Introduction To Machine Learning

Dmytro Kedyk

Learning Outcomes

- Understand the nature of learning, what it can and can't do
- Understand basic algorithms
- Be prepared to understand guest speaker lectures
- Understand what big data is and ignore the hype

Topics

- How Learning Works + Basic Algorithms
- Estimation Error + Linear SVM
- Approximation Error + Kernel SVM
- Other Theoretical Ideas
- Improving Performance + Random Forest

Representing Decidable Situations

- A decidable situation must be fully represented as an object in some **feature space** *X*
- *X* is usually a vector space where every object is a vector of *D* **features**, selected based on domain knowledge
- E.g. how to detect free riders in online team games? Can use number of met enemies and % of explored game space as features
- Human rule – a player who met 0 enemies and explored < 5% of the game space is a free rider

Automatic Learning

- Want **predictor** *f* that decides correctly in specific situations
- Creating one manually can be hard (how exactly to locate a face in a photo?) or time-consuming
- Instead pick a knowledge representation and somehow train it from available data – which one and how and why is machine learning

Setup

- Learning finds *f* that **partitions** *X* into possibly uncountably many pieces, assigning $\forall x \in X$ partition label $ID \in \{ID\}$
- Assume *x* follows distribution *P* and *ID* conditional $P|x$.
- Want an algorithm *A* that sees a finite set *S* of *n* examples z_i, from which it hopefully learns *f* that **generalizes** to unseen *z*.
- $\forall z_i$ have x_i and **hint information** $y_i \in Y$.

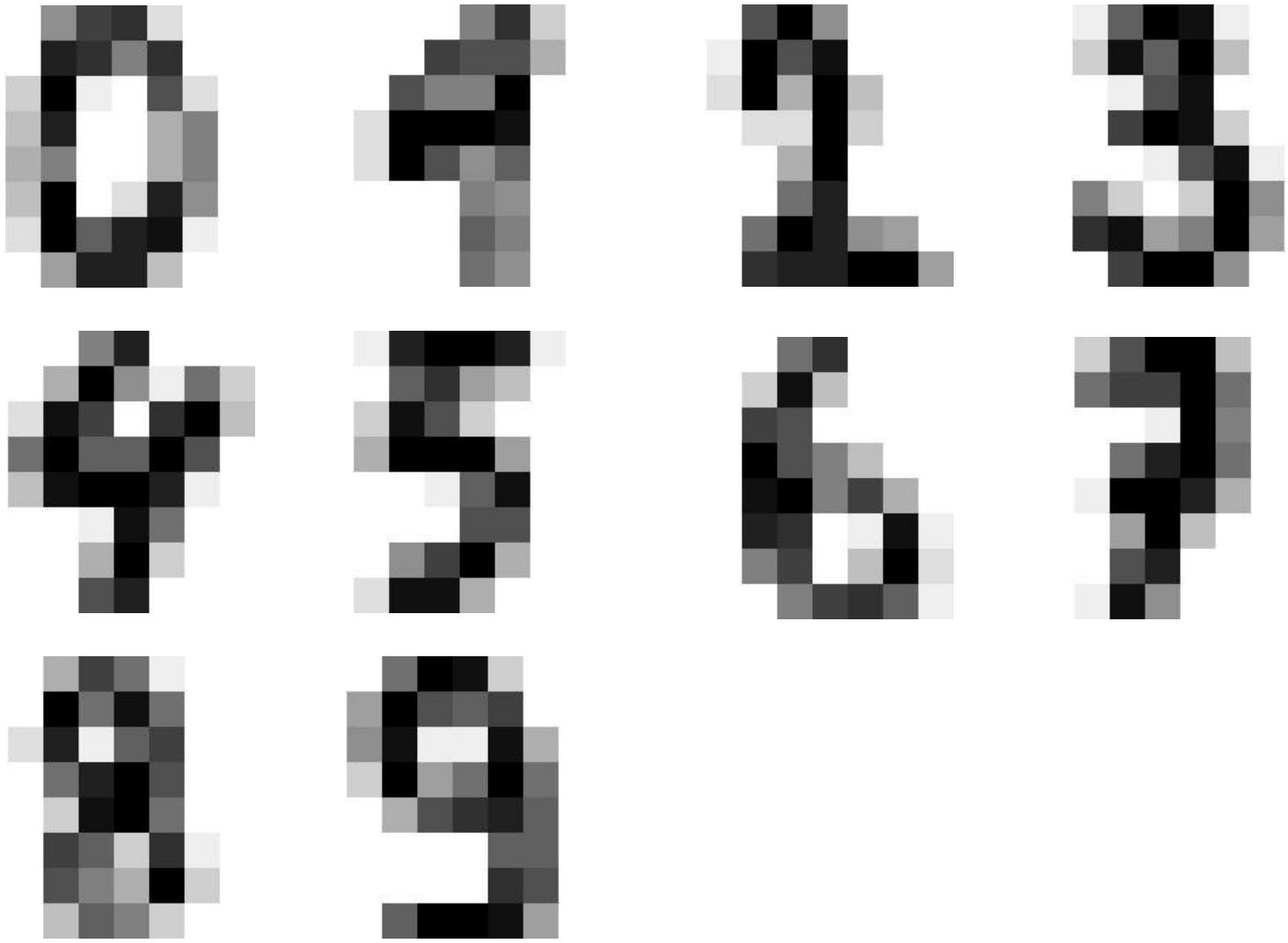
Setup

- For **prediction** tasks $ID_i = y_i$ = sample from $P|x_i$
- Generally, *y* is a function of *x*, sampled *ID*, and possibly other information
- A **loss function** $L_f(z) = L_f(x, y)$ measures error of *f* on z. *L* must return minimum possible finite value, usually 0, for perfect performance.
- **Risk** $R_f(P) = E_z[L_f(z)]$ measures generalization error of *f*.
- *Z* and *L* determine a **learning task**, and *P* a **problem** within it.

Prediction Tasks

- **Classification** – *ID* = *y* = **class label** $\in [0, k-1]$. Assuming right decisions have profit 0 and wrong cost 1, $L_f(x, y) = (f(x) \neq y)$ with corresponding R_f = E[% of misclassified *z*].
- **Regression** – *ID* = *y* are real valued. Typically assume quadratic cost of errors, i.e. $L_f(x, y) = (f(x) - y)^2$.
- **Reinforcement learning** – hint is assigned to a sequence of decisions. E.g. for chess some logic tells what is a good move, and after a game such logic wins or loses.

Digit Data



Iris Data

- Measurements of the iris leaf – want to decide what type of flower it is.

