Introduction To Machine Learning

Dmytro Kedyk

Learning Outcomes

- Be able to analyze various learning algorithms using theoretical ideas such as estimation and approximation errors and bias/variance decomposition
- Be familiar with some of the main algorithms for learning, such as decision tree, random forest, and SVM
- Understand some methods for data that is big in number and dimension.

Topics

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

Formalizing a Decidable Situation

- It must be fully represented as an object in some **feature space** *X*
- *X* is usually a vector space—an object is a vector of *D* **features**
- E.g. how to detect free riders in online team games?
  - Use the number of met enemies and % of explored game space as features
  - A player who met 0 enemies and explored < 5% of the game space is a free rider

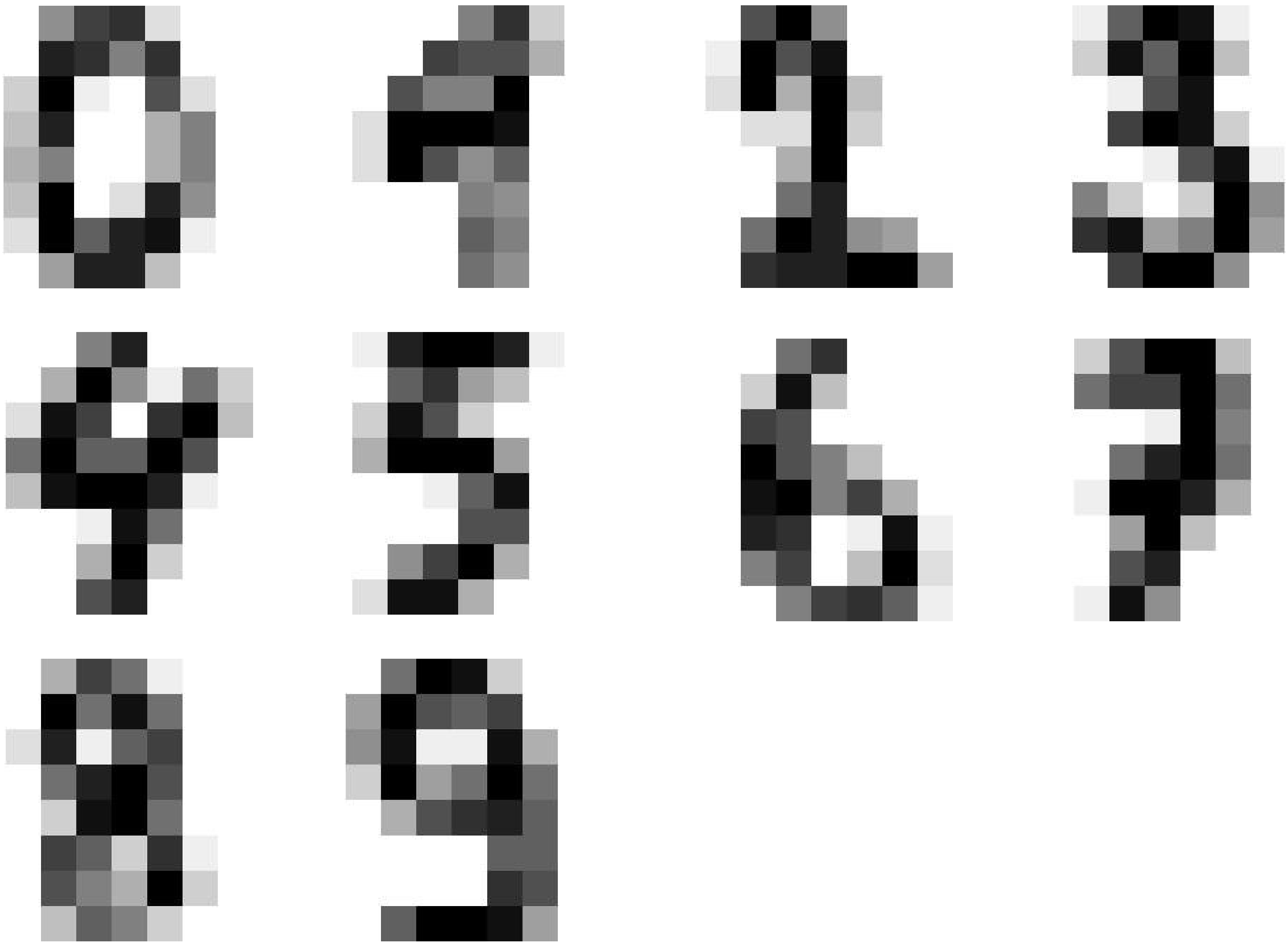
Automatic Learning

- Nature generates data—can never know the true model
- Want **predictor** *f* that decides correctly in specific situations
- So pick a surrogate model and fit it to the data
- The data set S = n examples z_i
- Data reduces uncertainly in the model
  - Never know the model exactly

Classification

- $\forall z_i$ have x_i and **class label** $y \in [0, k - 1]$.
- A **loss function** $L_f(z) = L_f(x, y)$ measures error of f on z.
  - Binary Loss—right decisions have profit 0 and wrong cost 1.
- **Risk** $R_f(P) = E_z[L_f(z)]$ measures generalization error of f.
  - For binary loss R_f = E[% of misclassified z]

