

Large-Scale Machine Learning

Jean-Philippe Vert

jean-philippe.vert@{mines-paristech,curie,ens}.fr



Outline

- 1 Introduction
- 2 Standard machine learning
 - Dimension reduction: PCA
 - Clustering: k -means
 - Regression: ridge regression
 - Classification: kNN, logistic regression and SVM
 - Nonlinear models: kernel methods
- 3 Large-scale machine learning
 - Scalability issues
 - The tradeoffs of large-scale learning
 - Random projections
 - Random features
 - Approximate NN
 - Shingling, hashing, sketching
- 4 Conclusion

Acknowledgement

In the preparation of these slides I got inspiration and copied several slides from several sources:

- Sanjiv Kumar's "Large-scale machine learning" course:
<http://www.sanjivk.com/EECS6898/lectures.html>
- Ala Al-Fuqaha's "Data mining" course:
<https://cs.wmich.edu/alfuqaha/summer14/cs6530/lectures/SimilarityAnalysis.pdf>
- Léon Bottou's "Large-scale machine learning revisited" conference
<https://bigdata2013.sciencesconf.org/conference/bigdata2013/pages/bottou.pdf>

Outline

1 Introduction

2 Standard machine learning

- Dimension reduction: PCA
- Clustering: k -means
- Regression: ridge regression
- Classification: kNN, logistic regression and SVM
- Nonlinear models: kernel methods

3 Large-scale machine learning

- Scalability issues
- The tradeoffs of large-scale learning
- Random projections
- Random features
- Approximate NN
- Shingling, hashing, sketching

4 Conclusion

TECH**2017 is the year of Machine Learning. Here's why**

■ GAURAV SANGWANI | 0 | JAN 13, 2017, 12:51 PM

Facebook

LinkedIn

Twitter

Google+

Reddit



Machine learning is maybe the most sweltering thing in Silicon Valley at this moment. Particularly deep learning. The reason why it is so hot is on the grounds that it can assume control of numerous repetitive, thoughtless tasks. It'll improve doctors, and make lawyers better lawyers. What's more, it makes cars drive themselves.

Perception



Communication

The image shows a browser window at translate.google.fr. The Google logo is at the top left. Below it, the word "Traduction" is displayed in red. To the right, there is a link to "Désactiver la traduction instantanée" and a star icon. The language selection area shows "Français" selected for the target language and "Japonais - détecté" for the source language. A blue "Traduire" button is visible. The input text is "猿も木から落ちる" (Sarumokikaraochiru) and the output is "Même les singes tombent des arbres". The page footer contains links for "À propos de Google Traduction", "Communauté", "Mobile", "G+", "B", "À propos de Google", "Confidentialité et conditions d'utilisation", "Aide", and "Envoyer des commentaires".

Google

Traduction [Désactiver la traduction instantanée](#)

Anglais Français Arabe Japonais - détecté

Français Anglais Arabe Traduire

猿も木から落ちる

Même les singes tombent des arbres

Sarumokikaraochiru

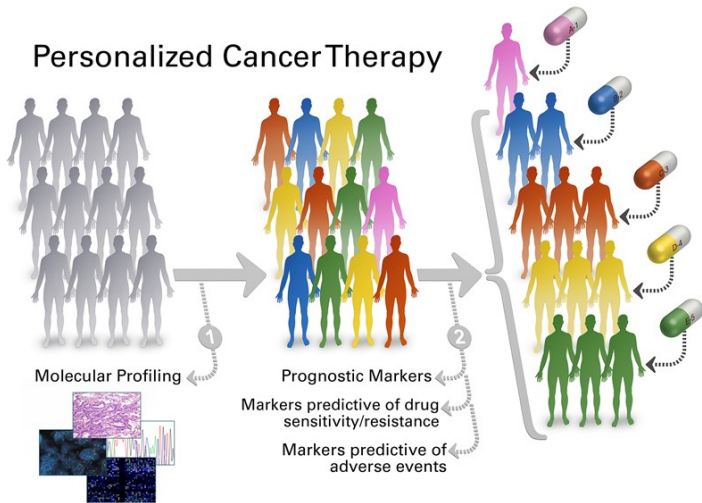
À propos de Google Traduction Communauté Mobile G+ B

À propos de Google Confidentialité et conditions d'utilisation Aide Envoyer des commentaires

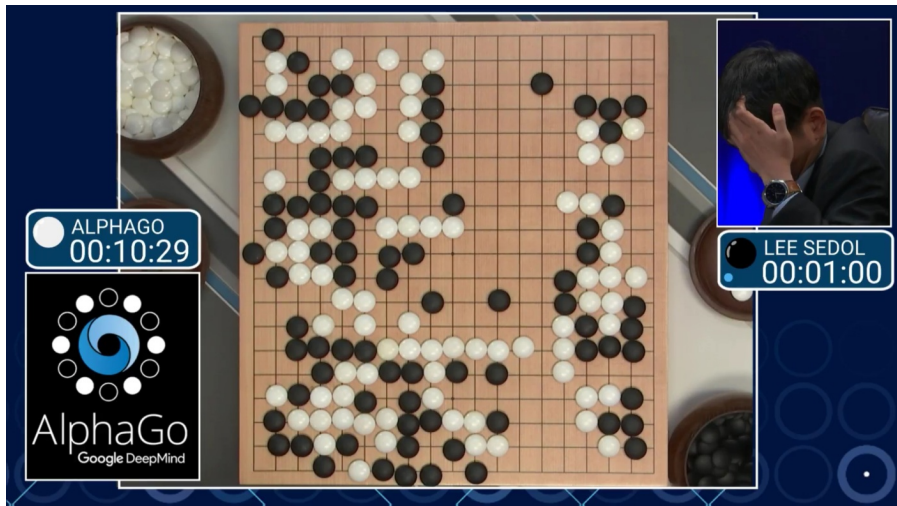
Mobility



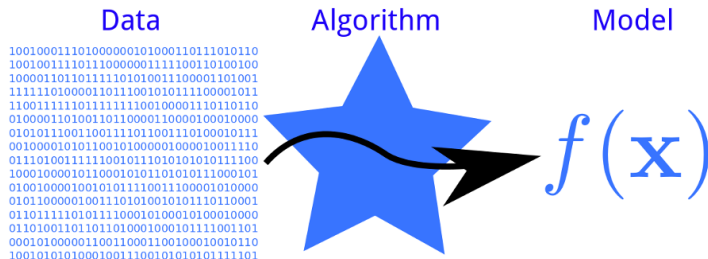
Personalized Cancer Therapy



Reasoning



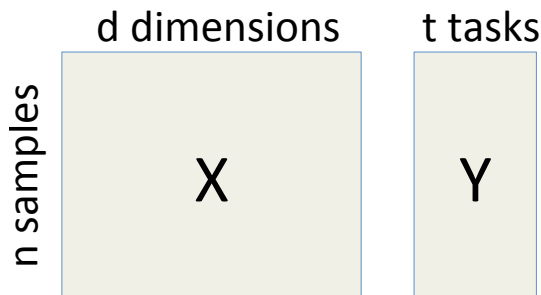
A common process: learning from data



<https://www.linkedin.com/pulse/supervised-machine-learning-pega-decisioning-solution-nizam-muhammad>

- Given examples (training data), make a machine learn how to predict on new samples, or discover patterns in data
- Statistics + optimization + computer science
- Gets better with more training examples and bigger computers

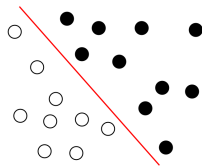
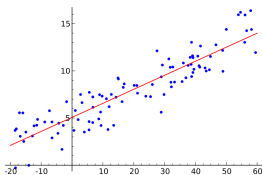
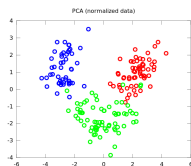
Large-scale ML?



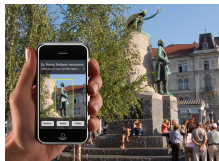
- Iris dataset: $n = 150, d = 4, t = 1$
- Cancer drug sensitivity: $n = 1k, d = 1M, t = 100$
- Imagenet: $n = 14M, d = 60k+, t = 22k$
- Shopping, e-marketing $n = O(M), d = O(B), t = O(100M)$
- Astronomy, GAFA, web... $n = O(B), d = O(B), t = O(B)$

Today's goals

1 Review a few standard ML techniques



2 Introduce a few ideas and techniques to scale them to modern, big datasets



Outline

1 Introduction

2 Standard machine learning

- Dimension reduction: PCA
- Clustering: k -means
- Regression: ridge regression
- Classification: kNN, logistic regression and SVM
- Nonlinear models: kernel methods

3 Large-scale machine learning

- Scalability issues
- The tradeoffs of large-scale learning
- Random projections
- Random features
- Approximate NN
- Shingling, hashing, sketching

4 Conclusion

Main ML paradigms

- Unsupervised learning
 - Dimension reduction
 - Clustering
 - Density estimation
 - Feature learning
- Supervised learning
 - Regression
 - Classification
 - Structured output classification
- Semi-supervised learning
- Reinforcement learning

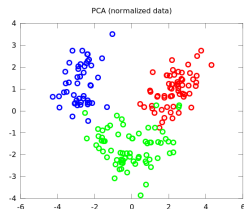
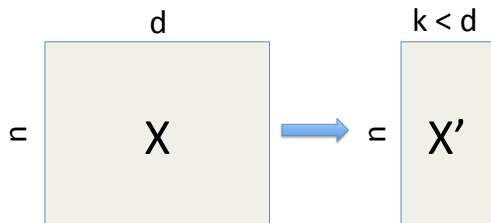
Main ML paradigms

- Unsupervised learning
 - Dimension reduction: PCA
 - Clustering: k-means
 - Density estimation
 - Feature learning
- Supervised learning
 - Regression: OLS, ridge regression
 - Classification: kNN, logistic regression, SVM
 - Structured output classification
- Semi-supervised learning
- Reinforcement learning

Outline

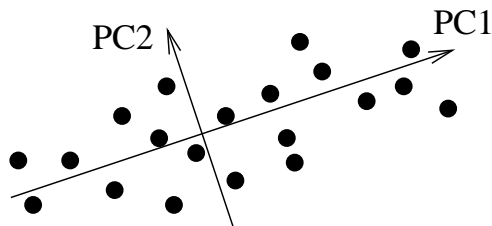
- 1 Introduction
- 2 Standard machine learning
 - Dimension reduction: PCA
 - Clustering: k -means
 - Regression: ridge regression
 - Classification: kNN, logistic regression and SVM
 - Nonlinear models: kernel methods
- 3 Large-scale machine learning
- 4 Conclusion

Motivation



- Dimension reduction
- Preprocessing (remove noise, keep signal)
- Visualization ($k = 2, 3$)
- Discover structure

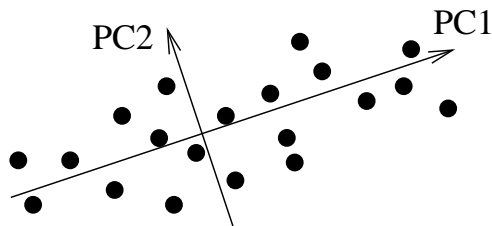
PCA definition



- Training set $\mathcal{S} = \{x_1, \dots, x_n\} \subset \mathbb{R}^d$
- For $i = 1, \dots, k \leq d$, PC_i is the linear projection onto the direction that captures the largest amount of variance and is orthogonal to the previous ones:

$$u_i \in \underset{\|u\|=1, u \perp \{u_1, \dots, u_{i-1}\}}{\operatorname{argmax}} \sum_{j=1}^n \left(x_j^\top u - \frac{1}{n} \sum_{j=1}^n x_j^\top u \right)^2$$

PCA solution

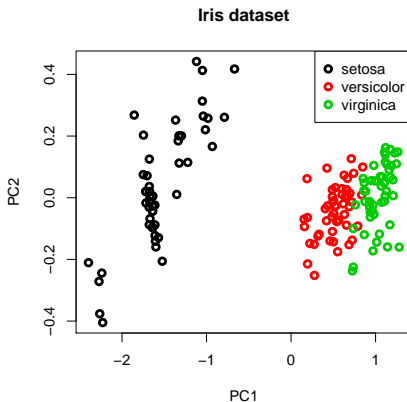


- Let \tilde{X} be the **centered** $n \times d$ data matrix
- PCA solves, for $i = 1, \dots, k \leq d$:

$$u_i \in \underset{\|u\|=1, u \perp \{u_1, \dots, u_{i-1}\}}{\operatorname{argmax}} \quad u^\top \tilde{X}^\top \tilde{X} u$$

- Solution: u_i is the i -th eigenvector of $C = \tilde{X}^\top \tilde{X}$, the empirical covariance matrix

PCA example



```
> data(iris)
> head(iris, 3)
  Sepal.Length Sepal.Width Petal.Length Petal.Width Species
1          5.1          3.5          1.4          0.2  setosa
2          4.9          3.0          1.4          0.2  setosa
3          4.7          3.2          1.3          0.2  setosa
> m <- princomp(log(iris[,1:4]))
```

PCA complexity

- Memory: store X and C : $O(\max(nd, d^2))$
- Compute C : $O(nd^2)$
- Compute k eigenvectors of C (power method): $O(kd^2)$

Computing C is more expensive than computing its eigenvectors ($n > k$)!