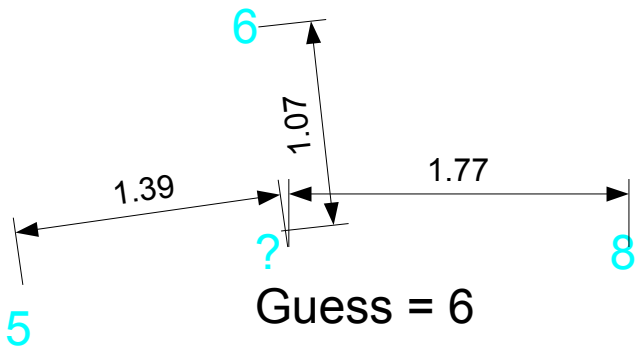
Type	Sepal Length	Sepal Width	Petal Length	Petal Width
Setosa	5.1	3.5	1.4	0.2
Setosa	5.4	3.9	1.7	0.4
Versicolor	6.4	3.2	4.5	1.5
Versicolor	5.6	2.9	3.6	1.3
Virginica	6.2	3.4	5.4	2.3
Virginica	7.2	3.6	6.1	2.5

Data Preparation and Cleaning

- Data hay have **missing values**
  - Delete examples or features
- May not be **numerical**
  - Convert each **categorical** value into 0/1 vector, increasing *D*
- Some algorithms like **normalized** scale
  - ∀ feature make range [0, 1]
  - Or mean 0 and variance 1
- Remove useless features like record id
  - Don't want to predict a tank from sky color

Nearest Neighbor

- Must have appropriate distance function for the problem and/or scale the value
- E.g. Euclidean distance in age-height space doesn't mean much
- To train create an efficient index (usually **vantage point tree**) from examples
- To predict return the class of the closest example



Evaluating Performance

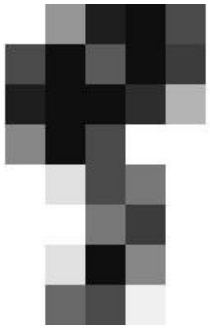
- Estimate risk by **empirical risk** $R_{f,n} = (\sum L_f(z_i))/n$.
- **Hoeffding inequality** better for $L \in [0, 1]$:
  - Let $e(p) = \sqrt{\ln(1/p)/(2n)}$
  - Then with probability $\geq 1 - p$:
    - Upper bound: $X \leq m + e$
    - *Lower bound: $X \geq m - e$*
    - *Confidence interval: $X \in m \pm e(p/2)$*
- Works for independent iid test set only

Evaluating Performance

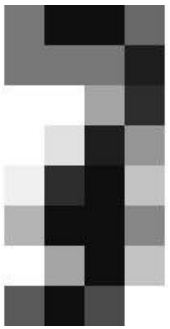
- For classification R_f is expected error, i.e. E[% of misclassified examples]
- Empirical **accuracy** $= 1 - R_{f,n}$
- E.g. nearest neighbor gets ≈98% accuracy on the digit data
- Problem: can never have 100% accuracy with 100% probability
- Ok for many applications, but not for others
- E.g. self-driving car can never be perfectly safe

Some NN Mistakes on Digit Data

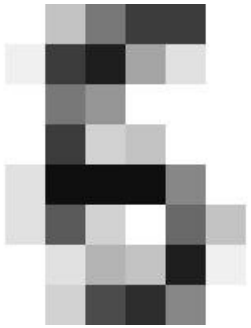
Inferred: 5. Actual: 9



Inferred: 7. Actual: 3



Inferred: 6. Actual: 5



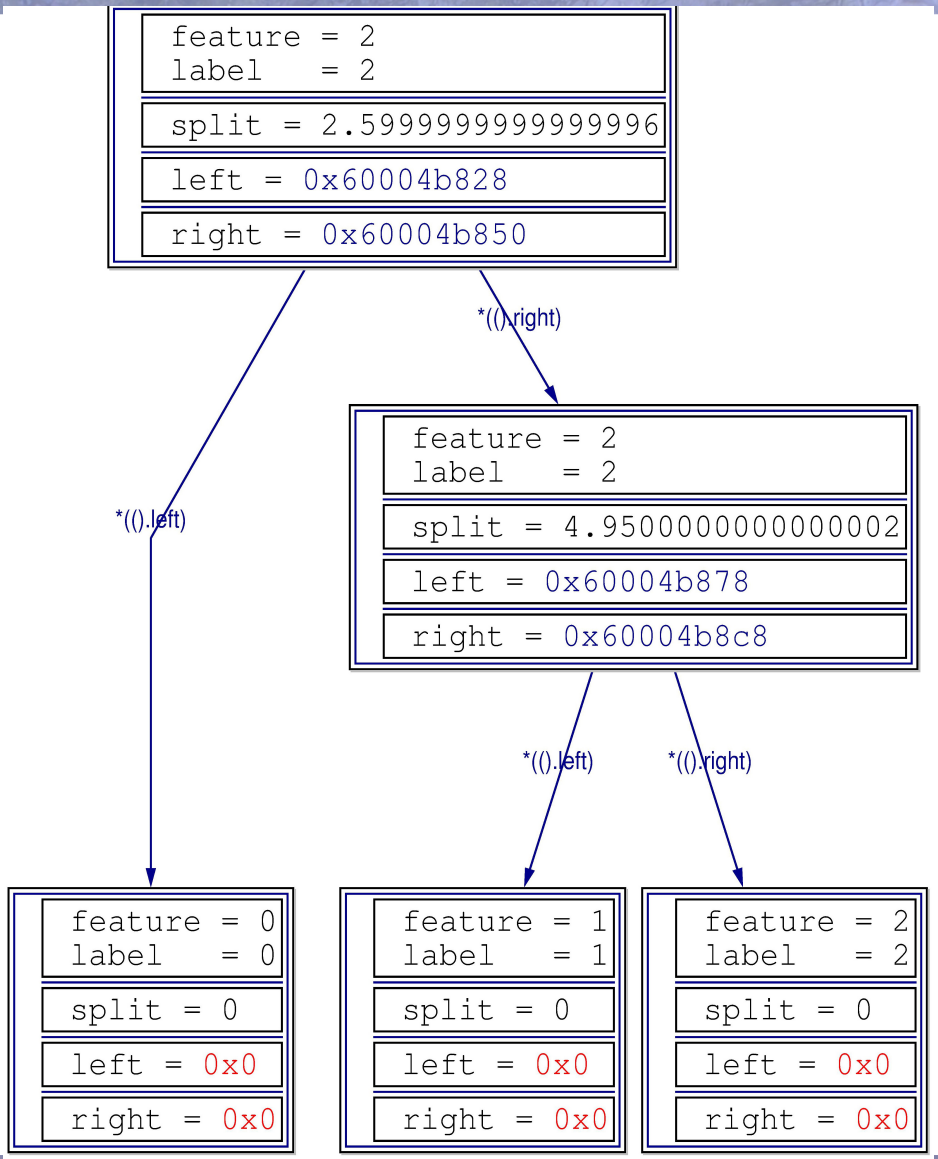
Problems with NN

- Poor accuracy on most data sets, despite good results on digits and some others
- Inefficient
  - O(n) memory and
  - Index query needs almost O(n) time with large D
- No explanation for decisions
  - E.g. credit denied because you look like a family from 10 years ago who defaulted

Decision Tree

- Binary tree
- Nodes look at values of specific feature values to decide which branch to take
- Leaf nodes give the resulting class

Decision Tree



Entropy Minimization

- Limit on number of bits needed to encode a string over an alphabet, given each symbol has some probability p.
- E.g. in English 'e' more likely than 'z' so encodings like Morse give short code to it
- For S **entropy** of labels = $\sum H(p_i)$, with $H(p) = p\lg(p)$ and p_i = % of examples of class i
- Want split that minimizes total entropy after split = #left child examples × entropy(left) + #right child examples × entropy(right)

Efficient Incremental Calculation

- For a numerical feature have $n - 1$ possible split points at the root
- Sort the feature values
- Consider all splits points one by one from left to right
- Update after each split left and right count tables $\forall$ class, recomputing entropy from these
- Considering all splits at the root takes $O(n \lg(n) D k)$ time

Decision Tree

- Find best split using incremental calculation
- Split data based on it into left and right parts
- Recurse on the left and the right parts, until get a pure node or exceed some depth *m*
- **Prune** to not overfit – will explain later
  - Use sign test with default std = 1
- O(*mn*lg(*n*)*Dk*) worst case, O(*n*lg(*n*)^2*Dk*) expected under equal size splits.
- Can explain decisions, but still makes mistakes

Sign Test Pruning

- Sub-tree must be better than it's root to not be pruned
- Given two players, one is better if wins significantly more than ½ of all games, including draws.
- Modeled as binomial(#games/2, # games), can approximate by normal
- For pruning better only if z score of best player's performance > 1 (or 0.5) std
- $z = (wins - nGames/2)/\sqrt{nGames/4}$

