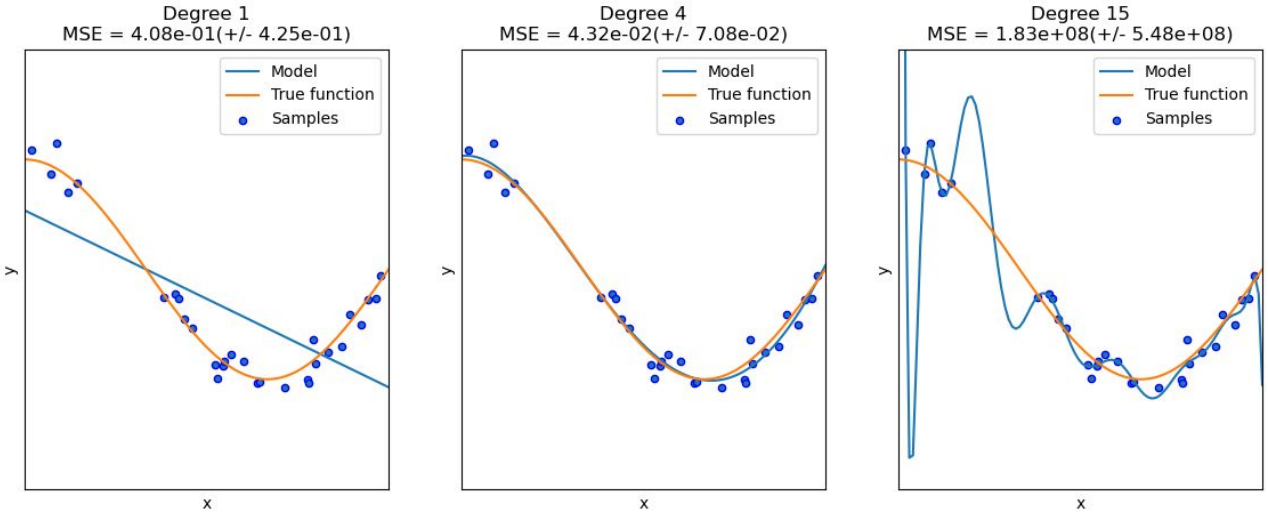
Generalization: model works well equally on the train and unseen datasets

Example B: multivariate linear regression

Hyperparameter: design constants of algorithms (e.g. learning rate)

Overfit: a model has "memorized data". Works well on train data but poorly on independent samples. Typical for complex model + low data statistics

Generalization: model works well equally on the train and unseen datasets



From scikit webpage

In this case…

A polynomial of a higher power (hyperparameter!) makes the model more complex, or flexible, and as a result a model can overfit.

Cross-Validation

Two datasets

- **Train set** to optimize your model
- **Test set** to measure performance

… we want to use the test set only once at the end, never want to tune anything on it!