Digit Data



Iris Data

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

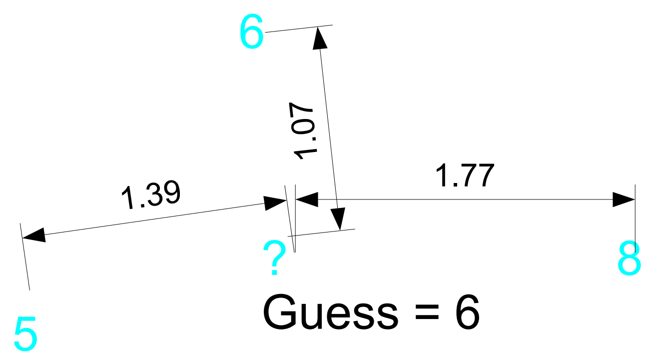
Type	Sepal Length	Sepal Width	Petal Length	Petal Width
Setosa	5.1	3.5	1.4	0.2
Setosa	5.4	3.9	1.7	0.4
Versicolor	6.4	3.2	4.5	1.5
Versicolor	5.6	2.9	3.6	1.3
Virginica	6.2	3.4	5.4	2.3
Virginica	7.2	3.6	6.1	2.5

Data Preparation and Cleaning

- Data may 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
  - $\forall$ feature make range [0, 1]
  - Or mean 0 and variance 1
- Remove useless features like record id
  - Infamous—spot a military tank from sky color

Nearest Neighbor Model

- 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

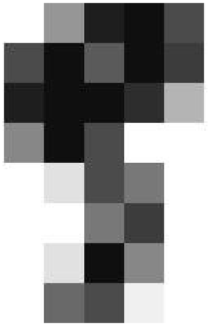


Evaluating Performance

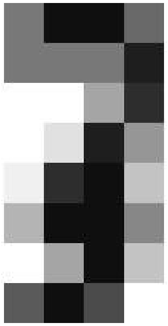
- R_f by **empirical risk** $R_{f,n} = (\sum L_f(z_i))/n$.
- **Accuracy** $= 1 - R_{f,n}$
- Nearest neighbor gets ≈98% accuracy on the digit data

Some NN Mistakes on Digit Data

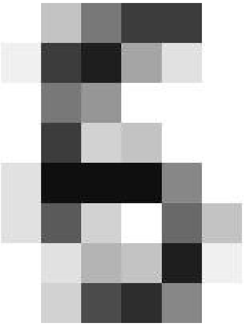
Inferred: 5. Actual: 9



Inferred: 7. Actual: 3



Inferred: 6. Actual: 5



Is Accuracy Accurate?

- **Hoeffding inequality** 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:* $X \in m \pm e(p/2)$
- Digits test set has $n = 1798$, so with 95% probability have 0.98 ± 0.032
- Works for iid test set only

General Problem

- Can never have 100% accuracy with 100% probability
- Ok for many applications, but not for others
- E.g. data-trained self-driving car can never be perfectly safe

