Lecture 4: Ensemble Learning

Pilsung Kang

School of Industrial Management Engineering

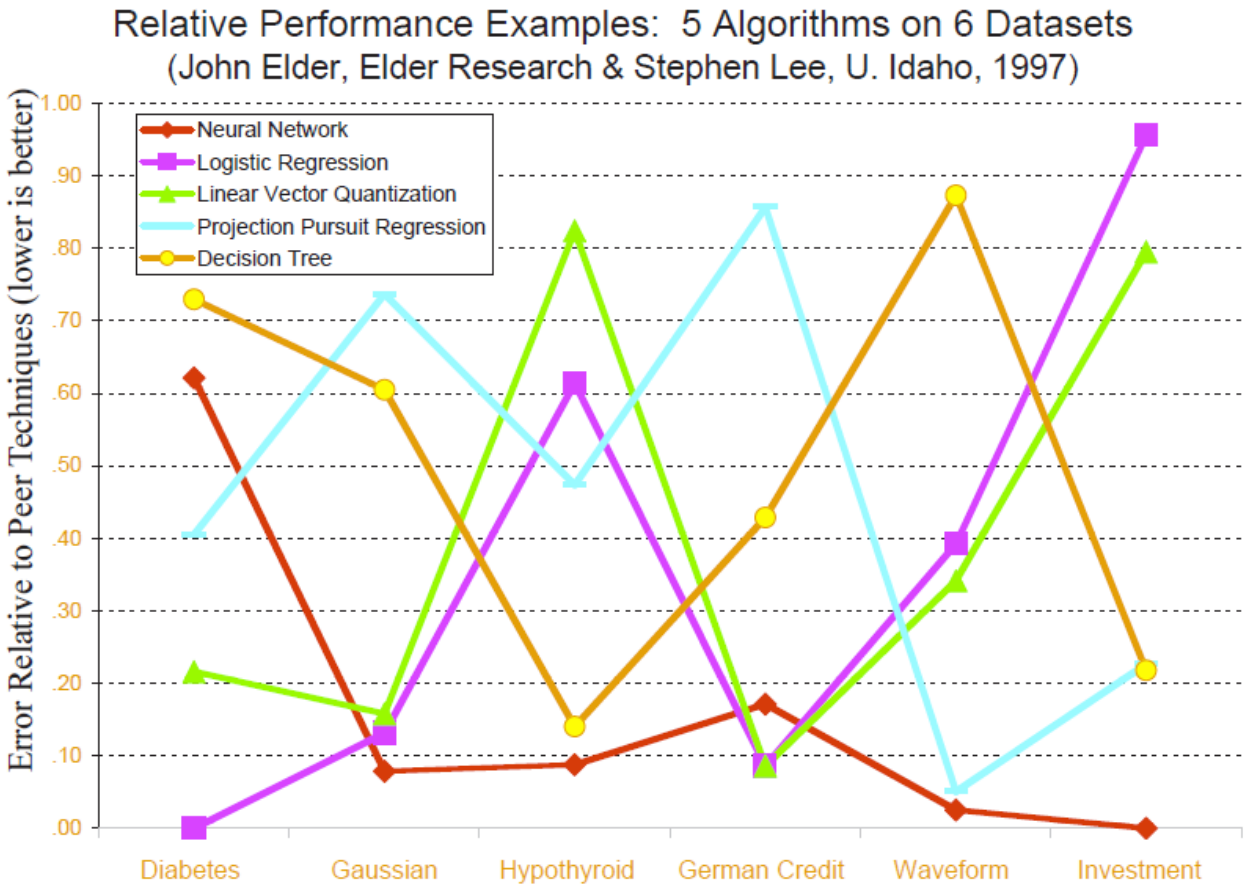
Korea University

AGENDA

Backgrounds

Seni and Elder (2010)

- Can we have a superior algorithm for all datasets?
  - ✓ Every algorithm scored best or next-to-best on at least two of the six data sets.