Cross-Validation

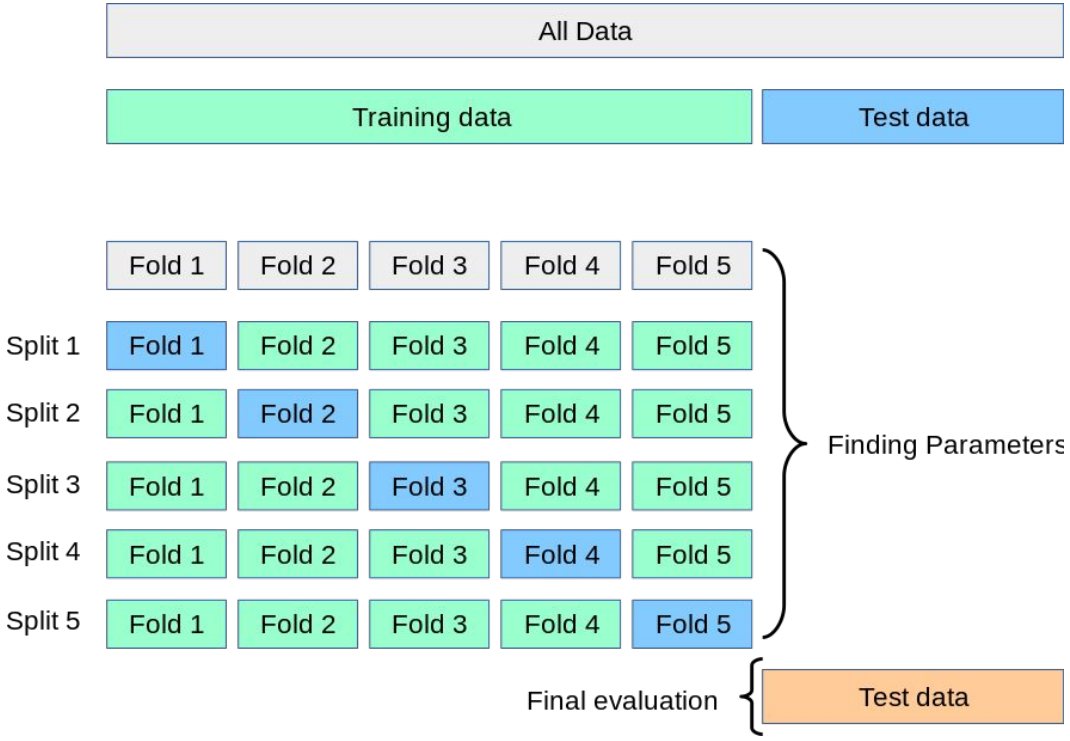
Two datasets

- **Train set** to optimize your model
- **Test set** to measure performance

... we want to use the test set only once at the end, never want to tune anything on it!

Strategy:

- Split the train set into k-folds
- Use *i*-th set as a "**validation set**" to measure the performance of a model trained on the rest (k-1 combined).
- Repeat *k*-times and take the mean as a performance.



From scikit webpage

Regularization

There may be more than one solution (i.e. local minimum in GD)… prefer a *simpler* solution over complicated ones (=may help generalization).

$$\mathcal{L}_{\text{total}} = \underbrace{\mathcal{L}\left(\mathbf{y}, Q(\mathbf{x}, \mathbf{w})\right)}_{\text{model loss}} + \underbrace{\alpha R(\mathbf{w})}_{\text{regularization loss}}$$

Regularization

There may be more than one solution (i.e. local minimum in GD)… prefer a *simpler* solution over complicated ones (=may help generalization).

$$\mathcal{L}_{\text{total}} = \underbrace{\mathcal{L}\left(\mathbf{y}, Q(\mathbf{x}, \mathbf{w})\right)}_{\text{model loss}} + \underbrace{\alpha R(\mathbf{w})}_{\text{regularization loss}}$$

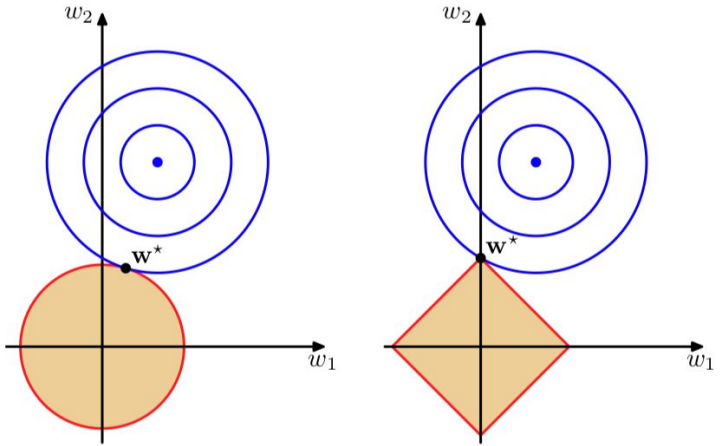
Basic regularization terms are L1, L2, and L1+L2

$$L2 = \|\mathbf{w}\|^2 = \sum_i w_i^2$$

L2 favors weights with smaller values

$$L1 = \|\mathbf{w}\| = \sum_i |w_i|$$

L1 favors sparse solution (also called LASSO)



Regularization

There may be more than one solution (i.e. local minimum in GD)… prefer a *simpler* solution over complicated ones (=may help generalization).

$$\mathcal{L}_{\text{total}} = \underbrace{\mathcal{L}\left(\mathbf{y}, Q(\mathbf{x}, \mathbf{w})\right)}_{\text{model loss}} + \underbrace{\alpha R(\mathbf{w})}_{\text{regularization loss}}$$

Basic regularization terms are L1, L2, and L1+L2

$$L2 = \|\mathbf{w}\|^2 = \sum_i w_i^2$$

L2 favors weights with smaller values

$$L1 = \|\mathbf{w}\| = \sum_i |w_i|$$

L1 favors sparse solution (also called LASSO)