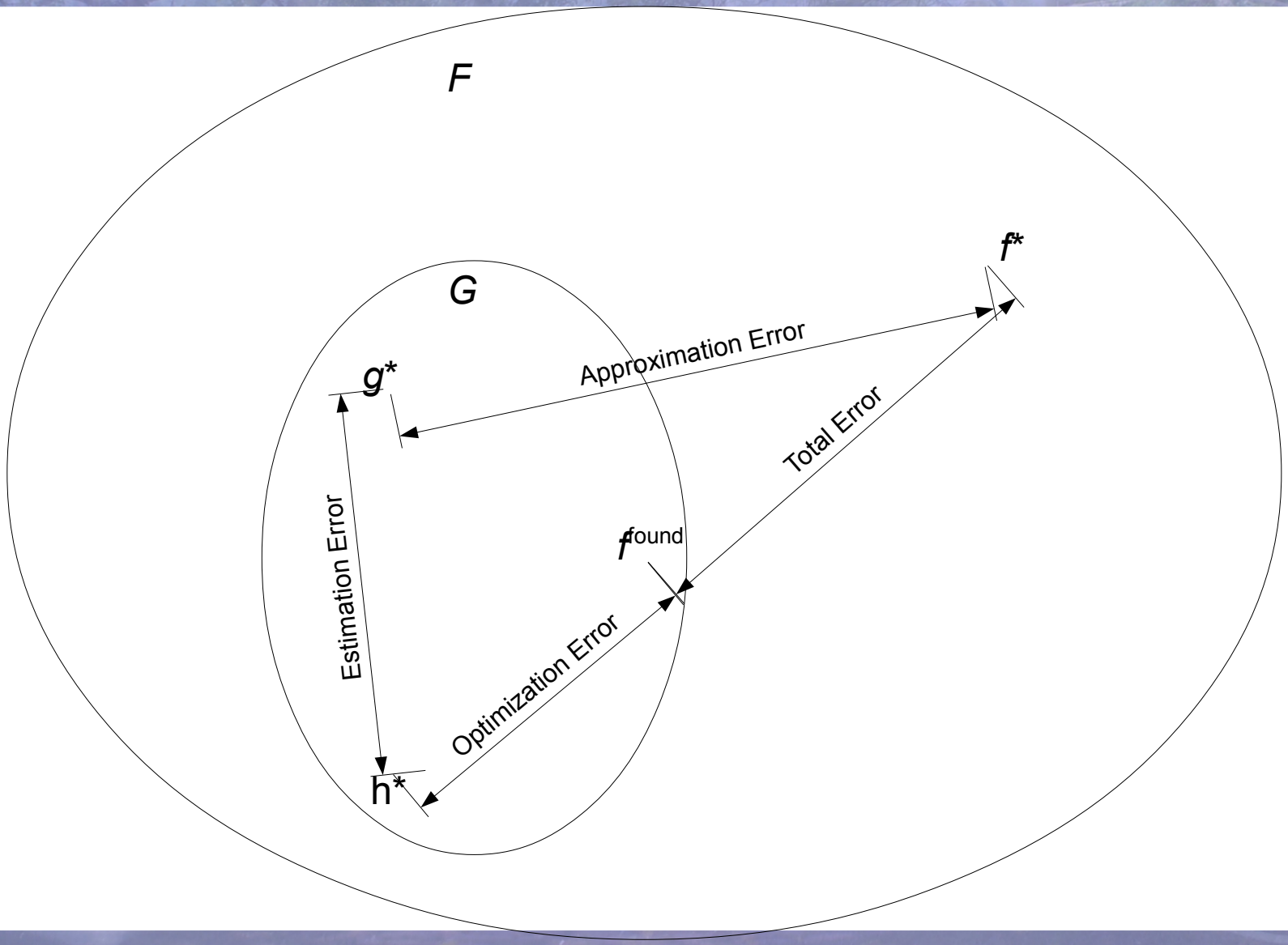
Topics

- How Learning Works + Basic Algorithms
- Estimation Error + Linear SVM
- Approximation Error + Kernel SVM
- Other Theoretical Ideas
- Improving Performance + Random Forest

Error Decomposition

- $R_f \leq$ sum of:
  - **feature informativeness error** $R_{oB} - 0$, due to not having informative features
  - **approximation error** $\min_{h \in G} R_h - R_{oB}$, due to not using G with best h
  - **estimation error** $R_f - \min_{h \in G} R_h$, for $f = \text{argmin}_{h \in G} \text{SearchObjective}(h, n)$, due to not knowing X
  - **optimization error** $R_g - R_f$, for $g = \text{approx argmin}_{h \in G} \text{SearchObjective}(h, n)$, due to optimizing approximately

Error Decomposition



Estimation Error Control

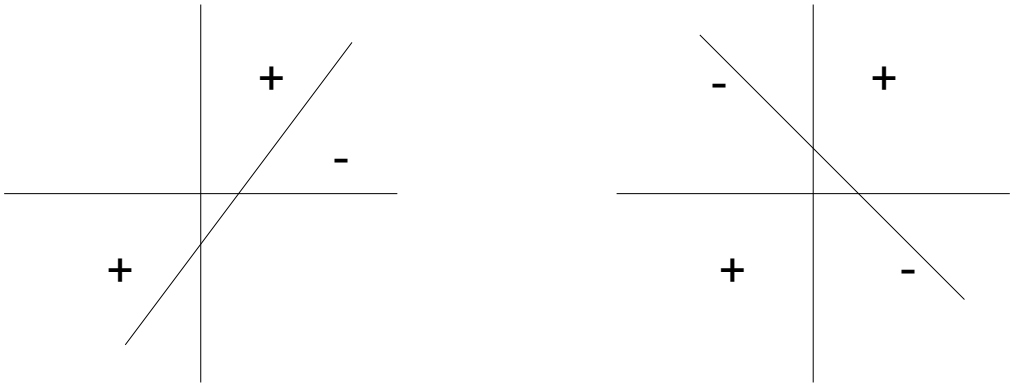
- For some G have a **finite sample** estimator of R_f, i.e. $\forall f \in G \; |R_f - R_{f,n}| \leq B(n, f, p)$ with probability $\geq 1 - p$, such that $\forall e > 0$ and $p > 0$ $\exists n(e, f, p)$ such that $B \leq e$ (**PAC** property)
- If B is same $\forall f$, G has **uniform convergence** and $B(n, f, p) = B(n, C(G), p)$, where C measures of G's **capacity to overfit**
- E.g. G = all hyperplanes has it, i.e. linear regression is good for this reason
- Finite sample bounds hold simultaneously over $\forall f \in G$, in particular over any f returned by A

Estimation Error

- Heuristic *C* is number of parameters to be estimated
- E.g. given *n* credit card and phone numbers, can construct a polynomial that predicts card number from phone number
- This doesn't work if restrict degree of polynomial to a constant < *n*.
- Restricting capacity prevents over-fitting by excluding *f* capable of accurately modeling noise in *S*

VC-dimension for Binary Classifiers

- Max d such that $\exists d$ x arranged such that $\forall$ assignment of binary labels to them, $\exists f \in G$ that can separate them
- E.g. a line can separate any 3 points in 2D, but not 4 points, thus for 2D lines $d = 3$



Using VC-dimension

- With probability $\geq 1 - p$, $R_f \leq R_{f,n} + \sqrt{2d\ln(en/d)/n} + \sqrt{-\ln(p)/(2n)}$
- Some conclusions:
  - If $n/d \rightarrow \infty$ as $n \rightarrow \infty$, $R_{f,n} \rightarrow R_f$
  - Even hyperplanes can overfit for $D \approx n$.
  - Bounds are usually loose since assume worse case, but still conceptually useful
- By restricting on range of parameters can make d smaller, e.g. for sin functions VC-dimension $= \infty$