$n = 1B, d = 100M$

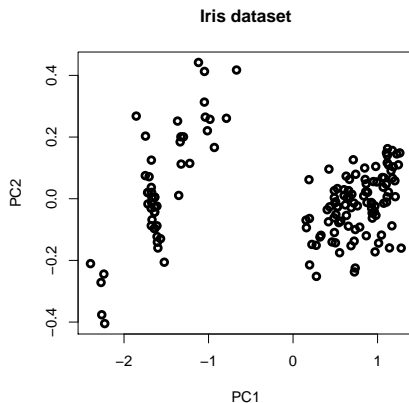
Store C : **40,000TB**

Compute C : $2 \times 10^{25} FLOPS = 20yottaFLOPS$ (about 300 years of the most powerful supercomputer in 2016)

Outline

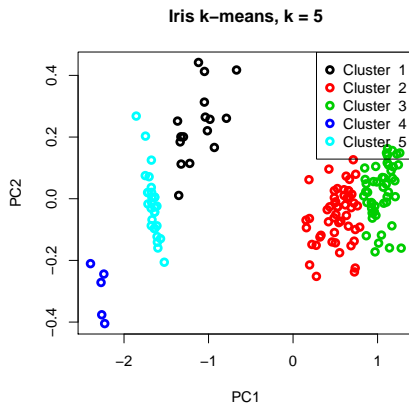
- 1 Introduction
- 2 Standard machine learning
 - Dimension reduction: PCA
 - **Clustering: k -means**
 - Regression: ridge regression
 - Classification: kNN, logistic regression and SVM
 - Nonlinear models: kernel methods
- 3 Large-scale machine learning
- 4 Conclusion

Motivation



- Unsupervised learning
- Discover groups
- Reduce dimension

Motivation



- Unsupervised learning
- Discover groups
- Reduce dimension

k -means definition

- Training set $\mathcal{S} = \{x_1, \dots, x_n\} \subset \mathbb{R}^d$
- Given k , find $C = (C_1, \dots, C_n) \in \{1, k\}^n$ that solves

$$\min_C \sum_{i=1}^n \|x_i - \mu_{C_i}\|^2$$

where μ_i is the barycentre of data in class i .

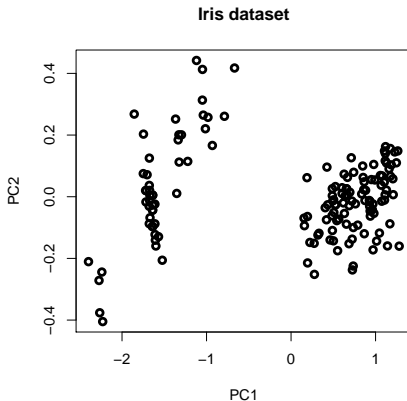
- This is an NP-hard problem. k -means finds an approximate solution by iterating
 - 1 Assignment step: fix μ , optimize C

$$\forall i = 1, \dots, n, \quad C_i \leftarrow \arg \min_{c \in \{1, \dots, k\}} \|x_i - \mu_c\|$$

- 2 Update step

$$\forall i = 1, \dots, k, \quad \mu_i \leftarrow \frac{1}{|C_i|} \sum_{j: C_j=i} x_j$$

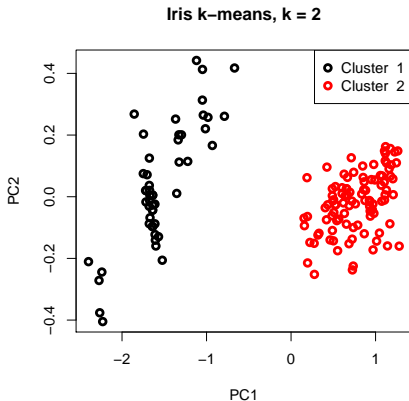
k-means example



```
> irisCluster <- kmeans(log(iris[, 1:4]), 3, nstart = 20)
> table(irisCluster$cluster, iris$Species)
```

	setosa	versicolor	virginica
1	0	48	4
2	50	0	0
3	0	2	46

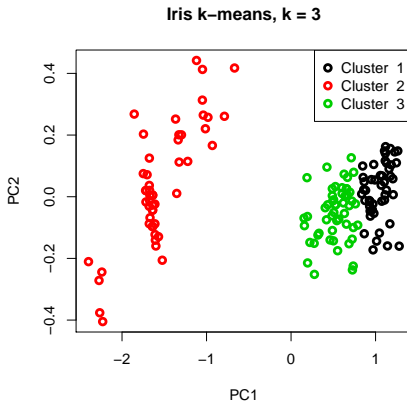
k-means example



```
> irisCluster <- kmeans(log(iris[, 1:4]), 3, nstart = 20)
> table(irisCluster$cluster, iris$Species)
```

	setosa	versicolor	virginica
1	0	48	4
2	50	0	0
3	0	2	46

k-means example

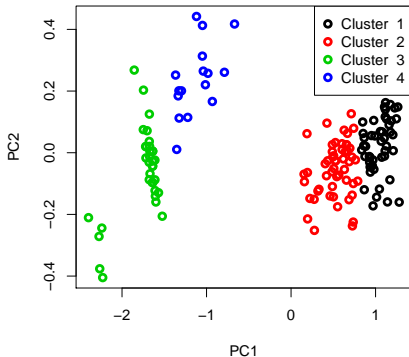


```
> irisCluster <- kmeans(log(iris[, 1:4]), 3, nstart = 20)
> table(irisCluster$cluster, iris$Species)
```

	setosa	versicolor	virginica
1	0	48	4
2	50	0	0
3	0	2	46

k-means example

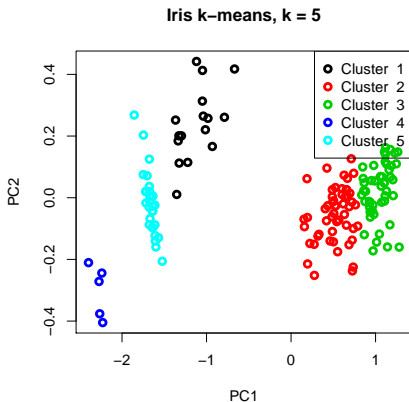
Iris k-means, k = 4



```
> irisCluster <- kmeans(log(iris[, 1:4]), 3, nstart = 20)
> table(irisCluster$cluster, iris$Species)
```

	setosa	versicolor	virginica
1	0	48	4
2	50	0	0
3	0	2	46

k-means example



```
> irisCluster <- kmeans(log(iris[, 1:4]), 3, nstart = 20)
> table(irisCluster$cluster, iris$Species)
```

	setosa	versicolor	virginica
1	0	48	4
2	50	0	0
3	0	2	46

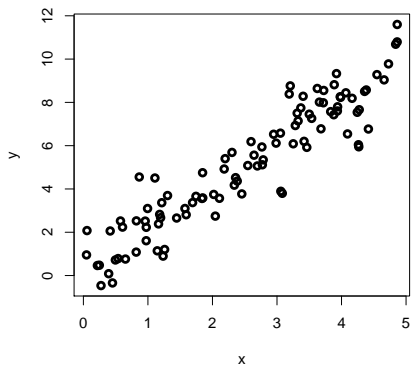
k -means complexity

- Each update step: $O(nd)$
- Each assignment step: $O(ndk)$

Outline

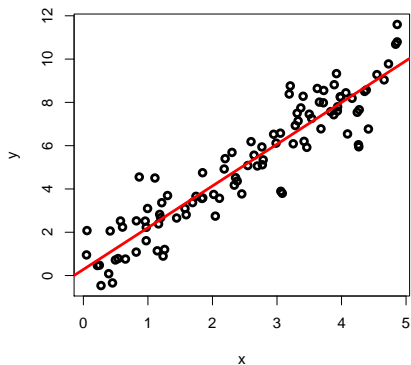
- 1 Introduction
- 2 **Standard machine learning**
 - Dimension reduction: PCA
 - Clustering: k -means
 - **Regression: ridge regression**
 - Classification: kNN, logistic regression and SVM
 - Nonlinear models: kernel methods
- 3 Large-scale machine learning
- 4 Conclusion

Motivation



- Predict a continuous output from an input

Motivation



- Predict a continuous output from an input

Model

- Training set $\mathcal{S} = \{(x_1, y_1), \dots, (x_n, y_n)\} \subset \mathbb{R}^d \times \mathbb{R}$
- Fit a linear function:

$$f_{\beta}(x) = \beta^{\top} x$$

- Goodness of fit measured by residual sum of squares:

$$RSS(\beta) = \sum_{i=1}^n (y_i - f_{\beta}(x_i))^2$$

- Ridge regression minimizes the regularized RSS:

$$\min_{\beta} RSS(\beta) + \lambda \sum_{i=1}^d \beta_i^2$$

- Solution (set gradient to 0):

$$\hat{\beta} = (X^{\top} X + \lambda I)^{-1} X^{\top} Y$$

Ridge regression complexity

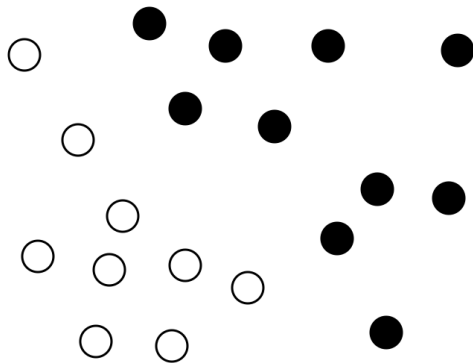
- Compute $X^T X$: $O(nd^2)$
- Inverse $(X^T X + \lambda I)$: $O(d^3)$

Computing $X^T X$ is more expensive than inverting it!

Outline

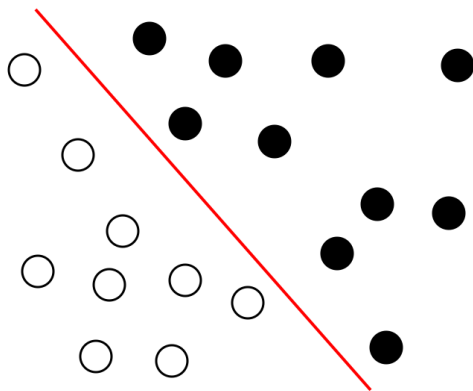
- 1 Introduction
- 2 **Standard machine learning**
 - Dimension reduction: PCA
 - Clustering: k -means
 - Regression: ridge regression
 - **Classification: kNN, logistic regression and SVM**
 - Nonlinear models: kernel methods
- 3 Large-scale machine learning
- 4 Conclusion

Motivation



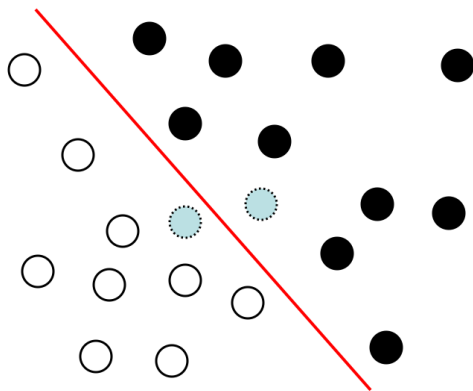
- Predict the category of a data
- 2 or more (sometimes many) categories

Motivation



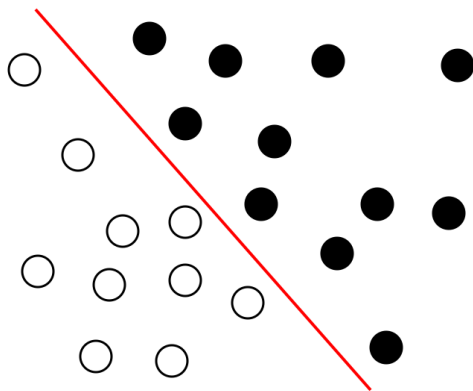
- Predict the category of a data
- 2 or more (sometimes many) categories

Motivation



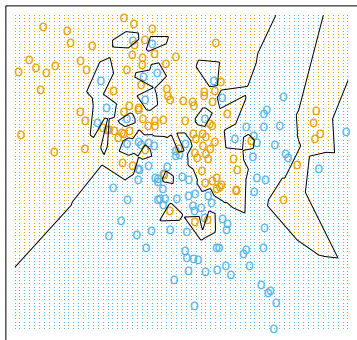
- Predict the category of a data
- 2 or more (sometimes many) categories

Motivation



- Predict the category of a data
- 2 or more (sometimes many) categories

k -nearest neighbors (kNN)



(Hastie et al. *The elements of statistical learning*. Springer, 2001.)