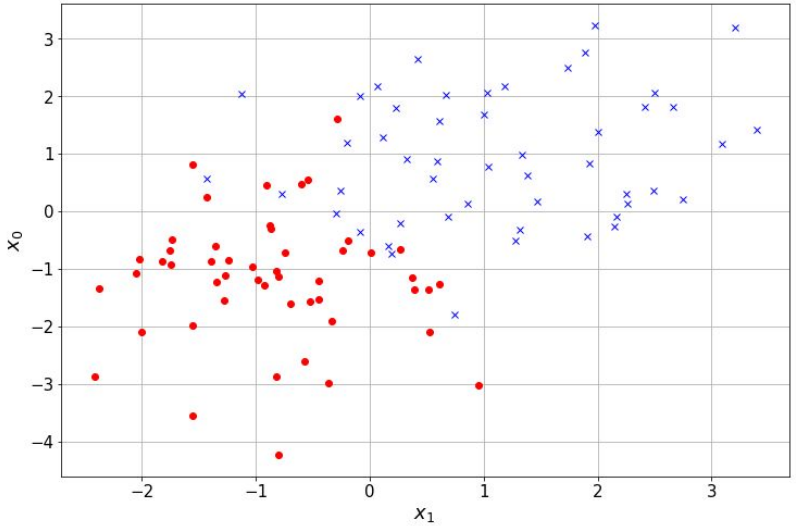
$$L1 + L2 = \sum_i \left(\beta w_i^2 + |w_i|\right)$$

Can also combine (“elastic net”)

Intermediate summary

- "Learning" is a process of tuning a model Q(x) to make a better approximation of inaccessible, underlying function P(x)
  - Define a model and a loss function, use Gradient Descent (GD) to tune the parameters
  - Stochastic GD (SGD) uses a random subset of train data per parameter update
  - Minimization of D_{KL}, maximization of likelihood (MLE)
- Linear regression = linear transformation of features (i.e. linear to model parameters)
  - Exact solution is available and can be compared to the optimization outcome
  - Popular loss functions = Mean Squared Error (MSE), Mean Absolute Error (MAE) ...
- Train set = dataset used to optimize Q(x)
- Test set = dataset used to benchmark the performance of Q(x)
- Generalization = model performs on the test dataset as well as the train dataset
- Overfit = *memorization* of data, a model performs well on train set but poorly on test set
- Cross validation = splitting the train set to k-folds to create validation set(s) and measure model performance and/or tune hyperparameters
- Regularization = additional constraints on model parameters, can help avoiding overfitting
  - L2 prefers smaller weight values, L1 may lead to a sparse solution

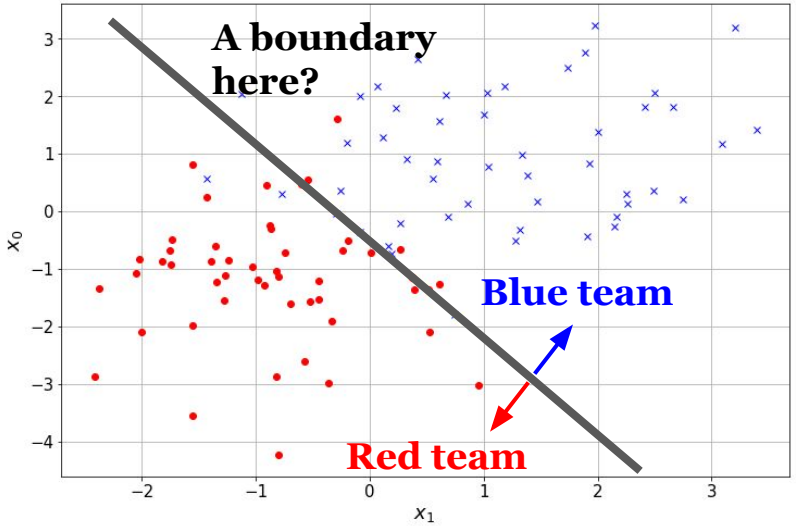
Example C: logistic regression for classification



Full disclosure about data (left):

- Red data points ~ Normal (μ=(-1,-1), σ=(1,1))
- Blue data points ~ Normal (μ=(1,1), σ=(1,1))

