Universidad Zaragoza

Machine Learning (69152)

Machine Learning Fundamentals

Universidad Zaragoza

Rubén Martínez Cantín
rmcantin@unizar.es, office 0.09, Ada Byron bldg
Dpto. Informática e Ingeniería de Sistemas.

Content

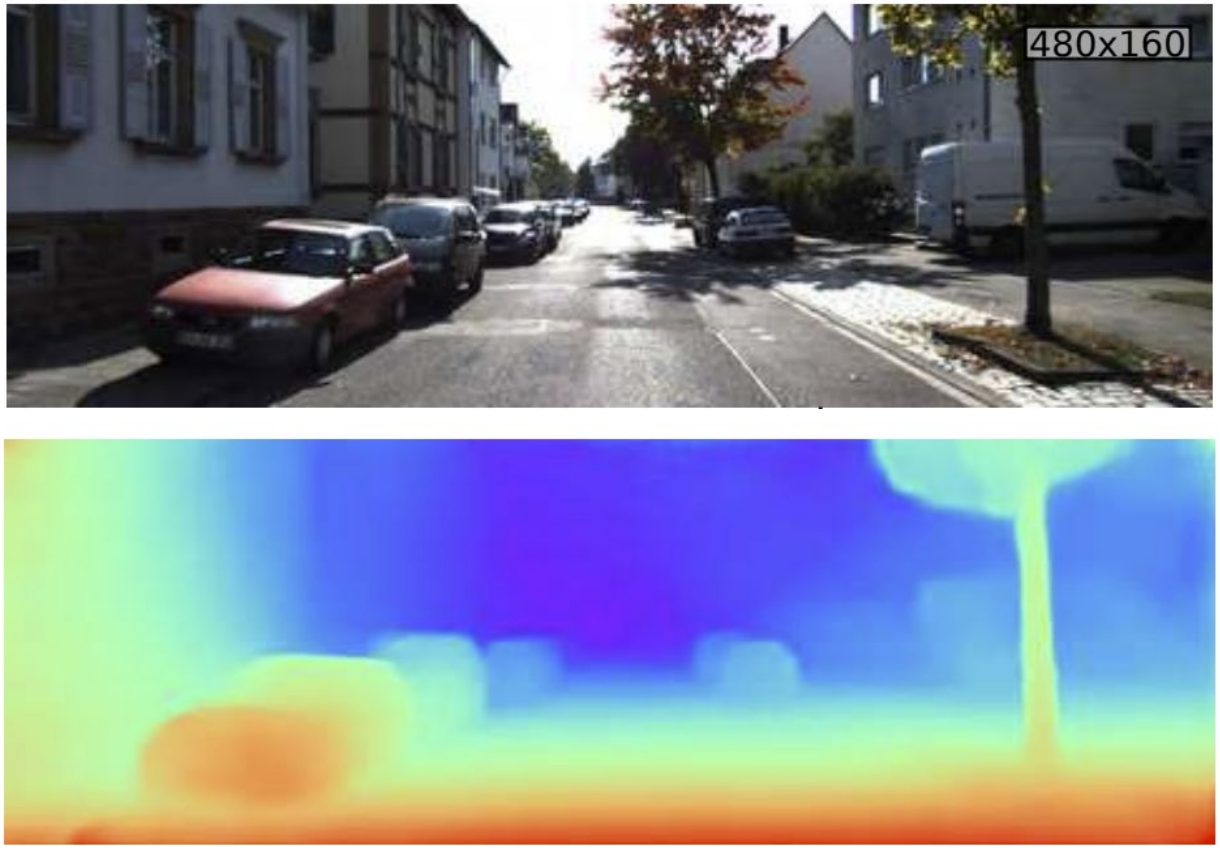
- Introduction
- K-nearest neighbours (as a simple example)
- **Regression**
    - Linear regression
    - Non-linear regression
    - Overfitting, regularization, model selection
    - Ridge regression
    - Robust regression
- **Classification**
    - Logistic regression
    - SVM
- **Unsupervised methods**
    - Dimensionality reduction: PCA
    - Clustering: K-means, GMM
- **Evaluation metrics**

Bibliography

- **Kevin P. Murphy, Probabilistic Machine Learning: An Introduction, The MIT Press, 2022** (available online https://probml.github.io/pml-book/book1.html)
- Marc Peter Deisenroth, A. Aldo Faisal, Cheng Soon Ong, Mathematics for Machine Learning, Cambridge University Press, 2020 (available online https://mml-book.com/)
- Christopher M. Bishop, Pattern Recognition and Machine Learning, Springer, 2006 (available online https://www.microsoft.com/en-us/research/uploads/prod/2006/01/Bishop-Pattern-Recognition-and-Machine-Learning-2006.pdf)
- Aurélien Géron, Hands-On Machine Learning with Scikit-Learn, Keras & TensorFlow, O'Reilly, 2019 (2nd edition)
- Online courses
  - Andrew Ng, Stanford CS229: Machine Learning, https://www.youtube.com/playlist?list=PLoROMvodv4rMiGQp3WXShtMGgzqpfVfbU
  - Ayush Singh and @freecodecamp, ML for beginners, https://www.youtube.com/watch?v=NWONeJKn6kc
- Python resources
  - The Python tutorial: https://docs.python.org/3/tutorial/
  - numpy tutorial: https://cs231n.github.io/python-numpy-tutorial/
  - scikit-learn tutorials: https://scikit-learn.org/stable/tutorial/index.html

Regression

- Learns a function that relates an input vector with a **continuous output**.
- Examples: depth prediction, weather forecasting, time-to-destination estimation…



Linear regression

For more details, check Murphy chapter 11, Deisenroth chapter 9

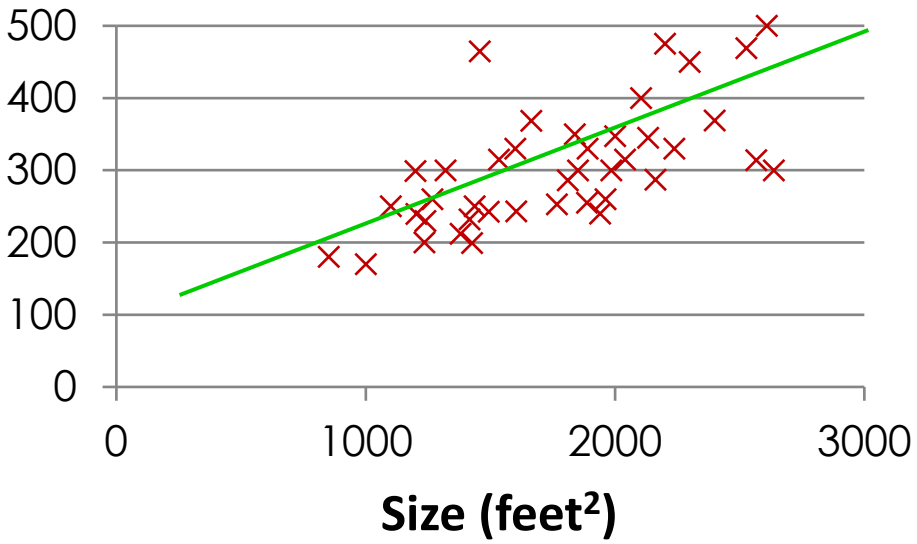
- The input/output relation is linear in the model parameters

$$y = x^T w + \epsilon \quad \epsilon \sim N(0, \sigma^2) \quad \text{one data point}$$

$$\boldsymbol{y} = Xw + \epsilon I_n \quad \epsilon \sim N(0, \sigma^2) \quad \text{dataset (size n)}$$

- Example: Housing prices and size

$y \rightarrow$ output variables



$x \rightarrow$ independent/explanatory variables

Linear regression

- Frequentist approach
  - Assume there is one true model w^*
  - If we had infinite data, we would get the exact model.
  - With finite data, we get an approximation $\hat{w}$.

- How? → By Maximum-likelihood (ML)

$$\hat{w} = \underset{w}{\operatorname{argmax}} \log p(\boldsymbol{y}|X, w)$$

  - If data is independent $X = \{\boldsymbol{x}_1, \dots, \boldsymbol{x}_n\}$ and $\boldsymbol{y} = \{y_1, \dots, y_n\}$

$$\hat{w} = \underset{w}{\operatorname{argmax}} \log \prod_{i=1}^{n} p(y_i|\boldsymbol{x}_i, w) = \underset{w}{\operatorname{argmax}} \sum_{i=1}^{n} \log p(y_i|\boldsymbol{x}_i, w)$$

How do we get w?

- What is the likelihood?

$$p(y_i|\boldsymbol{x}_i, w) = N(y\,|\boldsymbol{x}_i^T w, \sigma^2) \quad \text{one data point}$$

$$p(\boldsymbol{y}|X, w) = N(\boldsymbol{y}\,|\,Xw, \sigma^2 I_n) \quad \text{joint distribution dataset}$$

Ignore all the terms without w

$$\operatorname*{argmax}_w \sum_i \log(p(y_i|\boldsymbol{x}_i, w)) =$$

$$= \operatorname*{argmax}_w \sum_i \log\left(\exp\left(-\frac{1}{2\sigma^2}\left(y_i - \boldsymbol{x}_i^T w\right)^2\right)\right) =$$

$$= \operatorname*{argmin}_w \sum_i \left(y_i - \boldsymbol{x}_i^T w\right)^2 = \operatorname*{argmin}_w \sum_i r_i^2$$

Assume σ^2 is constant

Least squares!

ML = Least squares

How do we get w?

- Minimize Negative Log-Likelihood (NLL).
  - Follow the gradient!

- Rewrite NLL as a function easy to derivate

$$J(w) = \frac{1}{2}(\boldsymbol{y} - Xw)^T(\boldsymbol{y} - Xw)$$

$$\nabla J(w) = X^T Xw - X^T \boldsymbol{y}$$

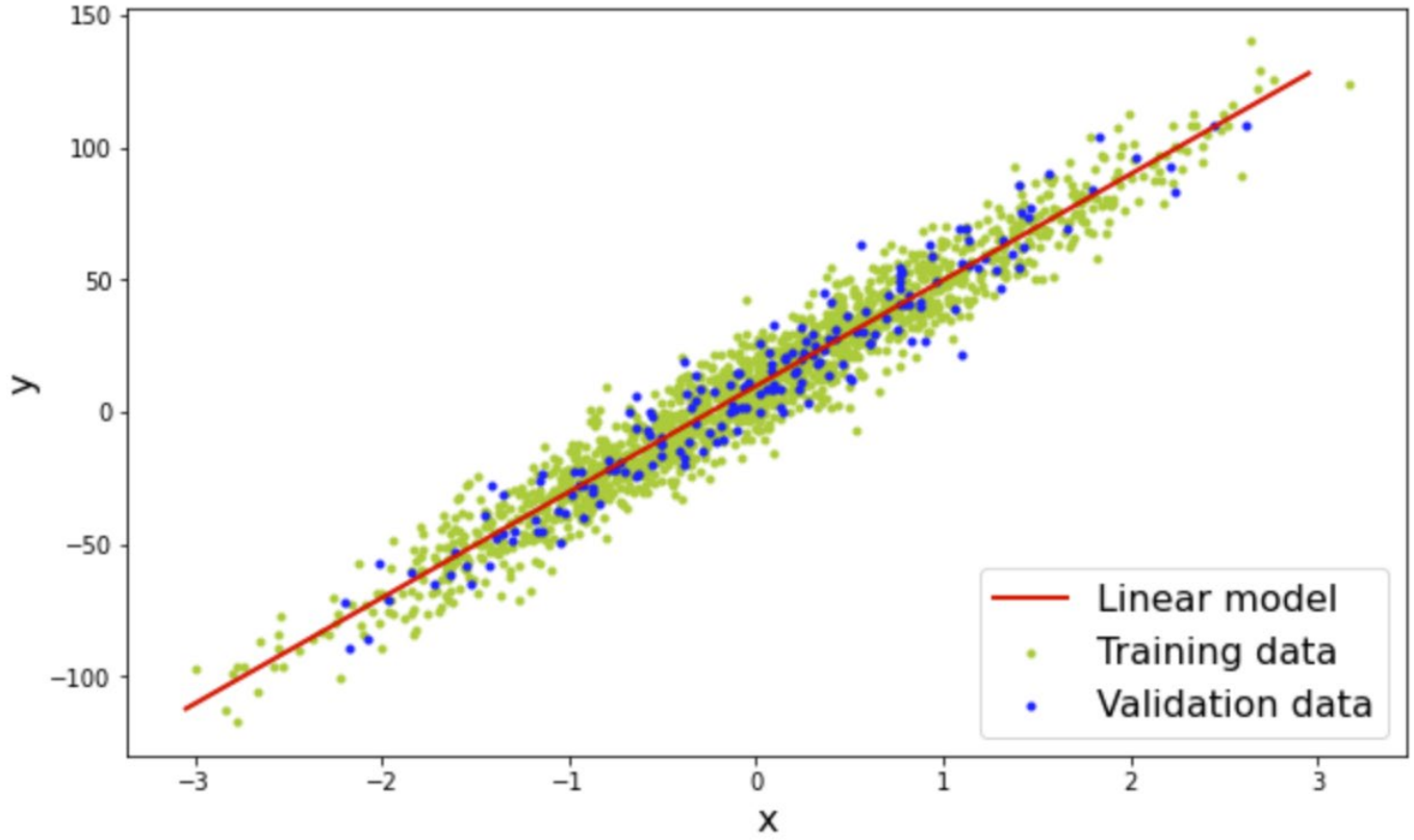
$$\hat{\boldsymbol{w}} = \arg min_w J(w) \Rightarrow \nabla J(\hat{w}) = 0$$

- For the linear-Gaussian case → Analytical solution

$$\hat{\boldsymbol{w}} = (X^T X)^{-1} X^T \boldsymbol{y}$$

Simple linear regression example

- Colab link: https://colab.research.google.com/drive/1EXDJHTIvp4zK3qavGsXO03NMwYRr59JM



Nonlinear Regression (linear in w_i)

- Basis function expansión
  - Linear regression can be made to model non-linear relations by using a feature space $x \rightarrow \phi(x)$

$$y = \phi(x)^T w + \epsilon \quad \epsilon \sim N(0, \sigma^2) \qquad \text{one data point } \{\mathrm{x}, \mathrm{y}\}$$

$$\mathbf{y} = \Phi_X^T w + \epsilon I_n \qquad \text{dataset } \{\mathrm{X}, \mathbf{y}\} \text{ (size n)}$$

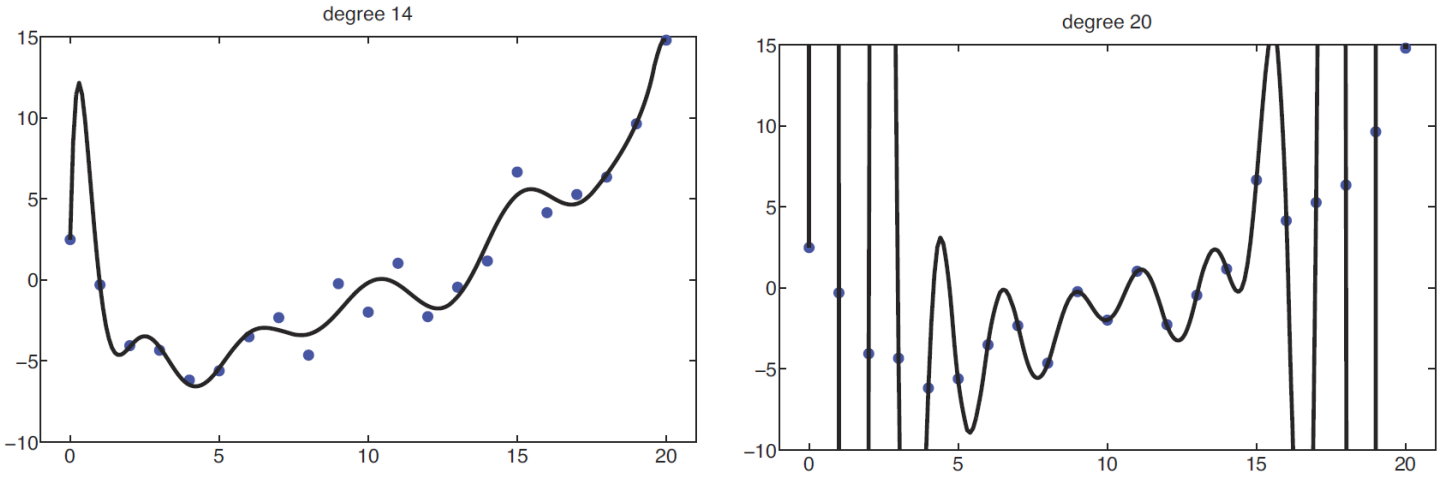
- Example:
  - In the previous example, we were computing the intercept

$$\phi(x) = [1, x]^T$$

$$y = \phi(x)^T w + \epsilon = w_0 + w_1 x_1 + w_2 x_2 + \cdots + w_n x_n + \epsilon$$

Nonlinear Regression (linear in w_i!)

- Example:
  - Polinomial basis function $\phi(x) = [1, x, x^2, \cdots, x^d]^T$ $(x \in \mathbb{R}^1, \phi(x) \in \mathbb{R}^d)$
  - $y = \phi(x)^T w + \epsilon = \sum_{j=1}^{d} w_j x^j + \epsilon = w_0 + w_1 x^1 + w_2 x^2 + \cdots + w_d x^d + \epsilon$
  - Polynomia of degree 14 and 20 fit by LS (which one is better?)

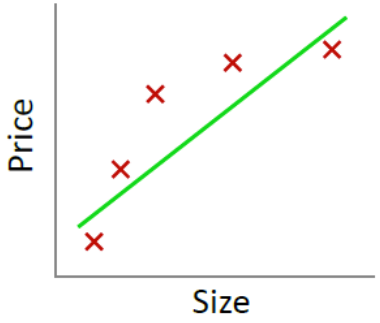


Example and figures taken from Murphy's book

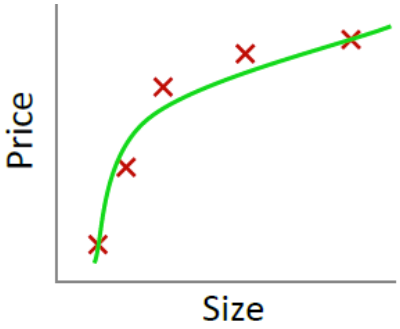
- We can use exponentials, sinusoids, step-functions…
  - We'll see more examples in the Bayesian slides)

Overfitting

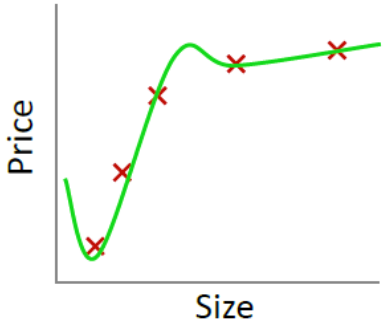
- The function $\phi(x)^T w$ fits the noise in addition to the underlying patterns. It "*memorizes*" the data instead of "*learning to generalize*".
- The function does not capture the structure of the data.
- The predictive capability of an overfitted model is poor.
- Techniques:
  - Cross-validation
  - Regularization



$\theta_0 + \theta_1 x$



$\theta_0 + \theta_1 x + \theta_2 x^2$

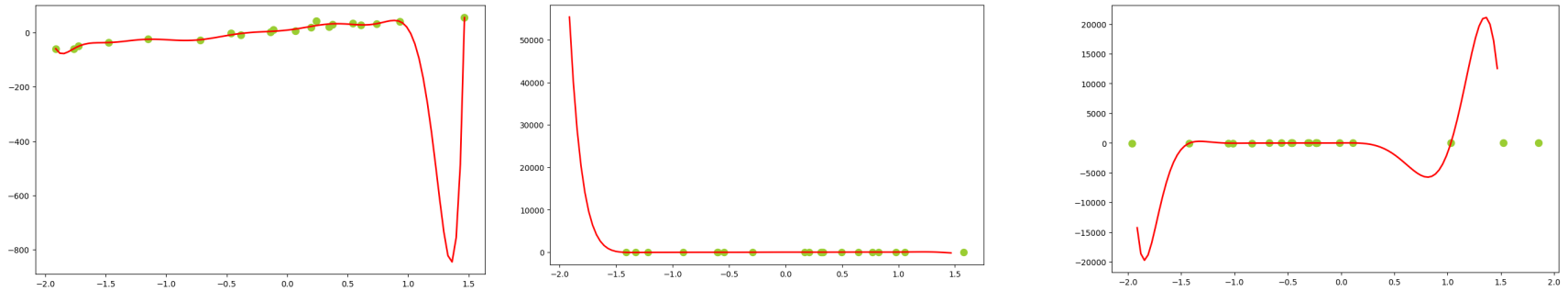


$\theta_0 + \theta_1 x + \theta_2 x^2 + \theta_3 x^3 + \theta_4 x^4$

$$J(\theta) = \frac{1}{2} \sum_{i=1}^{N} (h_\theta(x^{(i)}) - y^{(i)})^2 \approx 0$$

Regularization

- Bias-variance trade off.
  - ML is **unbiased** → If data → ∞ then $\mathbb{E}(w^* - \hat{w}) \to 0$
  - Overfitting results in large **variance**
    - If we fit slightly different data, the model is completely different