- Training set $\mathcal{S} = \{(x_1, y_1), \dots, (x_n, y_n)\} \subset \mathbb{R}^d \times \{-1, 1\}$
- No training
- Given a new point $x \in \mathbb{R}^d$, predict the majority class among its k nearest neighbors (take k odd)

kNN properties

Uniform Bayes consistency [Stone, 1977]

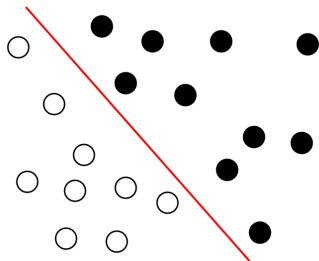
- Take $k = \sqrt{n}$ (for example)
- Let P be any distribution over (X, Y) pairs
- Assume training data are random pairs sampled i.i.d. according to P
- Then the k -NN classifier \hat{f}_n satisfies almost surely:

$$\lim_{n \rightarrow +\infty} P(\hat{f}_n(X) \neq Y) = \inf_{f \text{ measurable}} P(f(X) \neq Y)$$

Complexity:

- Memory: store X is $O(nd)$
- Training time: 0
- Prediction: $O(nd)$ for each test point

Linear models for classification



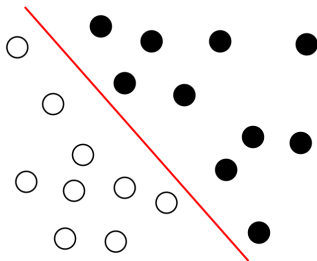
- Training set $\mathcal{S} = \{(x_1, y_1), \dots, (x_n, y_n)\} \subset \mathbb{R}^d \times \{-1, 1\}$
- Fit a linear function

$$f_{\beta}(x) = \beta^{\top} x$$

- The prediction on a new point $x \in \mathbb{R}^d$ is:

$$\begin{cases} +1 & \text{if } f_{\beta}(x) > 0, \\ -1 & \text{otherwise.} \end{cases}$$

Large-margin classifiers



- For any $f : \mathbb{R}^d \rightarrow \mathbb{R}$, the **margin** of f on an (x, y) pair is

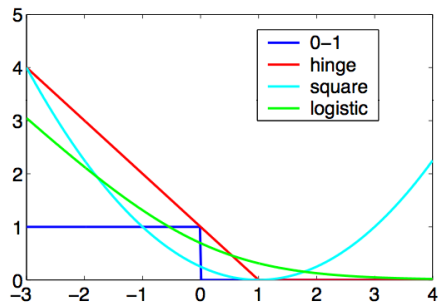
$$yf(x)$$

- Large-margin classifiers fit a classifier by maximizing the margins on the training set:

$$\min_{\beta} \sum_{i=1}^n \ell(y_i f_{\beta}(x_i)) + \lambda \beta^T \beta$$

for a convex, non-increasing loss function $\ell : \mathbb{R} \rightarrow \mathbb{R}_+$

Loss function examples



Loss	Method	$\ell(u)$
0-1	none	$1(u \leq 0)$
Hinge	Support vector machine (SVM)	$\max(1-u, 0)$
Logistic	Logistic regression	$\log(1+e^{-u})$
Square	Ridge regression	$(1-u)^2$

Ridge logistic regression [Le Cessie and van Houwelingen, 1992]

$$\min_{\beta \in \mathbb{R}^p} J(\beta) = \sum_{i=1}^n \ln \left(1 + e^{-y_i \beta^\top x_i} \right) + \lambda \beta^\top \beta$$

- Can be interpreted as a regularized conditional maximum likelihood estimator
- No explicit solution, but smooth convex optimization problem that can be solved numerically by Newton-Raphson iterations:

$$\beta^{new} \leftarrow \beta^{old} - \left[\nabla_{\beta}^2 J \left(\beta^{old} \right) \right]^{-1} \nabla_{\beta} J \left(\beta^{old} \right) .$$

- Each iteration amounts to solving a weighted ridge regression problem, hence the name iteratively reweighted least squares (IRLS).
- Complexity $O(\textit{iterations} * (nd^2 + d^3))$

SVM [Boser et al., 1992]

$$\min_{\beta \in \mathbb{R}^p} \sum_{i=1}^n \max(0, 1 - y_i \beta^\top x_i) + \lambda \beta^\top \beta$$

- A non-smooth convex optimization problem (convex quadratic program)
- Equivalent to the dual problem

$$\max_{\alpha \in \mathbb{R}^n} 2\alpha^\top Y - \alpha^\top X X^\top \alpha \quad \text{s.t.} \quad 0 \leq \mathbf{y}_i \alpha_i \leq \frac{1}{2\lambda} \text{ for } i = 1, \dots, n$$

- The solution β^* of the primal is obtained from the solution α^* of the dual:

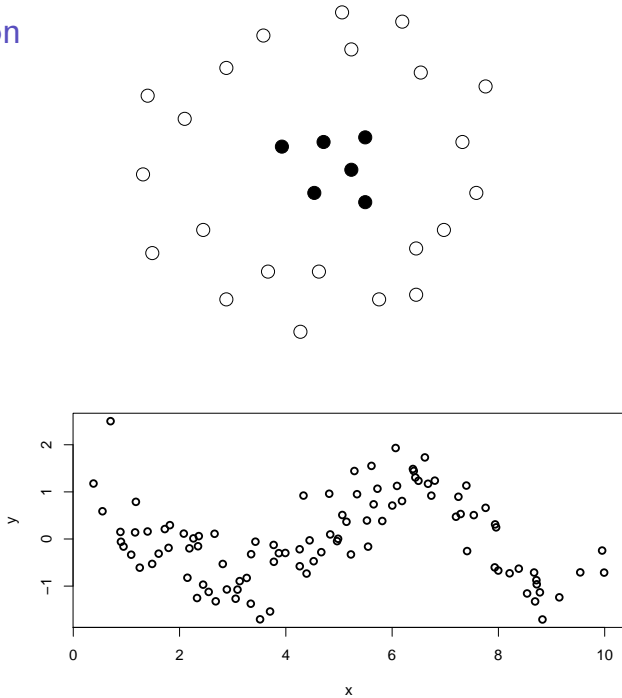
$$\beta^* = X^\top \alpha^* \quad f_{\beta^*}(x) = (\beta^*)^\top x = (\alpha^*)^\top Xx$$

- Training complexity: $O(n^2)$ to store XX^\top , $O(n^3)$ to find α^*
- Prediction: $O(d)$ for $(\beta^*)^\top x$, $O(nd)$ for $(\alpha^*)^\top Xx$

Outline

- 1 Introduction
- 2 **Standard machine learning**
 - Dimension reduction: PCA
 - Clustering: k -means
 - Regression: ridge regression
 - Classification: kNN, logistic regression and SVM
 - **Nonlinear models: kernel methods**
- 3 Large-scale machine learning
- 4 Conclusion

Motivation



Model

- Learn a function $f : \mathbb{R}^d \rightarrow \mathbb{R}$ of the form

$$f(x) = \sum_{i=1}^n \alpha_i K(x_i, x)$$

- For a positive definite (p.d.) kernel $K : \mathbb{R}^d \times \mathbb{R}^d \rightarrow \mathbb{R}$, such as

Linear $K(x, x') = x^\top x'$

Polynomial $K(x, x') = (x^\top x' + c)^p$

Gaussian $K(x, x') = \exp\left(-\frac{\|x - x'\|^2}{2\sigma^2}\right)$

Min/max $K(x, x') = \sum_{i=1}^d \frac{\min(|x_i|, |x'_i|)}{\max(|x_i|, |x'_i|)}$

Feature space

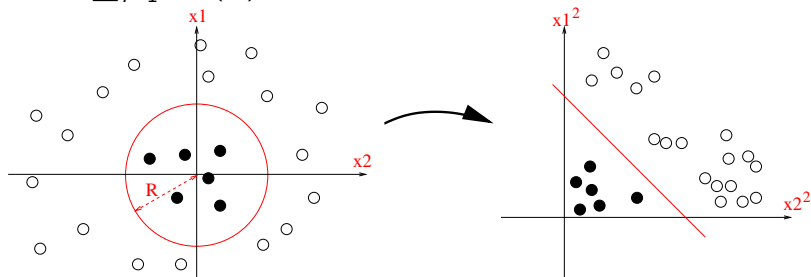
- A function $K : \mathbb{R}^d \times \mathbb{R}^d \rightarrow \mathbb{R}$ is a p.d. kernel if and only if there exists a mapping $\Phi : \mathbb{R}^d \rightarrow \mathbb{R}^D$, for some $D \in \mathbb{N} \cup \{+\infty\}$, such that

$$\forall x, x' \in \mathbb{R}^d, \quad K(x, x') = \Phi(x)^\top \Phi(x')$$

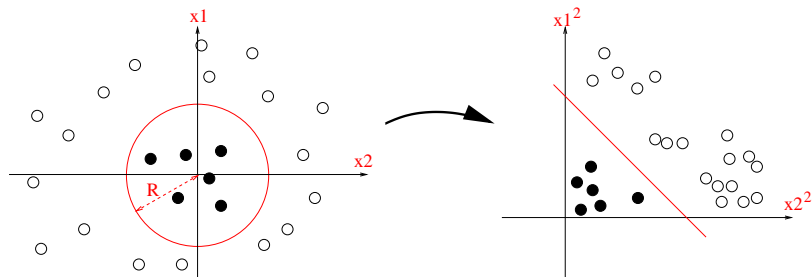
- f is then a linear function in \mathbb{R}^D :

$$f(x) = \sum_{i=1}^n \alpha_i K(x_i, x) = \sum_{i=1}^n \alpha_i \Phi(x_i)^\top \Phi(x) = \beta^\top \Phi(x)$$

for $\beta = \sum_{i=1}^n \alpha_i \Phi(x_i)$.



Learning



- We can learn $f(x) = \sum_{i=1}^n \alpha_i K(x_i, x)$ by fitting a linear model $\beta^\top \Phi(x)$ in the feature space
- Example: ridge regression / logistic regression / SVM

$$\min_{\beta \in \mathbb{R}^D} \sum_{i=1}^n \ell(y_i, \beta^\top \Phi(x_i)) + \lambda \beta^\top \beta$$

- But D can be very large, even infinite...