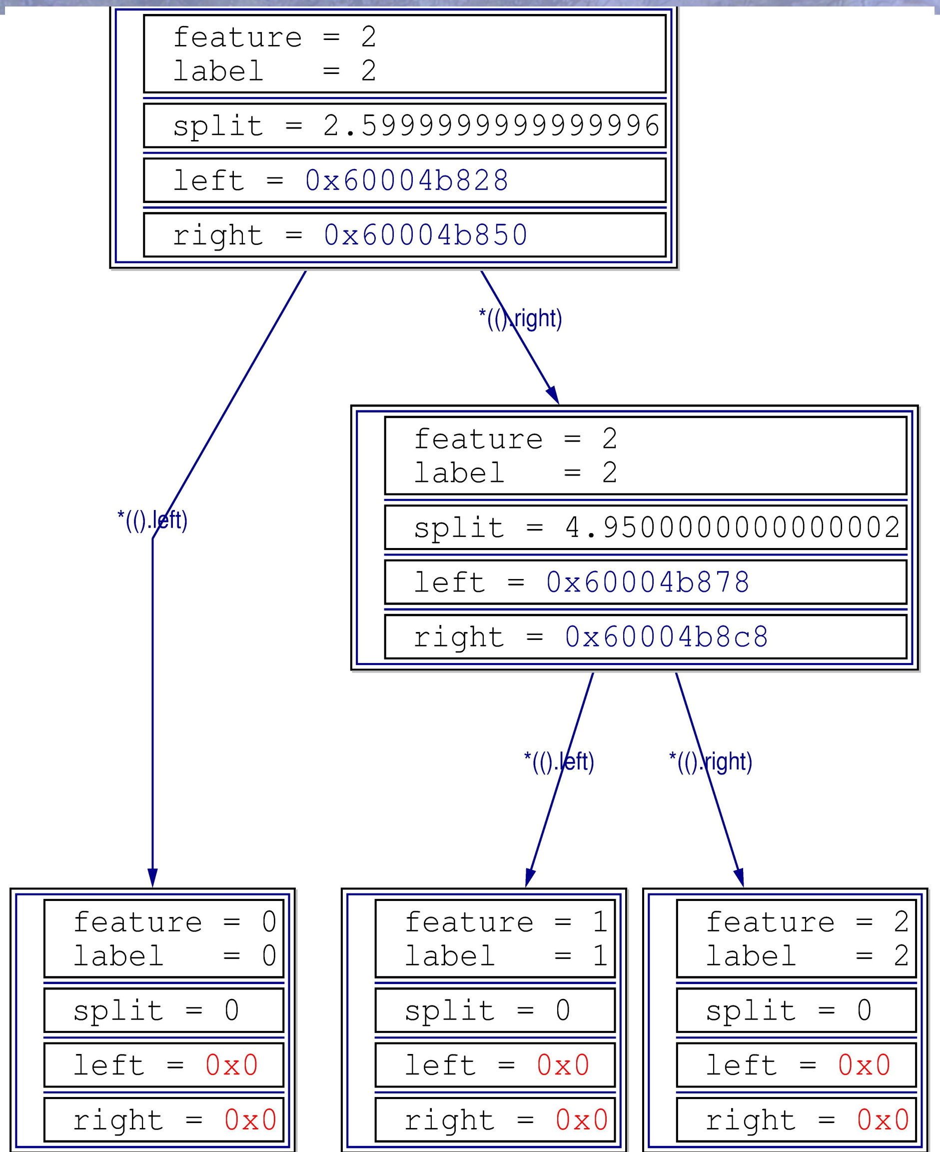
Problems with NN

- Poor accuracy on most data sets despite good results on digits
- Inefficient
  - O(n) memory
  - Index query needs ≈O(n) time with large D
- No explanation of 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 a specific feature to decide which branch to take
- Leaf nodes give the resulting class

Decision Tree for Iris



Picking Split Feature and Value

- For S, **entropy** of labels = $\sum H(p_i)$
  - $H(p) = p\lg(p)$ and p_i = % of examples of class i
- Want a split that minimizes the total entropy after splitting:
  - #left child examples × entropy(left) + #right child examples × entropy(right)
- I.e. pick a split that gains most information

Decision Tree

- Find best split using incremental calculation
- Split data based on it into left and right parts
- Recurse on them until get a pure node or exceed some depth *m* (50?)
- **Prune** to not overfit
- Can explain decisions
- But still makes mistakes
  - ≈87% on Iris and ≈84% on digits

Efficient Incremental Calculation

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

Sign Test Pruning

- Subtree must be better than its root to not be pruned
- Test idea: 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
- Significantly better only if *z* score of best subtree's performance > 1 std (not the usual 2!)
- $z = (\text{nWins} - \text{nGames}/2)/\sqrt{\text{nGames}/4}$

Topics

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

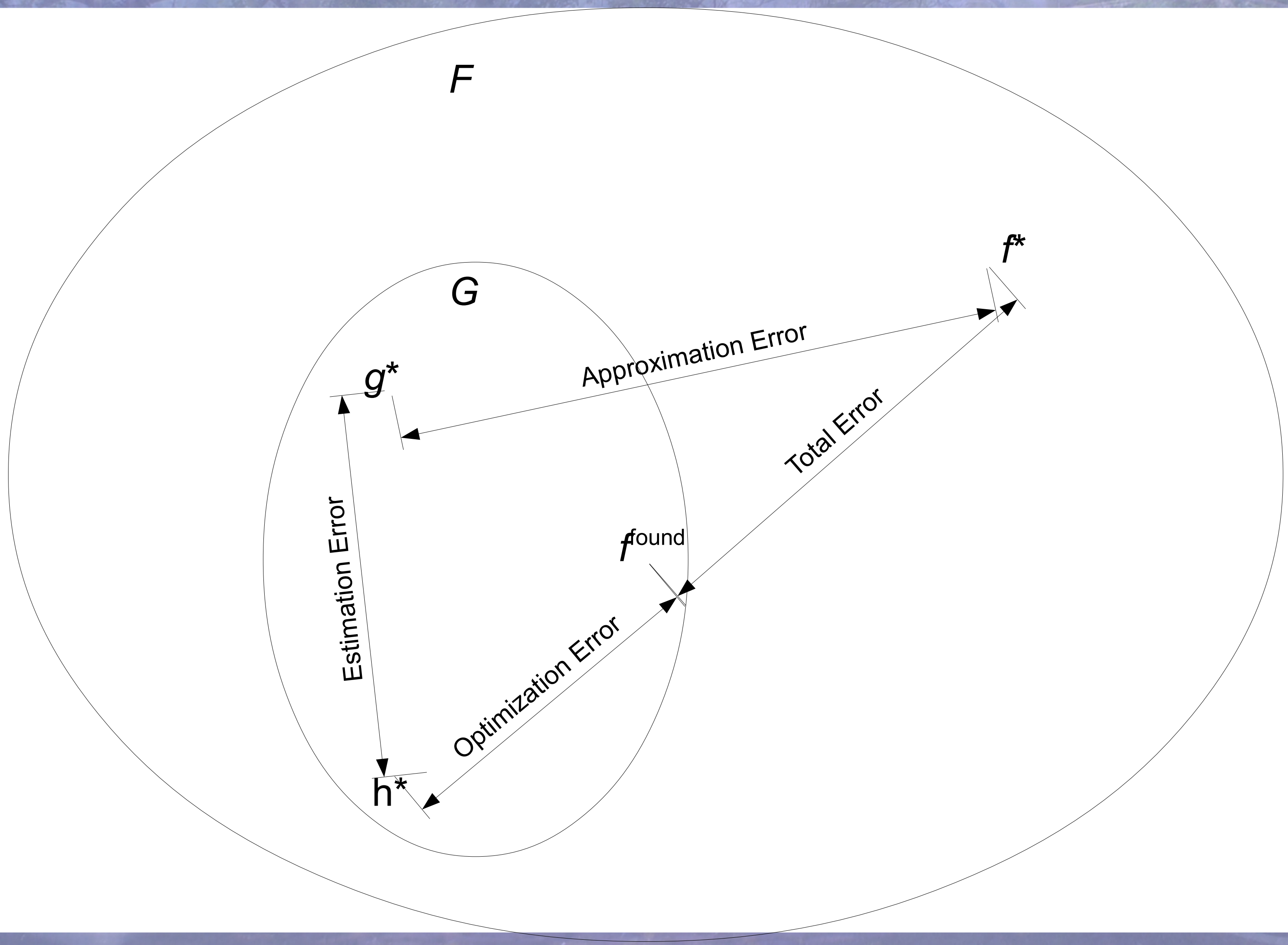
Is There the Best Model?