Using VC-dimension

- For $k = 2$:
  - VC-dimension of a tree is the number of leaves
  - Getting a tree with d leaves means that with $p = 0.05$ have complexity term $\approx \sqrt{d \ln(2en/d)/n} + \sqrt{1.5/n}$
- Good reason for pruning – give up accuracy to reduce complexity and minimize overall risk

Empirical Risk Minimization

- Use $R_{f,n}$ as search objective
- If G has uniform convergence, B is same $\forall f \in G$, and the found one will have finite sample estimation error bound
- Needs small C or very large n to come up with small B
- Only useful for few special cases such as linear regression

Structural Risk Minimization

- Can construct G with a finite sample bound
- Consider $G = \cup H_i$ and corresponding $w_i \in (0, 1)$ picked before seeing the data such that $\sum w_i \leq 1$ and $\forall H_i$ have uniform convergence with bound B.
- Due to multiple testing $R_f \leq B$ doesn't hold, but by Bonferroni correction $\forall f \in S \; |R_f - R_{f,n}| \leq B(n, C(H_i(h)), w_i(f)p)$ with probability $\geq 1 - p$.
- E.g. consider polynomials with increasing degree i and $w_i = 6/(\pi(i+1))^2$

Structural Risk Minimization

- For G with finite sample estimator use search objective = $R_{f,n} + B(n, f, p)$
- Start by looking at simpler models first and stop when for best found f and all h not yet considered $R_{f,n} + B(n, f, p) \leq B(n, h, p)$
- Solution will have finite sample bound
- E.g. can prune decision trees this way – assume $k = 2$ and greedily fold least accurate leaf pair while R_f bound improves. Sign test more efficient though.

Occam Risk Minimization

- Can do SRM over any countable set of hypothesis
- Assume H_i consists of a single hypothesis h and $L \in [0, M]$
- Then $R_f \leq R_{f,n} + M\sqrt{(\ln(1/w_i) + \ln(2/p))/(2n)}$ with probability $\geq 1 - p$.
- Let every h be represented in binary using some code, e.g. gamma, and $w_i = 2^{-|h|}$
- Find $f = \text{argmin}_{h \in G}(R_{h,n} + M\sqrt{((|h|\ln(2) + \ln(2/p))/(2n))}$

ORM for Decision Tree

- Encode structure with 2 bits per node, i.e. using depth first traversal write node, then 1 bit for left child, and 1 bit for the right
- Feature id needs $\lfloor \lg(D) \rfloor$ bits and label $\lfloor \lg(k) \rfloor$
- For split point can scale data into [0, 1] and represent to precision 0.001, so about 10 bits
- With $m/2$ internal nodes and $m/2 + 1$ leaves, $|h| \approx m(10 + \lg(kD)/2)$
- With $p = 0.05$ complexity term = $\sqrt{|h|\ln(2) + \ln(40))/m}$,
- VC bound is better though, but only applies for $k = 2$. Can likewise prune with ORM.

Occam Risk Minimization

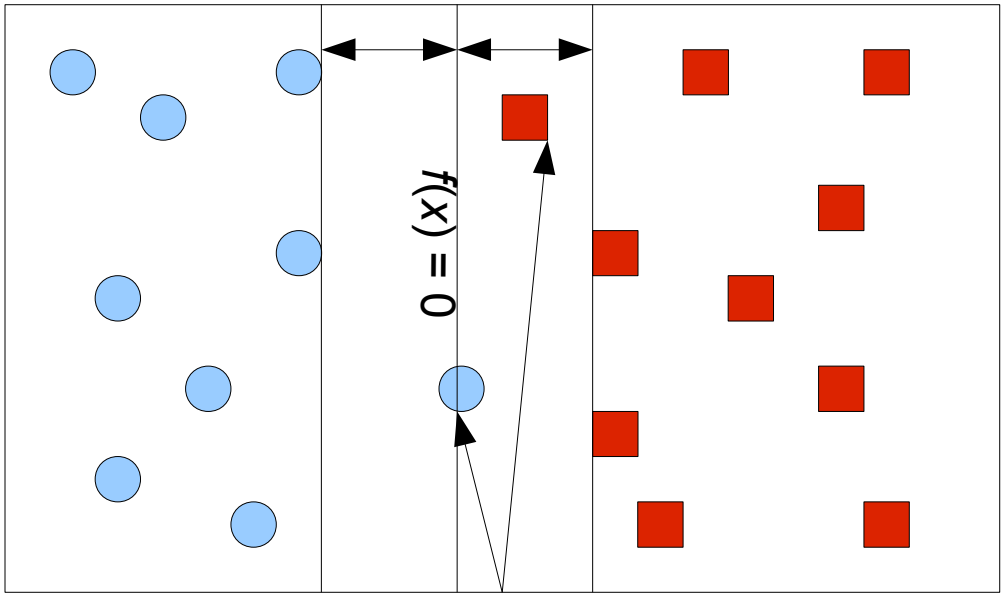
- Very general – it applies ∀task with bounded *L*, and it's usually easy to define and search a suitable *G*
- **Occam razor** – don't make things more complex than needed
- Seems like the ultimate induction principle, but complexity bounds are very loose
- Mostly of conceptual value in practice, but don't increase complexity without need

Margin

- Distance of patches of examples of particular class to partition boundaries
- Intuitively, the larger the margin, the more certain is the partition, so in some cases can bound risk or *d* as function of the margin
- A very complex *f* can have small margins and much smaller risk bounds than suggested by complexity bounds
- Many successful algorithms implicitly or explicitly maximize margins in some way.
- Still, differences of algorithms in risk aren't fully explained by margins

Linear SVM

- Given two classes, compute a linear separator $f(x) = wx + b$ with maximum margins
- Put linear penalty on outsize of margin examples



Linear SVM

min $\frac{1}{2}w^2 + C\sum e_i$ subject to

$y_i f(x_i) \geq 1 - e_i$

$e_i \geq 0$

- Constant $C > 0$ defines trade-off between margin size and accuracy
- Motivation – can bound d as function of margin and radius of the data's enclosing hypersphere
- When walking in a minefield, it's best to walk between mines and not close to any of them

Linear SVM

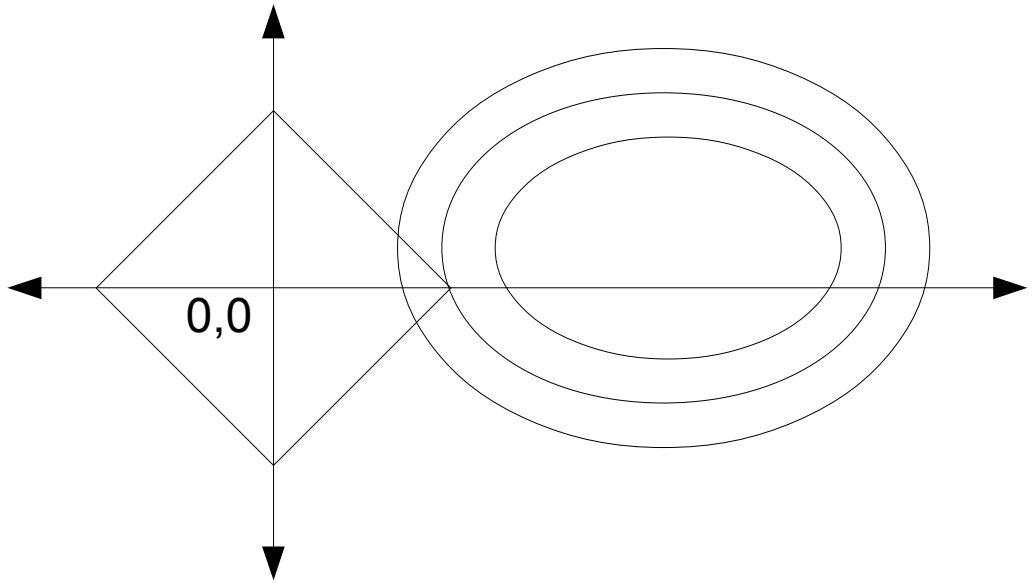
- Cases:
  - $y_i f(x_i) > 1$ – correctly classified and beyond the margin
  - $y_i f(x_i) = 1$ – correctly classified and on the margin
  - $0 < y_i f(x_i) < 1$ – correctly classified and inside the margin
  - $y_i f(x_i) < 0$ – incorrectly classified and on wrong side of the margin