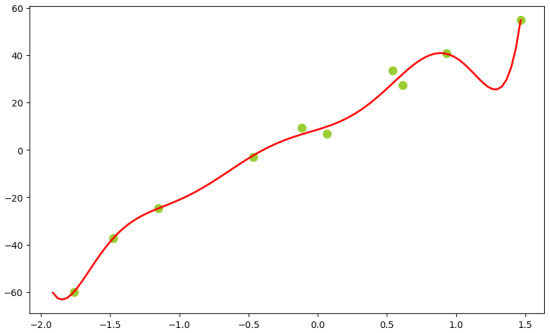
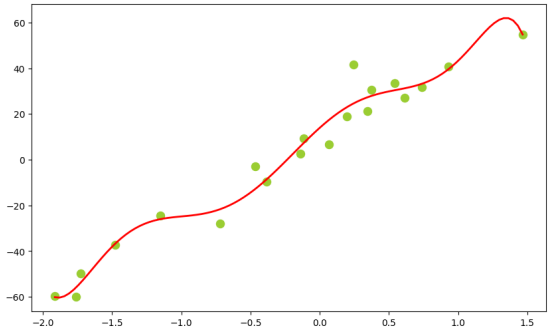
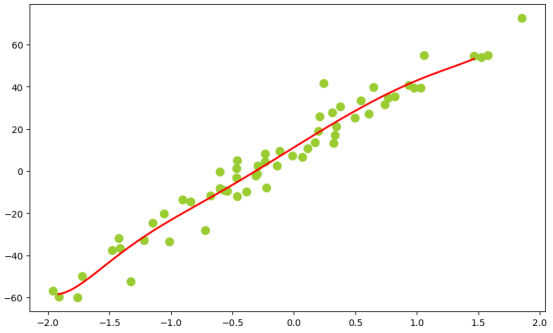


Polynomial regression d=12 for data generated by $y = 33x + 10 + \epsilon$ with $\epsilon \sim N(0,7)$

- Constrain the model while learning.
  - Model restrictions → **Lower variance**
  - ML+restrictions is **biased** → If data → ∞ then $\mathbb{E}(w^* - \hat{w}) \neq 0$

Regularization

- For complex models → Get more data
  - Same polynomial d=8

- If more data is not possible. Many variants:
  - **L2-regularization** (Ridge regression), Weight decay
  - L1 (Lasso)
  - Early stopping
  - Data augmentation
  - Dropout
  - Batch normalization
  - …

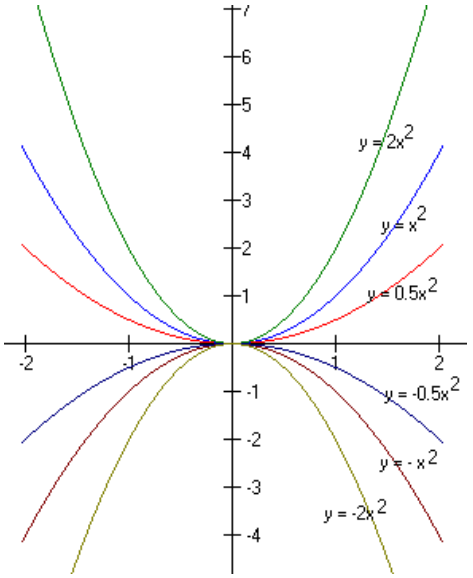
Check deep learning slides

Ridge regression

- Different ways to reach the same idea.
- We want to constraint the weights

$$\underset{w}{\operatorname{argmin}} \frac{1}{N} \sum_i \left(y^{(i)} - w^T x^{(i)} \right)^2$$

$$\text{subject to } \|w\|_2^2 \leq t$$

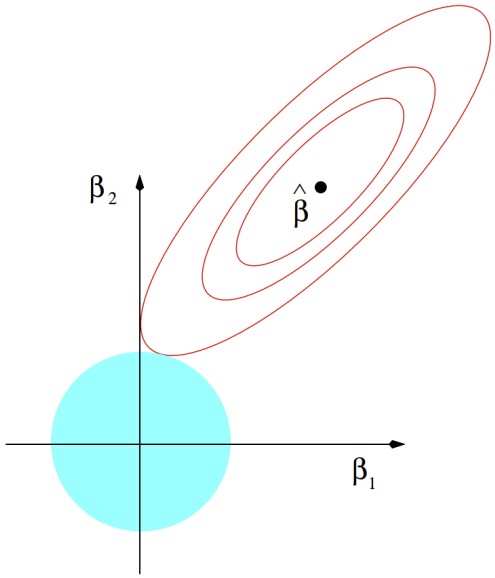


- Equivalent: Lagrange optimizer

$$\underset{w}{\operatorname{argmin}} \frac{1}{N} \sum_i \left(y^{(i)} - w^T x^{(i)} \right)^2 + \lambda \|w\|_2^2;$$

$$\hat{w} = (X^T X + \lambda I_d)^{-1} X^T \boldsymbol{y}$$

- Which is equivalent to assuming a Gaussian prior on **w** and do *maximum a posteriori (MAP)*



Bayesian interpretation

- Remember the Maximum Likelihood

$$\hat{w} = \underset{w}{\operatorname{argmax}} \log p(\boldsymbol{y}|X, w)$$

- We can specify a prior over the model parameters $p(w)$
  - For example, avoid large values $p(w) \sim \mathrm{N}(0, \tau^2 I_d)$
- Use Bayes's rule to compute the posterior

$$p(w|\boldsymbol{y}, X) = \frac{p(\boldsymbol{y}|X, w)p(w)}{p(\boldsymbol{y})} \propto p(\boldsymbol{y}|X, w)p(w)$$

- Maximum a posteri

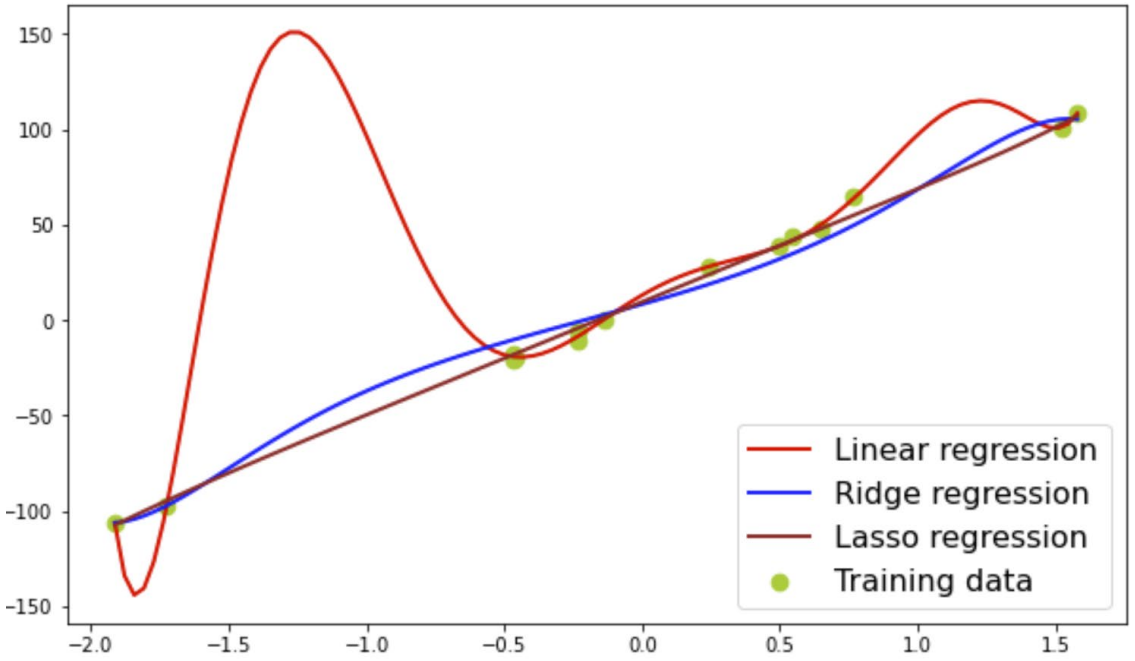
$$\hat{w} = \underset{w}{\operatorname{argmax}} \log p(w|\boldsymbol{y}, X)$$

$$\hat{w} = (X^T X + \lambda I_d)^{-1} X^T \boldsymbol{y} \qquad \text{with: } \lambda = \frac{\sigma^2}{\tau^2}$$

Ridge Regression

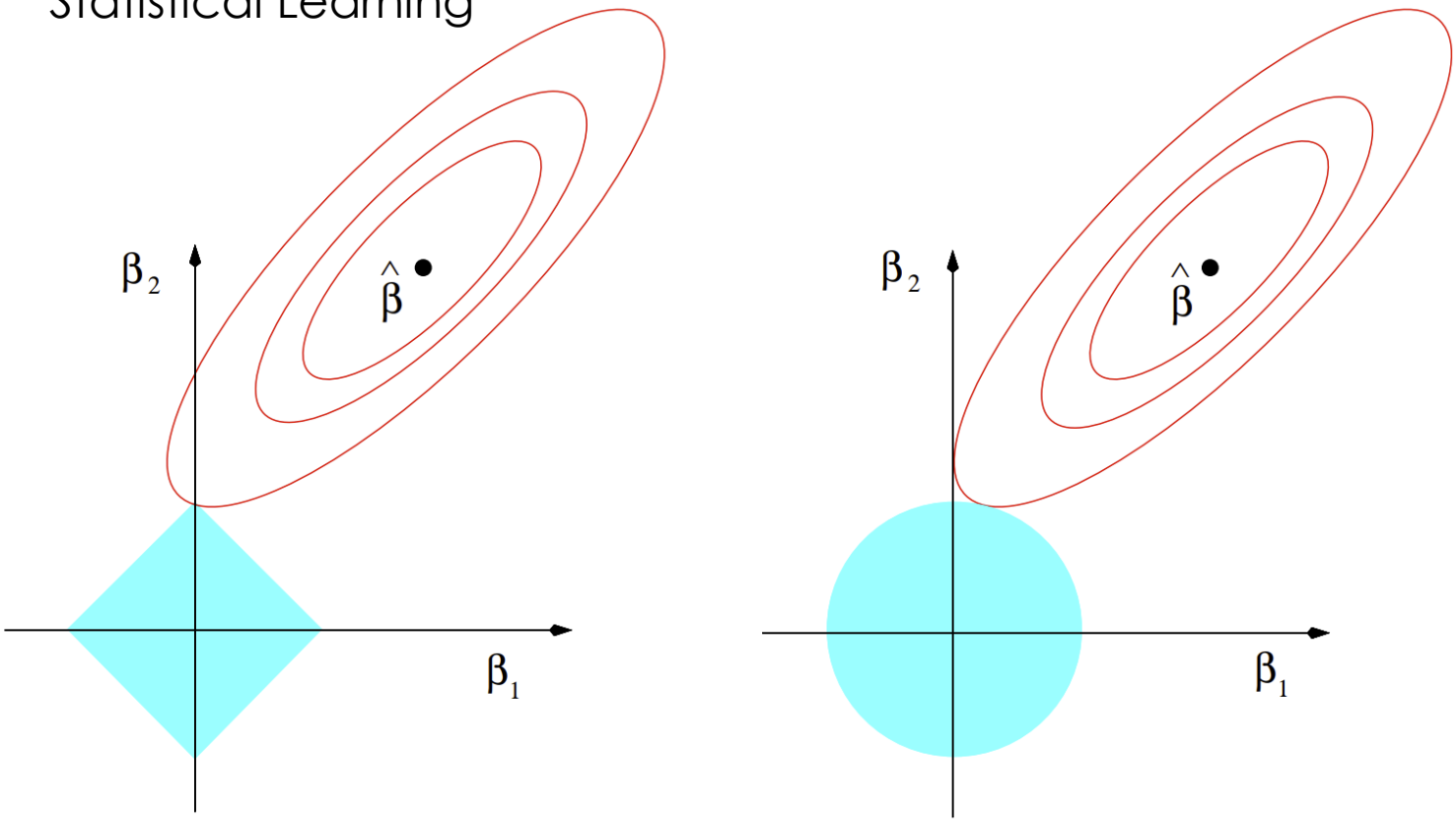
Colab link:
https://drive.google.com/file/d/1EnrFqx8Ks0L3-Uy99IB6DJ4DdEHyP5pl

- Avoids overfitting
- $\lambda w^T w = \lambda \|w\|_2$ similar to "weight decay" in deep learning
- $\lambda \|w\|_1$ LASSO (Least Absolute Shrinkage and Selection Operator)
  - Assumes a Laplacian distribution prior for the weights (then $\lambda = \frac{2\sigma^2}{b^2}$)
  - Can assign zero values to some coefficients (feature selection)



L1 norm leads to sparse models

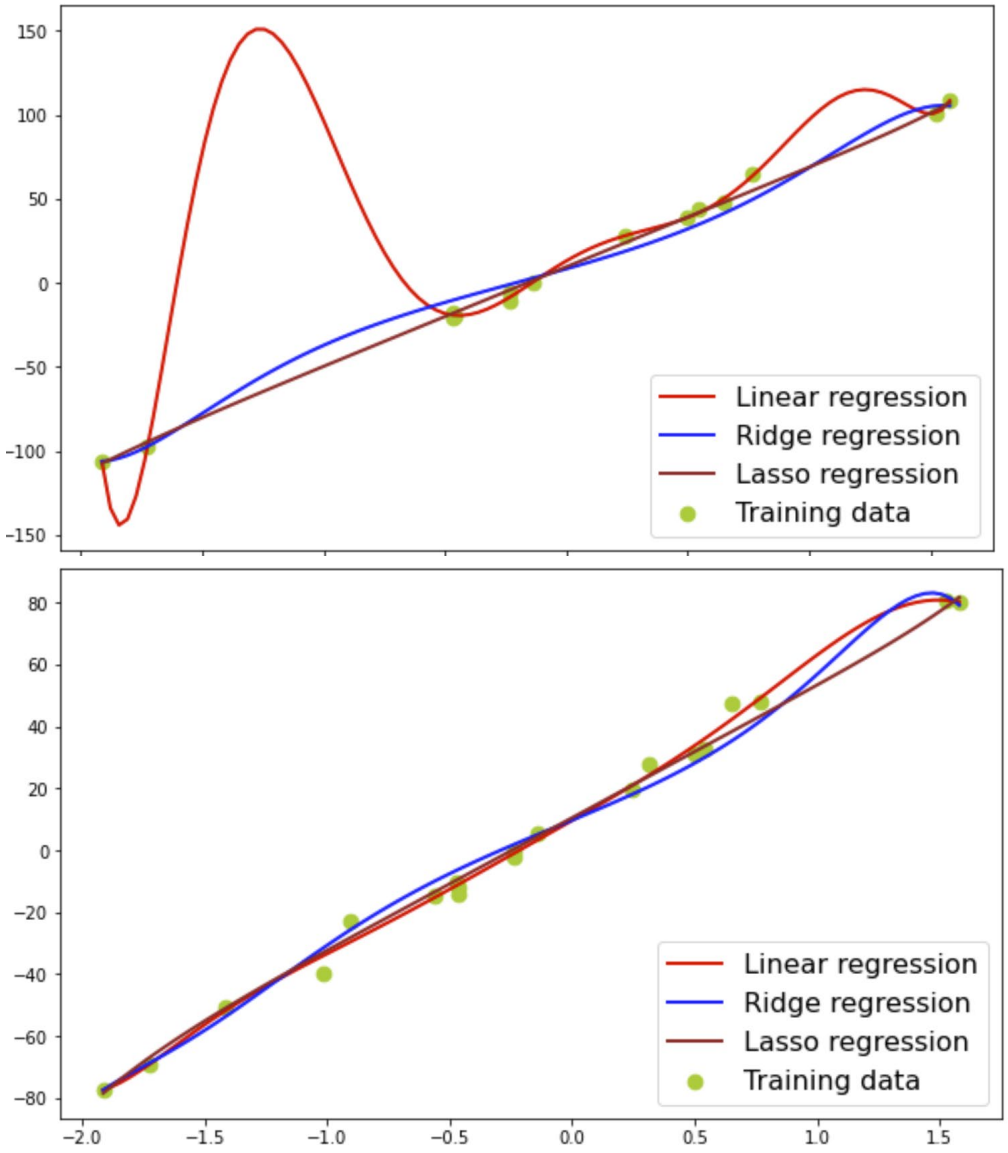
- Sparse models
  - Some coefficients are equal to zero
  - Feature selection → uninformative features are given zero weight
  - Geometrical interpretation: Fig. 3.11, Hastie et al., The Elements of Statistical Learning



Data regularization

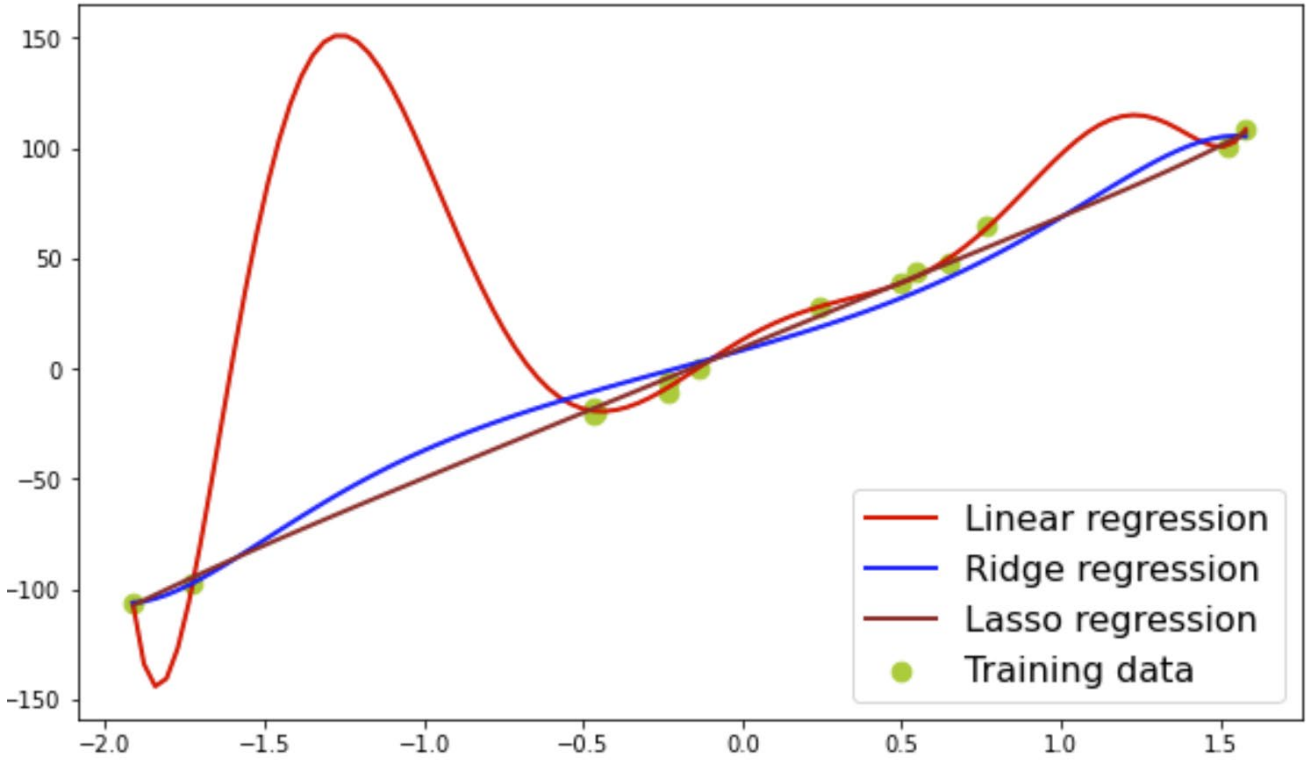
Colab link:
https://colab.research.google.com/drive/1j3kYQAHeGSbEFf66U_f2P9wlSatT1Erl

- BTW, in this example, check the regularization effect of adding more data
  - 15 data samples
  - 20 data samples



Summary: avoiding overfitting

- Do cross-validation to select the best model
- Add constraints that do not involve training data (e.g., weight decay)
- Add more training data!

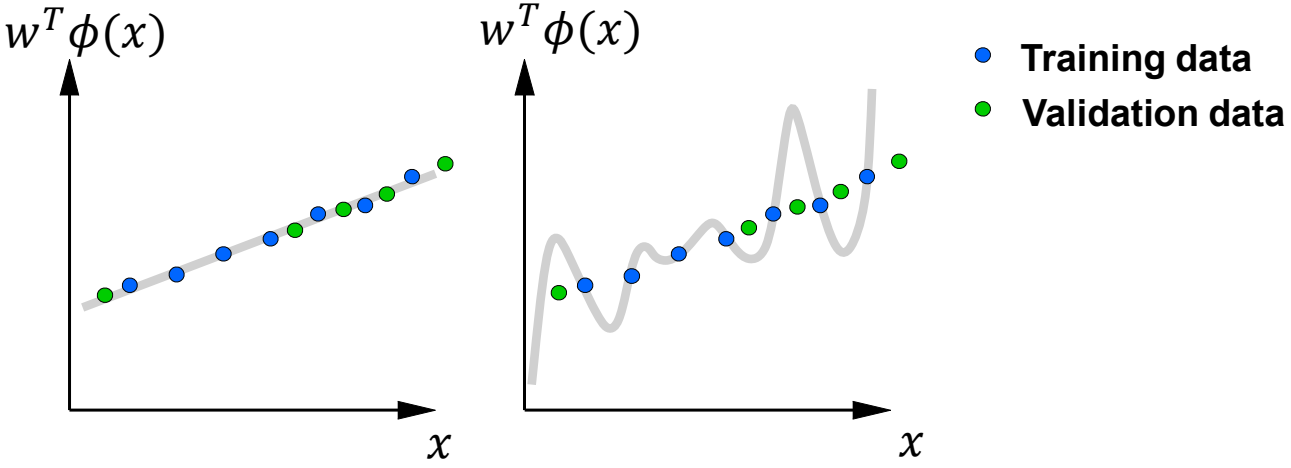


Hyperparameter and model selection

- How do we select λ regularizer?
  - … or the model complexity?
  - … or other hyperparameters* of the learning process?