Example C: logistic regression for classification



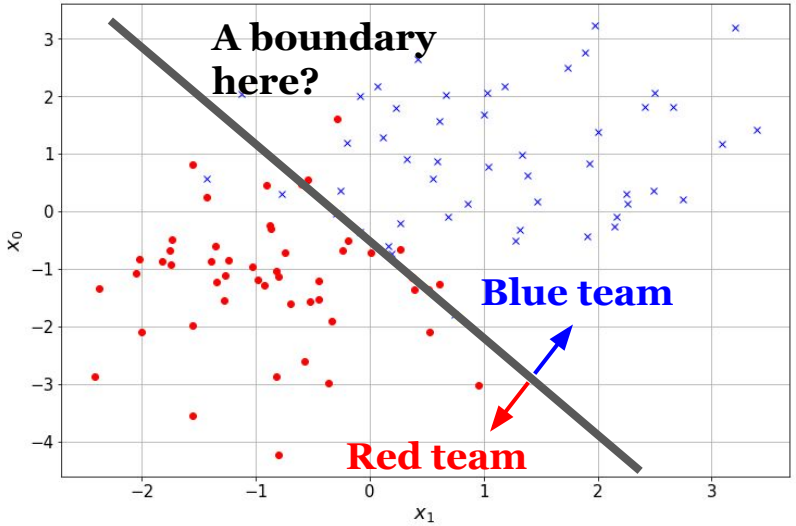
Full disclosure about data (left):

- Red data points ~ Normal (μ=(-1,-1), σ=(1,1))
- Blue data points ~ Normal (μ=(1,1), σ=(1,1))

What should be Q and loss function?

- A straight line to separate two colors
- Classification confidence low on and near the line, more confident away from the line

Example C: logistic regression for classification



Full disclosure about data (left):

- Red data points ~ Normal (μ=(-1,-1), σ=(1,1))
- Blue data points ~ Normal (μ=(1,1), σ=(1,1))

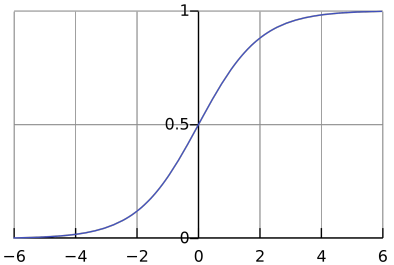
What should be Q and loss function?

- A straight line to separate two colors
- Classification confidence low on and near the line, more confident away from the line

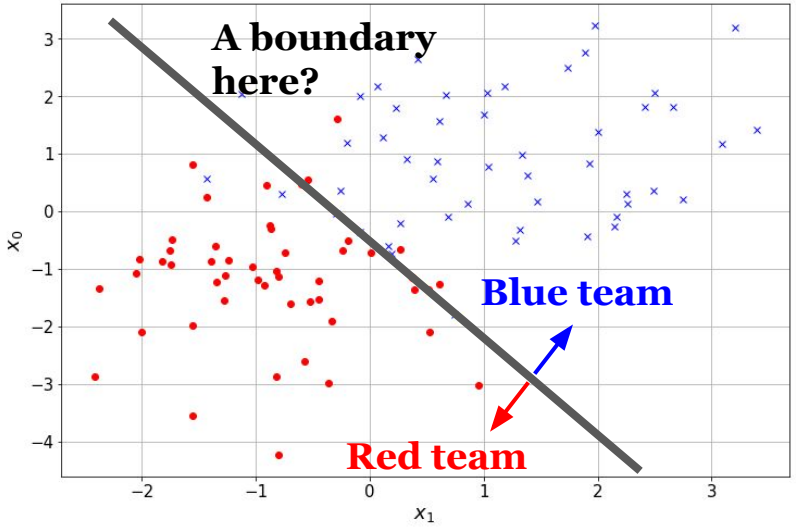
$$Q(\mathbf{x}, \mathbf{w}) = \frac{1}{1 + \exp(-\mathbf{w}^T \cdot \mathbf{x})}$$

Popular one: logistic (a.k.a. “sigmoid”) function

- Satisfies what we want + the output is bounded [0,1] (probabilistic)



Example C: logistic regression for classification



Full disclosure about data (left):

- Red data points ~ Normal (μ=(-1,-1), σ=(1,1))
- Blue data points ~ Normal (μ=(1,1), σ=(1,1))

What should be Q and loss function?