Linear SVM and Sparcity

- Most useful for large n and D, so make an important adjustment – use $\boldsymbol{L_1}$ **weight size penalty** instead
- Also solve equivalent unconstrained problem
- min $l|w|+ \sum\max(0, 1 - y_i f(x_i))$ with $l = 1/C$
- This is convex non-differentiable optimization problem
- L_1 penalty leads to a sparse solution, with many $w_i = 0$

Sparcity

- Solution is sparse because in the equivalent constrained problem the feasible region contains cusps, corresponding to some variables set to 0, and one of them is likely to attain the best feasible level set

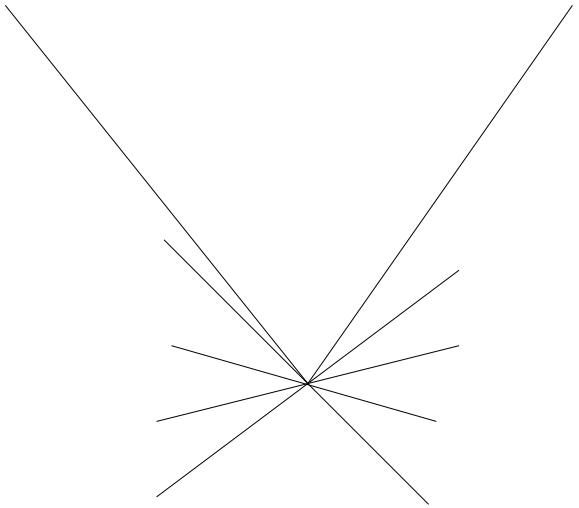


Stochastic Gradient Descent

- Staring from any initial solution, take a step of some size *s* into direction of the expected gradient
- Repeat until convergence, adjusting *s* as necessary
- **Robbins-Munro conditions** – for a convex problem converges if:
  - $\sum_{0 \le i \le \infty} s_i = \infty$ – must be able to walk far enough
  - $\sum_{0 \le i \le \infty} s_i^2 < \infty$ – can't diverge

Subgradients

- Linear SVM objective not differentiable, so use sub-gradient instead
- In 2D, any tangent line to the cusp
- Same result as using gradients, also slow $O(1/\sqrt{n})$ convergence, but good for generalization



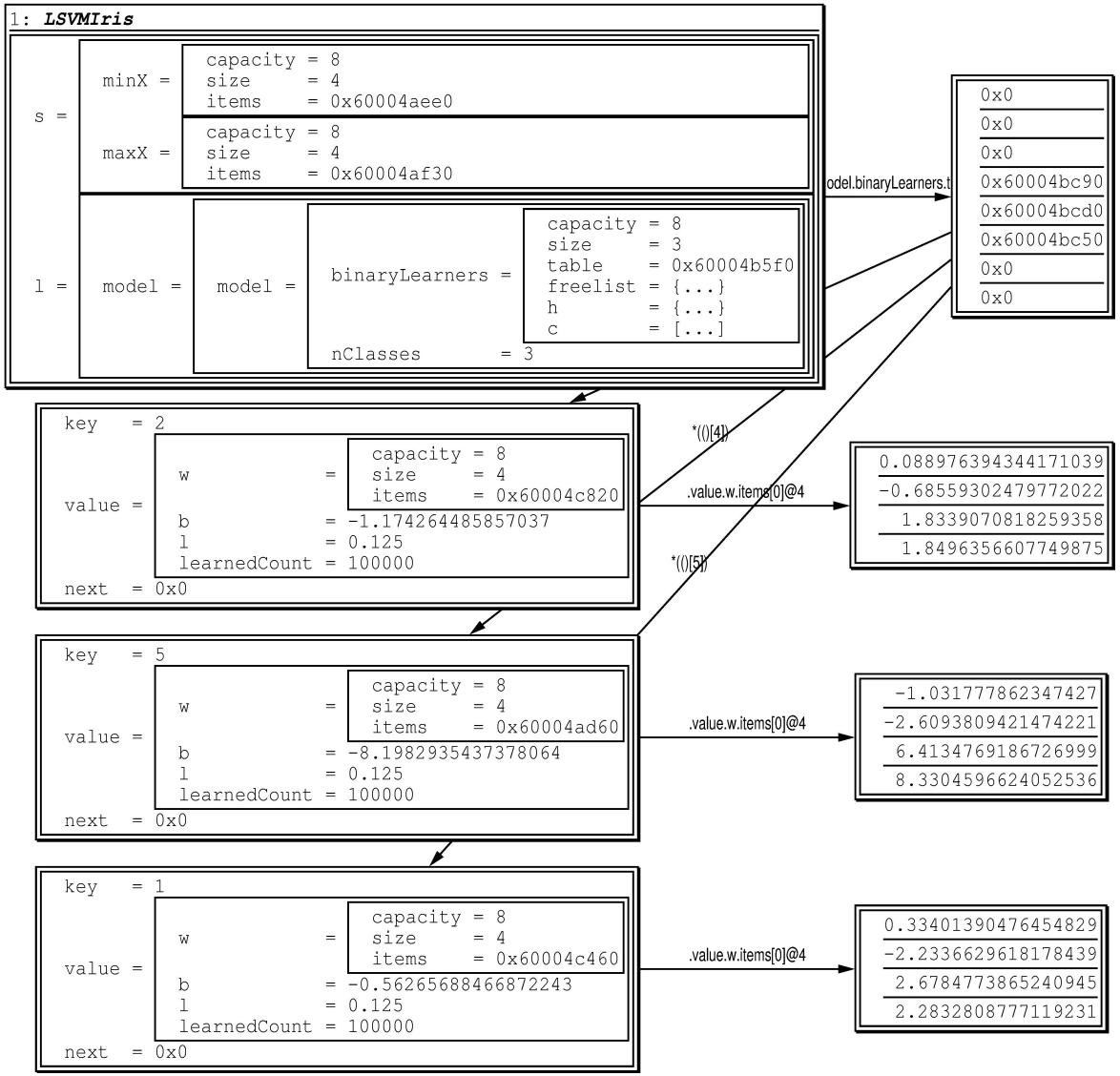
Solving Linear SVM with SGD

- Need expression whose expected value is a subgradient of the function, calculated off single example
- For a given I start with 0's and update:
  - $\forall 0 \leq j \leq D\ w_{ij} \mathrel{-}= s_i(w_{ij} > 0\ ?\ 1 : -1)I - n(y_i f(x_i) > 1\ ?\ 0 : y_i x_i))$
  - $b \mathrel{+}= s_i(y_i f(x_i) > 1\ ?\ 0 : y_i)$
- To be disk friendly make $\lceil 10^5/n \rceil$ passes over the data instead of using random examples