Kernel tricks

- $K(x, x') = \Phi(x)^\top \Phi(x')$ can be quick to compute even if D is large (even infinite)
- For a set of training samples $\{x_1, \dots, x_n\} \subset \mathbb{R}^d$ let K_n the $n \times n$ **Gram matrix**:

$$[K_n]_{ij} = K(x_i, x_j)$$

- For $\beta = \sum_{i=1}^n \alpha_i \Phi(x_i)$ we have

$$\beta^\top \Phi(x_i) = [K\alpha]_i \quad \text{and} \quad \beta^\top \beta = \alpha^\top K\alpha$$

- We can therefore solve the equivalent problem in $\alpha \in \mathbb{R}^n$

$$\min_{\alpha \in \mathbb{R}^n} \sum_{i=1}^n \ell(y_i, [K\alpha]_i) + \lambda \alpha^\top K\alpha$$

Example: kernel ridge regression (KRR)

$$\min_{\beta \in \mathbb{R}^d} \sum_{i=1}^n \left(y_i - \beta^\top \Phi(x_i) \right)^2 + \lambda \beta^\top \beta$$

- Solve in \mathbb{R}^D :

$$\hat{\beta} = \underbrace{\left(\Phi(X)^\top \Phi(X) + \lambda I \right)^{-1}}_{D \times D} \Phi(X)^\top Y$$

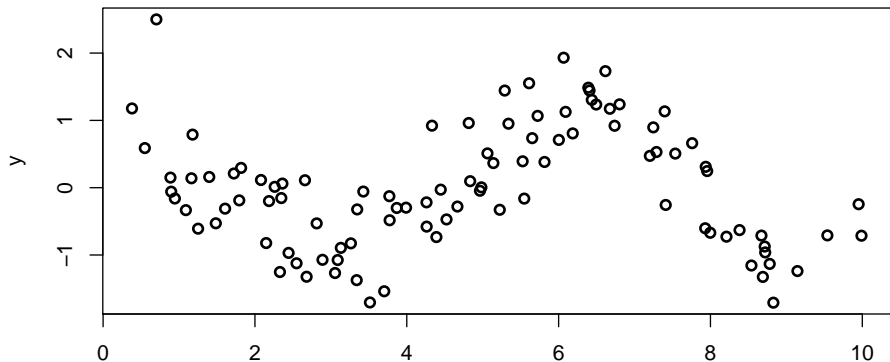
- Solve in \mathbb{R}^n :

$$\hat{\alpha} = \underbrace{\left(K + \lambda I \right)^{-1}}_{n \times n} Y$$

KRR with Gaussian RBF kernel

$$\min_{\beta \in \mathbb{R}^d} \sum_{i=1}^n \left(y_i - \beta^\top \Phi(x_i) \right)^2 + \lambda \beta^\top \beta$$

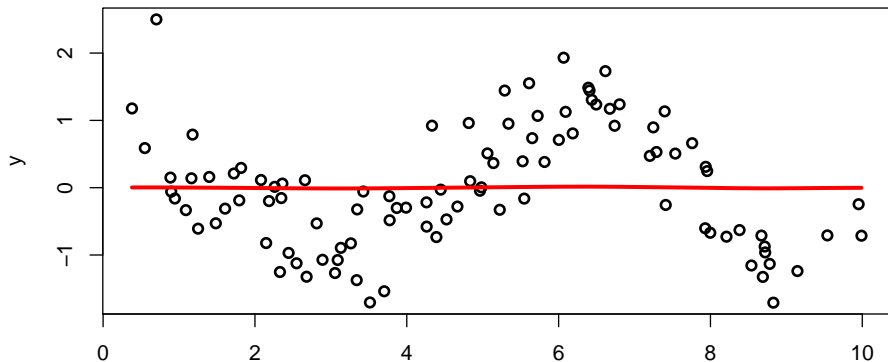
$$K(x, x') = \exp \left(-\frac{\|x - x'\|^2}{2\sigma^2} \right)$$



KRR with Gaussian RBF kernel

$$\min_{\beta \in \mathbb{R}^d} \sum_{i=1}^n \left(y_i - \beta^\top \Phi(x_i) \right)^2 + \lambda \beta^\top \beta \quad K(x, x') = \exp \left(-\frac{\|x - x'\|^2}{2\sigma^2} \right)$$

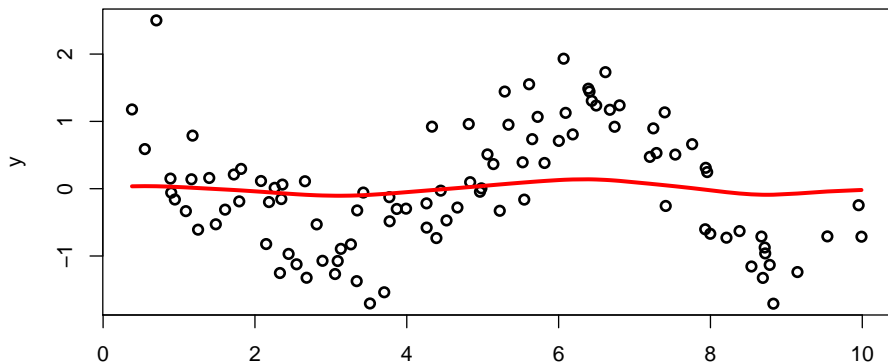
lambda = 1000



KRR with Gaussian RBF kernel

$$\min_{\beta \in \mathbb{R}^d} \sum_{i=1}^n \left(y_i - \beta^\top \Phi(x_i) \right)^2 + \lambda \beta^\top \beta \quad K(x, x') = \exp \left(-\frac{\|x - x'\|^2}{2\sigma^2} \right)$$

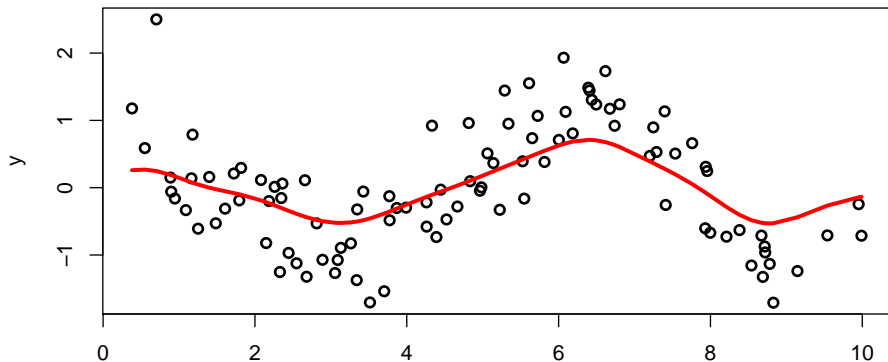
lambda = 100



KRR with Gaussian RBF kernel

$$\min_{\beta \in \mathbb{R}^d} \sum_{i=1}^n \left(y_i - \beta^\top \Phi(x_i) \right)^2 + \lambda \beta^\top \beta \quad K(x, x') = \exp \left(\frac{\|x - x'\|^2}{2\sigma^2} \right)$$

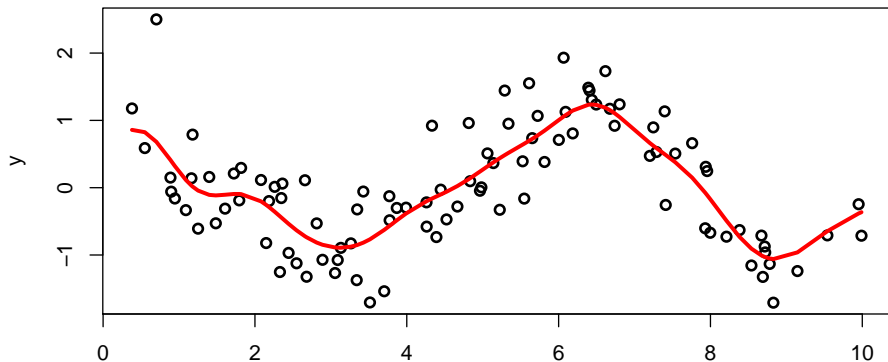
lambda = 10



KRR with Gaussian RBF kernel

$$\min_{\beta \in \mathbb{R}^d} \sum_{i=1}^n \left(y_i - \beta^\top \Phi(x_i) \right)^2 + \lambda \beta^\top \beta \quad K(x, x') = \exp \left(\frac{\|x - x'\|^2}{2\sigma^2} \right)$$

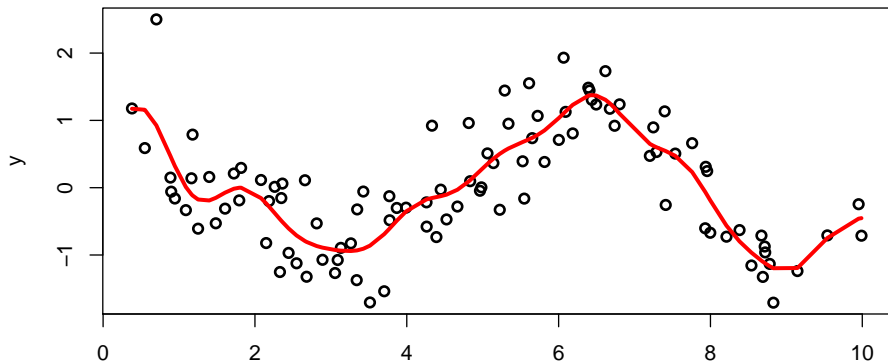
lambda = 1



KRR with Gaussian RBF kernel

$$\min_{\beta \in \mathbb{R}^d} \sum_{i=1}^n \left(y_i - \beta^\top \Phi(x_i) \right)^2 + \lambda \beta^\top \beta \quad K(x, x') = \exp \left(-\frac{\|x - x'\|^2}{2\sigma^2} \right)$$

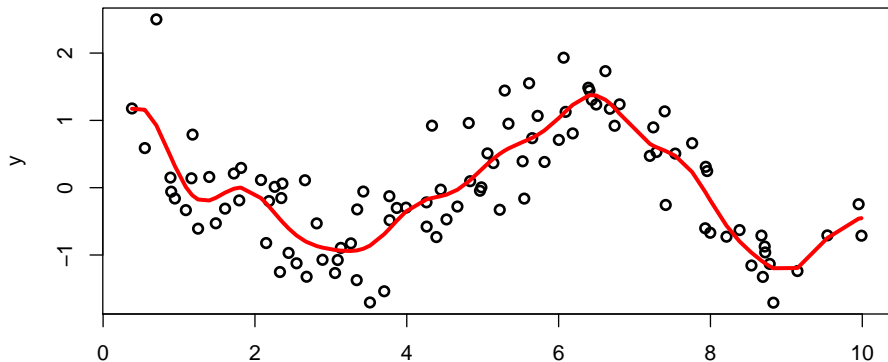
lambda = 0.1



KRR with Gaussian RBF kernel

$$\min_{\beta \in \mathbb{R}^d} \sum_{i=1}^n \left(y_i - \beta^\top \Phi(x_i) \right)^2 + \lambda \beta^\top \beta \quad K(x, x') = \exp \left(-\frac{\|x - x'\|^2}{2\sigma^2} \right)$$

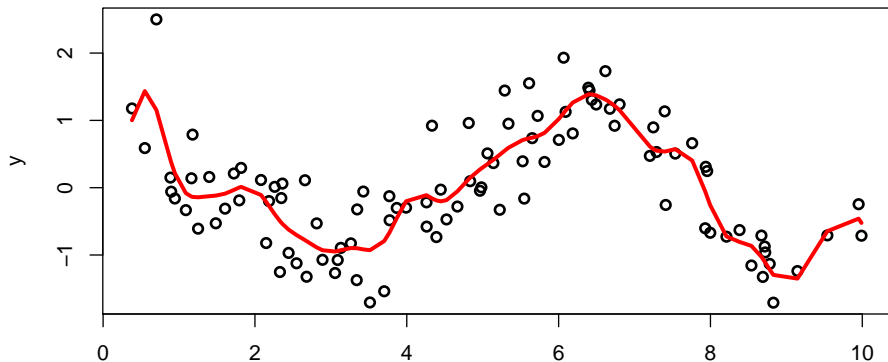
lambda = 0.01



KRR with Gaussian RBF kernel

$$\min_{\beta \in \mathbb{R}^d} \sum_{i=1}^n \left(y_i - \beta^\top \Phi(x_i) \right)^2 + \lambda \beta^\top \beta \quad K(x, x') = \exp \left(\frac{\|x - x'\|^2}{2\sigma^2} \right)$$

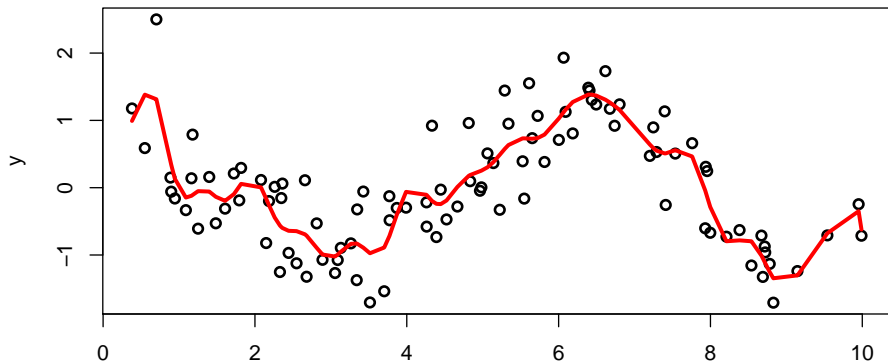
lambda = 0.001



KRR with Gaussian RBF kernel

$$\min_{\beta \in \mathbb{R}^d} \sum_{i=1}^n \left(y_i - \beta^\top \Phi(x_i) \right)^2 + \lambda \beta^\top \beta \quad K(x, x') = \exp \left(-\frac{\|x - x'\|^2}{2\sigma^2} \right)$$