- A straight line to separate two colors
- Classification confidence low on and near the line, more confident away from the line

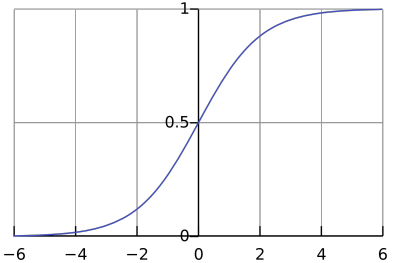
$$Q(\mathbf{x}, \mathbf{w}) = \frac{1}{1 + \exp(-\mathbf{w}^T \cdot \mathbf{x})}$$

Popular one: logistic (a.k.a. "sigmoid") function

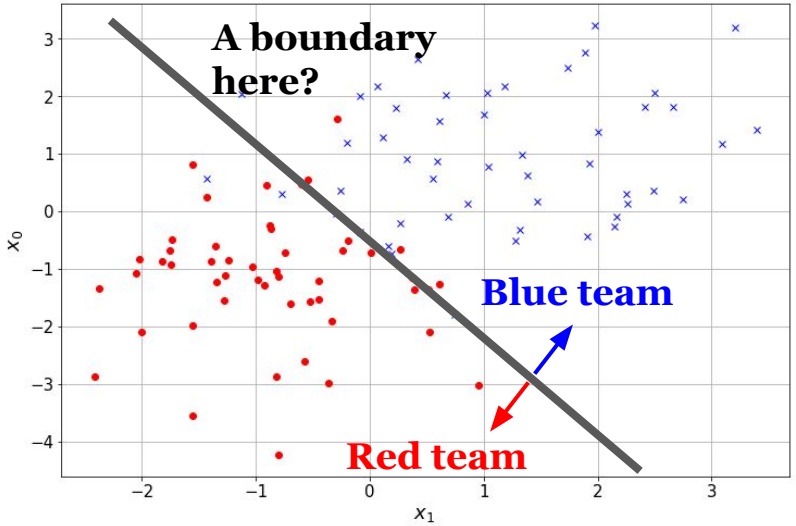
- Satisfies what we want + the output is bounded [0,1] (probabilistic)

$$\text{Likelihood} = \prod_i^m (y_i Q(\mathbf{x}_i, \mathbf{w}) + (1 - y_i)(1 - Q(\mathbf{x}_i, \mathbf{w}))$$

y_i = 0 if red, 1 if blue



Example C: logistic regression for classification



Full disclosure about data (left):

- Red data points ~ Normal (μ=(-1,-1), σ=(1,1))
- Blue data points ~ Normal (μ=(1,1), σ=(1,1))

What should be Q and loss function?

- A straight line to separate two colors
- Classification confidence low on and near the line, more confident away from the line

$$Q(\mathbf{x}, \mathbf{w}) = \frac{1}{1 + \exp(-\mathbf{w}^T \cdot \mathbf{x})}$$

Popular one: logistic (a.k.a. "sigmoid") function

- Satisfies what we want + the output is bounded [0,1] (probabilistic)

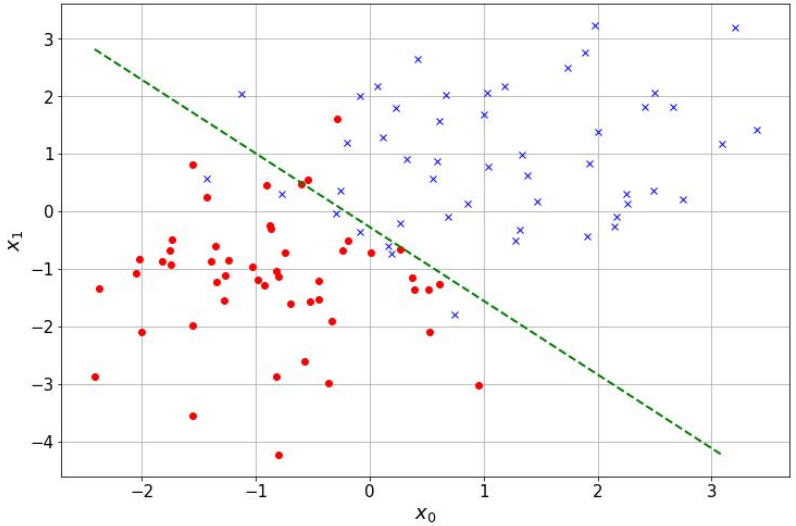
$$\mathcal{L} = -\sum_i^m [y_i \log(Q(\mathbf{x}_i, \mathbf{w})) + (1 - y_i) \log(1 - Q(\mathbf{x}_i, \mathbf{w}))]$$