Linear SVM for Many Classes

- Break up multi-class problem into two class problems using **one vs one** (OVO) decomposition
- For k classes train $O(k^2)$ learners for all combinations of binary classifiers
- For prediction choose the class with most votes
- Some good properties – simplicity, asymptotic efficiency for super-linear A, and lowest approximation error among alternatives

Linear SVM for Many Classes



Tuning Parameters

- How to pick *l* for linear SVM?
- Choice of parameter is property of *A* and not *f*,
- Theoretical solution – use SRM or equivalent method, but problem with loose bounds
- Old fashioned way – have a statistician look at the problem and figure out the answer by common sense – costly, slow, and doesn't scale
- Practical solution – **cross-validation**, works well almost always

Cross-Validation

- Partition the data into *k* equal subsets
- *k* times train *A* on *k* − 1 subsets and tests on the remaining one
- Return average risk as performance estimate

Fold	Data Use				
1	Train	Train	Train	Train	Test
2	Train	Train	Train	Test	Train
3	Train	Train	Test	Train	Train
4	Train	Test	Train	Train	Train
5	Test	Train	Train	Train	Train

Cross-Validation

- Impossible to estimate variance of risk estimate
- Estimate usually very good for picking params, though maybe not for predicting risk of *A*
- Sources of variance:
  - *A* itself may be randomized
  - Using different data or its order can give different answer, **stratification helps**
  - Fold results dependent because training data overlaps

Picking Continuous Parameters

- Cross-validation works for discrete sets
- **Grid search** – from very small min to very large max, step by factor of 4
- Usually method of choice, feasible for ≤ 3 params
- For more no clear answer, but **random search** and compass search show some promise
- **Compass search** – starting from some initial solution, take a step in some direction, cycling thru directions

Big Data

- Large *n*, sometimes *D*, **that's it**
- Need scalable *A*, linear SVM or its almost twin logistic regression will do
- More data than is needed for good generalization – can do single pass
- **Incremental SGD** – go through examples in disk-friendly way
- May want to randomly permute – reduces to sorting, which is IO-fast
- Lots of research on parallelizable and online *A*

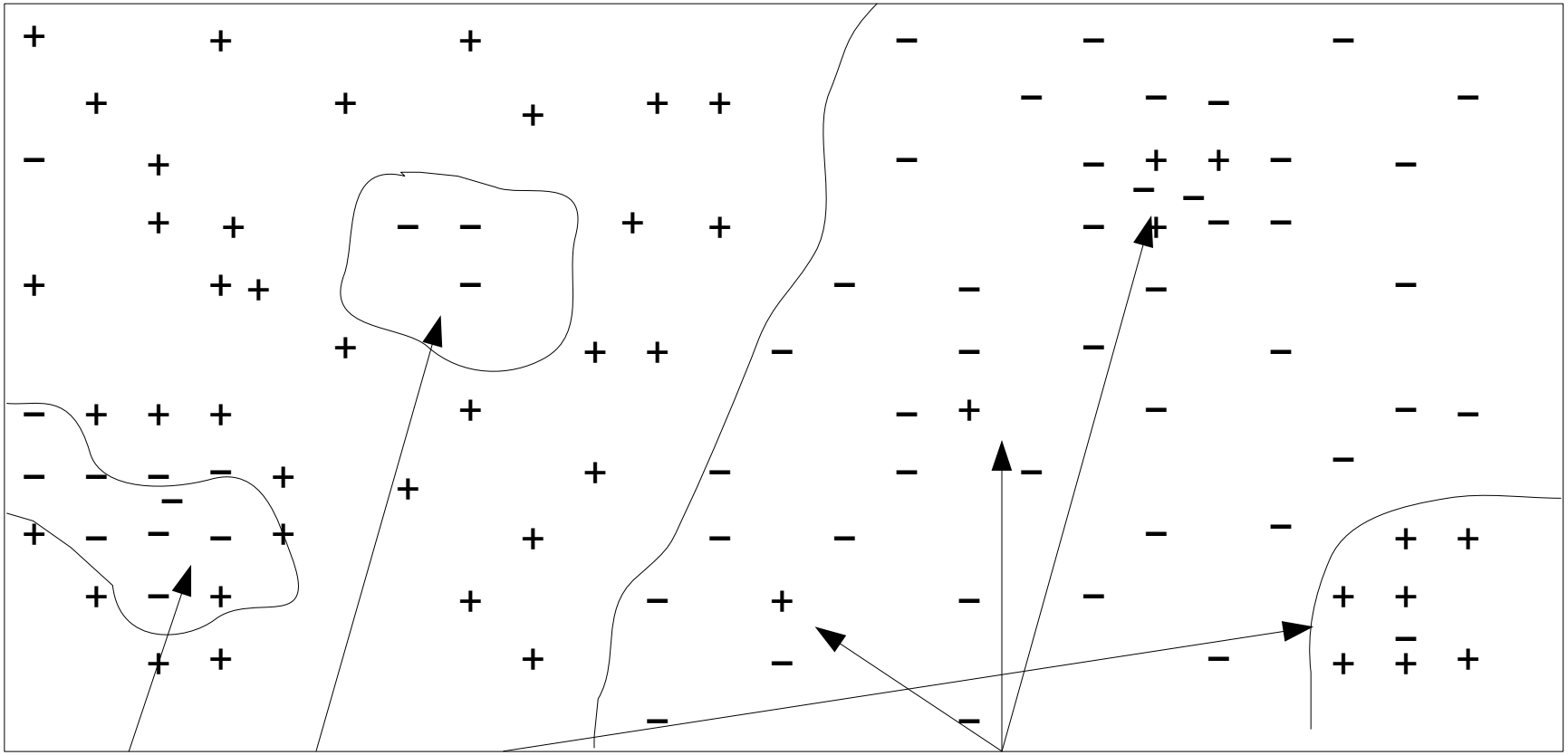
Topics

- How Learning Works + Basic Algorithms
- Estimation Error + Linear SVM
- Approximation Error + Kernel SVM
- Other Theoretical Ideas
- Improving Performance + Random Forest

Approximation Error

- Comes from inability of any *G* to compute the optimal Bayes partition
- The latter for a particular problem may be **arbitrarily complex**
- And therefore need arbitrarily many examples to express
- Some examples seem to be **outliers** because it can be hard to distinguish **hint noise** (some y_i are wrong) from valid information
- **Local patches** of *X* can have different behavior

Approximation Error



Approximation Error

- It's hopeless to try to discover the exact best partition, at least better than asymptotically
- At best can approximate it using results like Weierstrass theorem that can approximate any continuous function arbitrarily well by a polynomial
- Thus usually pick most expressive *G* where have satisfactory control of estimation error
- Unlike for estimation error, don't have general finite sample bounds, only asymptotic ones

Kernels

- Allow efficiently adding features that are combinations of other features
- Done by some feature mapping function *F*
- Computationally feasible only when **kernel trick** applies, i.e. when *A* only uses a dot product, e.g. for linear regression *wx* for some weight vector *w* becomes $F(w)F(x) = K(w, x)$,
- where *K* is a **kernel** function specific to *F*. *K* is computable directly, without mapping to the enhanced space first.