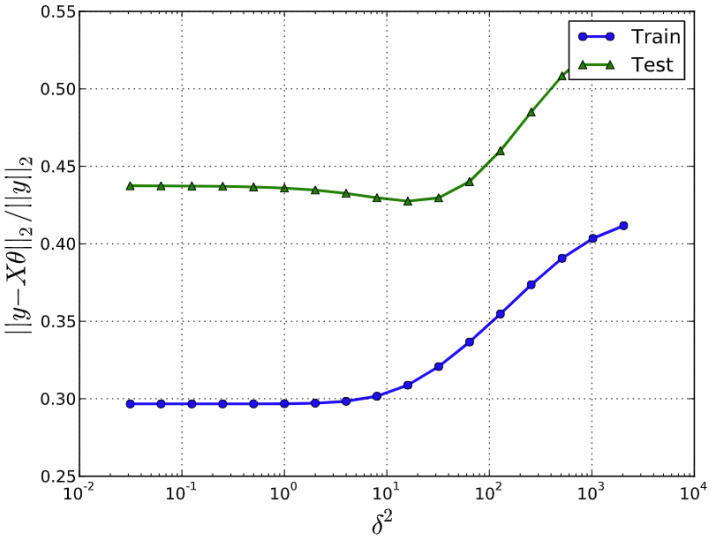
* Hyperparameters are all the model/algorithm numbers that we select that are not part of the trainable parameters w

- Remember to partition the data: train/validation/test!
  - The **training set** is used to estimate the model parameters w
  - The **validation set** is used to decide among a set of hyperparameters (models)
  - What we care about is generalization! → Final performance metrics should be given for a **test set**



Cross-validation

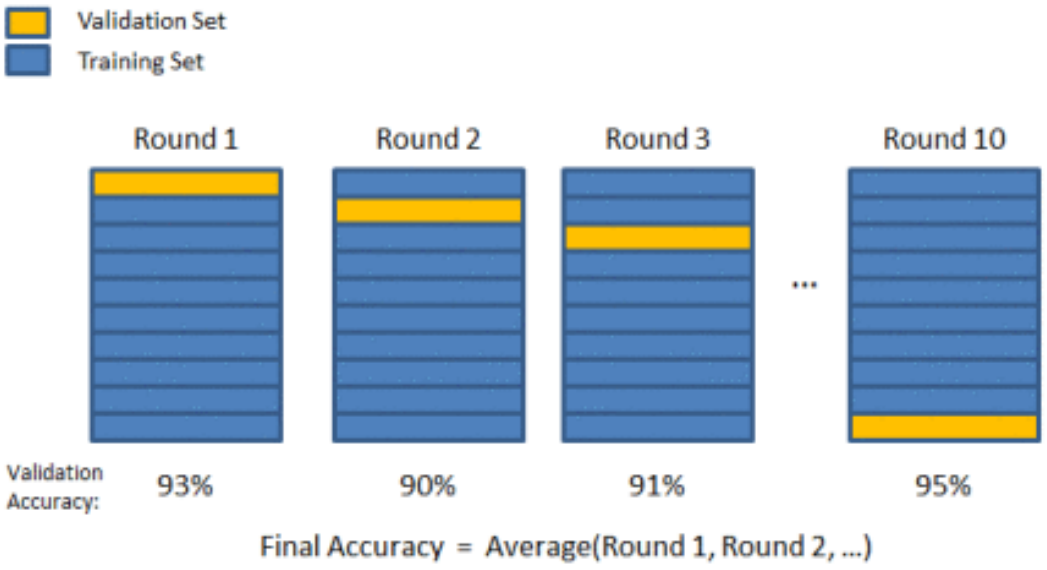
- Compute evaluation metrics (e.g. : prediction error) on the validation data for different hyperparameters/models



- How do we explore the hyperparameter space?
  - Randomly
  - Exhaustively
  - Intelligently ← Check Bayesian optimization slides!

K-fold cross-validation

- Common practice to have a higher degree of invariance to the validation/test split.
  - Separate the data into k folds
  - Do k training-validation rounds. A usual value for k is 10.
  - The validation/training sets are changed each round.
  - Metrics are averaged over all folds.
  - If k=N, the we have **leave-one-out cross-validation** or LOOCV



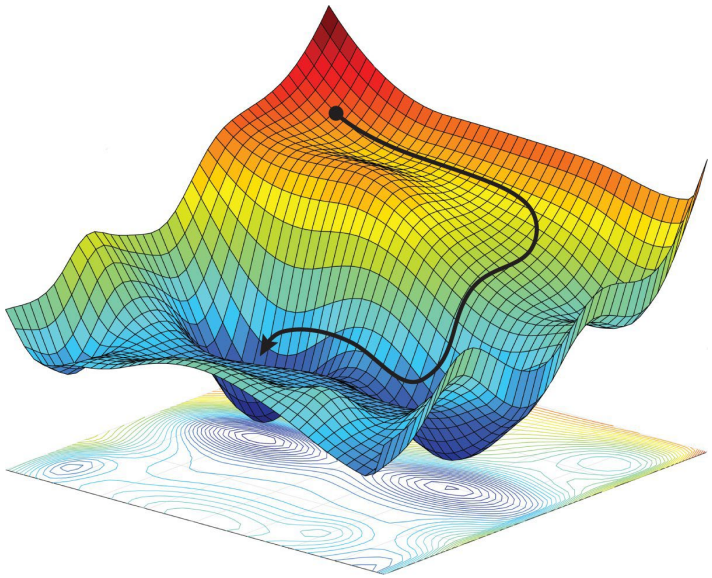
Remainder: data splits

- Training data (~80% of the data)
  - Used to learn the model parameters
- Validation (~10% of the data)
  - Used to select the hyper-parameters
  - After used, can be added to the training data to train the final model
  - Training/validation splits can change during training (k-fold cross-validation)
- Test (~10% of the data)
  - Never used at the training stage!
  - Used to obtain the final specs of our model
  - After used, can be added to the training data to train the final model

Analytical solution may not work…

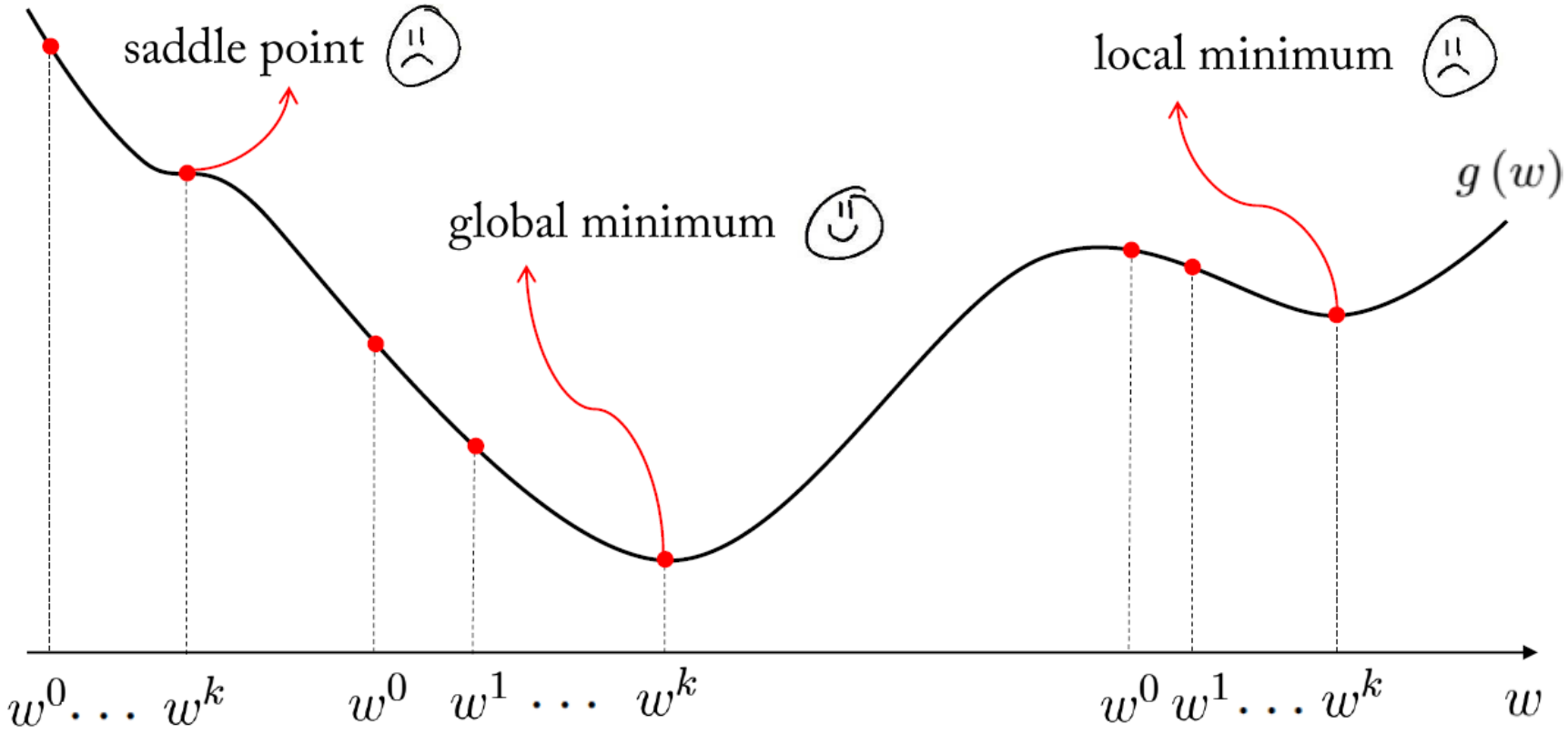
- Most algorithms in Machine Learning do not have an analytical solution.

- Large datasets → Huge $X^T X$ matrix
  - Minimization avoids computing inverse $(X^T X)^{-1}$
  - Follow the gradient $\nabla J(w)$
  - We can even optimize one data point at a time



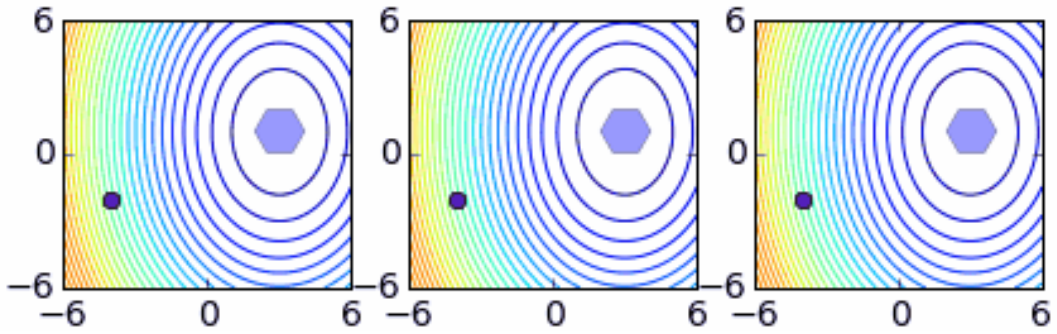
Gradient descent

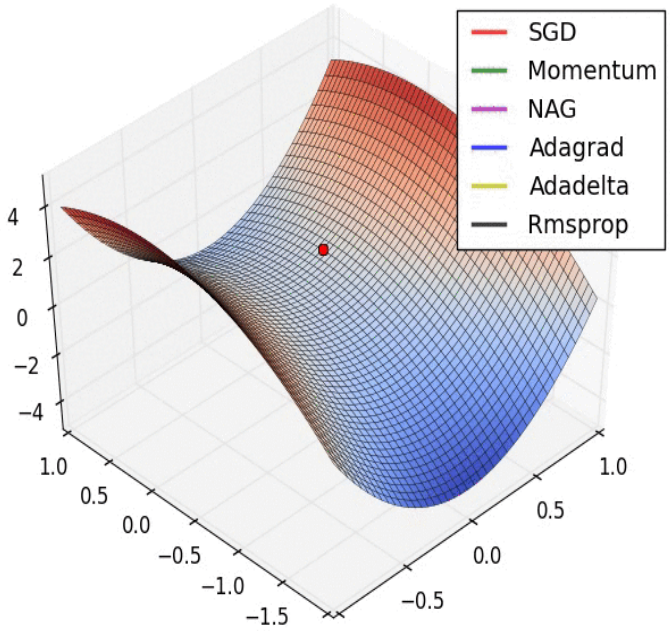
- Gradient descent → Iterative weight updates in the downward direction, $w_{k+1} = w_k - \eta_k \nabla J(w)$
- In general, it converges to a local minimum



Gradient descent

- Issues:
  - Select the initial seed w_0
  - Select the learning rate η_k
    - Small $\eta_k \rightarrow$ slow convergence, large $\eta_k \rightarrow$ fail to converge





Stochastic gradient descent

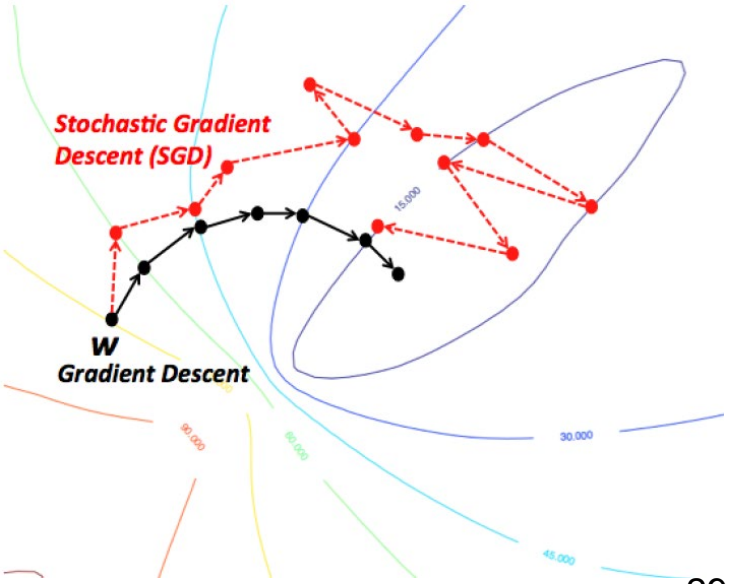
- Remember:

$$\hat{w} = \underset{w}{\operatorname{argmax}} \log \prod_{i=1}^{n} p(y_i|\boldsymbol{x}_i, w) = \underset{w}{\operatorname{argmax}} \sum_{i=1}^{n} \log p(y_i|\boldsymbol{x}_i, w)$$

- We are computing a loss based on an expectation

$$\hat{w} = \underset{w}{\operatorname{argmax}} \mathbb{E}[\log p(y_i|\boldsymbol{x}_i, w)]$$

- Then, we can optimize one data at a time and still find the optimum.
- One of the most important algorithms in the last decades!

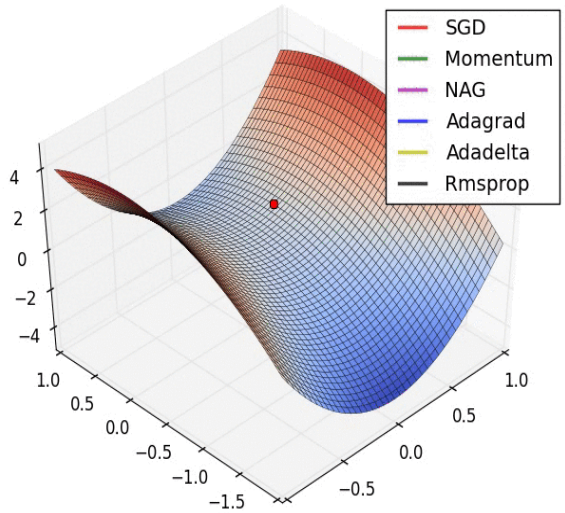


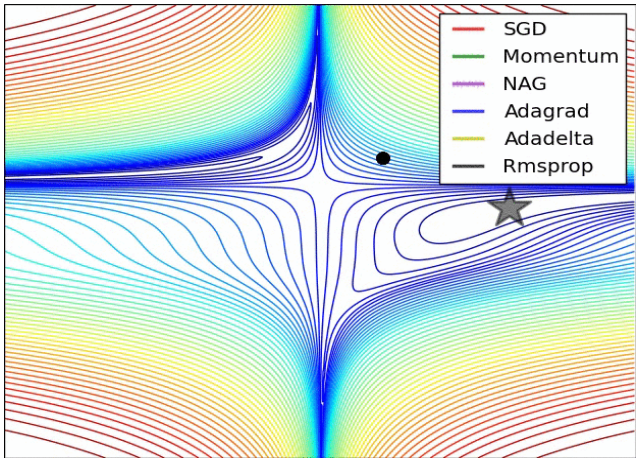
Stochastic gradient descent

- In our linear-Gausssian case

$$\nabla J_i(w) = -2\boldsymbol{x}_i(\boldsymbol{x}_i^T\boldsymbol{w} - y_i)$$

- Highly efficient for large datasets.
- **WARNING!**
  - At each iteration, we must use data **independent** from previous iterations.
  - For example, select data **randomly** from training set.
- Random → Might avoid local minima and saddle points!
- Every time we go over all data points is called an **epoch**.





GD vs SGD

- Batch gradient descent
  - Expensive for large datasets
  - N → whole dataset
  - Low variance

$$w_{k+1} = w_k - \eta_k \sum_{i=1}^{N} \nabla J_i(w)$$

- Stochastic gradient descent
  - Cheap but more grad steps
  - Might avoid local optima
  - High variance/fluctuations

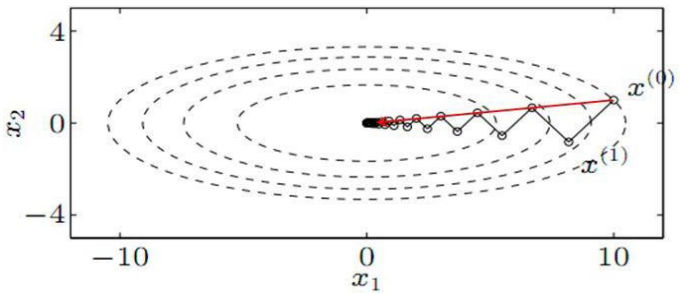
$$w_{k+1} = w_k - \eta_k \nabla J_i(w)$$

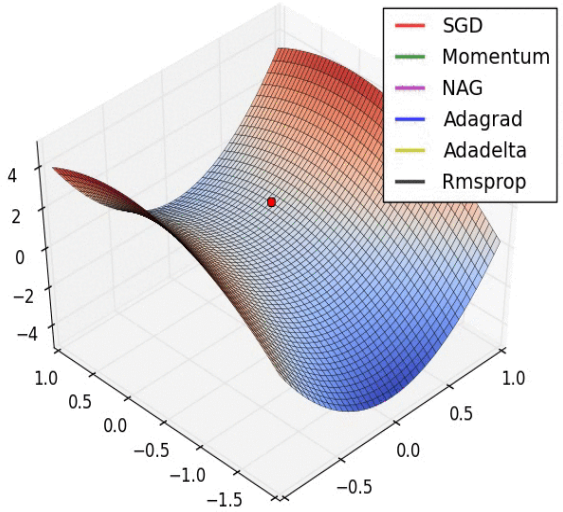
- SGD with mini-batch
  - Best of both worlds
  - Choose a small batch size ($n = 2, 4, 8, \dots, 128, 256, \dots$)

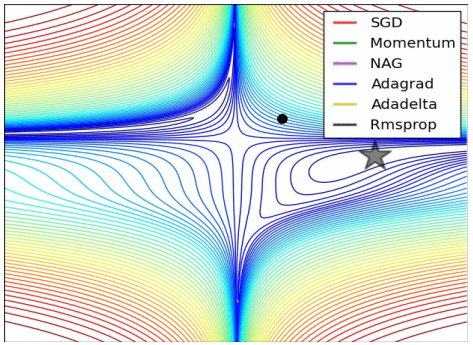
$$w_{k+1} = w_k - \eta_k \sum_{i=1}^{n} \nabla J_i(w)$$

SGD variants

- Momentum can stabilize/smooth SGD variance
- Adaptive methods adapt the learning rate for each feature depending on the estimated geometry of the problem.
- Many in ML libraries: AdaGrad, AdaDelta, RMSProm, Adam,…