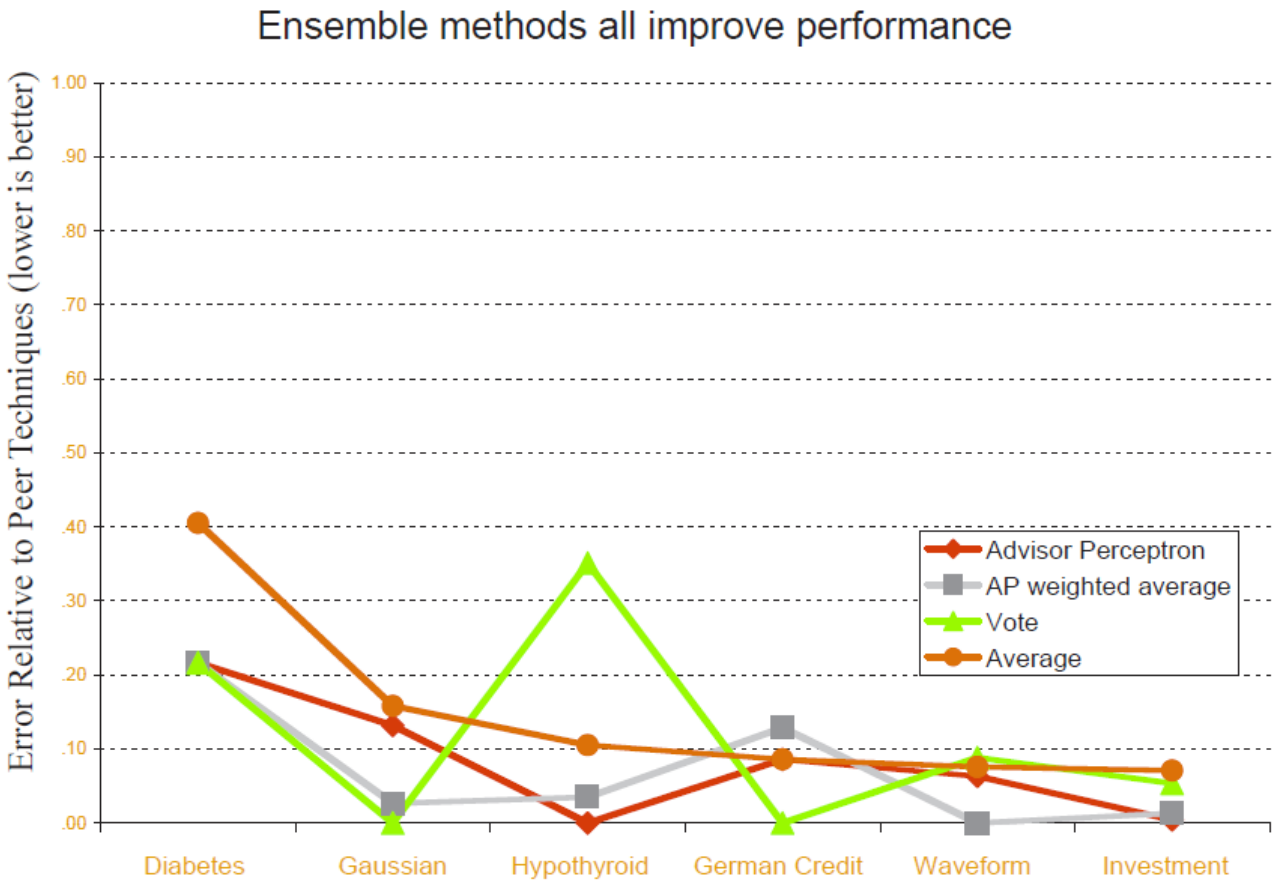


Backgrounds

- No Free Lunch Theorem
  - ✓ Can we expect any classification method to be superior or inferior overall?
  - ✓ No Free Lunch Theorem: No
  - ✓ If the goal is to obtain good generalization performance, there is no context-independent or usage-independent reasons to favor one algorithm over others
  - ✓ If one algorithm seems to outperform another in a particular situation, it is a consequence of its fit to a particular pattern recognition problem
  - ✓ In practice, experience with a broad range of techniques is the best insurance for solving arbitrary new classification problems

Motivation

- However, if they are properly combined…
  - ✓ Every ensemble method competes well against the best of the individual algorithms



Empirical Evidence

Opitz and Maclin (1999)