Kernels

- K is valid iff the all examples $n \times n$ matrix M such that $M[i, j] = K_{ij}$ is **symmetric and positive definite** (SPSD)
- Intuitively, K measures similarity
- Applies to non-vector data, e.g. can use edit distance between strings to construct a K
- Even with non-vector X K is an inner product of vectors – has all properties of **Hilbert space**
- Specific choice of K represents prior domain knowledge about the data

Kernels

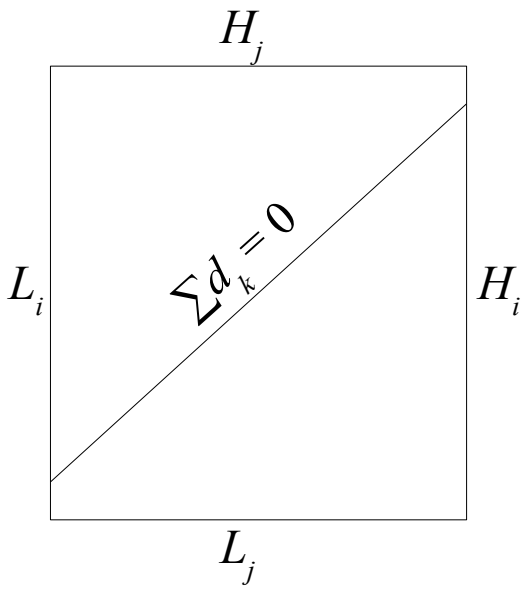
- Useful K for numeric vector X is the **radial basis function** or **Gaussian kernel** defined by $K(x_1, x_2) = \exp(-||x_1 - x_2||/q)$
- Here q is the width parameter, similar to standard deviation of normal distribution
- Can represent any continuous partition boundary – low approximation error
- Can define analogous K from any metric distance for non-vector data, i.e. strings and graphs, though they have specialized K

Kernel SVM

- Replace dot product by kernel, in case of Gaussian add width parameter
- Can't represent *w* explicitly, solve **dual problem** instead:
  - max $\sum y_i d_i - \frac{1}{2}\sum d_i d_i K_{ij}$ subject to
  - $L_i \le d_i \le H_i$
  - $\sum d_i = 0$
  - $[L_i, H_i] = [0, C]$ if $y_i = 1$ and $[-C, 0]$ otherwise
- Given optimal d_i^*, $f(x) = \sum d_i^* K(x_i, x) + b$
- **Support vectors** are x_i for which $|d_i^*| > 0$

Kernel SVM

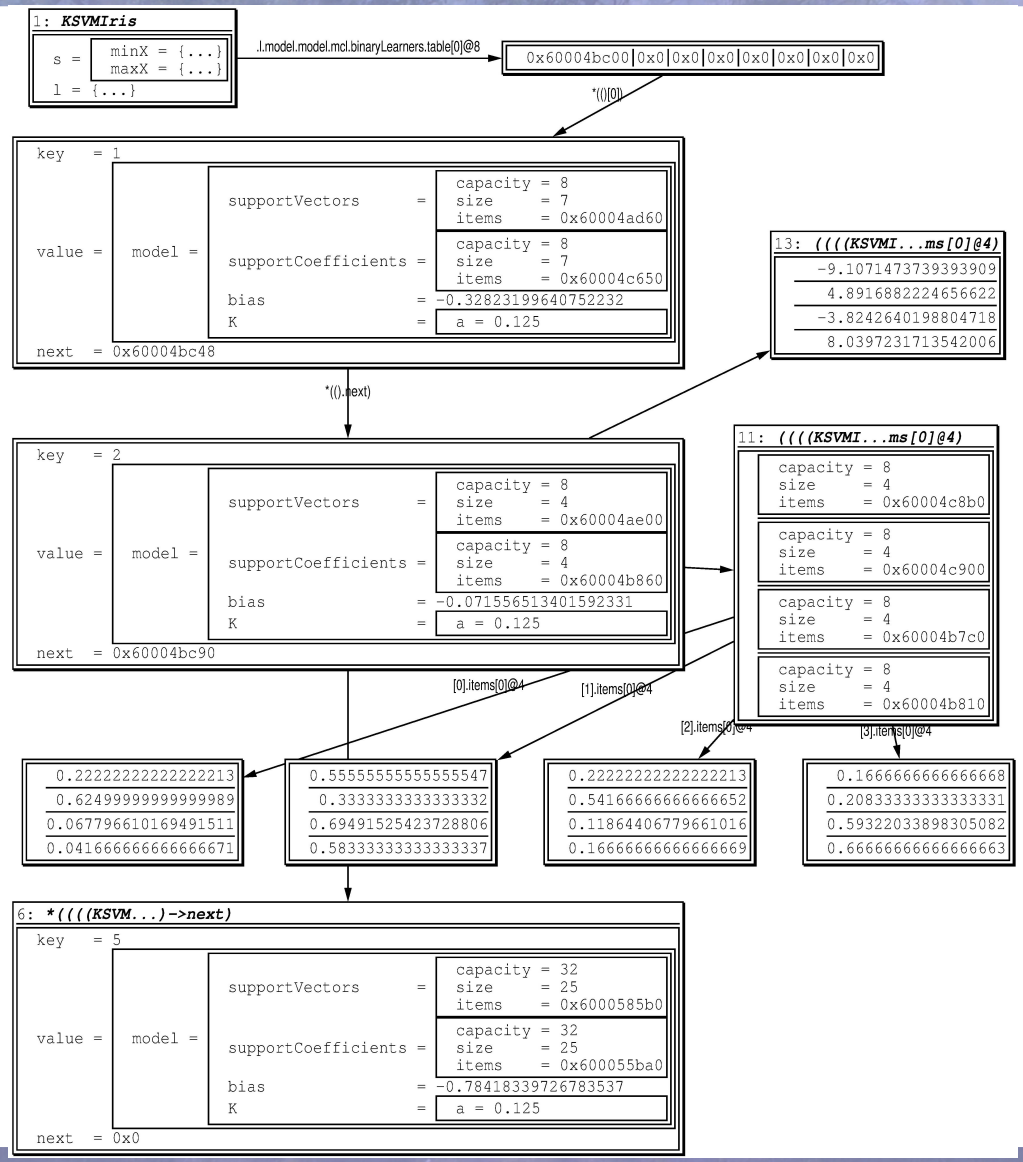
- **Quadratic programming problem**, but black box solvers too slow and need kernel matrix in memory
- Use **SMO algorithm** instead – until convergence greedily pick 2 variables and optimize them



Kernel SVM

- Start with all $d_i = 0$
- Proven convergence, also most d_i remain 0
- Also maintain gradients for variable selection
- If solution isn't optimal, $\exists\, i$ and j such that $d_i < H_i$, $d_j > L_j$, and can increase V by increasing d_i and decreasing d_j by some step s.
- Choose of such i and j where gap = $g_i - g_j$ is maximal
- For efficiency cache values of K

Kernel SVM


1: KSVMIris
s = minX = {...} maxX = {...}
l = {...}
.l.model.model.mcl.binaryLearners.table[0]@8
0x60004bc00|0x0|0x0|0x0|0x0|0x0|0x0|0x0
*(()[0])
key = 1
value = model = supportVectors = capacity = 8 size = 7 items = 0x60004ad60
supportCoefficients = capacity = 8 size = 7 items = 0x60004c650
bias = -0.32823199640752232
K = a = 0.125
next = 0x60004bc48
*(().next)
13: ((((KSVMI...ms[0]@4)
-9.1071473739393909
4.8916882224656622
-3.8242640198804718
8.0397231713542006
key = 2
value = model = supportVectors = capacity = 8 size = 4 items = 0x60004ae00
supportCoefficients = capacity = 8 size = 4 items = 0x60004b860
bias = -0.071556513401592331
K = a = 0.125
next = 0x60004bc90
11: ((((KSVMI...ms[0]@4)
capacity = 8 size = 4 items = 0x60004c8b0
capacity = 8 size = 4 items = 0x60004c900
capacity = 8 size = 4 items = 0x60004b7c0
capacity = 8 size = 4 items = 0x60004b810
[0].items[0]@4
[1].items[0]@4
[2].items[0]@4
[3].items[0]@4
0.22222222222222213
0.62499999999999989
0.067796610169491511
0.041666666666666671
0.55555555555555547
0.3333333333333332
0.69491525423728806
0.58333333333333337
0.22222222222222213
0.54166666666666652
0.11864406779661016
0.16666666666666669
0.1666666666666668
0.20833333333333331
0.59322033898305082
0.66666666666666663
6: *((((KSVM...)->next)
key = 5
value = model = supportVectors = capacity = 32 size = 25 items = 0x6000585b0
supportCoefficients = capacity = 32 size = 25 items = 0x600055ba0
bias = -0.78418339726783537
K = a = 0.125
next = 0x0