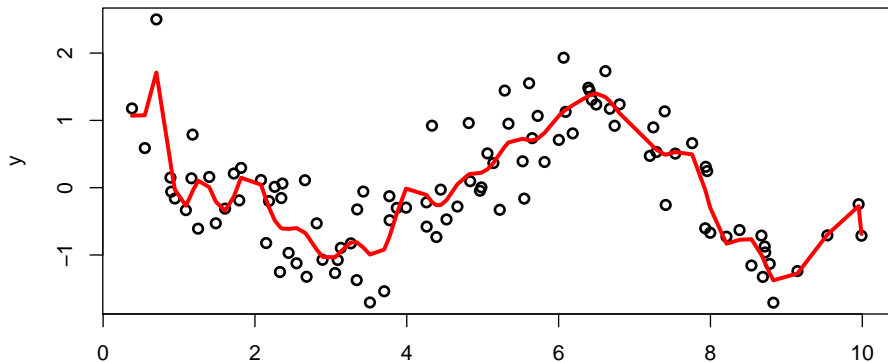
lambda = 0.0001



KRR with Gaussian RBF kernel

$$\min_{\beta \in \mathbb{R}^d} \sum_{i=1}^n \left(y_i - \beta^\top \Phi(x_i) \right)^2 + \lambda \beta^\top \beta \quad K(x, x') = \exp \left(-\frac{\|x - x'\|^2}{2\sigma^2} \right)$$

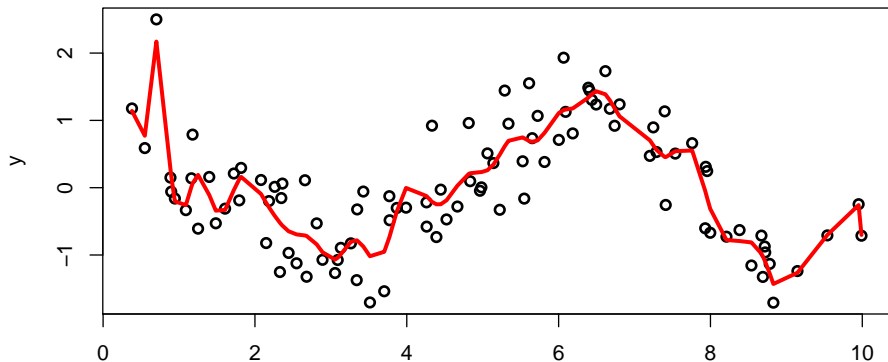
lambda = 0.00001



KRR with Gaussian RBF kernel

$$\min_{\beta \in \mathbb{R}^d} \sum_{i=1}^n \left(y_i - \beta^\top \Phi(x_i) \right)^2 + \lambda \beta^\top \beta \quad K(x, x') = \exp \left(\frac{\|x - x'\|^2}{2\sigma^2} \right)$$

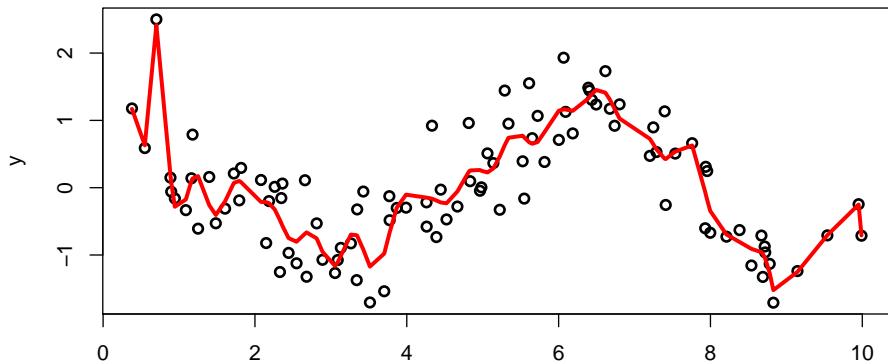
lambda = 0.000001



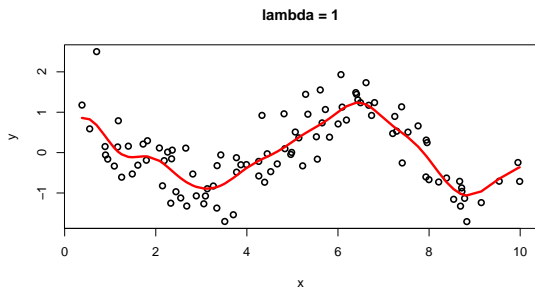
KRR with Gaussian RBF kernel

$$\min_{\beta \in \mathbb{R}^d} \sum_{i=1}^n \left(y_i - \beta^\top \Phi(x_i) \right)^2 + \lambda \beta^\top \beta \quad K(x, x') = \exp \left(\frac{\|x - x'\|^2}{2\sigma^2} \right)$$

lambda = 0.0000001



Complexity



- Compute K : $O(dn^2)$
- Store K : $O(n^2)$
- Solve α : $O(n^2 \sim 3)$
- Compute $f(x)$ for one x : $O(nd)$
- Unpractical for $n > 10 \sim 100k$

Outline

- 1 Introduction
- 2 Standard machine learning
 - Dimension reduction: PCA
 - Clustering: k -means
 - Regression: ridge regression
 - Classification: kNN, logistic regression and SVM
 - Nonlinear models: kernel methods
- 3 Large-scale machine learning
 - Scalability issues
 - The tradeoffs of large-scale learning
 - Random projections
 - Random features
 - Approximate NN
 - Shingling, hashing, sketching
- 4 Conclusion

Outline

- 1 Introduction
- 2 Standard machine learning
- 3 Large-scale machine learning
 - Scalability issues
 - The tradeoffs of large-scale learning
 - Random projections
 - Random features
 - Approximate NN
 - Shingling, hashing, sketching
- 4 Conclusion

What is "large-scale"?

- Data cannot fit in RAM
- Algorithm cannot run on a single machine in reasonable time (algorithm-dependent)
- Sometimes even $O(n)$ is too large! (e.g., nearest neighbor in a database of $O(B+)$ items)
- Many tasks / parameters (e.g., image categorization in $O(10M)$ classes)
- Streams of data



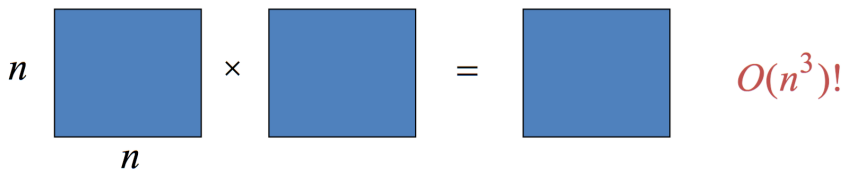
Things to worry about

- Training time (usually offline)
- Memory requirements
- Test time
- Complexities so far

Method	Memory	Training time	Test time
PCA	$O(d^2)$	$O(nd^2)$	$O(d)$
k -means	$O(nd)$	$O(ndk)$	$O(kd)$
Ridge regression	$O(d^2)$	$O(nd^2)$	$O(d)$
kNN	$O(nd)$	0	$O(nd)$
Logistic regression	$O(nd)$	$O(nd^2)$	$O(d)$
SVM, kernel methods	$O(n^2)$	$O(n^3)$	$O(nd)$

Techniques for large-scale machine learning

- Good baselines:
 - Subsample data and run standard method
 - Split and run on several machines (depends on algorithm)
- Need to revisit standard algorithms and implementation, taking into account **scalability**


$$n \begin{matrix} \square \\ n \end{matrix} \times \begin{matrix} \square \\ n \end{matrix} = \begin{matrix} \square \\ n \end{matrix} \quad O(n^3)!$$

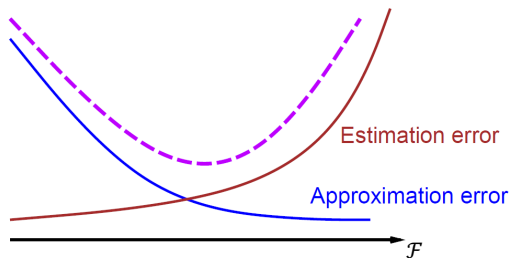
- Trade exactness for scalability
- Compress, sketch, hash data in a smart way

Outline

- 1 Introduction
- 2 Standard machine learning
- 3 Large-scale machine learning**
 - Scalability issues
 - The tradeoffs of large-scale learning**
 - Random projections
 - Random features
 - Approximate NN
 - Shingling, hashing, sketching
- 4 Conclusion

Motivation

- Classical learning theory analyzes the trade-off between:
 - approximation error (how well we approximate the true function)
 - estimation errors (how well we estimate the parameters)



- But reaching the best trade-off for a given n may be impossible with limited computational resources
- We should include in the trade-off the computational budget, and see which optimization algorithm gives the best trade-off!
- Seminal paper of Bottou and Bousquet [2008]

Classical ERM setting

- Goal: learn a function $f : \mathbb{R}^d \rightarrow \mathcal{Y}$ ($\mathcal{Y} = \mathbb{R}$ or $\{-1, 1\}$)
- P unknown distribution over $\mathbb{R}^d \times \mathcal{Y}$
- Training set: $\mathcal{S} = \{(X_1, Y_1), \dots, (X_n, Y_n)\} \subset \mathbb{R}^d \times \mathcal{Y}$ i.i.d. following P
- Fix a class of functions $\mathcal{F} \subset \{f : \mathbb{R}^d \rightarrow \mathbb{R}\}$
- Choose a loss $\ell(y, f(x))$
- Learning by empirical risk minimization

$$f_n \in \arg \min_{f \in \mathcal{F}} R_n[f] = \frac{1}{n} \sum_{i=1}^n \ell(Y_i, f(X_i))$$

- Hope that f_n has a small risk:

$$R[f_n] = E \ell(Y, f_n(X))$$

Classical ERM setting

- The best possible risk is

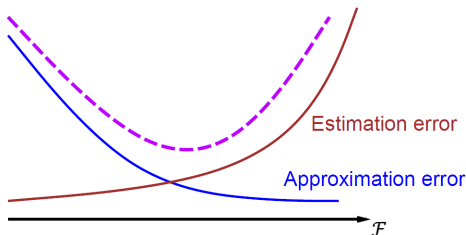
$$R^* = \min_{f: \mathbb{R}^d \rightarrow \mathcal{Y}} R[f]$$

- The best achievable risk over \mathcal{F} is

$$R_{\mathcal{F}}^* = \min_{f \in \mathcal{F}} R[f]$$

- We then have the decomposition

$$R[f_n] - R^* = \underbrace{R[f_n] - R_{\mathcal{F}}^*}_{\text{estimation error } \epsilon_{\text{est}}} + \underbrace{R_{\mathcal{F}}^* - R^*}_{\text{approximation error } \epsilon_{\text{app}}}$$



Optimization error

- Solving the ERM problem may be hard (when n and d are large)
- Instead we usually find an **approximate solution** \tilde{f}_n that satisfies

$$R_n[\tilde{f}_n] \leq R_n[f_n] + \rho$$

- The excess risk of \tilde{f}_n is then

$$\epsilon = R[\tilde{f}_n] - R^* = \underbrace{R[\tilde{f}_n] - R[f_n]}_{\text{optimization error } \epsilon_{opt}} + \epsilon_{est} + \epsilon_{app}$$

A new trade-off

$$\epsilon = \epsilon_{app} + \epsilon_{est} + \epsilon_{opt}$$

Problem

- Choose \mathcal{F}, n, ρ to make ϵ as small as possible
- Subject to a limit on n and on the computation time T

Table 1: Typical variations when \mathcal{F}, n , and ρ increase.

		\mathcal{F}	n	ρ
\mathcal{E}_{app}	(approximation error)	\searrow		
\mathcal{E}_{est}	(estimation error)	\nearrow	\searrow	
\mathcal{E}_{opt}	(optimization error)	\dots	\dots	\nearrow
T	(computation time)	\nearrow	\nearrow	\searrow

Large-scale or small-scale?