- Yes—**optimal Bayes** (oB)
- Assume that know the distribution of *y* given *x*
- But still R_{oB} need not be 0
- If have a coin that come up heads 90% of the time, R_{oB} = 10%
- Don't know oB in practice—purely theoretical ideal

Error Decomposition

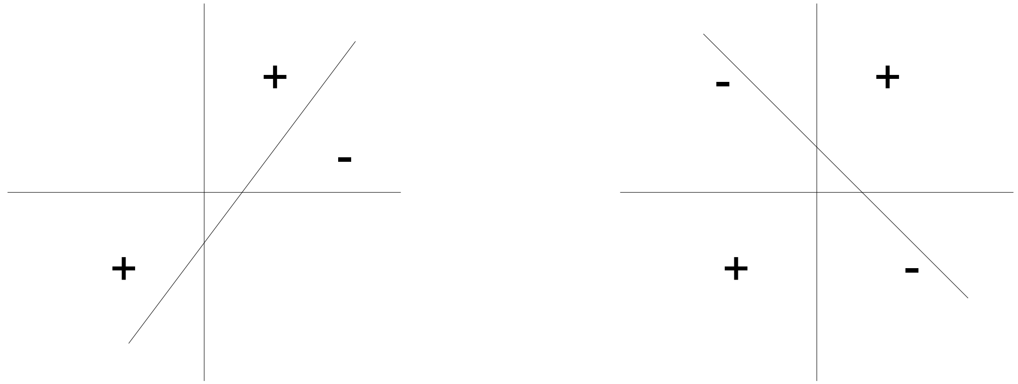
- $R_f \leq$ sum of:
  - **feature informativeness error** R_{oB}, due to not having informative features
  - **approximation error** $\min_{h \in G} R_h - R_{\mathrm{oB}}$, due to not using G with best h
  - **estimation error** $R_f - \min_{h \in G} R_h$, for $f = \mathrm{argmin}_{h \in G}$ SearchObjective(h,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

Error Decomposition



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, so for 2D lines $d = 3$



Using VC-dimension

- Theorem: with probability $\geq 1 - p$, $R_f \leq R_{f,n} + \sqrt{2d\ln(en/d)/n} + \sqrt{-\ln(p)/(2n)}$
- Some conclusions:
  - If $n/d \to \infty$ as $n \to \infty$, $R_{f,n} \to R_f$
  - Even hyperplanes can overfit for $D \approx n$
  - Bounds are usually loose, but still conceptually useful

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
- Restricting parameters can make d smaller, e.g. for sin functions VC-dimension = ∞

General 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 the same $\forall f$, G has **uniform convergence**
  - $B(n, f, p) = B(n, C(G), p)$, where C measures of G's **capacity to overfit**
- E.g. G = all hyperplanes has it

General Estimation Error Control

- Heuristic *C* is the 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 the degree of polynomial to a constant < *n*
- Restricting *C* prevents overfitting by excluding *f* capable of accurately modeling noise in *S*

Empirical Risk Minimization

- Use $R_{f,n}$ as search objective to fit model
- If G has uniform convergence, B is the same $\forall f \in G$, so the found one will have a finite sample estimation error bound
- Problem:
  - Need small C or very large n to get with small B
  - So only useful for a few parametric models such as linear regression, and not for decision tree

Structural Risk Minimization

- For G with a finite sample estimator use search objective = $R_{f,n} + B(n, f, p)$
- Look at simpler models first
- Stop when for the best found f and all h not yet considered $R_{f,n} + B(n, f, p) \leq B(n, h, p)$
- The solution will have a finite sample bound
- E.g. to prune a decision trees—greedily fold least accurate leaf pair while the SRM objective improves

Structural Risk Minimization

- Can construct G with a finite sample bound
- Let $G = \cup H_i$ and corresponding $w_i \in (0, 1)$ be such that $\sum w_i \leq 1$ and $\forall H_i$ have uniform convergence
- Then $\forall f \in G\ |R_f - R_{f,n}| \leq B(n, C(H_i(h)), w_i(f)p)$ with probability $\geq 1 - p$ (by Bonferroni correction)
- With VC dimension can use $w_i = 6/(\pi(i+1))^2$

Occam Risk Minimization

- 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 $\geq 1 - p$ (by Hoeffding and Bonferroni)
- Let every h be represented in binary using some code, such as gamma, and $w_i = 2^{-|h|}$
- Find $f = \operatorname{argmin}_{h \in G}(R_{h,n} + M\sqrt{(|h|\ln(2) + \ln(2/p))/(2n)})$