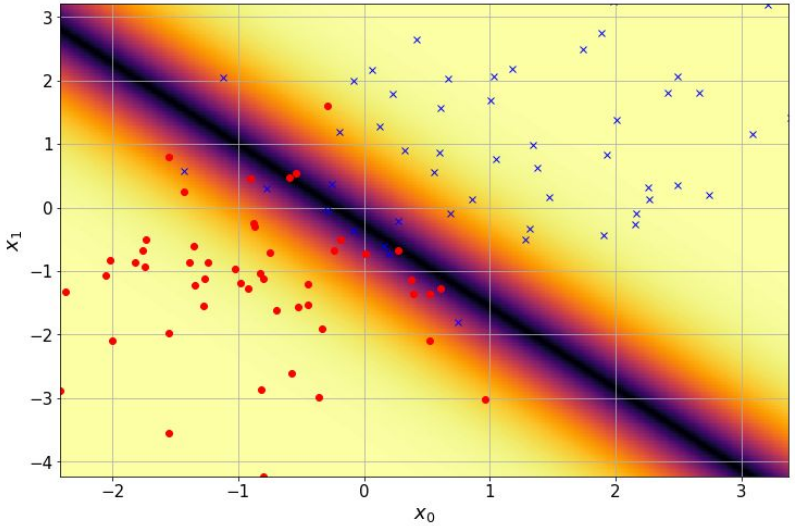
Minimization target = *Loss* = negative log likelihood

$$\frac{\partial \mathcal{L}}{\partial \mathbf{w}} = \frac{1}{m} \mathbf{x}^T (Q(\mathbf{x}, \mathbf{w}) - \mathbf{y})$$

For gradient update!

Example C: logistic regression for classification





Generalization to multiple targets (>2 categorization classes)

- Softmax function for Q + MLE yields cross-entropy loss function

$$Q_i(\mathbf{x}, \mathbf{w}_i) = \frac{\exp(-\mathbf{w}_i^T \cdot \mathbf{x})}{\sum_j \exp(-\mathbf{w}_j^T \cdot \mathbf{x})}$$

Q_i is a probability for being i-th class. Need a set of $\mathbf{w}$ per class!

$$\mathcal{L} = -\sum_i y_i \log Q_i(\mathbf{x}_i, \mathbf{w}_i)$$

$y_i = 1$ only for the correct class

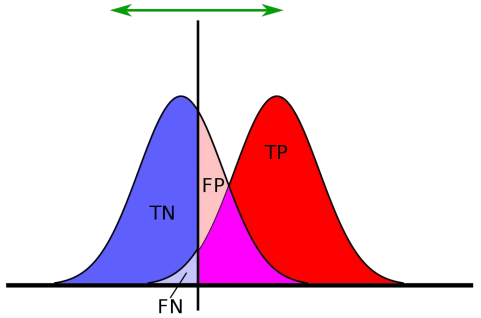
Classification performance metrics

How should we compare the classification performance that depends on an interpretation of score (i.e. Q)? This may be application specific, but a standard procedure exists with useful jargons :)

Classification performance metrics

How should we compare the classification performance that depends on an interpretation of score (i.e. Q)? This may be application specific, but a standard procedure exists with useful jargons :)

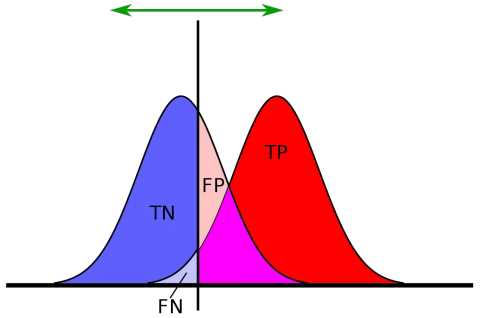
	Label P=1	Label P=0
Prediction Q=1	True Positive (TP)	False Positive (FP)
Prediction Q=0	False Negative (FN)	True Negative (TN)



Classification performance metrics

How should we compare the classification performance that depends on an interpretation of score (i.e. Q)? This may be application specific, but a standard procedure exists with useful jargons :)

	Label P=1	Label P=0
Prediction Q=1	True Positive (TP)	False Positive (FP)
Prediction Q=0	False Negative (FN)	True Negative (TN)



$$\text{Accuracy} = \frac{\text{TP} + \text{TN}}{\text{TP} + \text{FP} + \text{FN} + \text{TN}}$$

Classification performance metrics

How should we compare the classification performance that depends on an interpretation of score (i.e. Q)? This may be application specific, but a standard procedure exists with useful jargons :)

	Label P=1	Label P=0
Prediction Q=1	True Positive (TP)	False Positive (FP)
Prediction Q=0	False Negative (FN)	True Negative (TN)