- Empirical study 1: Single vs. Ensemble algorithms for 23 datasets

			Features		Neural Network			
Data Set	Cases	Class	Cont	Disc	Inputs	Outputs	Hiddens	Epochs
breast-cancer-w	699	2	9	-	9	1	5	20
credit-a	690	2	6	9	47	1	10	35
credit-g	1000	2	7	13	63	1	10	30
diabetes	768	2	9	-	8	1	5	30
glass	214	6	9	-	9	6	10	80
heart-cleveland	303	2	8	5	13	1	5	40
hepatitis	155	2	6	13	32	1	10	60
house-votes-84	435	2	-	16	16	1	5	40
hypo	3772	5	7	22	55	5	15	40
ionosphere	351	2	34	-	34	1	10	40
iris	159	3	4	-	4	3	5	80
kr-vs-kp	3196	2	-	36	74	1	15	20
labor	57	2	8	8	29	1	10	80
letter	20000	26	16	-	16	26	40	30
promoters-936	936	2	-	57	228	1	20	30
ribosome-bind	1877	2	-	49	196	1	20	35
satellite	6435	6	36	-	36	6	15	30
segmentation	2310	7	19	-	19	7	15	20
sick	3772	2	7	22	55	1	10	40
sonar	208	2	60	-	60	1	10	60
soybean	683	19	-	35	134	19	25	40
splice	3190	3	-	60	240	2	25	30
vehicle	846	4	18	-	18	4	10	40

Empirical Evidence

- Empirical study 1: Single vs. Ensemble algorithms for 23 datasets
  - ✓ Error rate: the lower, the better

	Neural Network					C4.5			
				Boosting				Boosting	
Data Set	Stan	Simp	Bag	Arc	Ada	Stan	Bag	Arc	Ada
breast-cancer-w	3.4	3.5	3.4	3.8	4.0	5.0	3.7	3.5	3.5
credit-a	14.8	13.7	13.8	15.8	15.7	14.9	13.4	14.0	13.7
credit-g	27.9	24.7	24.2	25.2	25.3	29.6	25.2	25.9	26.7
diabetes	23.9	23.0	22.8	24.4	23.3	27.8	24.4	26.0	25.7
glass	38.6	35.2	33.1	32.0	31.1	31.3	25.8	25.5	23.3
heart-cleveland	18.6	17.4	17.0	20.7	21.1	24.3	19.5	21.5	20.8
hepatitis	20.1	19.5	17.8	19.0	19.7	21.2	17.3	16.9	17.2
house-votes-84	4.9	4.8	4.1	5.1	5.3	3.6	3.6	5.0	4.8
hypo	6.4	6.2	6.2	6.2	6.2	0.5	0.4	0.4	0.4
ionosphere	9.7	7.5	9.2	7.6	8.3	8.1	6.4	6.0	6.1
iris	4.3	3.9	4.0	3.7	3.9	5.2	4.9	5.1	5.6
kr-vs-kp	2.3	0.8	0.8	0.4	0.3	0.6	0.6	0.3	0.4
labor	6.1	3.2	4.2	3.2	3.2	16.5	13.7	13.0	11.6
letter	18.0	12.8	10.5	5.7	4.6	14.0	7.0	4.1	3.9
promoters-936	5.3	4.8	4.0	4.5	4.6	12.8	10.6	6.8	6.4
ribosome-bind	9.3	8.5	8.4	8.1	8.2	11.2	10.2	9.3	9.6
satellite	13.0	10.9	10.6	9.9	10.0	13.8	9.9	8.6	8.4
segmentation	6.6	5.3	5.4	3.5	3.3	3.7	3.0	1.7	1.5
sick	5.9	5.7	5.7	4.7	4.5	1.3	1.2	1.1	1.0
sonar	16.6	15.9	16.8	12.9	13.0	29.7	25.3	21.5	21.7
soybean	9.2	6.7	6.9	6.7	6.3	8.0	7.9	7.2	6.7
splice	4.7	4.0	3.9	4.0	4.2	5.9	5.4	5.1	5.3
vehicle	24.9	21.2	20.7	19.1	19.7	29.4	27.1	22.5	22.9