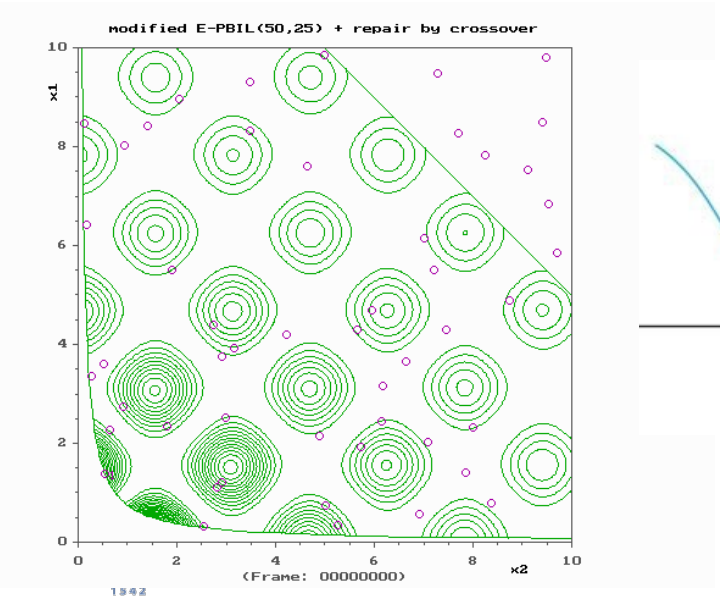


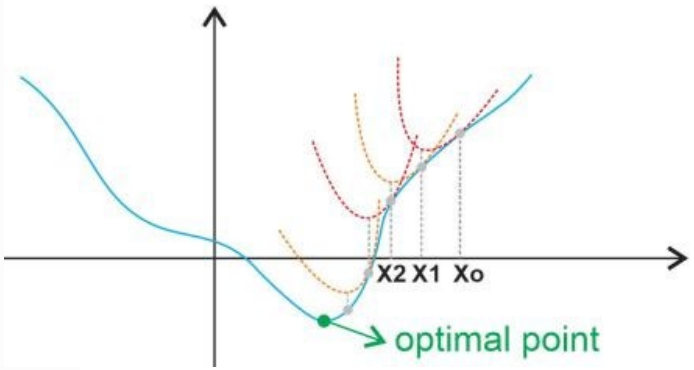


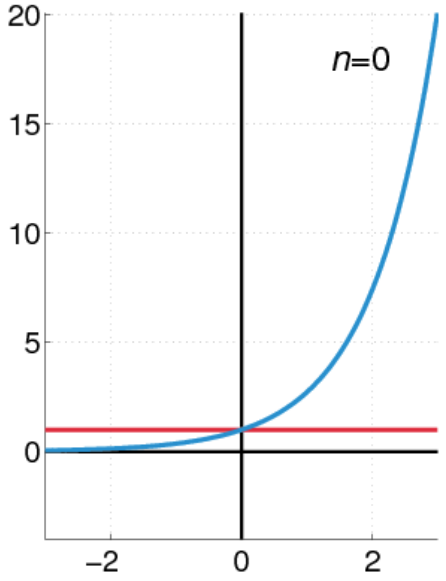
Universidad Zaragoza
1542

Other optimization algorithms

- They are ALL very computationally intensive for large models/data.
- Bayesian optimization → Later in the course. ☺
- Evolutionary algorithms
  - No need for gradient. Global optimization. Poor dimensional scaling.
- Second order methods:
  - Newton, Conjugate Gradient, BFGS, L-BFGS,...
  - They approximate the Hessian (second derivative of cost)

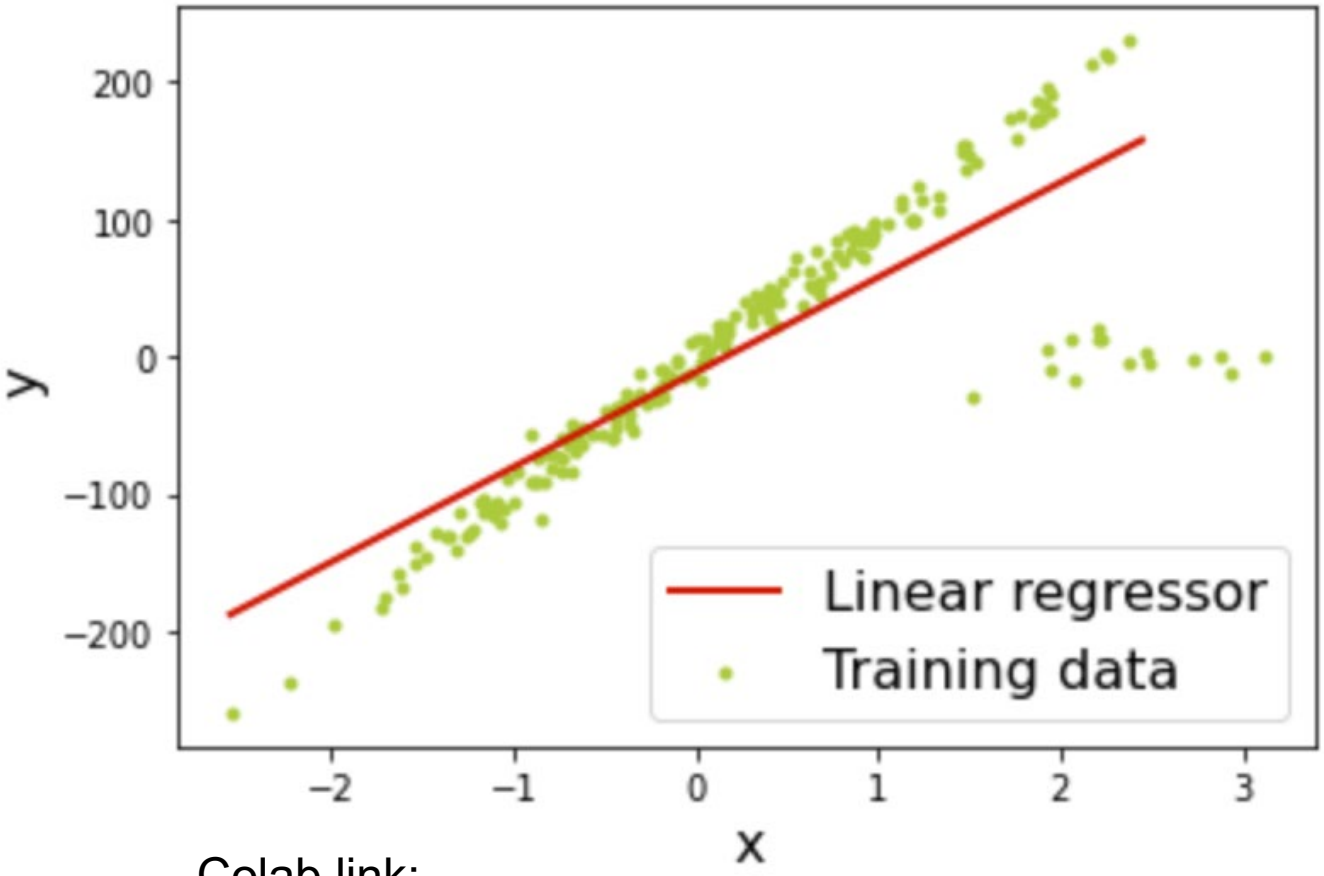






Robust Regression

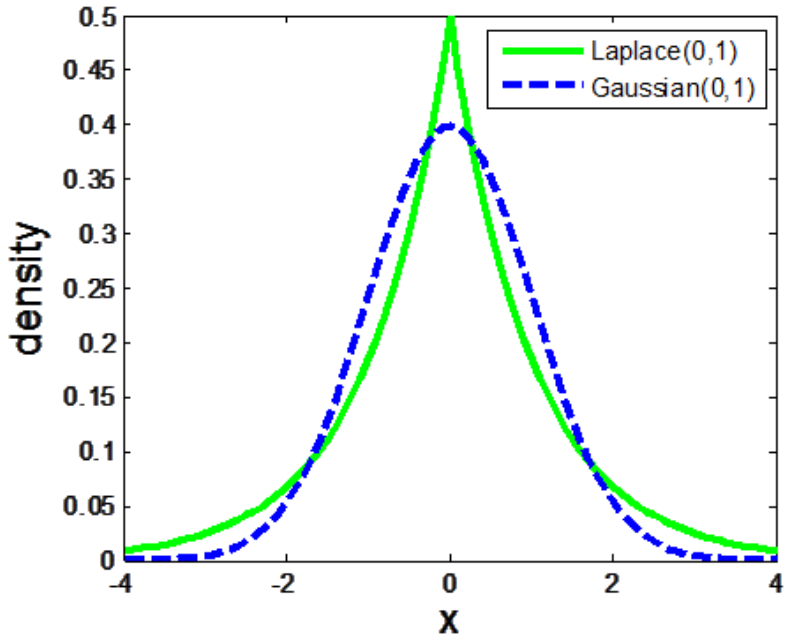
- The quadratic error penalty from the Gaussian noise gives a high weight to outliers



Colab link:
https://drive.google.com/file/d/1G5E6QHihfK1TyaYe6dxlF2FF5XC1kXn-

Robust Regression

- Probabilistic view: The low tolerance of Gaussian pdfs for high-noise data samples causes problems with outliers
- One possible solution: assume longer-tailed distributions, e.g., Laplace, Student's t, etc.
- Other: detect outliers and remove them
  - You will learn about RANSAC in the computer vision course



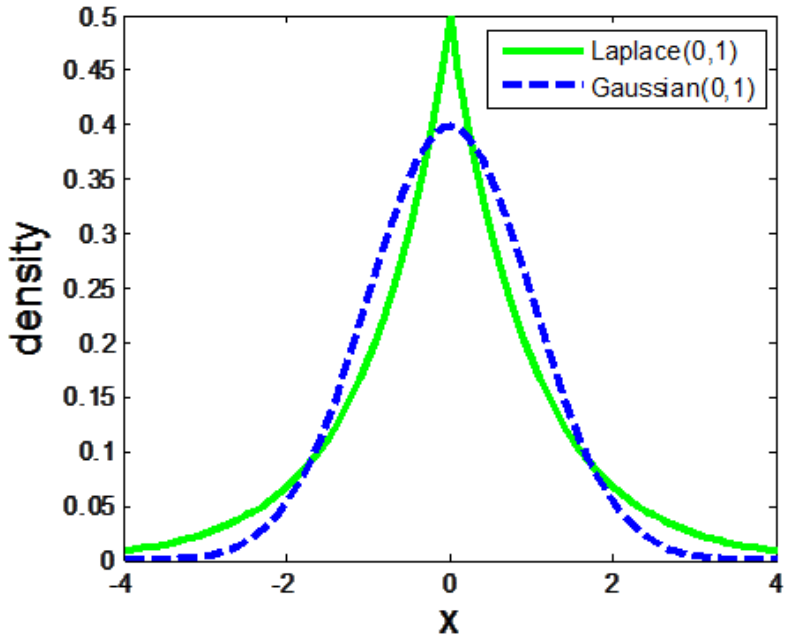
Robust Regression

- Laplace distribution:

$$p\left(y^{(i)}\middle|x^{(i)},w\right)=\exp\left(-\frac{1}{2b^2}\left|y^{(i)}-w^Tx^{(i)}\right|\right)\rightarrow\cdots\rightarrow$$

$$J(w)=\frac{1}{2}\sum_i\left|y^{(i)}-w^Tx^{(i)}\right|=\sum_i\left|r^{(i)}\right|$$

- Non-linear, hard to optimize!



Robust Regression: Huber loss function

- Equivalent to L2 near zero, to L1 for large errors

$$r^{(i)} = \begin{cases} \frac{r^{(i)^2}}{2} & if\ |r| \leq \delta \\ \delta|r| - \frac{\delta^2}{2} & if\ |r| > \delta \end{cases}$$

- Differentiable!

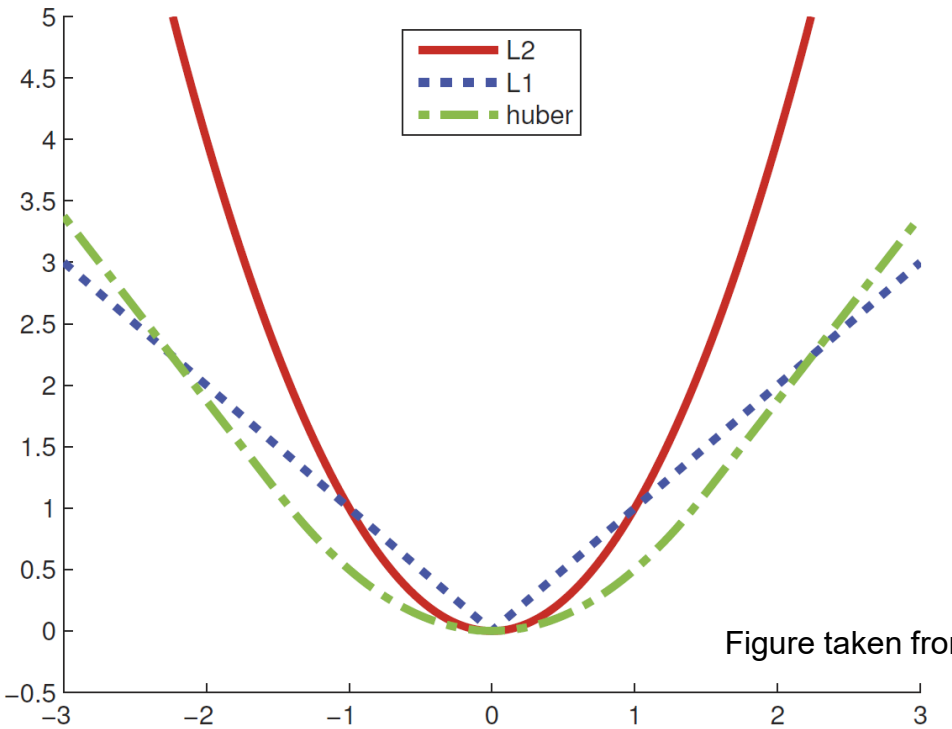
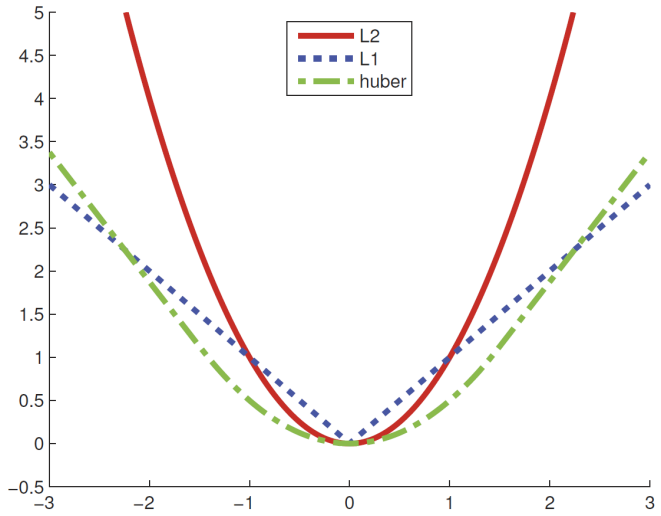


Figure taken from Murphy's book

Huber loss function

- How do we tune the parameter δ?

$$r^{(i)} = \begin{cases} \dfrac{{r^{(i)}}^2}{2} & if\ |r| \le \delta \\ \delta|r| - \dfrac{\delta^2}{2} & if\ |r| > \delta \end{cases}$$



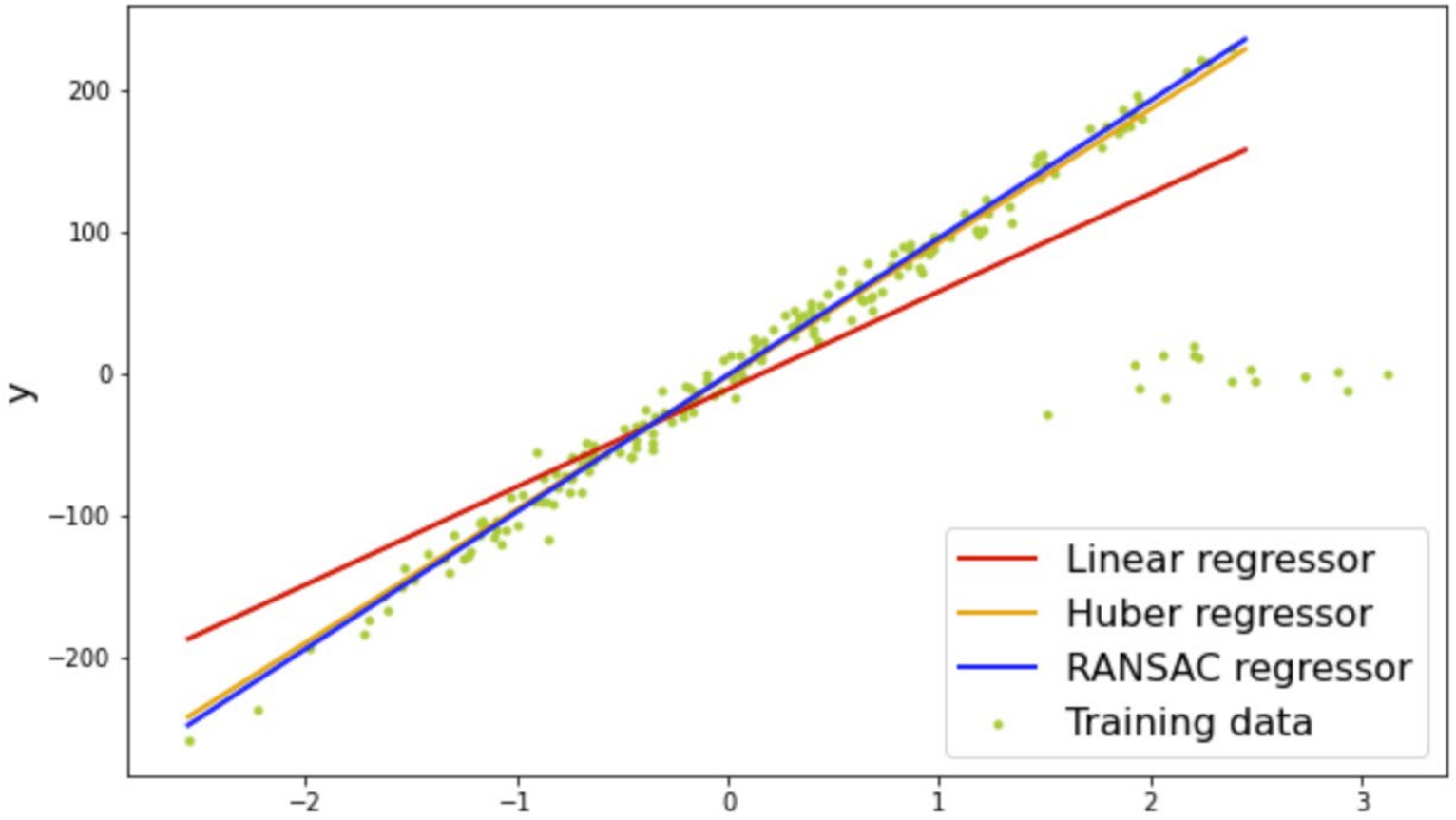
- δ is the "boundary" between inliers (Gaussian pdf assumed) and the outliers (Laplacian pdf assumed). So:

1. [Huber 1981], says: "good choices are between σ and 2σ"
   - σ is the standard deviation of the residuals $r^{(i)}$ of the inliers
2. If you do not know σ, you estimate it from the training data residuals $r^{(i)}$.
   - IMPORTANT! You have outliers, so use a robust estimator for σ. MAD (Median Absolute Deviation) is common:

$$\hat{\sigma} = 1.4826 \cdot \mathrm{MAD}, \mathrm{MAD} = \mathrm{median}\left(\left|r^{(1)} - \tilde{r}\right|, \dots, \left|r^{(N)} - \tilde{r}\right|\right), \tilde{r} = \mathrm{median}\left(r^{(1)}, \dots, r^{(N)}\right)$$

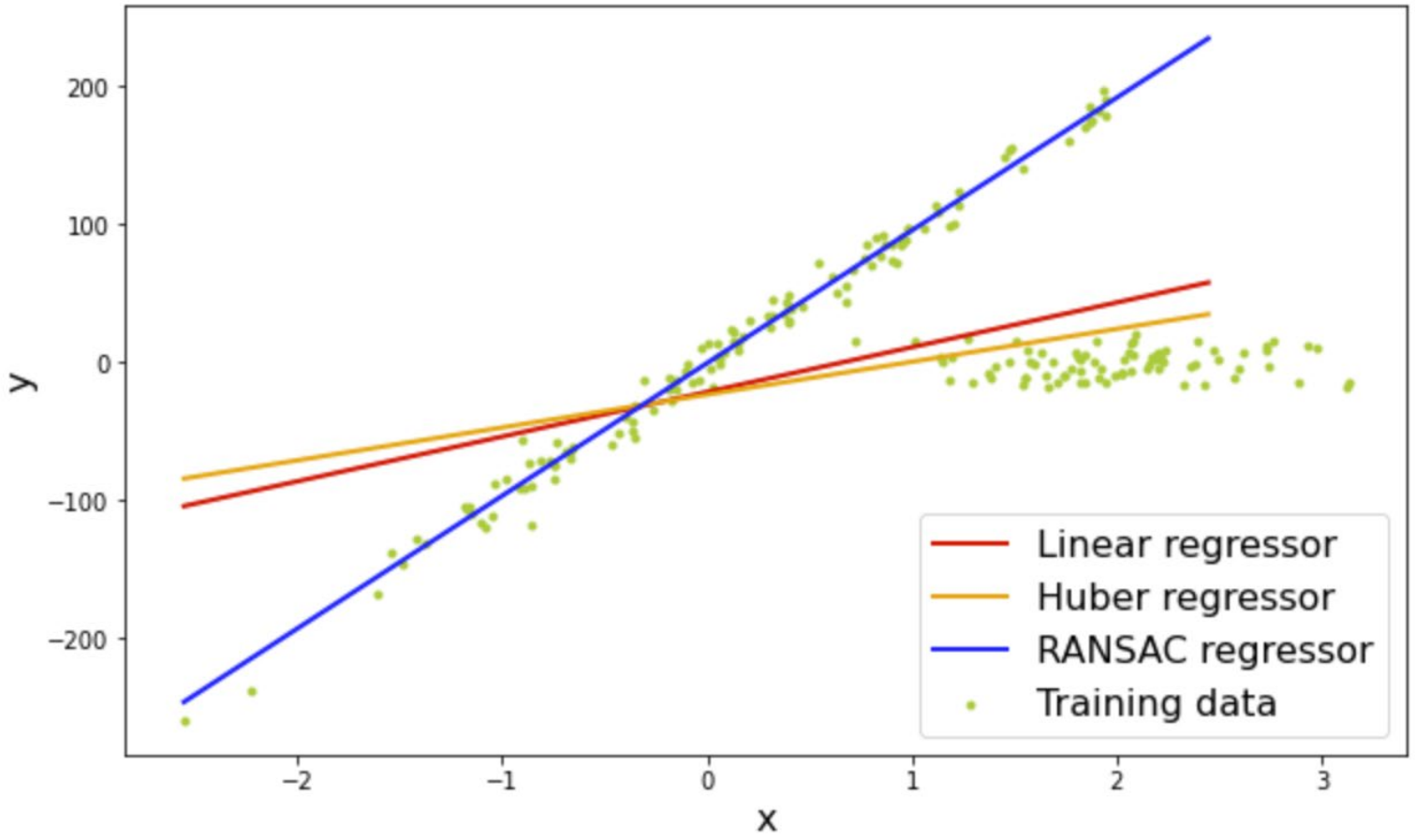
Robust regression example

- The Huber loss and RANSAC are robust to outliers



Robust regression example

- But RANSAC can tolerate a higher outlier ratio!