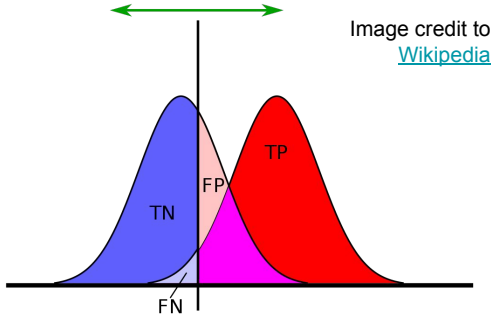


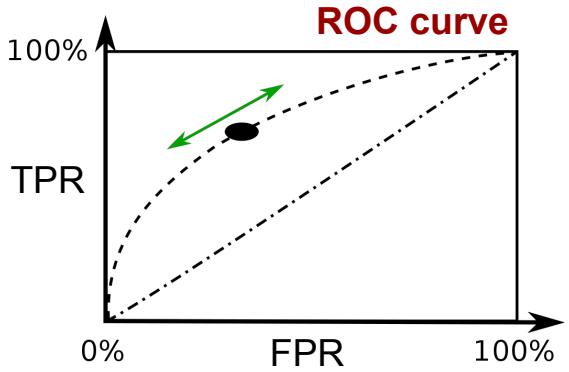
Image credit to Wikipedia

$$\text{Accuracy} = \frac{\text{TP} + \text{TN}}{\text{TP} + \text{FP} + \text{FN} + \text{TN}}$$

$$\text{False Positive Rate (FPR)} = \frac{\text{FP}}{\text{FP} + \text{TN}}$$

$$\text{True Positive Rate (TPR)} = \frac{\text{TP}}{\text{TP} + \text{FN}}$$

We can create a curve of FPR v.s. TPR by varying the score threshold. This is **Receiver Operating Characteristic** (**ROC**) curve. The area under this curve may be used as a performance metric.



Classification performance metrics

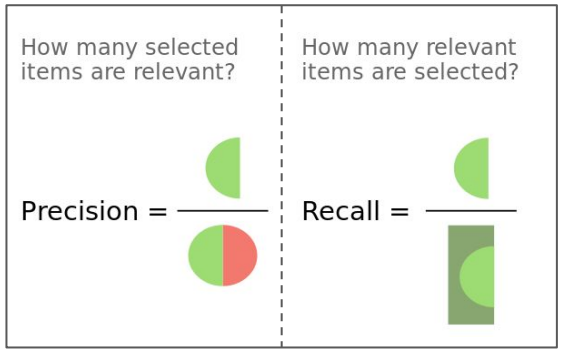
How should we compare the classification performance that depends on an interpretation of score (i.e. Q)? This may be application specific, but a standard procedure exists with useful jargons :)

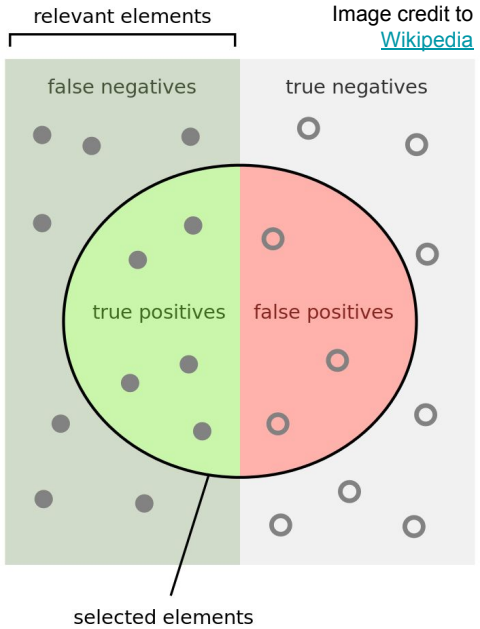
	Label P=1	Label P=0
Prediction Q=1	True Positive (TP)	False Positive (FP)
Prediction Q=0	False Negative (FN)	True Negative (TN)

Precision and Recall are also often used metrics. In physics they are sometimes called "purity" and "efficiency" of the prediction!

$$\text{Precision} = \frac{\text{TP}}{\text{TP} + \text{FP}}$$

$$\text{Recall} = \frac{\text{TP}}{\text{TP} + \text{FN}}$$





More dictionary

- Logistic regression is a task to classify an input into predefined set of classes
  - Logistic loss function gives a probabilistic interpretation of a score for binary classification
  - Generalization to multinomial regression using softmax function and cross-entropy loss.
  - (S)GD can be used to optimize a linear model
- Classification results with labels can be grouped into 4 categories
  - True Positive or TP... (prediction, label) = (1,1)
  - False Positive or FP ... (prediction, label) = (1,0)
  - False Negative or FN ... (prediction, label) = (0,1)
  - True Negative or TN ... (prediction, label) = (0,0)
  - Accuracy = (TP+TN) / ALL
  - False Positive Rate or FPR = FP / (FP+TN)
  - True Positive Rate or TPR = TP / (TP+FN)
  - Precision = TP / (TP+FP)
  - Recall = TP / (TP+FN)
- FPR v.s. TPR = Receiver Operating Characteristic (ROC) curve, the area under the curve is larger for a better classifier
- Precision v.s. Recall curve is another metric that is more robust against class imbalance