ORM for Decision Tree

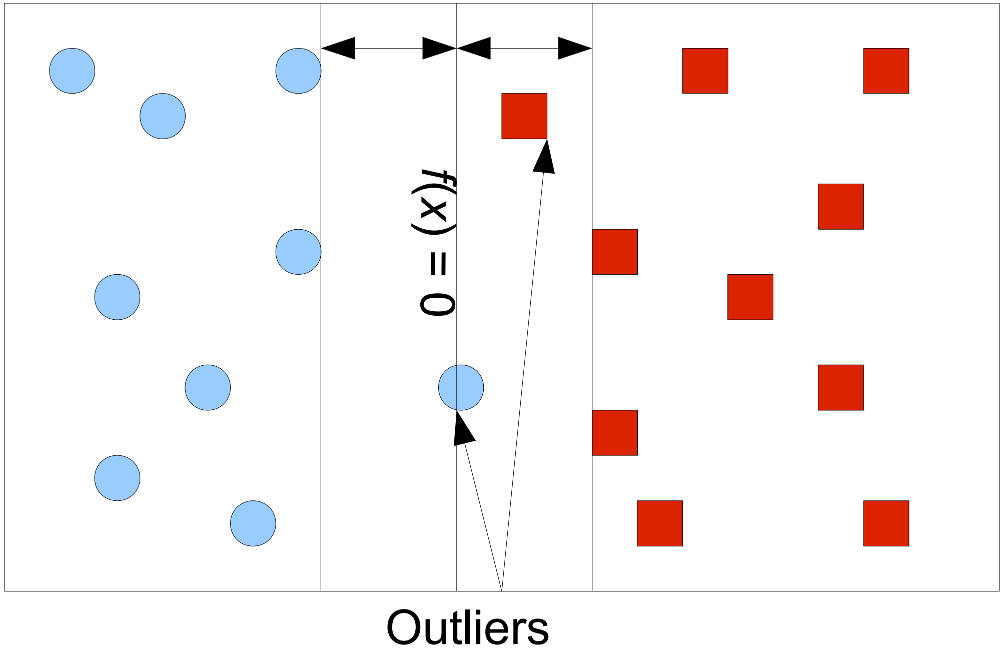
- Structure—2 bits per node, i.e. using DFS to write node, then 1 bit for left child, and 1 bit for the right
- Feature id needs $\lfloor \lg(D) \rfloor$ bits, and label $\lfloor \lg(k) \rfloor$
- Split point—scale data into [0, 1], and represent to precision 0.001, so ≈10 bits
- With $m/2$ internal nodes and $m/2 + 1$ leaves, $|h| \approx m(10 + \lg(kD)/2)$
- With $p = 0.05$, complexity term = $\sqrt{(|h|\ln(2) + \ln(40))/m}$

Occam Risk Minimization

- Very general – it applies $\forall$task with bounded L, not just classification with $k = 2$, 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
  - Hoeffding okay; Bonferroni guilty
- Mostly of conceptual value in practice, but don't increase complexity without need

Linear SVM

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



Margin

- Distance of patches of examples of a particular class to the partition boundaries
- When walking in a minefield, it's best to walk between mines and not close to any of them
- The larger the margin, the more certain is the partition
- A very complex *f* can have small margins
  - much smaller risk bounds than suggested by complexity bounds
- Many successful algorithms implicitly or explicitly maximize margins in some way

Linear SVM

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

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

$e_i \geq 0$

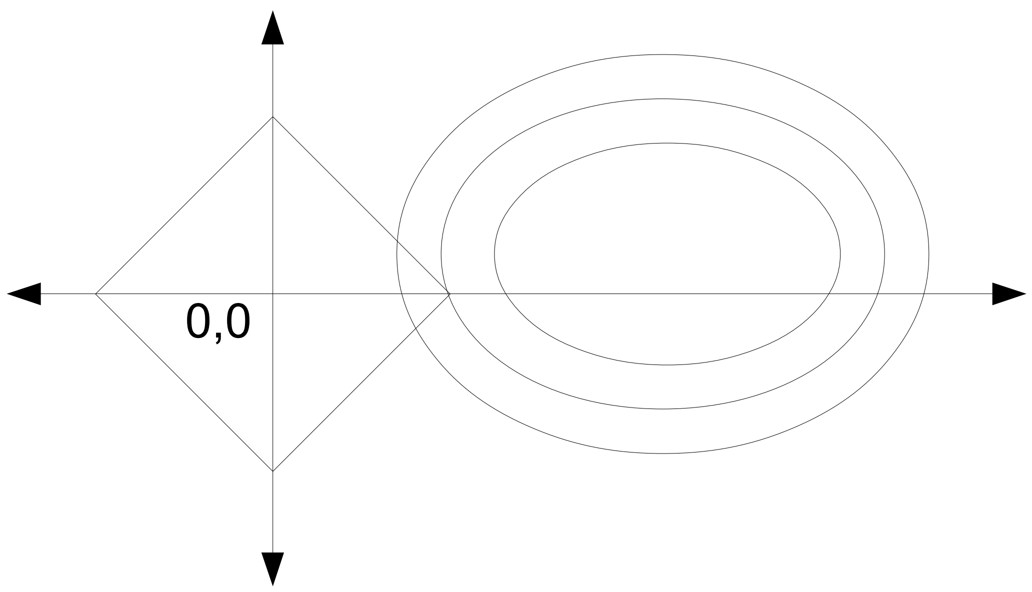
- Constant $C > 0$ defines trade-off between margin size and accuracy
- Can bound d as a function of margin and radius of the data's enclosing hypersphere

Linear SVM and Sparcity

- Most useful for large n and D, so make an important adjustment – use ***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 $w_i = 0$

SVM Sparcity

- In the equivalent constrained problem the feasible region contains cusps that correspond to some variables = 0
- One of them is likely to attain the best feasible level set:

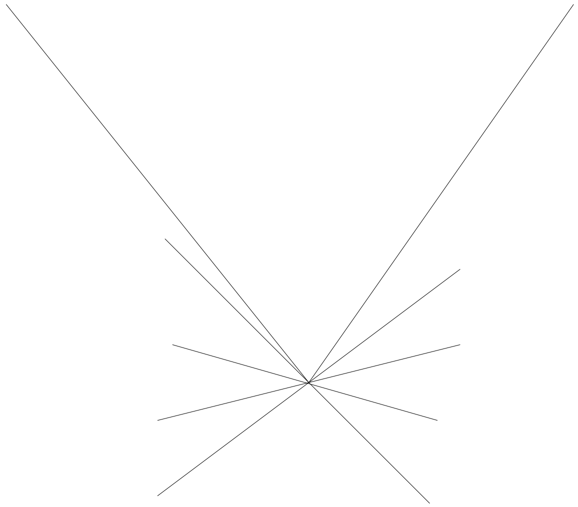


Solving—Stochastic Gradient Descent

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

Subgradients

- Linear SVM objective not differentiable—no gradients, so use subgradients
- In 2D, a subgradient is any tangent line to the cusp
- Slow $O(1/\sqrt{n})$ convergence, but good for generalization



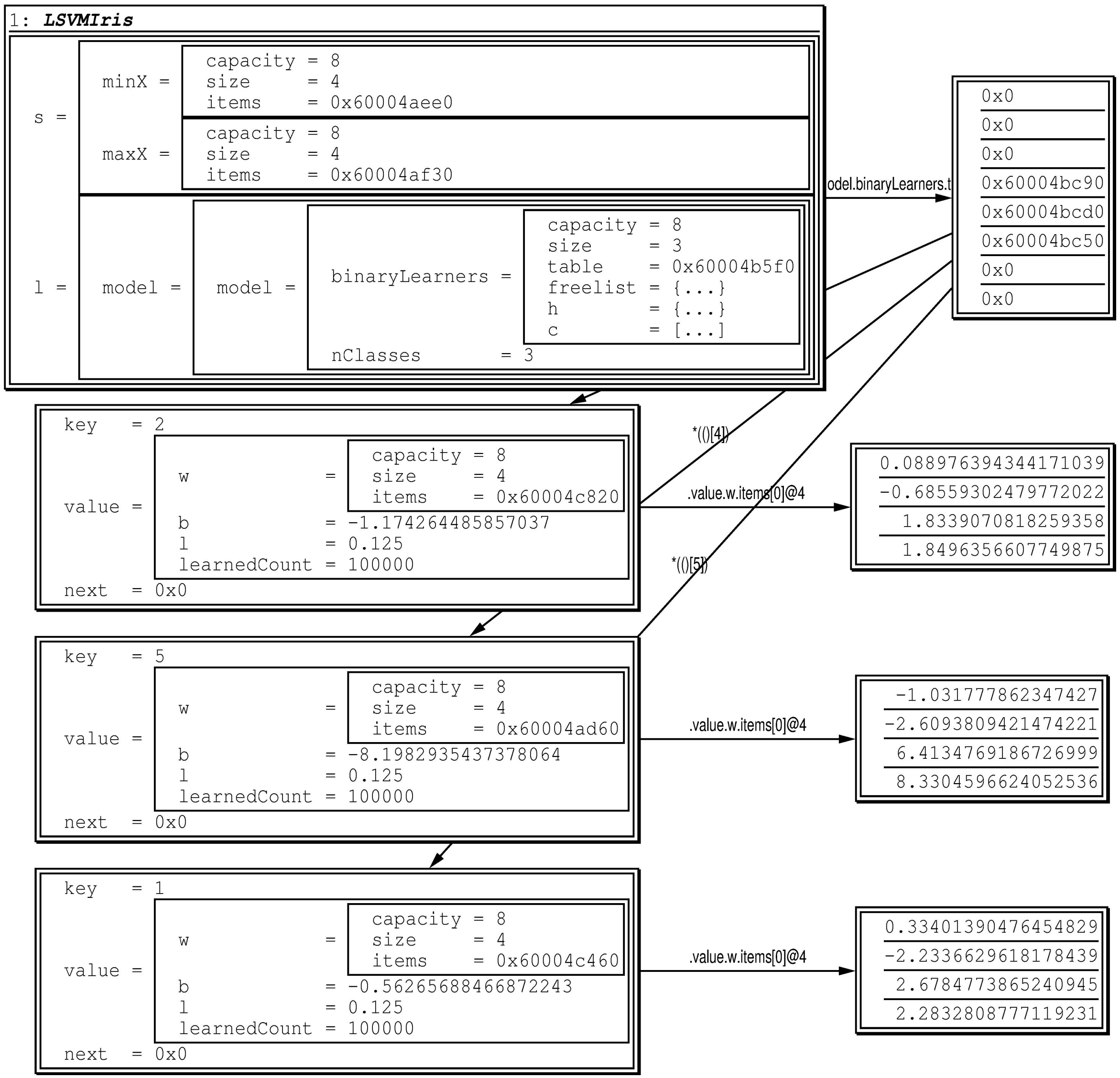
Solving Linear SVM with SGD

- Need expression whose expected value is a subgradient of the function, calculated from a single example
- For a given l start with 0's and update:
  - $\forall 0 \le j \le D$, $w_{ij} -= s_i(w_{ij} > 0 ~?~ 1 : -1)l - n(y_i f(x_i) > 1 ~?~ 0 : y_i x_i))$
  - b += $s_i(y_i f(x_i) > 1 ~?~ 0 : y_i)$
- 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 a multiclass 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 the most votes
- Some good properties – simplicity, asymptotic efficiency for a superlinear binary learner *A*, and the lowest approximation error among alternatives