Empirical Evidence

Caruana and Niculescu-Mizil (2006)

- Empirical study 2: 8 algorithms on 11 datasets
  - ✓ Algorithms
    - SVM, ANN, Logistic regression (LOGREG), Naïve Bayes (NB), KNN, Random Forests (RF), Decision Trees (DT), Bagged trees (BAG-DT), Boosted trees (BST-DT), Boosted stumps (BST-STMP)
  - ✓ Data sets

PROBLEM	#ATTR	TRAIN SIZE	TEST SIZE	%POZ
ADULT	14/104	5000	35222	25%
BACT	11/170	5000	34262	69%
COD	15/60	5000	14000	50%
CALHOUS	9	5000	14640	52%
COV_TYPE	54	5000	25000	36%
HS	200	5000	4366	24%
LETTER.P1	16	5000	14000	3%
LETTER.P2	16	5000	14000	53%
MEDIS	63	5000	8199	11%
MG	124	5000	12807	17%
SLAC	59	5000	25000	50%

Empirical Evidence

- Empirical study 2: 8 algorithms on 11 datasets
  - ✓ Normalized score by datasets

MODEL	CAL	COVT	ADULT	LTR.P1	LTR.P2	MEDIS	SLAC	HS	MG	CALHOUS	COD	BACT	MEAN
BST-DT	PLT	**.938**	.857	**.959**	**.976**	.700	.869	**.933**	.855	**.974**	**.915**	.878*	**.896***
RF	PLT	.876	.930	.897	.941	**.810**	.907*	.884	.883	.937	.903*	.847	.892
BAG-DT	–	.878	.944*	.883	.911	.762	.898*	.856	**.898**	.948	.856	**.926**	.887*
BST-DT	ISO	.922*	.865	.901*	.969	.692*	.878	.927	.845	.965	.912*	.861	.885*
RF	–	.876	.946*	.883	.922	.785	.912*	.871	.891*	.941	.874	.824	.884
BAG-DT	PLT	.873	.931	.877	.920	.752	.885	.863	.884	.944	.865	.912*	.882
RF	ISO	.865	.934	.851	.935	.767*	**.920**	.877	.876	.933	.897*	.821	.880
BAG-DT	ISO	.867	.933	.840	.915	.749	.897	.856	.884	.940	.859	.907*	.877
SVM	PLT	.765	.886	.936	.962	.733	.866	.913*	.816	.897	.900*	.807	.862
ANN	–	.764	.884	.913	.901	.791*	.881	.932*	.859	.923	.667	.882	.854
SVM	ISO	.758	.882	.899	.954	.693*	.878	.907	.827	.897	.900*	.778	.852
ANN	PLT	.766	.872	.898	.894	.775	.871	.929*	.846	.919	.665	.871	.846
ANN	ISO	.767	.882	.821	.891	.785*	.895	.926*	.841	.915	.672	.862	.842
BST-DT	–	.874	.842	.875	.913	.523	.807	.860	.785	.933	.835	.858	.828
KNN	PLT	.819	.785	.920	.937	.626	.777	.803	.844	.827	.774	.855	.815
KNN	–	.807	.780	.912	.936	.598	.800	.801	.853	.827	.748	.852	.810
KNN	ISO	.814	.784	.879	.935	.633	.791	.794	.832	.824	.777	.833	.809
BST-STMP	PLT	.644	**.949**	.767	.688	.723	.806	.800	.862	.923	.622	.915*	.791
SVM	–	.696	.819	.731	.860	.600	.859	.788	.776	.833	.864	.763	.781
BST-STMP	ISO	.639	.941	.700	.681	.711	.807	.793	.862	.912	.632	.902*	.780
BST-STMP	–	.605	.865	.540	.615	.624	.779	.683	.799	.817	.581	.906*	.710
DT	ISO	.671	.869	.729	.760	.424	.777	.622	.815	.832	.415	.884	.709
DT	–	.652	.872	.723	.763	.449	.769	.609	.829	.831	.389	.899*	.708
DT	PLT	.661	.863	.734	.756	.416	.779	.607	.822	.826	.407	.890*	.706
LR	–	.625	.886	.195	.448	.777*	.852	.675	.849	.838	.647	.905*	.700
LR	ISO	.616	.881	.229	.440	.763*	.834	.659	.827	.833	.636	.889*	.692
LR	PLT	.610	.870	.185	.446	.738	.835	.667	.823	.832	.633	.895	.685
NB	ISO	.574	.904	.674	.557	.709	.724	.205	.687	.758	.633	.770	.654
NB	PLT	.572	.892	.648	.561	.694	.732	.213	.690	.755	.632	.756	.650
NB	–	.552	.843	.534	.556	.011	.714	-.654	.655	.759	.636	.688	.481