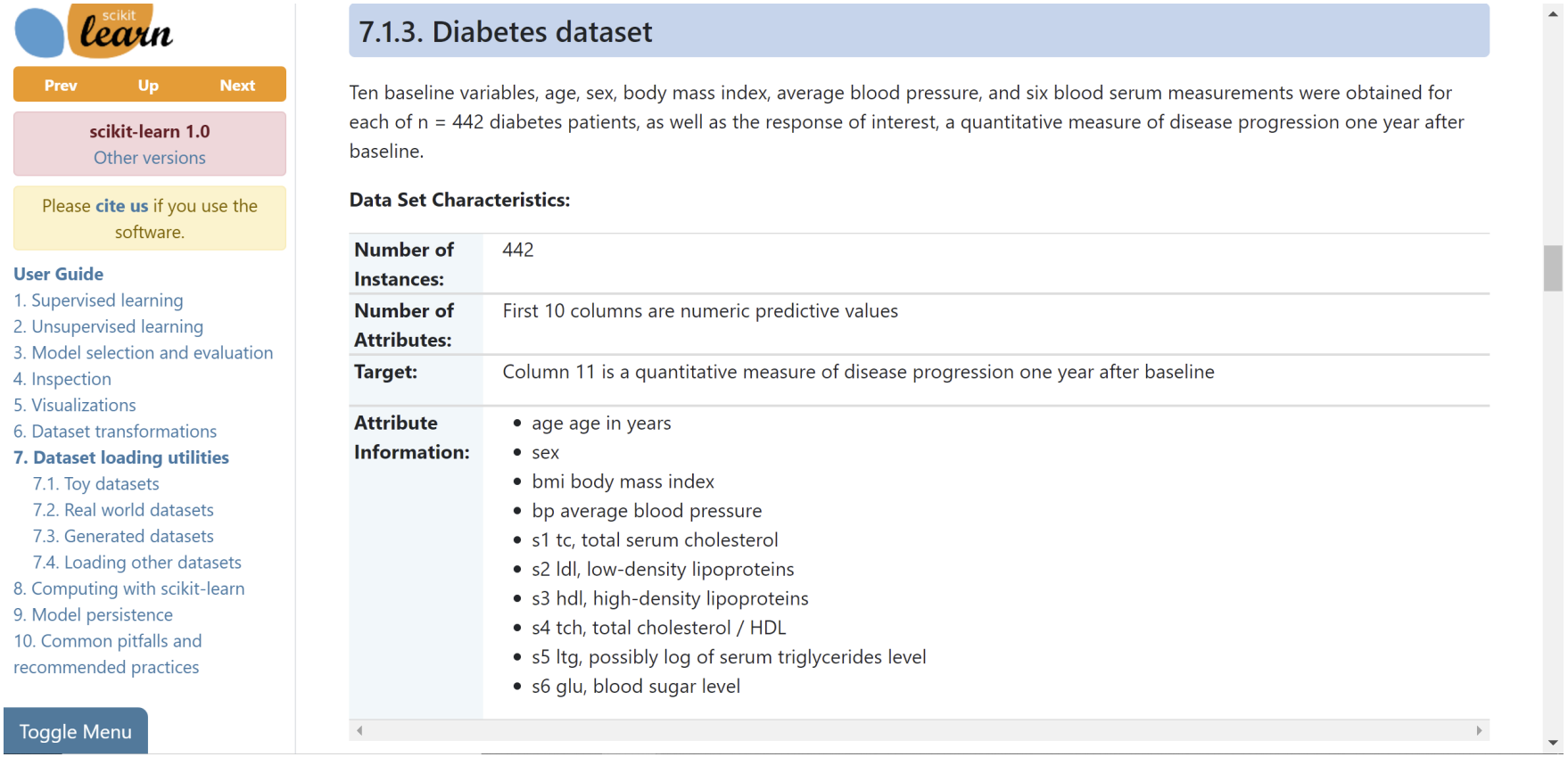
Regression metrics

- **Mean absolute error** $\rightarrow \frac{1}{N}\sum_{i=1}^{N}\left|y^{(i)} - w^T x^{(i)}\right|$
  - Small influence of large errors
- **Mean squared error** $\rightarrow \frac{1}{N}\sum_{i=1}^{N}\left(y^{(i)} - w^T x^{(i)}\right)^2$
  - Highly influenced by large errors
- **Root mean squared error** $\rightarrow \sqrt{\frac{1}{N}\sum_{i=1}^{N}(y^{(i)} - w^T x^{(i)})^2}$
- **Median absolute error** $\rightarrow \text{median}\left(\left|y^{(1)} - w^T x^{(1)}\right|, \ldots, \left|y^{(N)} - w^T x^{(N)}\right|\right)$
  - Robust to outliers!
- **R^2 score** $\rightarrow 1 - \frac{\sum_{i=1}^{N}\left(y^{(i)} - w^T x^{(i)}\right)^2}{\sum_{i=1}^{N}\left(y^{(i)} - \bar{y}\right)^2} \in [-\infty, 1], \bar{y} = \frac{1}{N}\sum_{i=1}^{N} y^{(i)}$
  - Proportion of variance in y explained by the variables of the model
- Many others! Maximum error, Mean absolute percentage error, explained variance score, etc.
- Compare against dumb baseline (mean of the training predictions?)

Regression example: The diabetes dataset

- Colab links:

Simple:
https://drive.google.com/file/d/1EIKbBCWetqEVKumrTX4GEpxdbRSFXwVw
With cross_validation, grid search and pipelines:
https://colab.research.google.com/drive/1xAhcsXMqoJD-sFRfGfMk-pB4FBuZE2Gf



scikit learn

Prev Up Next

scikit-learn 1.0
Other versions

Please cite us if you use the software.

User Guide
1. Supervised learning
2. Unsupervised learning
3. Model selection and evaluation
4. Inspection
5. Visualizations
6. Dataset transformations
7. Dataset loading utilities
   7.1. Toy datasets
   7.2. Real world datasets
   7.3. Generated datasets
   7.4. Loading other datasets
8. Computing with scikit-learn
9. Model persistence
10. Common pitfalls and recommended practices

Toggle Menu

7.1.3. Diabetes dataset

Ten baseline variables, age, sex, body mass index, average blood pressure, and six blood serum measurements were obtained for each of n = 442 diabetes patients, as well as the response of interest, a quantitative measure of disease progression one year after baseline.

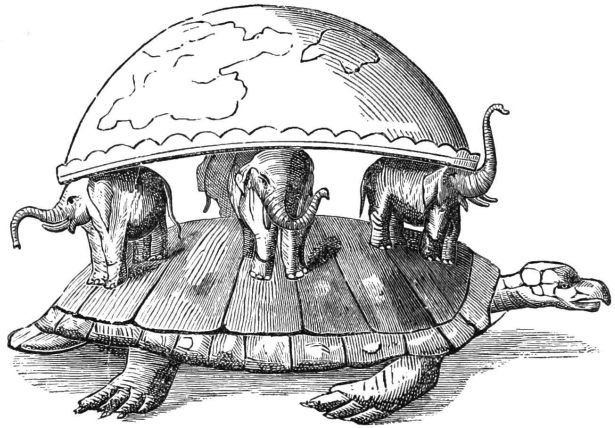
Data Set Characteristics:

Number of Instances:	442
Number of Attributes:	First 10 columns are numeric predictive values
Target:	Column 11 is a quantitative measure of disease progression one year after baseline
Attribute Information:	• age age in years • sex • bmi body mass index • bp average blood pressure • s1 tc, total serum cholesterol • s2 ldl, low-density lipoproteins • s3 hdl, high-density lipoproteins • s4 tch, total cholesterol / HDL • s5 ltg, possibly log of serum triglycerides level • s6 glu, blood sugar level

Bayesian Ridge Regression and ADR?

- They show the best performance in the Diabetes dataset!
- In the MAP estimate, we have a prior and likelihood with some parameters (noise, prior variance, etc.)
- Can we set up a prior on those parameters?

- Yes! That is called **hierarchical Bayesian model**!
  - *Turtles all the way down…*
  - We'll come back in the Bayesian learning, part



Data standardization

- Many ML algorithms (SVM, L2/L1 regularization) assume all dimensions of x has zero-mean and similar covariance.
- It also helps numerically.
- Most usual standardization:

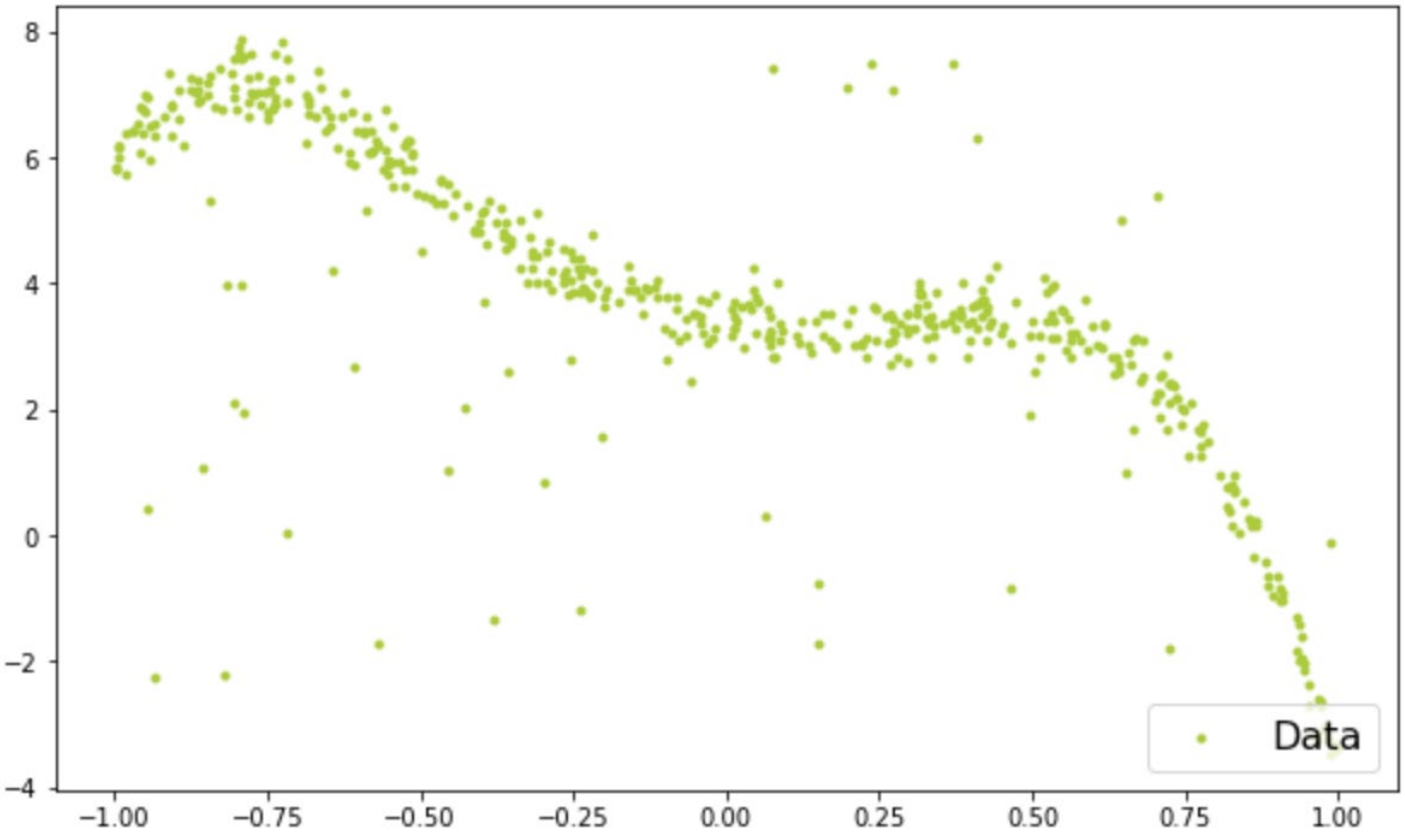
$$x = (1, x_1, \cdots, x_d)^T \rightarrow x' = (1, x'_1, \cdots, x'_d)^T$$

$$x'_j = \frac{x_j - \mu_j}{\sigma_j}$$

- For more details, you can check https://scikit-learn.org/stable/modules/preprocessing.html
  - For example, for data with outliers, you should use robust estimates for mean and covariance

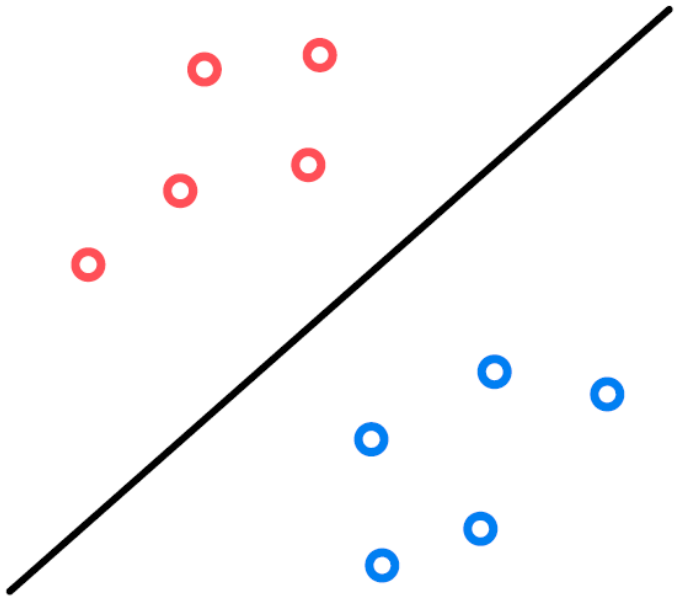
Exercise: Fit a polynomial model

- Link to Colab: https://drive.google.com/file/d/1G1N9C9Yry-3Lc9A1mL5yawM1rDB6TnKM
- Link to data: https://drive.google.com/file/d/1G_XseoB0lhII21jGdEkSYQyaEtMoSfQy



Linear Classification

- Classification: The output of the learned model is discrete
- Linear classification: The model is based on a linear combination of the input features
- Intuition: there is an hyperplane $\pi \in \mathbb{R}^{d-1}$ (e.g., a line for a 2D space) that separates the two classes/categories.

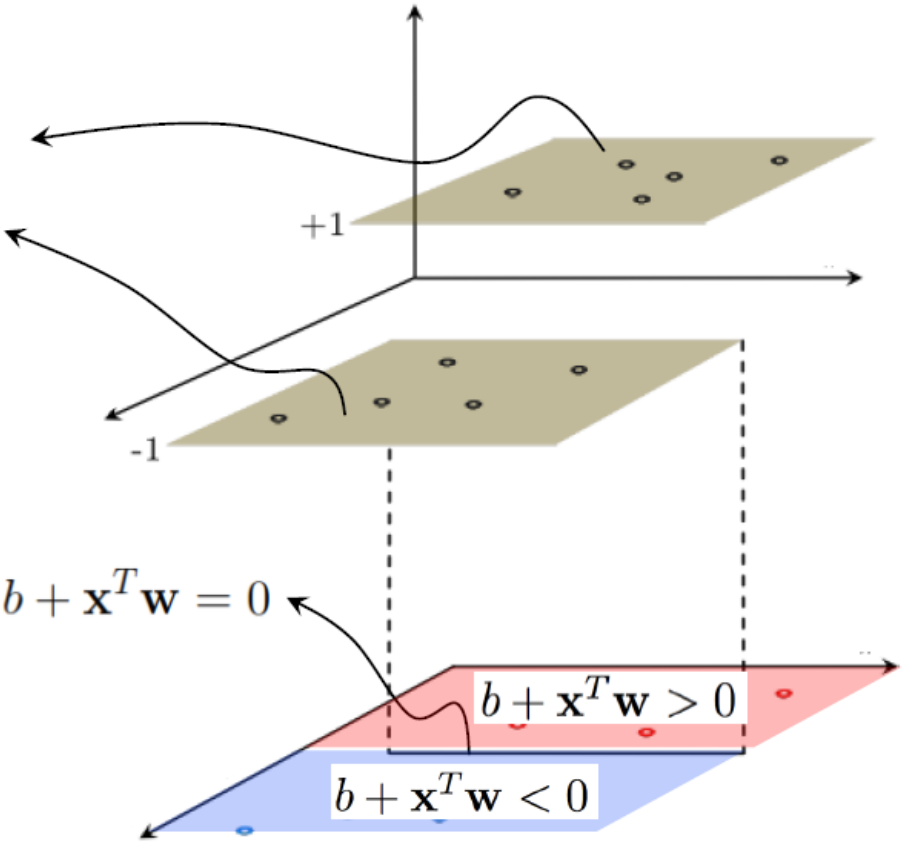


Binary Classification (intuition)

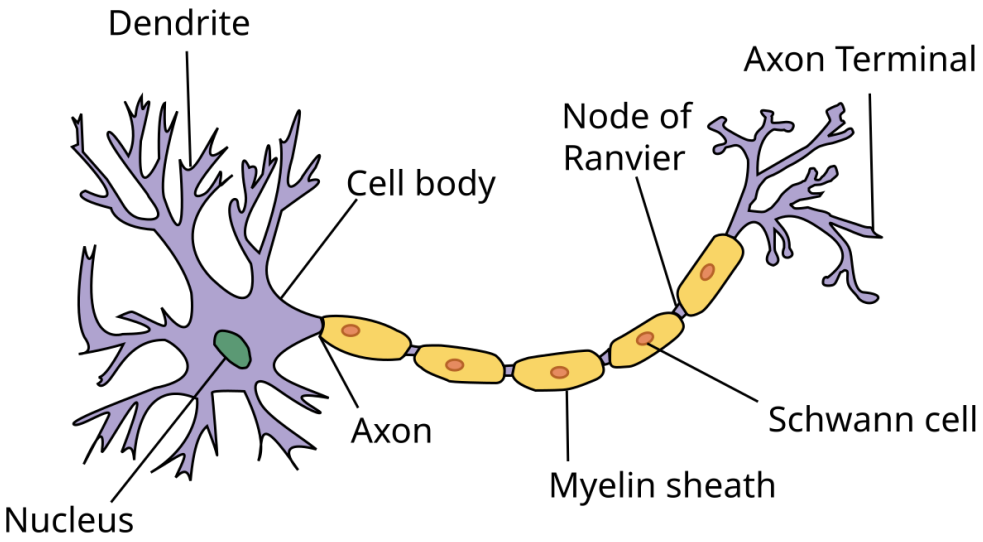
- We can get a discrete output $y \in \{-1, +1\}$ by taking the sign of the signed point-to-hyperplane distance
- The linear decision boundary is perpendicular to w
- The formulation can be extended easily to multi-class

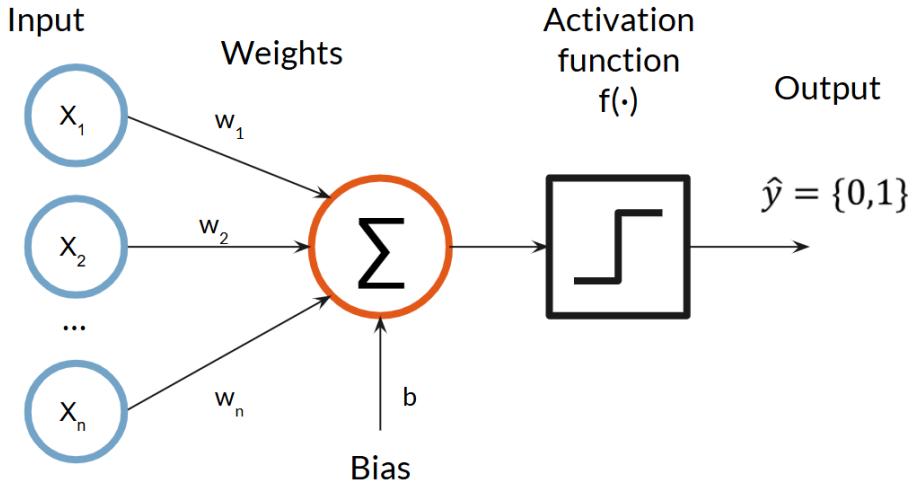
$$y(\mathbf{x}) = \text{sign}\left(b + \mathbf{x}^T\mathbf{w}\right)$$

Heaviside (step) function if $y \in \{0,1\}$



Perceptron (McCulloch-Pitts, 1943)

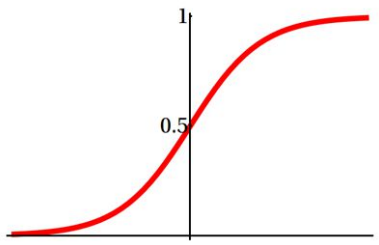




Activation functions

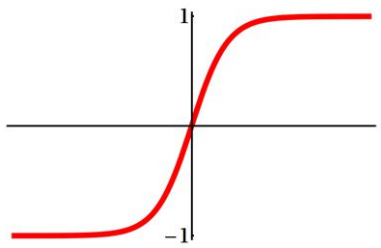
- Heaviside and sign() functions are not continuous.
  - Not differentiable!
- We can use continuous functions that approximate the sign function.
  - The sigmoid function
  - The tanh function
  - …