Topics

- How Learning Works + Basic Algorithms
- Estimation Error + Linear SVM
- Approximation Error + Kernel SVM
- Other Theoretical Ideas
- Improving Performance + Random Forest

Univeral Consistency

- As $n \to \infty$, sample distribution $\to$ true distribution and estimation error usually disappears and $R_{f,n}$ effectively measures approximation error
- A is **consistent** for a problem if $R_{f(A,\, n)} \to R_{\text{oB}}$ as $n \to \infty$, and **universally consistent** if it's consistent $\forall$ problem within a task
- Thus for a given problem $\forall e > 0$ and $p > 0$ $\forall$ consistent A finds f with a PAC bound after seeing enough samples
- This needs asymptotic and not finite sample bounds on estimation and approximation errors

No Free Lunch Popular Version

- $\exists$ problems where finite training data is harmful and not helpful, so that over all problems no A wins.
- If learning is deterministic and X and Y are bounded, can put uniform distribution on possible P so that every P is equally likely.
- Then $\forall A$ and finite training, if training data is excluded from risk calculation, $E_P R_{f(A,\ \text{data})} = E_{x \notin \text{data}} E_P L(f(x), y)$.
- Since under uniform distribution of P given x all y are equally likely, $E_P L(f(x), y) = EL(f(x)$, uniformly random $y) =$ risk of random guess.

Interpretation of NFL

- Over all problems, learning is impossible
- *A* that win on some problems, must lose on others
- This doesn't contract universal consistency, which is asymptotic
- Pessimism – solve only nice problems, use as much domain knowledge as possible, and don't trust performance comparisons
- Optimism – NFL is nonsense because nobody wants to solve problems where training data is harmful.
- Realism (?) – different real world problems favor different assumptions, e.g. digits aren't irises

NFL Devroye Version

- For classification, but extends to others because classification is the easiest problem
- $\forall A$ that uses n examples for learning f and $e > 0$ $\exists$problem with $R_{oB} = 0$ such that $R_f \geq ½ + e$
- Some problems need arbitrarily many examples, no finite sample approximation error control
- No A is best in all cases because for one that is bad for some problem another may do well, but some can be best in most practical cases

Stability

- Intuitively, stability for A means producing similar f from different data sets
- Let n iid sample training sets T_1 and T_2 differ in only 1 example.
- A is **uniformly *b*-stable** if given f from training on T_1 and h on T_2, $\forall z\ |Lf(z) - L_h(z)| \leq b$.
- If $L \leq M$ and A is b-stable, $\forall T$ consisting of n iid samples, if A trained on T produces f, then $R_f \leq R_{f,n} + b + (2nb + M)\sqrt{\ln(1/p)/(2n)}$ with probability $\geq 1 - p$.
- Thus if $b = O(1/n)$, estimation error is $O(1/\sqrt{n})$
- This holds for kernel SVM, but unknown in general

Topics

- How Learning Works + Basic Algorithms
- Estimation Error + Linear SVM
- Approximation Error + Kernel SVM
- Other Theoretical Ideas
- Improving Performance + Random Forest

Bias-Variance Decomposition

- **Bias** – preference for certain functional relationships over others due to prior knowledge
  - Unlike approximation error, bias takes into account all decisions, e.g. optimization error in using local search
- **Variance** – producing different *f* from different random *S* of same size due to not being able to estimate parameters effectively, mostly due to overfitting
  - Unlike estimation error, it takes part of optimization error and other randomness in decisions into account

Quantifying Bias and Variance

- Let p be a random predictor variable and it's **optimal combination** $C(p) = \text{argmin}_m \mathrm{E}_p[L(p, m)]$.
- E.g. for L being L_2 regression loss, $C(r)$ = the mean over, for L_1 loss the median, and for classification loss the mode.
- Optimal combination gives another way of defining oB(x) = $C(y|x)$, i.e. optimal Bayes can generate arbitrarily many samples from $y|x$ and combine them.

Quantifying Bias and Variance

- **Main predictor** $M(x) = C(f_S(x))$, i.e. combining predictions of trained f on randomly sampled S.
- **bias**(x) = $L(\text{oB}(x), M(x))$ – main predictor performance relative to oB
- **variance**$(x) = \text{E}_S[L(f_S(x), M(x))]$ – cost of performance difference from main predictor
- **noise**$(x) = \text{E}_S[L(\text{oB}(x), y(x))]$ – can't avoid noise, i.e. $\text{E}_x[\text{noise}(x)] = R_{\text{oB}}$
- $\forall x$ and metric L, $L(x, y) \leq \text{noise}(x) + \text{bias}(x) + \text{variance}(x)$. For L_2 loss for regression have =

Measuring Bias and Variance

- **Bagging** simulates the main predictor using bootstrap:
- *T* times for some *T* like 300
  - Create a resample of *S* of size *n*
  - Train *A* on it getting *f*
- To predict given *x*, form the main predictor out of all *f* and return it's answer
- Use **out of bag** estimates – run the bagged predictor on *S* and for an example combine only base *f* in training which it wasn't used

Measuring Bias and Variance

- For bias estimation, since don't know oB, use loss of the main predictor as combined bias + noise measure
- *A* with high enough variance is deemed **unstable**
- Decision tree is **unbiased** and unstable – bagged decision tree is a very good predictor. Pruning increases bias and reduces variance.
- Nearest nearest neighbor also has low bias and high variance. Combining several neighbors increases bias and reduces variance.

Random Forest

- Improves bagging decision tree – decorrellate the trees as much as possible to reduce bias, combination will take care of extra variance
- When building a tree from a resample, at every split randomly select and consider only $\sqrt{D}$ features
- No pruning, but should still cap depth
- $T = 300$ is a good default
- Extends to regression by using average to combine regression trees

Random Forest

- Top performance in many domains in terms of risk, for both classification and regression
- Fast, easily parallelizable
- Out of bag estimate removes need for cross-validation, can use more trees (1000 will do) for accurate estimate
- No params to tune as default number of trees and considered features seem to work well

What Really Controls Estimation Error?

- Randomization ensembles don't overfit more as $T \rightarrow \infty$
- Intuitively, this is because some finite number create a concentration that quickly converges to the set of solutions with similar generalization error
- E.g. majority vote for classification converges to fixed class probabilities
- Simplicity vs stability vs multiple testing – stability seems to win, but unsolved

Conclusions and Main Topics not Covered

- For classification – neural network, boosting, and naive Bayes
- Studies show that generally random forest is best and kernel SVM 2nd
- Classic network generally worse than SVM, deep impressive for some tasks, but not an effective black box yet
- Other leaning tasks – regression, reinforment leaning, clustering, association learning, etc.
- Feature selection – finding best feature subset

Ongoing Research

- Machine learning far from solved – thousands or researchers still working on many problems
- Better theory
- Better *A*
- Potential of deep learning
- Sparsity as way to scale for large *D*
- Computational methods to scale for large *n*

References

- Kedyk, D. (2016). *Commodity Algorithms and Data Structures in C++: Simple and Useful*. 2nd Edition. CreateSpace.