- Small-scale when constraint on n is active
- Large-scale when constraint on T is active

Comparing optimization methods

$$\min_{\beta \in \mathcal{B} \subset \mathbb{R}^d} R_n[f_\beta] = \sum_{i=1}^n \ell(y_i, f_\beta(x_i))$$

- Gradient descent (GD):

$$\beta_{t+1} \leftarrow \beta_t - \eta \frac{\partial R_n(f_{\beta_t})}{\partial \beta}$$

- Second-order gradient descent (2GD), assuming Hessian H known

$$\beta_{t+1} \leftarrow \beta_t - H^{-1} \frac{\partial R_n(f_{\beta_t})}{\partial \beta}$$

- Stochastic gradient descent (SGD):

$$\beta_{t+1} \leftarrow \beta_t - \frac{\eta}{t} \frac{\partial \ell(y_t, f_{\beta_t}(x_t))}{\partial \beta}$$

Results [Bottou and Bousquet, 2008]

Algorithm	Cost of one iteration	Iterations to reach ρ	Time to reach accuracy ρ	Time to reach $\mathcal{E} \leq c(\mathcal{E}_{\text{app}} + \epsilon)$
GD	$\mathcal{O}(nd)$	$\mathcal{O}\left(\kappa \log \frac{1}{\rho}\right)$	$\mathcal{O}\left(nd\kappa \log \frac{1}{\rho}\right)$	$\mathcal{O}\left(\frac{d^2 \kappa}{\epsilon^{1/\alpha}} \log^2 \frac{1}{\epsilon}\right)$
2GD	$\mathcal{O}(d^2 + nd)$	$\mathcal{O}\left(\log \log \frac{1}{\rho}\right)$	$\mathcal{O}\left((d^2 + nd) \log \log \frac{1}{\rho}\right)$	$\mathcal{O}\left(\frac{d^2}{\epsilon^{1/\alpha}} \log \frac{1}{\epsilon} \log \log \frac{1}{\epsilon}\right)$
SGD	$\mathcal{O}(d)$	$\frac{\nu \kappa^2}{\rho} + o\left(\frac{1}{\rho}\right)$	$\mathcal{O}\left(\frac{d\nu \kappa^2}{\rho}\right)$	$\mathcal{O}\left(\frac{d\nu \kappa^2}{\epsilon}\right)$

- $\alpha \in [1/2, 1]$ comes from the bound on ϵ_{est} and depends on the data
- In the last column, n and ρ are optimized to reach ϵ for each method
- 2GD optimizes much faster than GD, but limited gain on the final performance limited by $\epsilon^{-1/\alpha}$ coming from the estimation error
- SGD:
 - Optimization speed is catastrophic
 - Learning speed is the best, and independent of α
- This suggests that **SGD is very competitive** (and has become the de facto standard in large-scale ML)

Illustration

- **Results: Linear SVM**

$$\ell(\hat{y}, y) = \max\{0, 1 - y\hat{y}\} \quad \lambda = 0.0001$$

	Training Time	Primal cost	Test Error
SVMLight	23,642 secs	0.2275	6.02%
SVMPerf	66 secs	0.2278	6.03%
SGD	1.4 secs	0.2275	6.02%

- **Results: Log-Loss Classifier**

$$\ell(\hat{y}, y) = \log(1 + \exp(-y\hat{y})) \quad \lambda = 0.00001$$

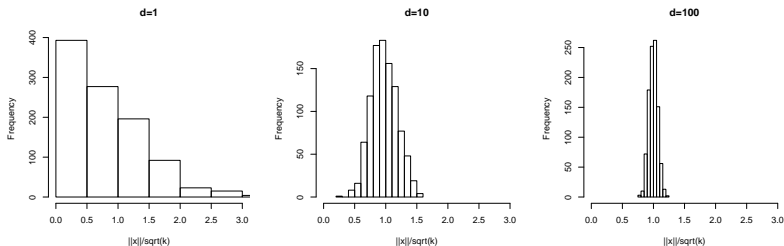
	Training Time	Primal cost	Test Error
TRON(LibLinear, $\varepsilon = 0.01$)	30 secs	0.18907	5.68%
TRON(LibLinear, $\varepsilon = 0.001$)	44 secs	0.18890	5.70%
SGD	2.3 secs	0.18893	5.66%

Outline

- 1 Introduction
- 2 Standard machine learning
- 3 **Large-scale machine learning**
 - Scalability issues
 - The tradeoffs of large-scale learning
 - **Random projections**
 - Random features
 - Approximate NN
 - Shingling, hashing, sketching
- 4 Conclusion

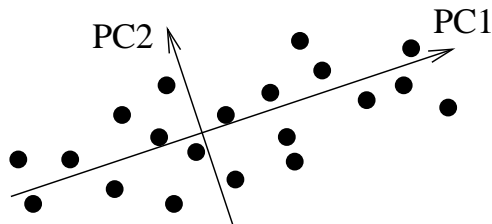
Motivation

- Affects scalability of algorithms, e.g., $O(nd)$ for kNN or $O(d^3)$ for ridge regression
- Hard to visualize
- (Sometimes) counterintuitive phenomena in high dimension, e.g., concentration of measure for Gaussian data



- Statistical inference degrades when d increases (curse of dimension)

Dimension reduction with PCA

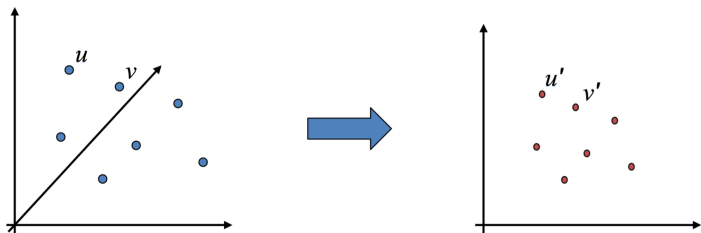


- Projects data onto $k < d$ dimensions that captures the largest amount of variance
- Also minimizes total reconstruction errors:

$$\min_{S_k} \sum_{i=1}^n \|x_i - \Pi_{S_k}(x_i)\|^2$$

- But computational expensive: $O(nd^2)$
- No theoretical guarantee on distance preservation

Linear dimension reduction



$$\underbrace{X'}_{n \times k} = \underbrace{X}_{n \times d} \times \underbrace{R}_{d \times k}$$

- Can we find R efficiently?
- Can we preserve distances?

$$\forall i, j = 1, \dots, n, \quad \|f(x_i) - f(x_j)\| \approx \|x_i - x_j\|$$

- Note: when $d > n$, we can take $k = n$ and preserve all distances exactly (kernel trick)

Random projections

Simply take a random projection matrix:

$$f(x) = \frac{1}{\sqrt{k}} R^\top x \quad \text{with} \quad R_{ij} \sim \mathcal{N}(0, 1)$$

Theorem [Johnson and Lindenstrauss, 1984]

For any $\epsilon > 0$ and $n \in \mathbb{N}$, take

$$k \geq 4 \left(\epsilon^2/2 - \epsilon^3/3 \right)^{-1} \log(n) \approx \epsilon^{-2} \log(n).$$

Then the following holds with probability at least $1 - 1/n$:

$$\forall i, j = 1, \dots, n \quad (1 - \epsilon) \|x_i - x_j\|^2 \leq \|f(x_i) - f(x_j)\|^2 \leq (1 + \epsilon) \|x_i - x_j\|^2$$

- k does not depend on d !
- $n = 1M, \epsilon = 0.1 \implies k \approx 5K$
- $n = 1B, \epsilon = 0.1 \implies k \approx 8K$

Proof (1/3)

- For a single dimension, $q_j = r_j^\top u$:

$$E(q_j) = E(r_j)^\top u = 0$$

$$E(q_j)^2 = u^\top E(r_j r_j^\top) u = \|u\|^2$$

- For the k -dimensional projection $f(u) = 1/\sqrt{k} R^\top u$:

$$\|f(u)\|^2 = \frac{1}{k} \sum_{j=1}^k q_j^2 \sim \frac{\|u\|^2}{k} \chi^2(k)$$

$$E\|f(u)\|^2 = \frac{1}{k} \sum_{j=1}^k E(q_j^2) = \|u\|^2$$

- Need to show that $\|f(u)\|^2$ is concentrated around its mean

Proof (2/3)

$$\begin{aligned} P [\|f\|^2 > (1 + \epsilon)\|u\|^2] &= P [\chi^2(k) > (1 + \epsilon)k] \\ &= P [e^{\lambda\chi^2(k)} > e^{\lambda(1+\epsilon)k}] \\ &\leq E [e^{\lambda\chi^2(k)}] e^{-\lambda(1+\epsilon)k} && \text{(Markov)} \\ &= (1 - 2\lambda)^{-\frac{k}{2}} e^{-\lambda(1+\epsilon)k} && \text{(MGF of } \chi^2(k) \text{ for } 0 \leq \lambda \leq 1/2) \\ &= ((1 + \epsilon)e^{-\epsilon})^{k/2} && \text{(take } \lambda = \epsilon/2(1 + \epsilon)) \\ &\leq e^{-(\epsilon^2/2 - \epsilon^3/3)k/2} && \text{(use } \log(1 + x) \leq x - x^2/2 + x^3/3) \\ &= n^{-2} && \text{(take } k = 4(\epsilon^2/2 - \epsilon^3/3) \log(n)) \end{aligned}$$

Similarly we get

$$P [\|f\|^2 < (1 - \epsilon)\|u\|^2] < n^{-2}$$

Proof (3/3)

- Apply with $u = x_i - x_j$ and use linearity of f to show that for an (x_i, x_j) pair, the probability of large distortion is $\leq 2n^{-2}$
- Union bound: for all $n(n-1)/2$ pairs, the probability that at least one has large distortion is smaller than

$$\frac{n(n-1)}{2} \times \frac{2}{n^2} = 1 - \frac{1}{n} \quad \square$$

Scalability

- $n = O(1B)$; $d = O(1M)$ $\implies k = O(10K)$
- Memory: need to store R , $O(dk) \approx 40GB$
- Computation: $X \times R$ in $O(ndk)$
- Other random matrices R have similar properties but better scalability, e.g.:

- "add or subtract" [Achlioptas, 2003], 1 bit/entry, size $\approx 1, 25GB$

$$R_{ij} = \begin{cases} +1 & \text{with probability } 1/2 \\ -1 & \text{with probability } 1/2 \end{cases}$$

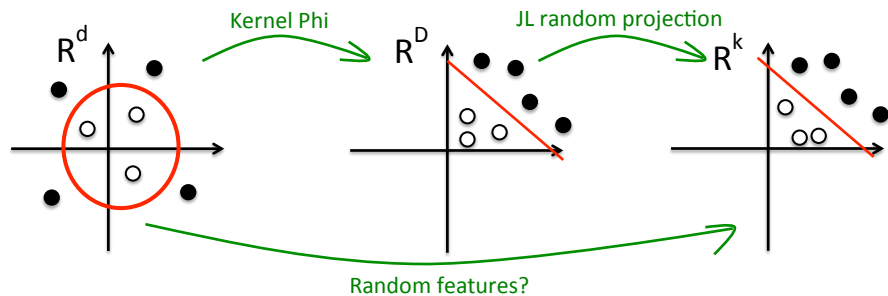
- Fast Johnson-Lindenstrauss transform [Ailon and Chazelle, 2009] where $R = PHD$, compute $f(x)$ in $O(d \log d)$

$$\begin{pmatrix} \text{Sparse} \\ \text{JL} \end{pmatrix}_{k \times d} \begin{pmatrix} \text{Walsh-} \\ \text{Hadamard} \end{pmatrix}_{d \times d} \begin{pmatrix} \pm 1 & & & \\ & \pm 1 & & \\ & & \ddots & \\ & & & \pm 1 \end{pmatrix}_{d \times d}$$

Outline

- 1 Introduction
- 2 Standard machine learning
- 3 Large-scale machine learning**
 - Scalability issues
 - The tradeoffs of large-scale learning
 - Random projections
 - Random features**
 - Approximate NN
 - Shingling, hashing, sketching
- 4 Conclusion

Motivation



Fourier feature space

Example: Gaussian kernel

$$\begin{aligned} e^{-\frac{\|x-x'\|^2}{2}} &= \frac{1}{(2\pi)^{\frac{d}{2}}} \int_{\mathbb{R}^d} e^{i\omega^\top(x-x')} e^{-\frac{\|\omega\|^2}{2}} d\omega \\ &= E_\omega \cos(\omega^\top(x-x')) \\ &= E_{\omega,b} \left[2 \cos(\omega^\top x + b) \cos(\omega^\top x' + b) \right] \end{aligned}$$

with

$$\omega \sim p(d\omega) = \frac{1}{(2\pi)^{\frac{d}{2}}} e^{-\frac{\|\omega\|^2}{2}} d\omega, \quad b \sim \mathcal{U}([0, 2\pi]).$$

This is of the form $K(x, x') = \Phi(x)^\top \Phi(x')$ with $D = +\infty$:

$$\Phi : \mathbb{R}^d \rightarrow L_2 \left(\left(\mathbb{R}^d, p(d\omega) \right) \times ([0, 2\pi], \mathcal{U}) \right)$$

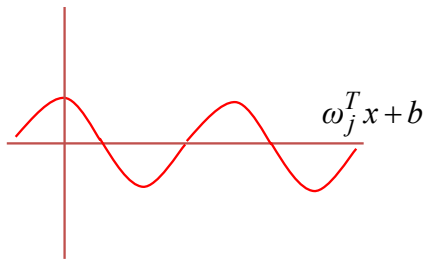
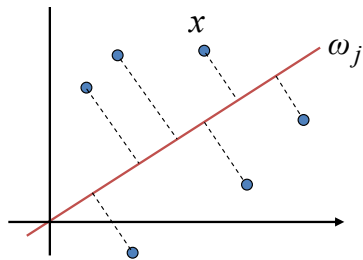
Random Fourier features [Rahimi and Recht, 2008]

- For $i = 1, \dots, k$, sample randomly:

$$(\omega_i, b_i) \sim p(d\omega) \times \mathcal{U}([0, 2\pi])$$

- Create random features:

$$\forall x \in \mathbb{R}^d, \quad f_i(x) = \sqrt{\frac{2}{k}} \cos(\omega_i^\top x + b_i)$$



Random Fourier features [Rahimi and Recht, 2008]

For any $x, x' \in \mathbb{R}^d$, it holds

$$\begin{aligned} E \left[f(x)^\top f(x') \right] &= \sum_{i=1}^k E \left[f_i(x) f_i(x') \right] \\ &= \frac{1}{k} \sum_{i=1}^k E \left[2 \cos \left(\omega^\top x + b \right) \cos \left(\omega^\top x' + b \right) \right] \\ &= K(x, x') \end{aligned}$$

and by Hoeffding's inequality,

$$P \left[\left| f(x)^\top f(x') - K(x, x') \right| > \epsilon \right] \leq 2e^{-\frac{k\epsilon^2}{2}}$$

This allows to approximate learning with the Gaussian kernel with a simple linear model in k dimensions!