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

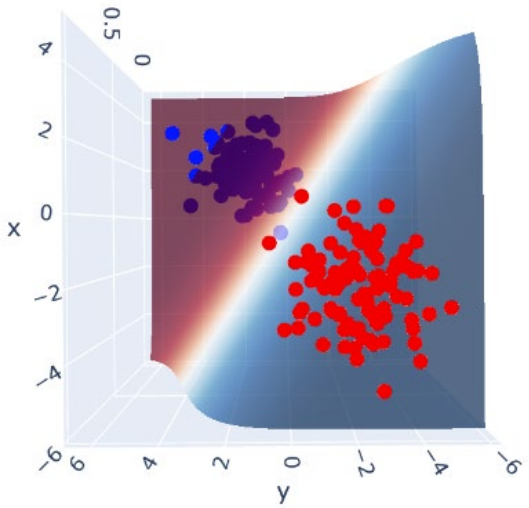


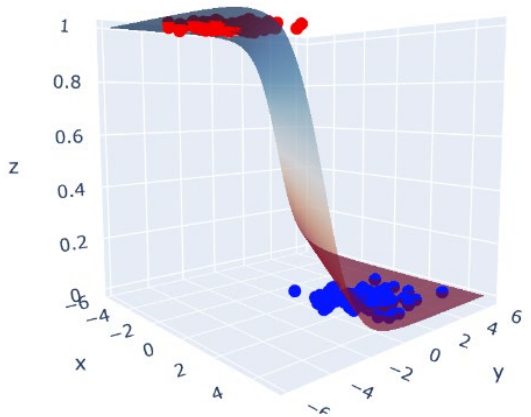
logistic (sigmoid, unipolar)

$$\tanh(\Sigma) = \frac{e^{\Sigma} - e^{-\Sigma}}{e^{\Sigma} + e^{-\Sigma}}$$



tanh (bipolar)





Logistic Regression (probabilistic interpretation)

- We can write the likelihood function as (see Math notes):

$$p(y|\boldsymbol{x}, w) = \text{Ber}(y|\lambda(\boldsymbol{x}))$$

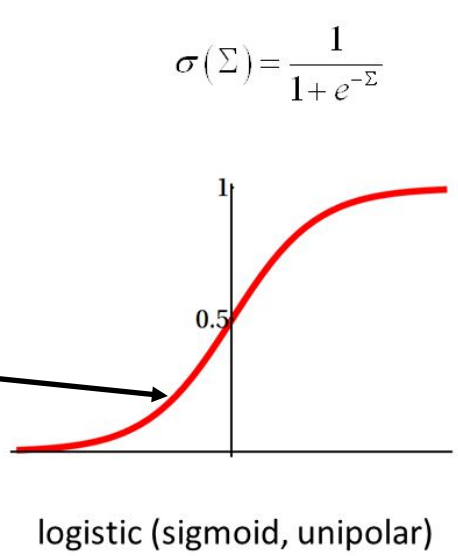
where $p(y = 1|\boldsymbol{x}) = \lambda(\boldsymbol{x})$ and $p(y = 0|\boldsymbol{x}) = 1 - \lambda(\boldsymbol{x})$

- The probability of the positive class is output of the sigmoid.

$$\lambda(\boldsymbol{x}) = \sigma(\boldsymbol{x}^\top w) = \frac{1}{1 + e^{-\boldsymbol{x}^\top w}}$$

Remember the trick to add a bias term $\phi(x) = [1, x]^T$

Probability of being class 1



logistic (sigmoid, unipolar)

Logistic Regression (probabilistic interpretation)

- We can write the likelihood function as (see Math notes):

$$p(y_i|\boldsymbol{x_i}, w) = \left[\frac{1}{1+e^{-\boldsymbol{x}_i^\top w}}\right]^{y_i}\left[1-\frac{1}{1+e^{-\boldsymbol{x}_i^\top w}}\right]^{1-y_i} = \lambda(\boldsymbol{x})^{y_i} + (1-\lambda(\boldsymbol{x}))^{1-y_i}$$

$$p(\boldsymbol{y}|X, w) = \prod_{i=1}^{N} p(y_i|\boldsymbol{x_i}, w)$$

assuming **iid data**.

- Then, we can compute logs:

$$\log p(y_i|\boldsymbol{x_i}, w) = y_i \log \lambda(\boldsymbol{x}) + (1-y_i)\log(1-\lambda(\boldsymbol{x}))$$

$$\log p(\boldsymbol{y}|X, w) = \sum_{i=1}^{N} \log p(y_i|\boldsymbol{x_i}, w)$$

Learning the parameters

- Remember maximum likelihood:

$$\hat{w} = \underset{w}{\operatorname{argmax}} \log p(\boldsymbol{y}|X, w) = \underset{w}{\operatorname{argmax}} \frac{1}{N} \sum_{i=1}^{N} \log p(y_i|\boldsymbol{x}_i, w)$$

- It is the same as minimize NLL:

$$\hat{w} = \underset{w}{\operatorname{argmin}} - \log p(\boldsymbol{y}|X, w) = \underset{w}{\operatorname{argmin}} \frac{1}{N} \sum_{i=1}^{N} -\log p(y_i|\boldsymbol{x}_i, w)$$

- What is $-\log p(y_i|\boldsymbol{x}_i, w)$? Cross-entropy! (Check Math notes)

$$-\log p(y_i|\boldsymbol{x}_i, w) = -y_i \log \lambda(\boldsymbol{x}) - (1 - y_i) \log(1 - \lambda(\boldsymbol{x}))$$

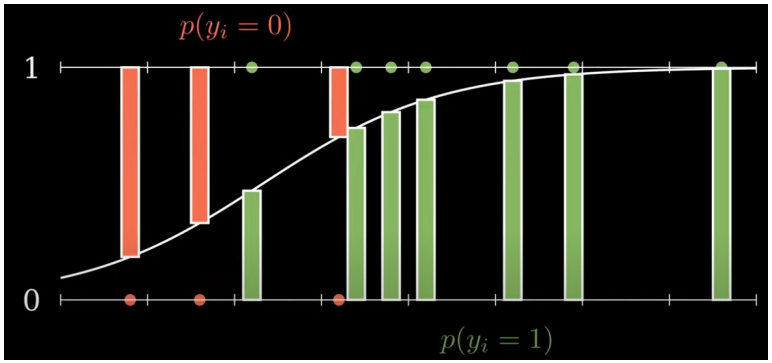
  - y_i is the **true** value (probability) of the class 1 $p(y_i = 1) = 1$
  - $\lambda(\boldsymbol{x})$ is **our model** probability of class 1
  - $\log \lambda(\boldsymbol{x})$ is **the information** of our model for class 1

Cross entropy vs ML

- Maximize likelihood = minimize cross-entropy
  - We want to find the model that minimizes the uncertainty of our predictions with respect to the *true world*.

$$\mathcal{L}(w) = -\frac{1}{N}\sum_{i=1}^{N} \log p(y_i|x_i, w) = -\left[\sum_{i=1}^{N} \underbrace{\frac{1}{N} y_i}_{\substack{true\ prob \\ class=1}} \log \lambda(x) + \sum_{i=1}^{N} \underbrace{\frac{1}{N}(1-y_i)}_{\substack{true\ prob \\ class=0}} \log(1-\lambda(x))\right]$$

- Intuition (check math notes):
  - Cross-entropy is the *expected surprise* of our model predictions averaged over the true probabilities of the elements *in the world*.
    - $-\log p(x)$ is the *surprise* of an event/observation.



Cross-entropy optimization

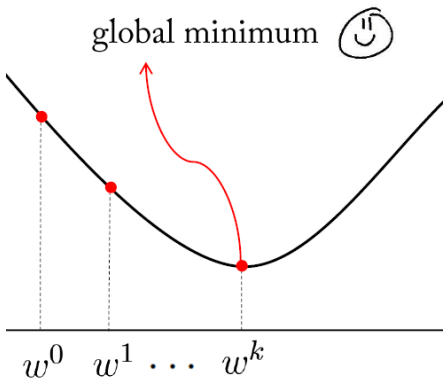
- To find the optimal model, we can use the same strategy as linear regression

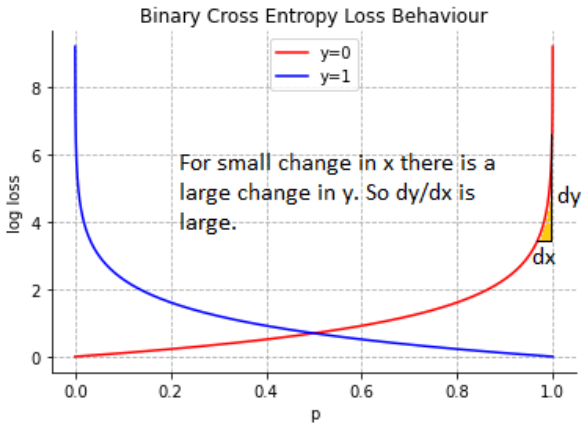
$$\mathcal{L}(w) = -\frac{1}{N}\sum_{i=1}^{N} y_i \log \lambda(\boldsymbol{x_i}) + (1 - y_i)\log(1 - \lambda(\boldsymbol{x_i}))$$

$$\nabla\mathcal{L}(w) = -\frac{1}{N}\sum_{i=1}^{N} \boldsymbol{x_i}^{\top}(\lambda(\boldsymbol{x_i}) - y_i) = -\frac{1}{N}\sum_{i=1}^{N} \boldsymbol{x_i}^{\top}(\sigma(\boldsymbol{x}^{\top}w) - y_i)$$

- Unfortunately, there is **no closed form** solution!
  - … but, the loss is **convex** and has a **single minimum**!
  - (Stochastic) Gradient Descent will take us to it.

Hessian has a quadratic form (positive definite)





Generalize to multi-class problems

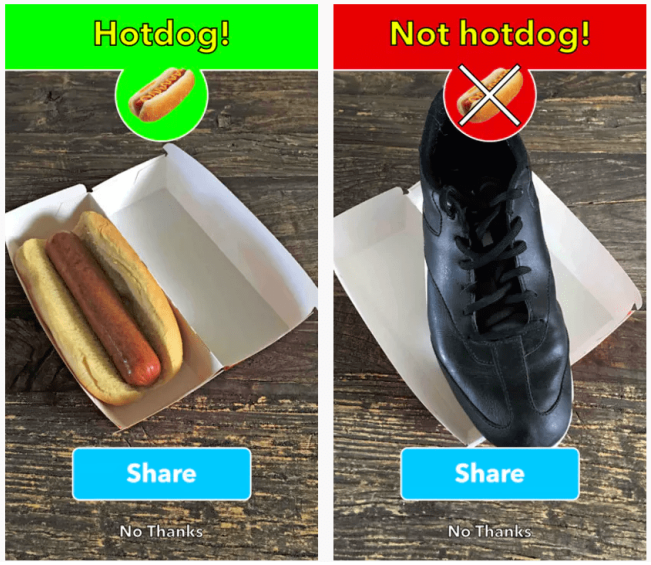
- Logistic regression works for binary classification problems

$$y = \{0,1\}$$

- What if we had multiple classes?

$$y = \{dog, cat, person, bed, \dots\}$$

- Solutions:
  - OVR: One-vs-Rest
    - Perform logistic regression for class 1 vs not class 1 (*hotdog/not hotdog*)
    - Repeat for all classes.
  - Replace Bernouilli → **Multinomial**
    - Distribution for ***k* categories** and *n* trials
    - It models throwing a k-sided dice n-times.
    - Multinomial(k=2, n=1) = Bernouilli
    - Bernouilli distribution models a coin flip.

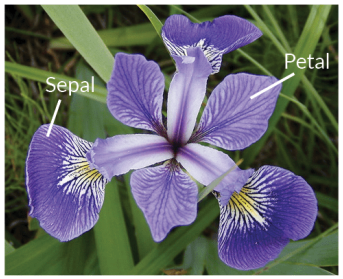


Multinomial logistic regression

Logistic regression in the Iris dataset

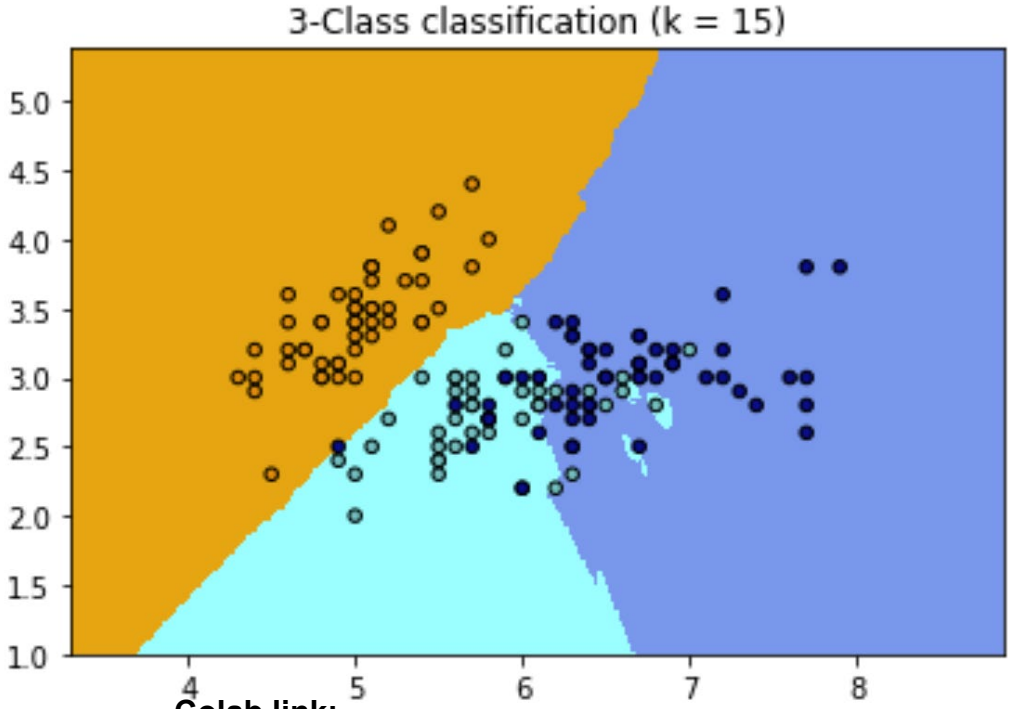
- Take-home messages:
  - Slightly outperforms k-nn (remember from our plots in slide 18, class boundaries are quite linear)
  - More sophisticated methods (SVM) do not improve, again class boundaries are quite linear
  - k-fold cross-validation IS relevant in this case!

Iris Setosa



Iris Versicolor

Iris Virginica



Colab link:
https://drive.google.com/file/d/1GftP9OQDIe6_7dP-AhIe_QgIZGiGPOjm

Welcome to deep learning

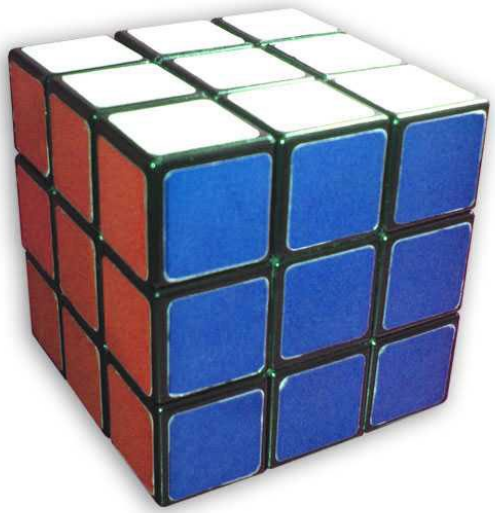
- Multilayer perceptron
  - Feedforward neural network
  - Trained using SGD
  - Forward *inference*, backward loss/gradient

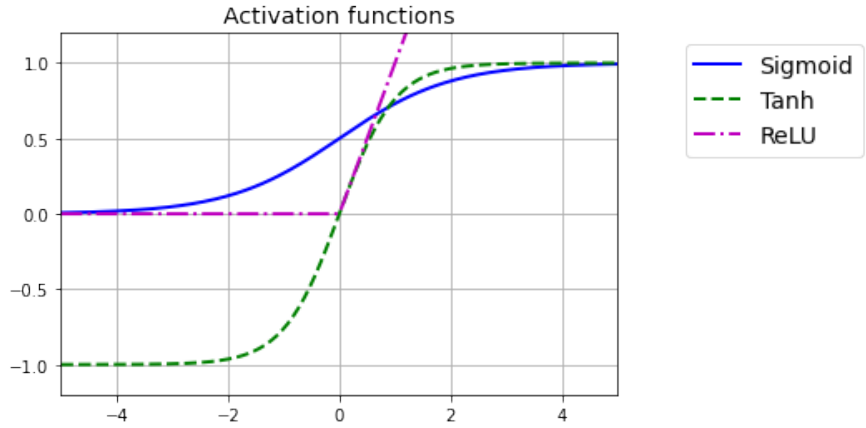
Welcome to deep learning

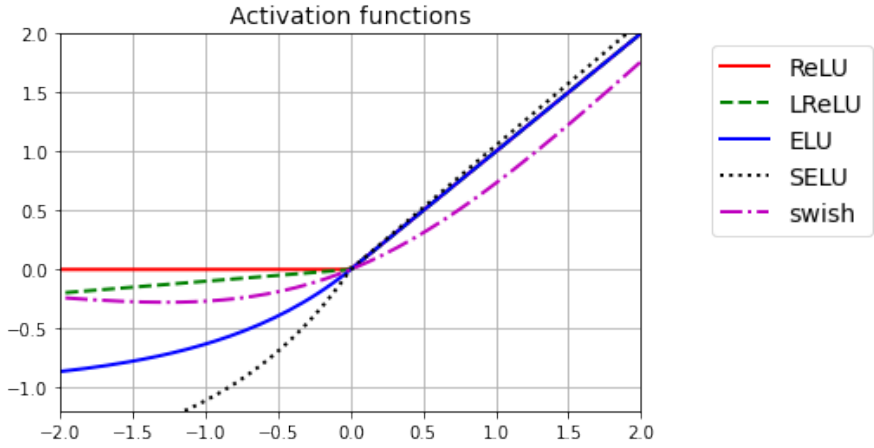
- General purpose architecture
  - Classification, regression
  - Unsupervised learning…

Activation functions

- Nonlinear component







https://colab.research.google.com/github/probml/pyprobml/blob/master/notebooks/book1/02/activation_fun_plot.ipynb

Types of error (binary classification)

- **True Positive (TP):** Positive data (correctly) classified as positive data.
- **True Negative (TN):** Negative data (correctly) classified as negative data.
- **False Positive / Type I Error (FP):** Negative data (not correctly) classified as positive data.
- **False Negative / Type II Error (FN):** Positive data (not correctly) classified as negative data.

	Ground Truth		
Prediction		Positive	Negative
	Positive	TP	FP
	Negative	FN	TN

Metrics: Precision and Recall

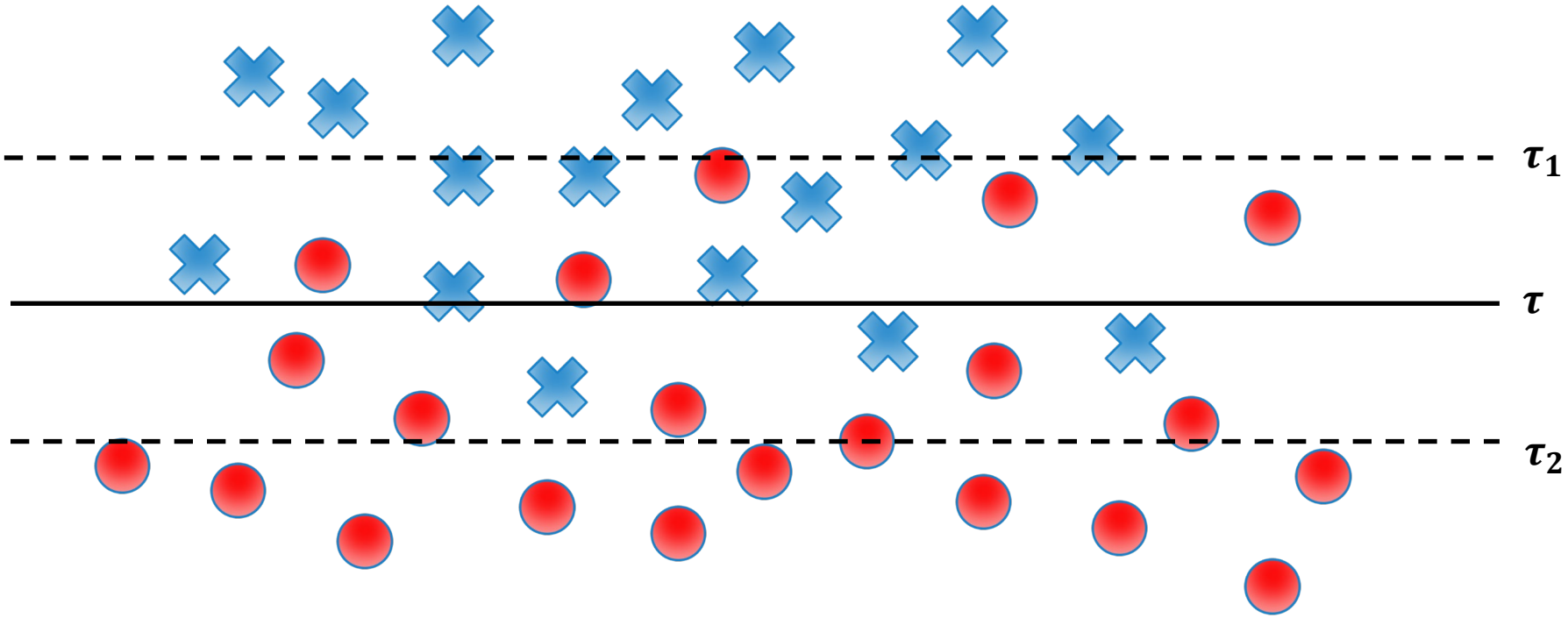
- **Accuracy, success rate, hit rate** $\rightarrow \frac{TP+TN}{TP+FP+TN+FN}$
  - Limitation: Not every error is equally important…
  - Best and worst success rate?
- **Precision** $\rightarrow \frac{TP}{TP+FP} \in [0,1]$
  - Ratio of the true positives over all the positive classifications.
  - Fraction of retrieved elements that are relevant.
- **Recall** $\rightarrow \frac{TP}{TP+FN} \in [0,1]$
  - Ratio of the true positives over all the positive samples.
  - Fraction of relevant elements that are retrieved.
- Both depend on the classification threshold.
- They are dependent, if one increases the other decreases. Sometimes given in pairs (e.g., precision at recall X).