Google nagoya food

All Maps Images News Shopping More Settings Tools Collections SafeSearch

miso nikomi udon | nagoya station | halal | aichi | restaurants | japan | aichi prefecture | yamamotoya | miso katsu | spaghetti | noodles | japanese | dishes | cuisine | eat | nagoy

Ultimate Nagoya Guide For F...
favy-jp.com


Food In Nagoya, Japan ...
trip101.com


The Top 5 Must Try Food in Nagoya
blog.gaijinpot.com


Ultimate Nagoya Guide For Food Lovers ...
favy-jp.com


The Top 5 Must Try Food in Nagoya
blog.gaijinpot.com


Nagoya Food - Top 10 dishes for Japan...
eatwanderexplore.com


Must-Eat Foods in Nagoya: ...
centrip-japan.com


Top 9 Famous Foods of Nagoya | Japan Info
jpninfo.com


Nagoya Food: Top 8 Dishes You Have to ...
mariaabroad.com


Nagoya: Home to Japan's most unique ...
cnn.com


Nagoya: Home to Japan's most unique ...
cnn.com


Ultimate Nagoya Guide For Food Lovers...
favy-jp.com


Nagoya Meshi （"Must Eat" Local Foods in ...
japanfoodculture.org


Nagoya Food: 8 Nagoya Meshi Dishes to ...
willflyforfood.net


Nagoya: Home to Japan's most unique ...
cnn.com


big destination for food in Japan
mic.com


Best Restaurants in Nagoya ...
en.compathy.net


The Top 5 Must Try Food in Nagoya
blog.gaijinpot.com


7 Best Nagoya Foods to Try – Trip-N-Tra...
trip-n-travel.com


Must-Eat Foods in Nagoya...
centrip-japan.com

Let's go to eat yummy food to Nagoya.
jpnfood.com


Nagoya Meshi （"Must Eat" Local Foods i...
japanfoodculture.org


Nagoya Popular Dish, Miso-Stewed U...
fooddiversity.today


5 Restaurants Near Na...
wow-j.com


The Top 5 Must Try Food in Nagoya
blog.gaijinpot.com


Nagoya Food Guide - 20 Local Foods in ...
nagoyafoodie.com

Nagoya: Home to Japan's most unique ...
cnn.com

... this is interesting ...

Next up

Played with linear models today.

What if we convolve multiple linear transformations?

Next topic: Neural Networks!

Scratch Slides

Core idea

- Consider 2 statements … **which one is information rich?**
  - A someone taught gorilla a linear algebra
  - A gorilla taught someone a linear algebra
  - … a surprising event contains more information
- **Information conte** $-\log P(A)$
  - Additive: information from event A + B: $-\log P(A) \cdot P(B) = -\left[\log P(A) + \log P(B)\right]$
  - e.g.) “I picked 13 of diamond from the stack”: $-\log\left(\frac{1}{13} \cdot \frac{1}{4}\right) = \log 52$
- **Total expected information content**
  - **Entropy:** S = $-\sum_{i}^{\text{all}} P_i \log P_i$

Core idea

- Consider a measurement
  - Total number of measurements: N
  - Observed occurrence N_i for an event type i

We cannot measure underlying probability p_i but can build a model q_i to approximate p_i, and say the task for machine learning is to learn a "good q_i".

How to define + find a "good q_i?"

- Maybe: "good" if q_i has a high likelihood to yield the observed data
- Probability to observe data x: $P(\vec{x}|\vec{q}) = q_0^{N_0} \cdot q_1^{N_1} \cdots q_k^{N_k} \times \frac{N!}{N_0! \cdot N_1! \cdots N_k!}$
  - When N and N_i are large ... $N_i \approx N \cdot p_i$ and $N_i! \approx N_i^{N_i}$
  - ... which yields $P(\vec{x}|\vec{q}) \approx \exp\left[-\sum_i^{\text{all}} p_i \log \frac{p_i}{q_i}\right]$

Maximize the likelihood = minimize the