Generalization

A translation-invariant (t.i.) kernel is of the form

$$K(x, x') = \varphi(x - x')$$

Bochner's theorem

For a continuous function $\varphi : \mathbb{R}^d \rightarrow \mathbb{R}$, K is p.d. if and only if φ is the Fourier-Stieltjes transform of a symmetric and positive finite Borel measure $\mu \in M(\mathbb{R}^d)$:

$$\varphi(x) = \int_{\mathbb{R}^d} e^{-i\omega^\top x} d\mu(\omega)$$

Just sample $\omega_i \sim \frac{d\mu(\omega)}{\mu(\mathbb{R}^d)}$ and $b_i \sim \mathcal{U}([0, 2\pi])$ to approximate any t.i. kernel K with random features

$$\sqrt{\frac{2}{k}} \cos(\omega_i^\top x + b_i)$$

Examples

$$K(x, x') = \varphi(x - x') = \int_{\mathbb{R}^d} e^{-i\omega^\top(x-x')} d\mu(\omega)$$

Kernel	$\varphi(x)$	$\mu(d\omega)$
Gaussian	$\exp\left(-\frac{\ x\ ^2}{2}\right)$	$(2\pi)^{-d/2} \exp\left(-\frac{\ \omega\ ^2}{2}\right)$
Laplace	$\exp(-\ x\ _1)$	$\prod_{i=1}^k \frac{1}{\pi(1+\omega_i^2)}$
Cauchy	$\prod_{i=1}^k \frac{2}{1+x_i^2}$	$e^{-\ \omega\ _1}$

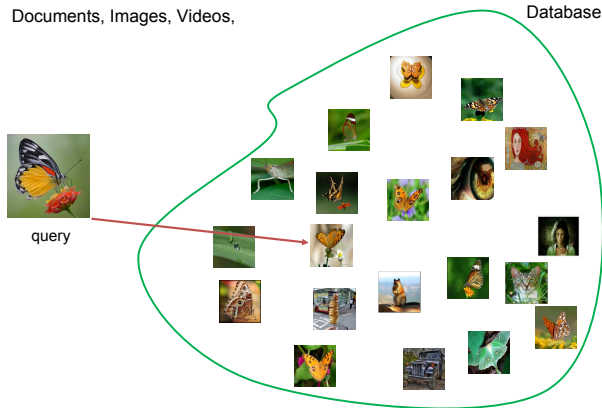
Performance [Rahimi and Recht, 2008]

Dataset	Fourier+LS	Binning+LS	CVM	Exact SVM
CPU regression 6500 instances 21 dims	3.6% 20 secs $D = 300$	5.3% 3 mins $P = 350$	5.5% 51 secs	11% 31 secs ASVM
Census regression 18,000 instances 119 dims	5% 36 secs $D = 500$	7.5% 19 mins $P = 30$	8.8% 7.5 mins	9% 13 mins SVMTorch
Adult classification 32,000 instances 123 dims	14.9% 9 secs $D = 500$	15.3% 1.5 mins $P = 30$	14.8% 73 mins	15.1% 7 mins SVM ^{light}
Forest Cover classification 522,000 instances 54 dims	11.6% 71 mins $D = 5000$	2.2% 25 mins $P = 50$	2.3% 7.5 hrs	2.2% 44 hrs libSVM
KDDCUP99 (see footnote) classification 4,900,000 instances 127 dims	7.3% 1.5 min $D = 50$	7.3% 35 mins $P = 10$	6.2% (18%) 1.4 secs (20 secs)	8.3% < 1 s SVM+sampling

Outline

- 1 Introduction
- 2 Standard machine learning
- 3 Large-scale machine learning
 - Scalability issues
 - The tradeoffs of large-scale learning
 - Random projections
 - Random features
 - **Approximate NN**
 - Shingling, hashing, sketching
- 4 Conclusion

Motivation



- Database $\mathcal{S} = \{x_1, \dots, x_n\} \subset \mathbb{R}^d$, query $q \in \mathbb{R}^d$
- Naively: $O(nd)$ to compute distances $\|q - x_i\|$ and find the smallest one
- For $n = 1B$, $d = 10k$, it takes 15 hours
- Projections $\mathbb{R}^d \rightarrow \mathbb{R}^k$ with $k < d$ is not good enough if n is large

ANN

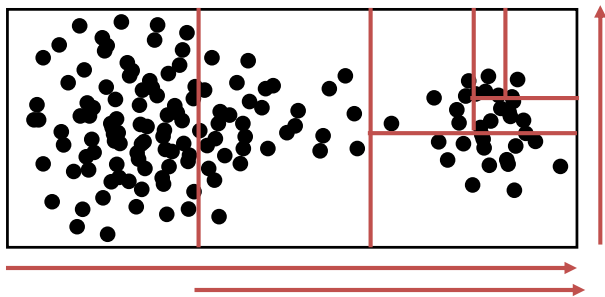
Given $\epsilon > 0$, the **approximate nearest neighbor (ANN)** problem is:

$$\text{Find } y \in \mathcal{S} \text{ such that } \|q - y\| \leq (1 + \epsilon) \min_{x \in \mathcal{S}} \|q - x\|$$

Two popular ANN approaches

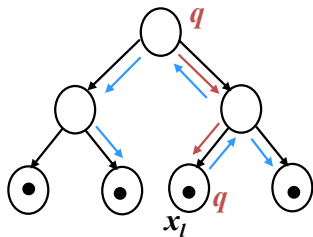
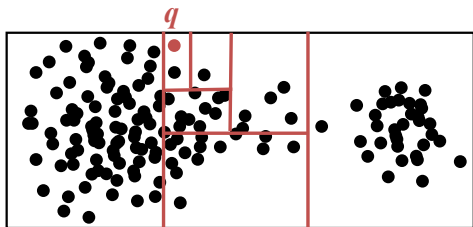
- 1 Tree approaches
 - Recursively partition the data: **Divide and Conquer**
 - Expected query time: $O(\log(n))$
 - Many variants: KDtree, Balltree, PCA-tree, Vantage Point tree
 - **Shown to perform very well in relatively low-dim data**
- 2 Hashing approaches
 - Each image in database represented as a code
 - Significant **reduction in storage**
 - Expected query time: $O(1)$ or $O(n)$
 - Compact codes preferred

KD tree



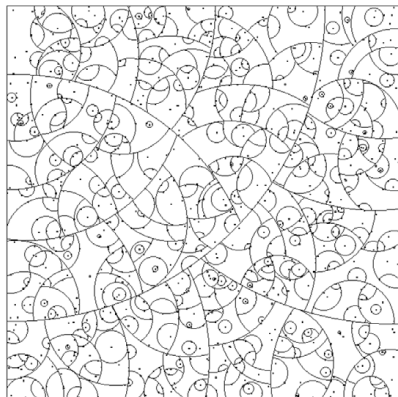
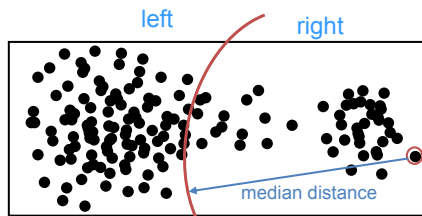
- Axis-parallel splits
- Along the direction of largest variance
- Split along the median \implies balanced partitioning
- Split recursively until each node has a single data point

Search in a KD tree



- Finds the leaf of the query in $O(\log(n))$
- But backtracking is needed to visit other leaves surrounding the cell
- As d increases, the number of leaves to visit grows exponentially
- Complexity: $O(nd \log(n))$ to build the tree, $O(nd)$ to store the original data
- Works fine up to $d = 10 \sim 100$

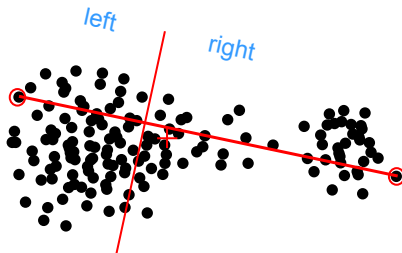
Variants



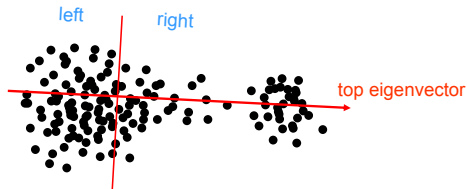
VP-Tree

Variants

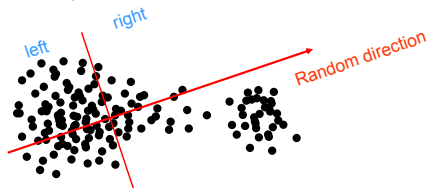
Ball tree



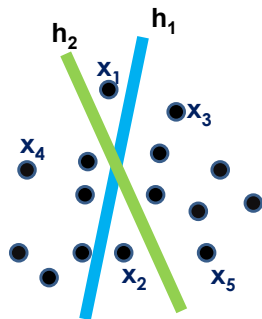
PCA tree



Random-Projection tree



Binary code using multiple hashing



X	x_1	x_2	x_3	x_4	x_5
y_1	0	1	1	0	1
y_2	1	0	1	0	1
y_m					
	010	100	111	001	110

No recursive partitioning, unlike trees

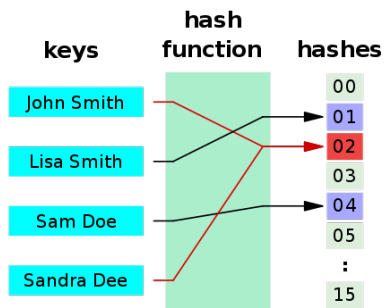
ANN with codes:

- 1 Choose a set of binary hashing functions to design a binary code
- 2 Index the database = compute codes for all points
- 3 Querying: compute the code of the query, and retrieve the points with similar codes

Hashing

A hash function is a function $h : \mathcal{X} \rightarrow \mathcal{Z}$ where

- \mathcal{X} is the set of data (\mathbb{R}^d for us)
- $\mathcal{Z} = \{1, \dots, N\}$ is a finite set of codes



https://en.wikipedia.org/wiki/Hash_function

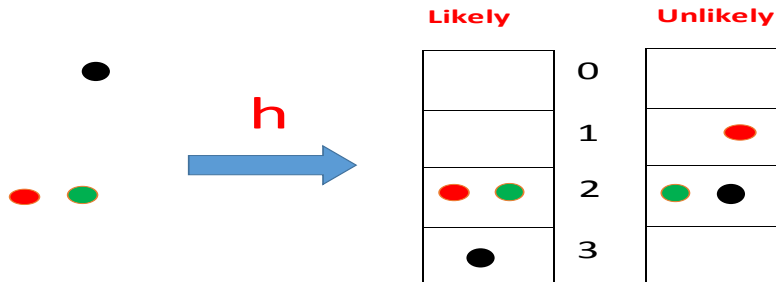
There is a **collision** when $h(x) = h(x')$ for two different entries $x \neq x'$

Locality sensitive hashing (LSH)