Precision and Recall (some intuition)

- The girlfriend/boyfriend example:
  - Your girlfriend/boyfriend makes birthday presents for you every year.
  - One year she/he asks: Do you remember all my presents from the last 10 years?
  - **Precision** is how many are correct from the ones you said you remember.
  - **Recall** is how many correct you can remember from all correct ones.
  - How can we have 100% precision or recall? What happens to the other?
  - What is the relation to the classification threshold?
- The airport security example:
  - FP and FN: Have both errors the same consequences?
  - How can we have 100% precision or recall?
- Do you need precision, recall or both?
  - The solution is, most often, a compromise…

Precision-Recall and thresholds

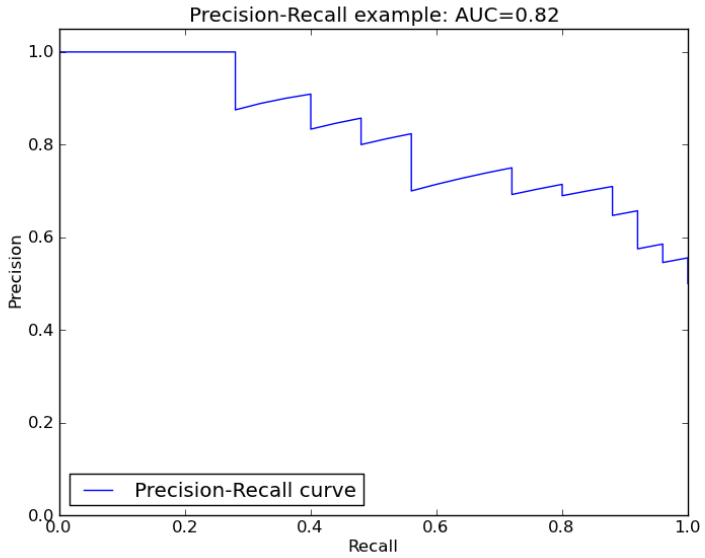
- Two-class problem



- What happens to the TP, TN, FP, FN, precision and recall if we move the boundary from τ to τ_1 or τ_2?

Precision-Recall curve

- 2D plot, recall in x-axis and precision in y-axis.
- It shows the precision-recall pairs when we change the classifier threshold τ
- $\tau \uparrow \Rightarrow FP \downarrow \Rightarrow FN \uparrow \Rightarrow P \rightarrow 1 \;\Rightarrow R \rightarrow 0$
- $\tau \downarrow \Rightarrow FP \uparrow \Rightarrow FN \downarrow \Rightarrow P \rightarrow 0 \;\Rightarrow R \rightarrow 1$
- Changing a classifier threshold τ it operates in different modes. We have to select the more appropriate for our purposes.
- For general purposes, the area under the curve (AUC) is usually taken as a general metric (the higher the better).
- What curve and AUC would have a perfect classifier? What curve and AUC would have the worst possible classifier?



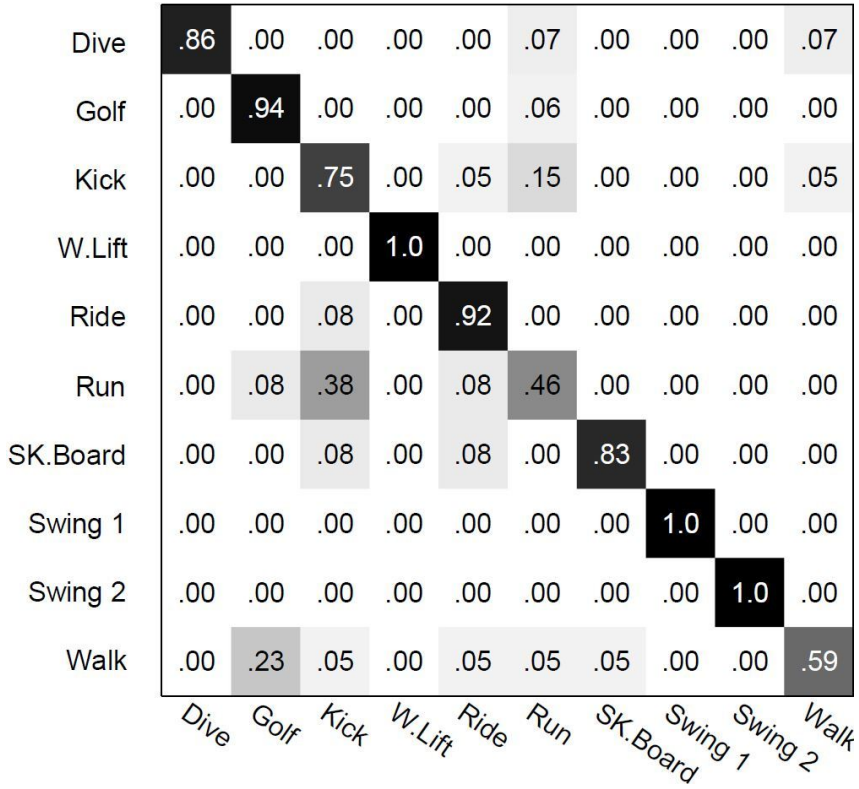
F1 score, confusion matrix

- **F1 score**
  - Harmonic mean of precision and recall (the higher the better)

$$F_1 = \frac{2}{{}^1/_P + {}^1/_R} = 2\frac{precision * recall}{precision + recall} \in [0,1]$$

  - It can be weighed to give β times more importance to precision than recall $F_1 = (1+\beta^2)\frac{precision * recall}{\beta^2 * precision + recall}$

- **Confusion matrix**
  - Rows: true categories.
  - Columns: predicted categories.
  - Cells: number/frequency of ocurrence.
  - More information than other metrics (for example, typical classification mistakes)
  - Difficult to display if many categories.
  - Difficult to summarize…

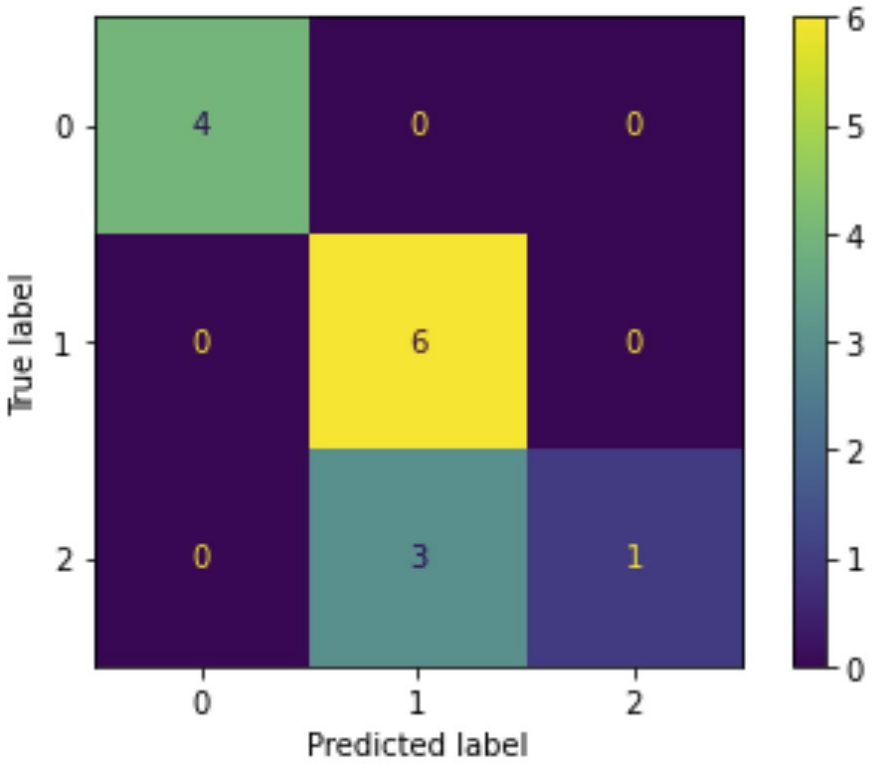


Confusion matrix for action recognition from videos

Taken from Qiu, Jiang, and Chellappa, "Sparse Dictionary-based Representation and Recognition of Human Action Attributes", ICCV 2011

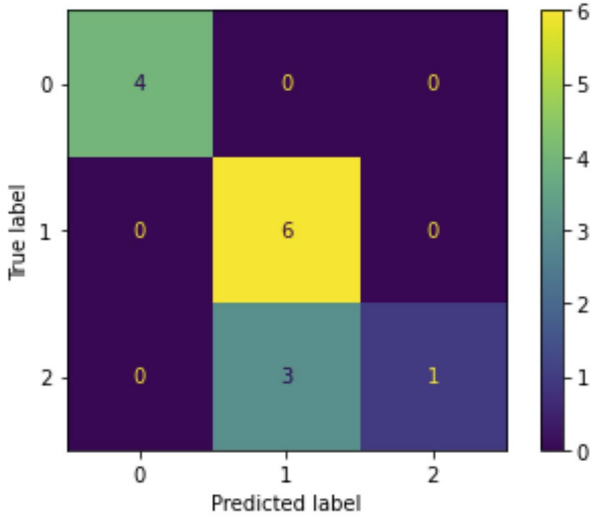
TP, TN, FP and FN from the confusion matrix

- Remember:
  - Rows: true categories.
  - Columns: predicted categories.
- **True positives per class:**
  - Diagonal elements [4, 6, 1]
- **False positives per class:**
  - Sum of the off-diagonal elements of each column [0, 3, 0]
- **False negatives per class:**
  - Sum of the off-diagonal elements of each row [0, 0, 3]
- **True negatives per class:**
  - Sum of all matrix elements except the row and column of the class [10, 5, 10]



Multiclass Precision/Recall

- **True positives per class:** [4, 6, 1], **True negatives per class:** [10, 5, 10]
- **False positives per class:** [0, 3, 0], **False negatives per class:** [0, 0, 3]



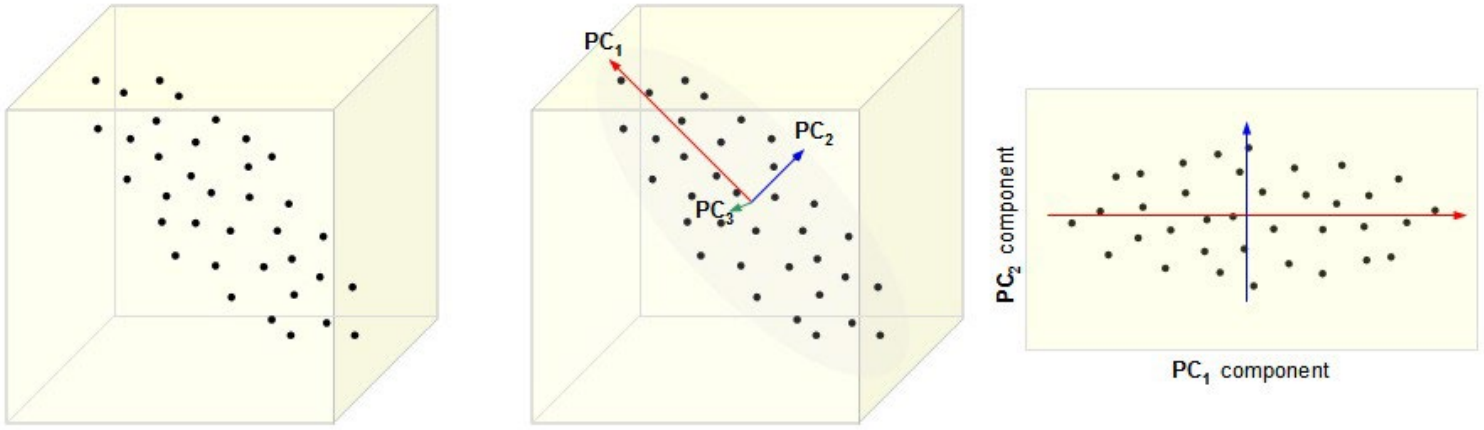
	Accuracy	Precision	Recall
Class 1	$\frac{4+10}{4+6+1+3} = \mathbf{1.0}$	$\frac{4}{4+0} = \mathbf{1.0}$	$\frac{4}{4+0} = \mathbf{1.0}$
Class 2	$\frac{6+5}{4+6+1+3} = \mathbf{0.79}$	$\frac{6}{6+3} = \mathbf{0.66}$	$\frac{6}{6+0} = \mathbf{1.0}$
Class 3	$\frac{1+10}{4+6+1+3} = \mathbf{0.79}$	$\frac{1}{1+0} = \mathbf{1.0}$	$\frac{1}{1+3} = \mathbf{0.25}$

- **Overall accuracy** $= \frac{correct}{total} = \frac{4+6+1}{4+6+1+3} = 0.78$
- **Average accuracy** $= \frac{1.0+0.79+0.79}{3} = 0.86$
- **Average Precision** $= \frac{1.0+0.66+1.0}{3} = 0.89$
- **Average Recall** $= \frac{1.0+1.0+0.25}{3} = 0.75$

If data is imbalanced, average metrics may be weighted…

Dimensionality reduction (unsupervised learning)

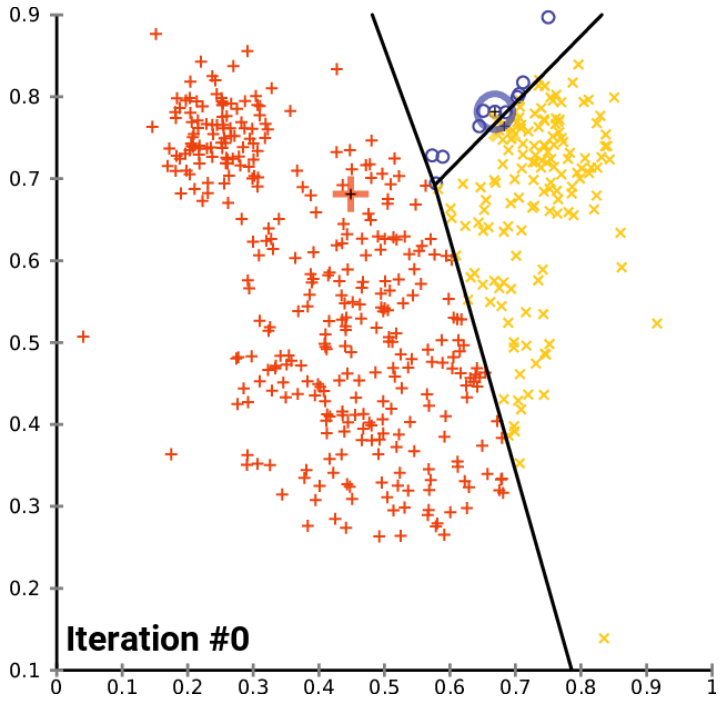
- PCA → Principal Component Analysis.
  - **Motivation:** in high dimensional data, some dimensions are redundant, or are highly correlated…
  - **Intuition:** Project the data into a low-dimensional space preserving the dimensions of maximum variance
  - SVD decomposition ($X = U\Sigma V^T$) of the training matrix $X = [x^{(1)} \quad \dots \quad x^{(N)}]$, the columns of V are the unit vectors of the principal components.
  - The low-d data is computed as $X_{d'} = XW_{d'}$; $W_{d'} \coloneqq V[1:d']$



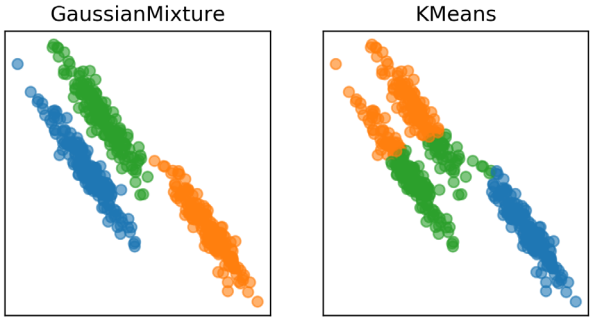
For more details, check Deisenroth chapter 10

Clustering (unsupervised learning)

- K-means:
  - Start with K random points (cluster means).
  - Repeat until convergence:
    - Assign each data point to the closest mean.
    - Recalculate means based on the new assignments



- Isotropic: clusters are spheres
  - Alternative: Gaussian mixture model (GMM) with expectation-maximization (EM)



- Requires knowing K
  - Alternative: X-means

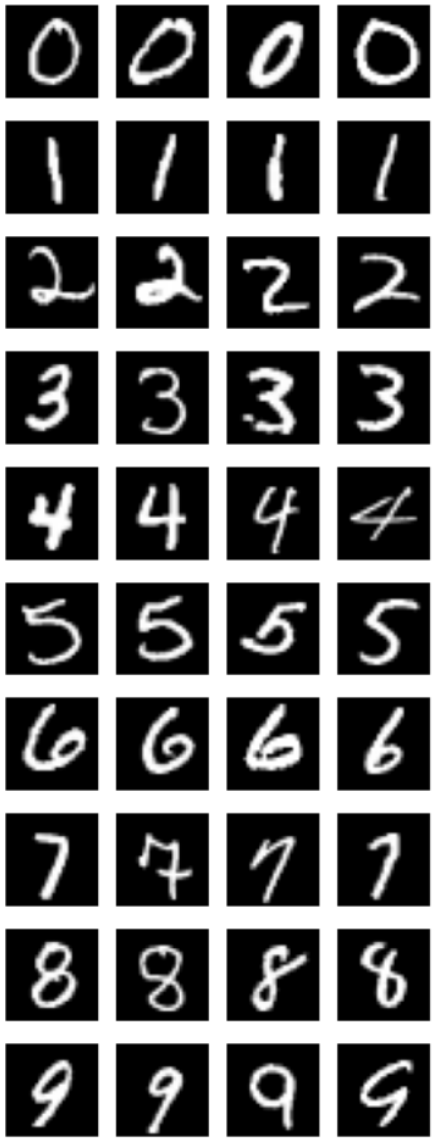
1-NN with and without PCA in MNIST

- Time per 10,000 queries in the test set
- Full-size input, $x \in \mathbb{R}^{784}$:

```
k=1, accuracy=97.22%, time=849.6 s
```

- Removing lowest-variance dimensions with PCA, up to 98% time reduction and same accuracy!