Linear SVM for Iris



Tuning Parameters

- How to pick / for linear SVM?
- Theoretical—use SRM or equivalent, but problem with loose bounds
- Old-fashioned—have a statistician look at the problem and figure out the answer by common sense—costly, slow, and doesn't scale
- Practical—use **cross-validation**, works well almost always

Cross-Validation for Estimating Performance

- 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
- The estimate is usually very good for picking parameters, but 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 are 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 a factor of 4
- Usually the method of choice, feasible for ≤ 3 parameters
- For more no clear answer, but **random search** shows some promise

Big Data

- Large *n*, sometimes *D*, **that's it**
- Need scalable *A*—linear SVM will do
- More data than is needed for good estimation error control – can do a single SGD pass
- **Incremental SGD** – go through examples in a disk-friendly way
- May want to randomly permute – reduces to sorting, which is I/O-fast
- Lots of research on paralellizable and online *A*

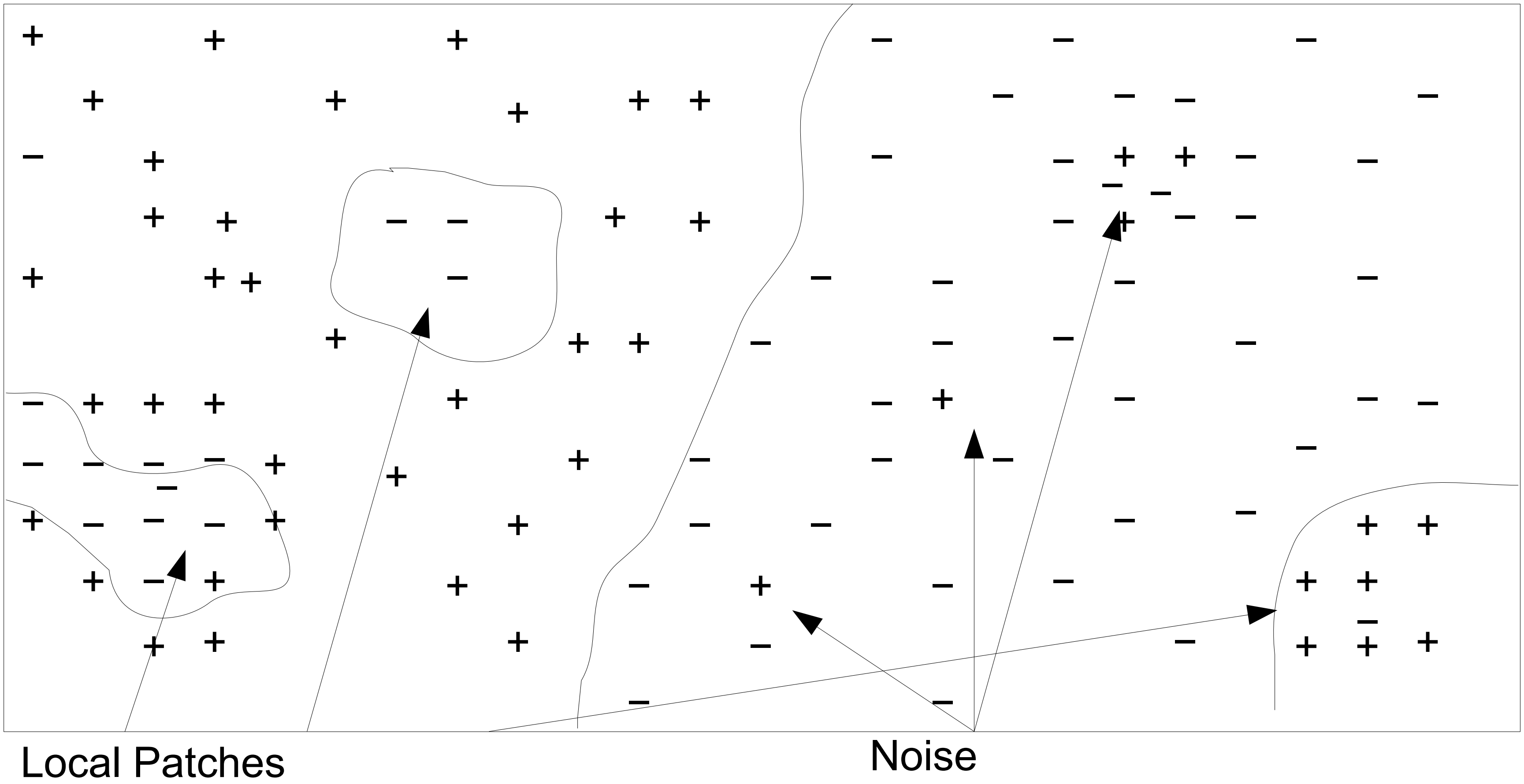
Topics

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

Approximation Error

- For many problems no *G* can compute the oB partition
- The latter for a particular problem may be **arbitrarily complex**
- And thus need arbitrarily many examples to express
- Some examples seem to be **outliers** because it can be hard to distinguish **label noise** (some y_i are wrong) from valid information
- **Local patches** of *X* can have different behavior

Approximation Error



Approximation Error

- At best can approximate the oB partition using results like Weierstrass theorem that can approximate any continuous function arbitrarily well by a polynomial
- So 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—Improve Linear SVM