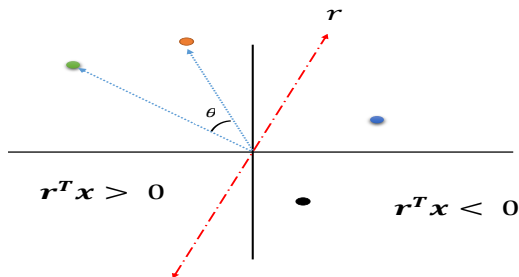
- Let a **random** hash function $h : \mathcal{X} \rightarrow \mathcal{Z}$
- It is a LSH with respect to a similarity function $sim(x, x')$ on \mathcal{X} if there exists a monotonically increasing function $f : \mathbb{R} \rightarrow [0, 1]$ such that:

$$\forall x, x' \in \mathcal{X}, \quad P [h(x) = h(x')] = f(sim(x, x'))$$

- **"Probability of collision increases with similarity"**



Example: simHash

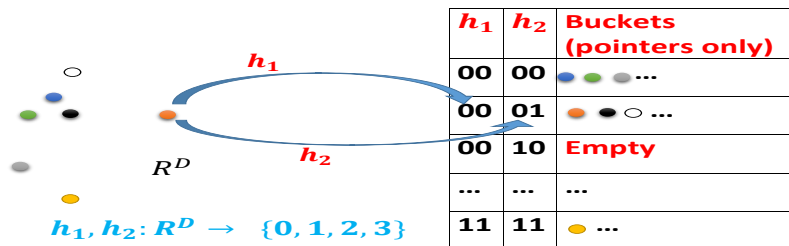


$$r \in \mathbb{R}^d \sim \mathcal{N}(0, Id) \quad h_r(x) = \begin{cases} 1 & \text{if } r^\top x \geq 0 \\ 0 & \text{otherwise.} \end{cases}$$

$$P[h_r(x) = h_r(x')] = 1 - \frac{\theta}{\pi}$$

LSH with respect to the cosine similarity $\text{sim}(x, x') = \cos(\theta)$ [Goemans and Williamson, 1995].

ANN with LSH



- $h_i(q) = h_i(x)$ implies high similarity (locality sensitive)

ANN with LSH

Table 1

h_1^1	...	h_K^1	Buckets
00	...	00	● ● ...
00	...	01	● ○ ...
00	...	10	Empty
...
11	...	11	...

...

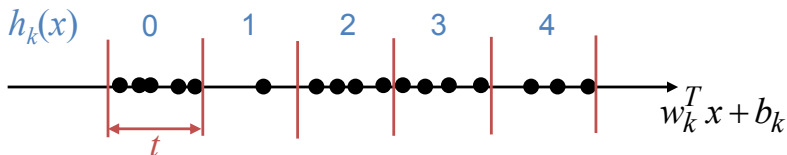
Table L

h_1^L	...	h_K^L	Buckets
00	...	00	● ● ...
00	...	01	● ● ...
00	...	10	○ ● ●
...
11	...	11	Empty

- $h_i(q) = h_i(x)$ implies high similarity (locality sensitive)
- Use K concatenations, repeated in L tables
- Querying: report union of K buckets
- Choice of K and L :
 - Large m increases precision but decreases recall
 - Large L increases recall but also storage
 - Optimization is possible to minimize run-time for a given application

LSH for $\|x - x'\|_s$?

$$h_k(x) = \left\lfloor \frac{w_k^\top x + b_k}{t} \right\rfloor \quad w_k \sim \prod_{i=1}^d P_s(w_k^i), \quad b_k \sim \mathcal{U}([0, t])$$



- P_s a s -stable distribution, i.e., for any $x \in \mathbb{R}^d$, and any w i.i.d. with $w^i \sim P_s$, $x^\top w \sim \|x\|_s w^1$.
- s -stable distributions exist for $p \in (0, 2]$:
 - Gaussian $\mathcal{N}(0, 1)$ is 2-stable
 - Cauchy $dx / (\pi(1 + x^2))$ is 1-stable
- Then $P[h_k(x) = h_k(x')]$ increases as $\|x - x'\|_s$ decreases

Outline

- 1 Introduction
- 2 Standard machine learning
- 3 Large-scale machine learning**
 - Scalability issues
 - The tradeoffs of large-scale learning
 - Random projections
 - Random features
 - Approximate NN
 - Shingling, hashing, sketching**
- 4 Conclusion

Motivation

- The hashing / LSH trick is a fast random projection to compact binary codes
- Initially proposed for ANN problems, it can also be used for more general learning problems
- It is particularly effective when data are first converted to huge binary vectors, using a specific similarity measure (the resemblance).
- Applications: texts, time series, images...

Shingling and resemblance

- Given some input space \mathcal{X} (e.g., texts, times series...), a **shingling** is a representation as large binary vector

$$x \in \{0, 1\}^D$$

- Equivalently, represent x as a subset of $S_x \subset \Omega = \{0, \dots, D - 1\}$
- Example: represent a text by the set of w -shingles it contains, i.e., sequences of w words. Typically, $w = 5$, 10^5 words, $D = 10^25$, but very sparse.
- A common measure of similarity between two such vectors is the **resemblance** (a.k.a. **Jaccart** or **Tanimoto** similarity):

$$R(x_1, x_2) = \frac{|S_1 \cap S_2|}{|S_1 \cup S_2|}$$

- But computing $R(x_1, x_2)$ is expensive, and not scalable for NN search or machine learning

Minwise hashing

- Let $\pi \in \mathbb{S}_D$ be a random permutation of Ω
- Let $h_\pi : \{0, 1\}^D \rightarrow \Omega$ assign to $S \subset \Omega$ the smallest index of $\pi(S)$:

$$h_\pi(x) = \min \{ \pi(i) : i \in S_x \}$$

Theorem [Broder, 1997]

Minwise hashing is a LSH with respect to the resemblance:

$$P [h_\pi(x_1) = h_\pi(x_2)] = R(x_1, x_2)$$

Proof:

- The smallest index $\min(h_\pi(x_1), h_\pi(x_2))$ correspond a random element of $S_1 \cup S_2$
- $h_\pi(x_1) = h_\pi(x_2)$ if it is in $S_1 \cap S_2$
- This happens with probability $R(x_1, x_2)$

Applications of minwise hashing

- If we pick k random permutations, we can represent x by $(h_1(x), \dots, h_k(x)) \in \{0, 1\}^{Dk}$
- Used for ANN, using the general LSH technique discussed earlier
- Learning linear models as an approximation to learning a nonlinear function with the resemblance kernel¹
- Various tricks to improve scalability
 - **b -bit minwise hashing** [Li and König, 2010]: only keep the last b bits of $h_\pi(x)$, which reduces the dimensionality of the hashed matrix to $2^b k$
 - **One-permutation hashing** [Li et al., 2012]: use a single permutation, keep the smallest index in each consecutive block of size k

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
$\pi(S_1)$:	0	0	1	0	1	0	0	1	0	0	0	0	0	1	0	0
$\pi(S_2)$:	1	0	0	1	0	0	1	0	0	0	0	0	0	1	0	0
$\pi(S_3)$:	1	1	0	0	0	0	0	0	0	0	1	0	1	0	0	0

¹This shows in particular that the resemblance is positive definite

Hash kernel [Shi et al., 2009]

- Goal: improve the scalability of random projections or minwise hashing, both in memory (sparsity) and processing time
- Simple idea:
 - Let $h : [1, d] \rightarrow [1, k]$ a hash function
 - For $x \in \mathbb{R}^d$ (or $\{0, 1\}^d$) let $\Phi(x) \in \mathbb{R}^k$ with

$$\forall i = 1, \dots, k \quad \Phi_i(x) = \sum_{j \in [1, d] : h(j)=i} x_j$$

- "Accumulate coordinates i of x for which $h(i)$ is the same
 - Repeat L times and concatenate if needed, to limit the effect of collisions
- Advantages
 - No memory needed for projections (vs. LSH)
 - No need for dictionary (just a hash function that can hash anything)
 - Sparsity preserving

Outline

- 1 Introduction
- 2 Standard machine learning
 - Dimension reduction: PCA
 - Clustering: k -means
 - Regression: ridge regression
 - Classification: kNN, logistic regression and SVM
 - Nonlinear models: kernel methods
- 3 Large-scale machine learning
 - Scalability issues
 - The tradeoffs of large-scale learning
 - Random projections
 - Random features
 - Approximate NN
 - Shingling, hashing, sketching
- 4 Conclusion

What we saw

- Most standard ML algorithms do not scale to modern, large-scale problems
- They are being revisited with **scalability** as new constraint, both in theory and in practice
- Generally, trading accuracy for fast approximations can be beneficial:
 - Optimization by SGD
 - Random projections, sketching
- Need to understand mathematics, statistics, algorithms, hardware

What we did not see

A lot!

- Hardware (distributed computing and storage, GPU, ...)
- Data streams
- Other models like deep learning or graphical models
- Other learning paradigms like reinforcement learning
- A lot of recent results (this is a very active research field!)

MERCI!

References I

- D. Achlioptas. Database-friendly random projections: Johnson-lindenstrauss with binary coins. *J. Comput. Syst. Sci.*, 66(4):671–687, 2003. doi: 10.1016/S0022-0000(03)00025-4. URL [http://dx.doi.org/10.1016/S0022-0000\(03\)00025-4](http://dx.doi.org/10.1016/S0022-0000(03)00025-4).
- N. Ailon and B. Chazelle. The fast Johnson-Lindenstrauss transform and approximate nearest neighbors. *SIAM J. Comput.*, 39(1):302–322, 2009. doi: 10.1137/060673096. URL <http://dx.doi.org/10.1137/060673096>.
- B. E. Boser, I. M. Guyon, and V. N. Vapnik. A training algorithm for optimal margin classifiers. In *Proceedings of the 5th annual ACM workshop on Computational Learning Theory*, pages 144–152, New York, NY, USA, 1992. ACM Press. URL <http://www.clopinet.com/isabelle/Papers/colt92.ps.Z>.
- L. Bottou and O. Bousquet. The tradeoffs of large scale learning. In J. C. Platt, D. Koller, Y. Singer, and S. T. Roweis, editors, *Adv. Neural. Inform. Process Syst.*, volume 20, pages 161–168. Curran Associates, Inc., 2008. URL <http://papers.nips.cc/paper/3323-the-tradeoffs-of-large-scale-learning.pdf>.
- A. Z. Broder. On the resemblance and containment of documents. In *Proceedings of the Compression and Complexity of Sequences*, pages 21–29, 1997. doi: 10.1109/SEQUEN.1997.666900. URL <http://dx.doi.org/10.1109/SEQUEN.1997.666900>.
- M. X. Goemans and D. P. Williamson. A general approximation technique for constrained forest problems. *SIAM J. Comput.*, 24(2):296–317, apr 1995. doi: 10.1137/S0097539793242618. URL <http://dx.doi.org/10.1137/S0097539793242618>.

References II

- W. B. Johnson and J. Lindenstrauss. Extensions of lipschitz mappings into a hilbert space. *Contemp. Math.*, 26:189–206, 1984. doi: 10.1090/conm/026/737400. URL <http://dx.doi.org/10.1090/conm/026/737400>.
- S. Le Cessie and J. C. van Houwelingen. Ridge estimators in logistic regression. *Appl. Statist.*, 41(1):191–201, 1992. URL <http://www.jstor.org/stable/2347628>.
- P. Li and A. C. König. b -bit minwise hashing. In *WWW*, pages 671–680, Raleigh, NC, 2010.
- P. Li, A. O., and C. hui Z. One permutation hashing. In F. Pereira, C. J. C. Burges, L. Bottou, and K. Q. Weinberger, editors, *Advances in Neural Information Processing Systems 25*, pages 3113–3121. Curran Associates, Inc., 2012. URL <http://papers.nips.cc/paper/4778-one-permutation-hashing.pdf>.
- A. Rahimi and B. Recht. Random features for large-scale kernel machines. In J. Platt, D. Koller, Y. Singer, and S. Roweis, editors, *Adv. Neural. Inform. Process Syst.*, volume 20, pages 1177–1184. Curran Associates, Inc., 2008. URL <http://papers.nips.cc/paper/3182-random-features-for-large-scale-kernel-machines.pdf>.
- Q. Shi, J. Petterson, G. Dror, J. Langford, A. Smola, and S. Vishwanathan. Hash kernels for structured data. *Journal of Machine Learning Research*, 10:2615–2637, 2009.
- C. Stone. Consistent nonparametric regression. *Ann. Stat.*, 8:1348–1360, 1977. URL <http://links.jstor.org/sici?sici=0090-5364%28197707%295%3A4%3C595%3ACNR%3E2.0.CO%3B2-0>.