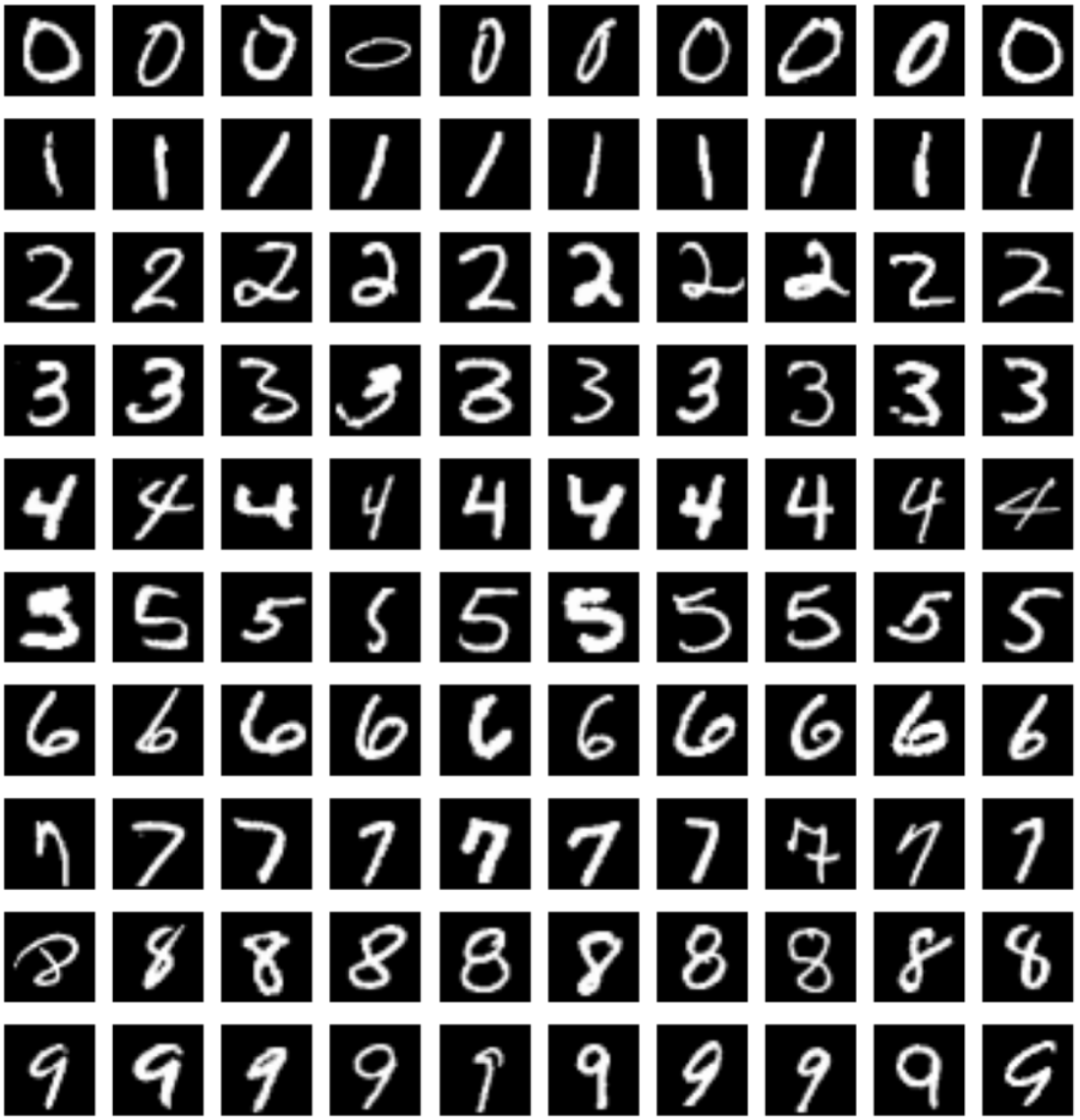
```
dims after PCA=10, accuracy=91.95%, time=1.3 s
dims after PCA=20, accuracy=96.60%, time=6.1 s
dims after PCA=30, accuracy=97.60%, time=14.8 s
dims after PCA=40, accuracy=97.43%, time=24.6 s
dims after PCA=50, accuracy=97.45%, time=36.3 s
dims after PCA=60, accuracy=97.56%, time=48.9 s
dims after PCA=70, accuracy=97.48%, time=62.0 s
dims after PCA=80, accuracy=97.51%, time=69.3 s
dims after PCA=90, accuracy=97.44%, time=81.7 s
dims after PCA=100, accuracy=97.54%, time=93.8 s
dims after PCA=200, accuracy=97.29%, time=159.8 s
dims after PCA=300, accuracy=97.24%, time=243.2 s
dims after PCA=400, accuracy=97.22%, time=318.2 s
dims after PCA=500, accuracy=97.24%, time=400.1 s
dims after PCA=600, accuracy=97.22%, time=473.3 s
```



Colab link:
https://colab.research.google.com/drive/1EdemME5vbZr01Q6CijYh0eo3hnRsjjKE

Assignment: Logistic regression in MNIST

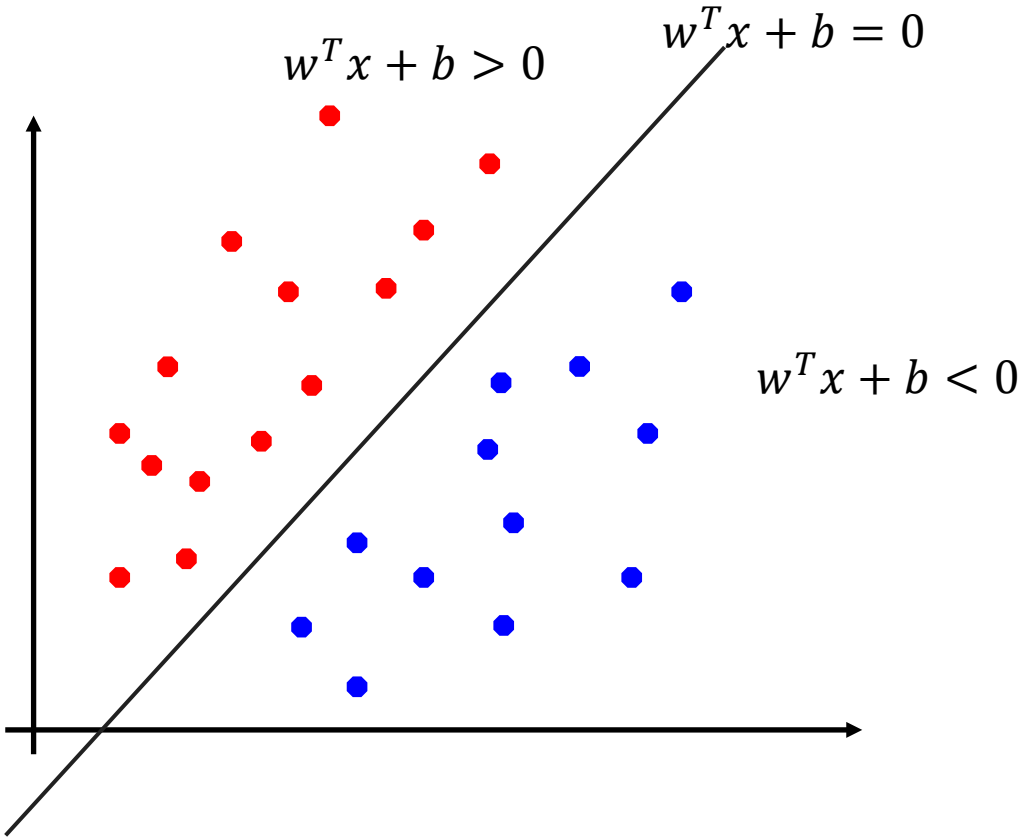
- Spoilers:
  - Excruciatingly slow!
    - Due to high dimensionality and large dataset size
    - Shortlist promising models using a reduced version of the data!
    - Use PCA!
  - 1-NN better than logistic regression…
    - why?



https://colab.research.google.com/drive/1HS9988xVWD6yqw9FRbe4z7qQfAFejL_V

Support Vector Machines (SVM)

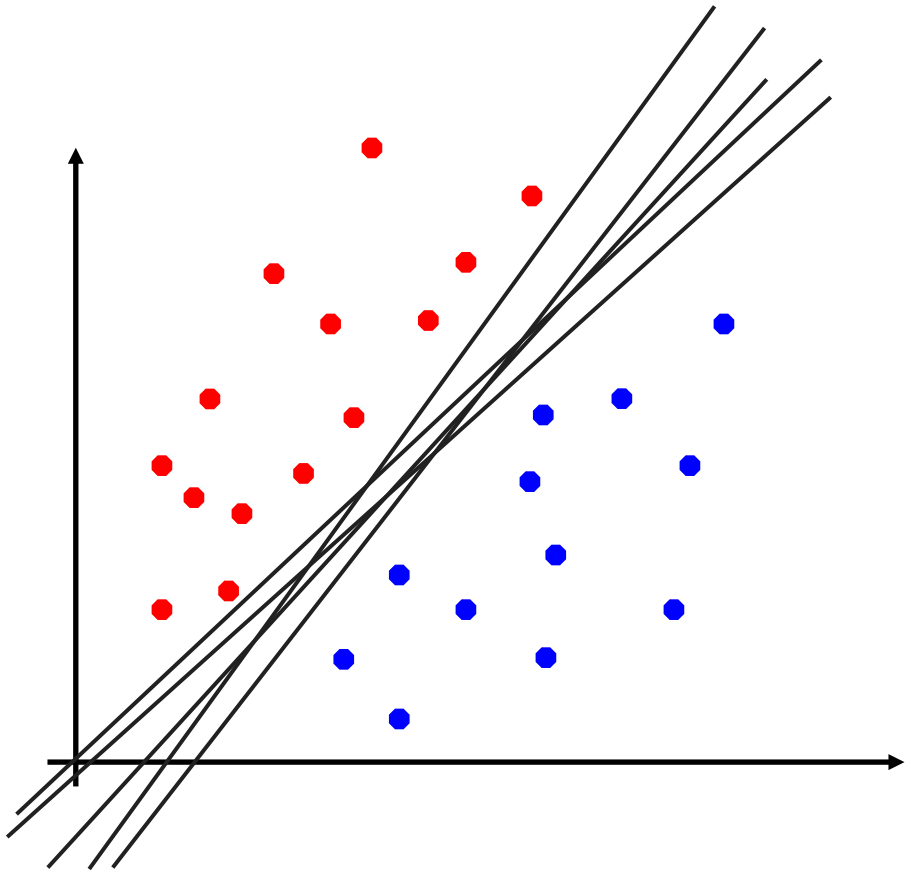
- Recap: Linear classification consists of finding an hyperplane in the feature space that separates the training data



For more details, check Deisenroth chapter 12

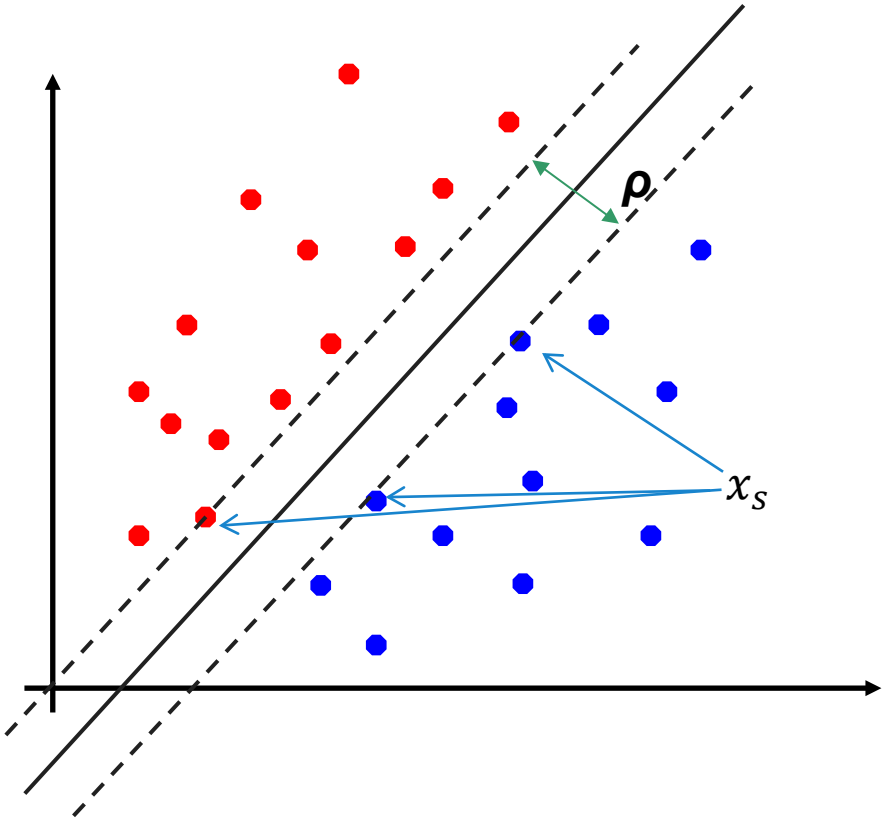
Support Vector Machines (SVM)

- But… which one of these hyperplanes would be the best one?



Support Vector Machines (SVM)

- SVM idea: let's take the one with **maximum margin** ρ
- Maximum margin:
  - The closest samples to the hyperplane are the support vectors x_s
  - The margin is the distance between the support vectors
  - Only support vectors matter! (which seems reasonable…)



"Hard-margin" SVM

- For each training example $(x^{(i)}, y^{(i)}), y^{(i)} \in \{-1, +1\}$

$$\begin{aligned} w^T x^{(i)} + b &\leq {}^{-\rho}/_{2} \quad if\ y^{(i)} = -1 \\ w^T x^{(i)} + b &\geq {}^{\rho}/_{2} \quad if\ y^{(i)} = +1 \end{aligned} \Rightarrow y^{(i)}\left(w^T x^{(i)} + b\right) \geq {}^{\rho}/_{2}$$

- If $x_s^{(i)}$ is a support vector, the above inequality is an equality
- After re-scaling w and b by ${}^{\rho}/_{2}$, the distance between each support vector $x_s^{(i)}$ and the hyperplane is ${}^{1}/_{\|w\|}$
- The margin is then $\rho = {}^{2}/_{\|w\|}$
- And the margin maximization can be formulated as

$$\hat{w}, \hat{b} = \arg max_{w,b}\ {}^{2}/_{\|w\|} \text{ (alternatively } \arg min_{w,b}\ \frac{1}{2}\|w\|^2\text{)}$$

subject to $y^{(i)}\left(w^T x^{(i)} + b\right) \geq 1$ (for re-scaled w and b)

"Soft-margin" SVM

- "Hard-margin SVM" does not allow data to be on the wrong side of the hyperplane
- For noisy data, we might want to allow that
- Idea: introduce a per-sample slack variable $\xi^{(i)} > 0$ modelling deviations from the correct side of the hyperplane
- The optimization is now

$$\hat{w}, \hat{b} = \arg min_{w,b} \frac{1}{2} \|w\|^2 + C \sum_i \xi^{(i)}$$

$$\text{subject to } y^{(i)}\left(w^T x^{(i)} + b\right) \geq 1 - \xi^{(i)} \text{(re-scaled } w \text{ and } b\text{)}$$

- C trades off the margin and the amount of slack we have
  - $C \downarrow\downarrow$ regularizes the solution (favors simple decision functions)
  - Usually tuned by cross-validation

Non-linear SVM

- The SVM parameters are learnt by solving the dual form of the optimization

$$\hat{\alpha} = \arg max_{\alpha}\ {}^{1}/_{2} \sum_{i}\sum_{j} y^{(i)} y^{(j)} \alpha^{(i)} \alpha^{(j)} \left\langle \boldsymbol{x}^{(i)}, \boldsymbol{x}^{(j)} \right\rangle - \sum_{i} \alpha^{(i)}$$

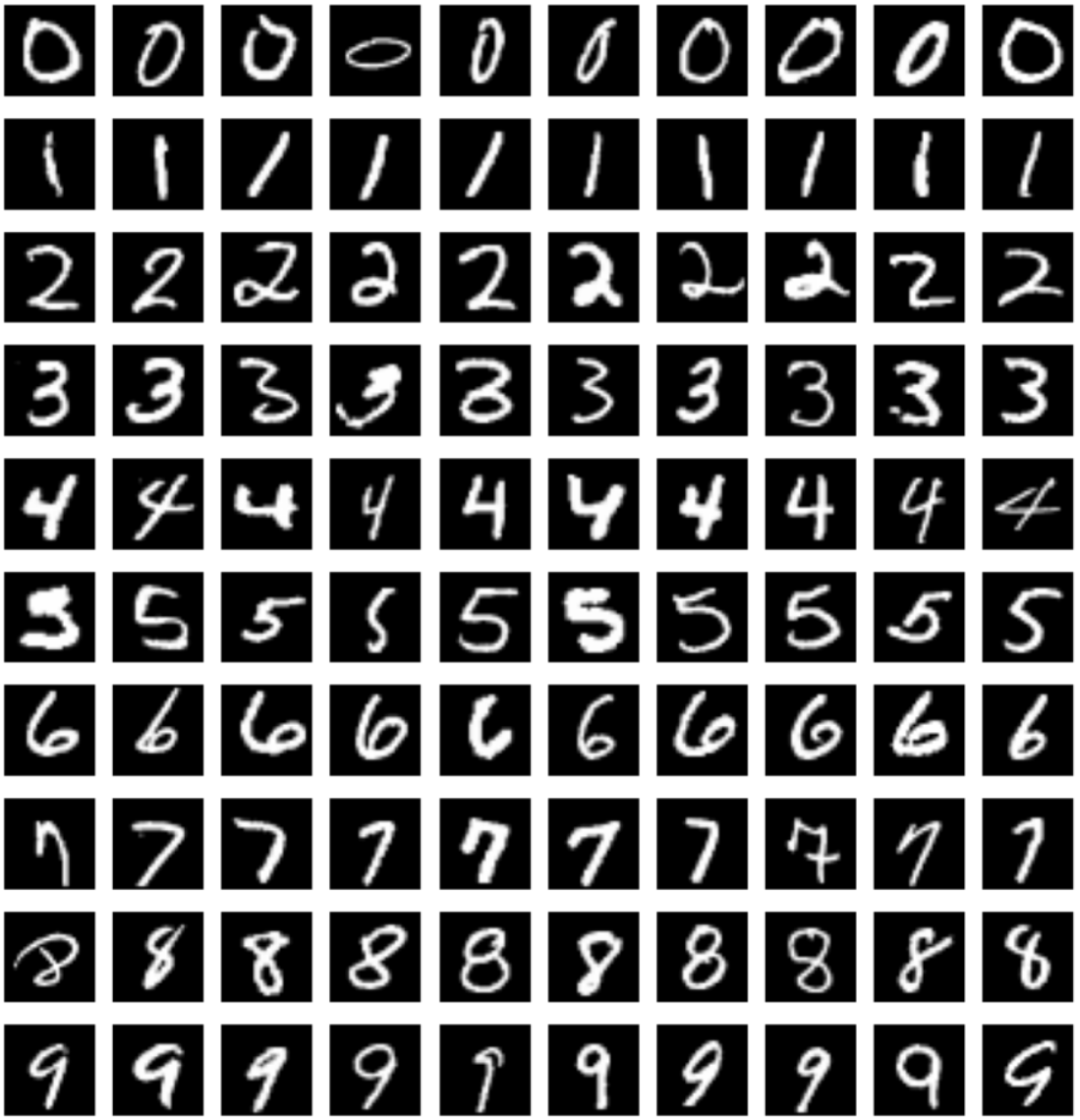
$$\text{subject to } \textstyle\sum_i y^{(i)} \alpha^{(i)} = 0,\ 0 \leq \alpha^{(i)} \leq \mathrm{C}$$

- The objective function is based in the inner product $\left\langle \boldsymbol{x}^{(i)}, \boldsymbol{x}^{(j)} \right\rangle$
- For non-linear functions of the data $\phi(x^{(i)})$, the only modification is on the inner product $\left\langle \boldsymbol{\phi}(\boldsymbol{x}^{(i)}), \boldsymbol{\phi}(\boldsymbol{x}^{(j)}) \right\rangle$
- Instead of choosing the non-linear function $\phi(x^{(i)})$, we choose the similarity function $k(x^{(i)}, x^{(j)}) = \langle \phi(x^{(i)}), \phi(x^{(j)}) \rangle_{\mathcal{H}}$
- The similarity function $k(x^{(i)}, x^{(j)})$ is a kernel (more about kernels later in the course…)

Non-linear SVM hyperparameters

- Obtained by cross-validation
- C parameter
  - Trades off the margin and the slack allowed to the data
- Kernel and kernel parameters. Usual choices
  - Linear kernel: $k\left(x^{(i)}, x^{(j)}\right) = x^{(i)}x^{(j)}$
  - Polynomial kernel: $k\left(x^{(i)}, x^{(j)}\right) = \left(\gamma x^{(i)}x^{(j)} + r\right)^d$
  - Gaussian kernel: $k\left(x^{(i)}, x^{(j)}\right) = \exp\left(-\frac{\left\|x^{(i)} - x^{(j)}\right\|^2}{\sigma^2}\right)$
  - Radial basis function: $k\left(x^{(i)}, x^{(j)}\right) = \exp\left(-\gamma\left\|x^{(i)} - x^{(j)}\right\|^2\right)$
  - Sigmoid kernel: $k\left(x^{(i)}, x^{(j)}\right) = \tanh\left(\gamma x^{(i)}x^{(j)} + r\right)$

Assignment: SVM in MNIST??



https://colab.research.google.com/drive/1HS9988xVWD6yqw9FRbe4z7qQfAFejL_V

Ensembling

- The machine learning equivalent to the wisdom of the crowd
  - Aggregating predictions from several models improve over the individual ones
- Example (from Géron 2019):
  - Aggregating the votes of 1,000 binary classifiers with individual accuracies of 51% gives us an accuracy of 75%!
- **Practical trick:** when, at the end of a machine learning project, you trained several good models, combine their predictions to obtain an even better one.

Aurélien Géron, Hands-on Machine Learning with Scikit-Learn, Keras and TensorFlow, O'Reilly, 2nd edition, 2019

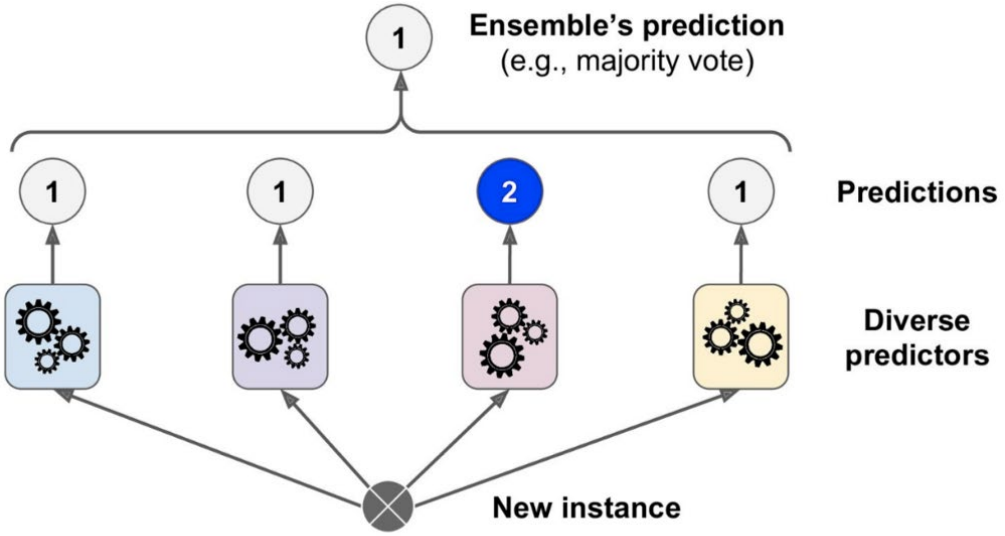


Figure 7-2. Hard voting classifier predictions

Universidad Zaragoza

Machine Learning (69152)

Máster in Robotics, Graphics and Computer Vision