- Allow efficiently adding features that are combinations of other features
  - Get more-complex-than-linear separation—reduce approximation error
- Done by some feature-mapping function *F*
- *A* must only use a dot product
  - For linear SVM wx becomes $F(w)F(x) = K(w, x)$
  - *K* is a **kernel function**, computable directly

Kernels

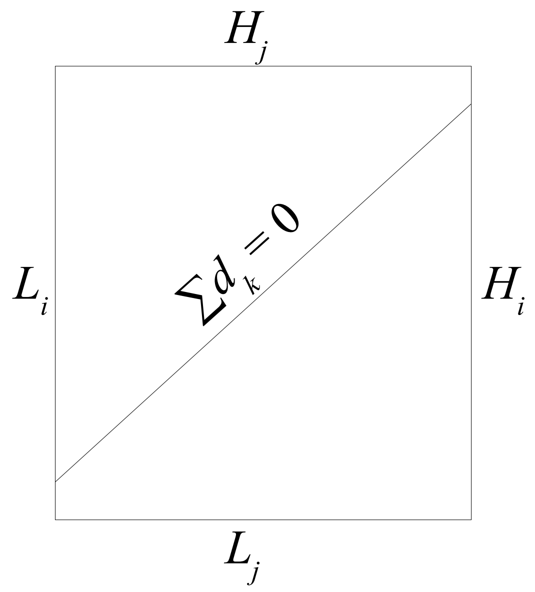
- **Gaussian kernel** is useful K for numeric vector X
  - $K(x_1, x_2) = \exp(-||x_1 - x_2||/\sigma)$; σ is width parameter
- Intuitively, K measures similarity
- Valid iff the all-example $n \times n$ matrix M such that $M[i, j] = K_{ij}$ is **symmetric and positive definite**
- Applies to non-vector data, e.g. can use edit distance between strings to construct a K

Kernel SVM

- Replace dot product by kernel, in case of Gaussian add width parameter
- Can't represent w explicitly, so solve the **dual**:
  - max $\sum y_i d_i - ½\sum d_i d_j K_{ij}$ 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 $|d_i^*| > 0$

Solving Kernel SVM

- **Quadratic programming problem**, but black-box solvers are too slow and need *M* in memory
- **SMO** idea—until convergence greedily pick 2 variables and optimize on the line in the box:



Kernel SVM Results

- Good:
  - Can approximate any continuous boundary with Gaussian Kernel
  - Similar error bound as for linear SVM
  - One of the best black-box *A*
- Bad:
  - Performance isn't perfect
  - SMO is still slow
  - Need O(*n*) memory for support vectors

Topics

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

Univeral Consistency

- A is **consistent** for a problem if $R_{f(A,\, n)} \to R_{\mathrm{oB}}$ as $n \to \infty$
- **Universally consistent** if it's consistent $\forall$ problem
- Intuitively, both estimation and approximation errors $\to$ 0
- kNN—NN which takes majority label of k nearest neighbors
    - Universally consistent if $k \to \infty$ and $k/n \to 0$
    - E.g. $k = \ln(n)$ works
- SVM too, with technical conditions
    - E.g. parameter optimization must not fail

No Free Lunch—Popular Version

- ∃ problems where finite training data is harmful and not helpful, so that over all problems no *A* wins
- Doesn't contract universal consistency, which is asymptotic
- Over all problems, learning is impossible
- *A* that win on some problems, must lose on others

Interpretations of NFL

- **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 are well-linearly-separable, but other data isn't

NFL—Devroye Version

- $\forall A$ that uses n examples for learning f and $e > 0$, $\exists$ a 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

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, 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, 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** $C(p) = \text{argmin}_m E_p[L(p, m)]$
- E.g. C = the mode(majority) for binary L
- $oB(x) = C(y|x)$ because generate arbitrarily many samples from $y|x$ and combine them
- **Main predictor** $M(x) = C(f_S(x))$
  - Combine predictions of trained f on randomly sampled S.

Quantifying Bias and Variance

- **Bias**(x) = L(oB(x), M(x)) – main predictor performance relative to oB
- **Variance**(x) = $E_S[L(f_S(x), M(x))]$ – cost of performance difference from main predictor
- **Noise**(x) = $E_S[L(\mathrm{oB}(x), y(x))]$ – can't avoid noise, i.e. $E_x[\mathrm{noise}(x)] = R_{\mathrm{oB}}$
- $\forall x$ and metric L, $L(x, y) \leq \mathrm{noise}(x) + \mathrm{bias}(x) + \mathrm{variance}(x)$.

Measuring Bias and Variance

- **Bagging** simulates the main predictor using bootstrap:
- *T* times for some *T* such as 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

- Don't know oB, so use loss of the main predictor as combined bias + noise measure
- *A* with high enough variance is **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 bagged decision tree—decorrelate 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

Random Forest

- Top performance in many domains in terms of risk
- Fast, easily parallelizable
- Out-of-bag estimate removes need for cross-validation
  - Use more trees (1000 will do) for an accurate estimate
- No parameters to tune
  - Default number of trees and considered features work well

What Really Controls Estimation Error?

- Randomization ensembles don't overfit more as $T \longrightarrow \infty$
- Intuitively, 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

- Generally random forest is best, and kernel SVM 2nd
- Deep network best for some tasks, but not an effective black box yet
- Machine learning far from solved – thousands or researchers still working on many problems
- Focus on data preparation
- Go for insight—decision tree and linear SVM first
- Best of random forest and kernel SVM next

References

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