Empirical Evidence

- Empirical study 2: 8 algorithms on 11 datasets
  - ✓ Normalized score by various metrics

MODEL	CAL	ACC	FSC	LFT	ROC	APR	BEP	RMS	MXE	MEAN	OPT-SEL
BST-DT	PLT	.843*	.779	**.939**	**.963**	**.938**	.929*	**.880**	**.896**	**.896**	**.917**
RF	PLT	.872*	.805	.934*	.957	.931	**.930**	.851	.858	.892	.898
BAG-DT	–	.846	.781	.938*	.962*	.937*	.918	.845	.872	.887*	.899
BST-DT	ISO	.826*	.860*	.929*	.952	.921	.925*	.854	.815	.885	.917*
RF	–	**.872**	.790	.934*	.957	.931	**.930**	.829	.830	.884	.890
BAG-DT	PLT	.841	.774	.938*	.962*	.937*	.918	.836	.852	.882	.895
RF	ISO	.861*	**.861**	.923	.946	.910	.925	.836	.776	.880	.895
BAG-DT	ISO	.826	.843*	.933*	.954	.921	.915	.832	.791	.877	.894
SVM	PLT	.824	.760	.895	.938	.898	.913	.831	.836	.862	.880
ANN	–	.803	.762	.910	.936	.892	.899	.811	.821	.854	.885
SVM	ISO	.813	.836*	.892	.925	.882	.911	.814	.744	.852	.882
ANN	PLT	.815	.748	.910	.936	.892	.899	.783	.785	.846	.875
ANN	ISO	.803	.836	.908	.924	.876	.891	.777	.718	.842	.884
BST-DT	–	.834*	.816	**.939**	**.963**	**.938**	.929*	.598	.605	.828	.851
KNN	PLT	.757	.707	.889	.918	.872	.872	.742	.764	.815	.837
KNN	–	.756	.728	.889	.918	.872	.872	.729	.718	.810	.830
KNN	ISO	.755	.758	.882	.907	.854	.869	.738	.706	.809	.844
BST-STMP	PLT	.724	.651	.876	.908	.853	.845	.716	.754	.791	.808
SVM	–	.817	.804	.895	.938	.899	.913	.514	.467	.781	.810
BST-STMP	ISO	.709	.744	.873	.899	.835	.840	.695	.646	.780	.810
BST-STMP	–	.741	.684	.876	.908	.853	.845	.394	.382	.710	.726
DT	ISO	.648	.654	.818	.838	.756	.778	.590	.589	.709	.774
DT	–	.647	.639	.824	.843	.762	.777	.562	.607	.708	.763
DT	PLT	.651	.618	.824	.843	.762	.777	.575	.594	.706	.761
LR	–	.636	.545	.823	.852	.743	.734	.620	.645	.700	.710
LR	ISO	.627	.567	.818	.847	.735	.742	.608	.589	.692	.703
LR	PLT	.630	.500	.823	.852	.743	.734	.593	.604	.685	.695
NB	ISO	.579	.468	.779	.820	.727	.733	.572	.555	.654	.661
NB	PLT	.576	.448	.780	.824	.738	.735	.537	.559	.650	.654
NB	–	.496	.562	.781	.825	.738	.735	.347	-.633	.481	.489

Empirical Evidence

Fernández-Delgado et al. (2014)

- Empirical study 3: 179 algorithms on 121 datasets

Data set	#pat.	#inp.	#cl.	%Maj.	Data set	#pat.	#inp.	#cl.	%Maj.
abalone	4177	8	3	34.6	energy-y1	768	8	3	46.9
ac-inflam	120	6	2	50.8	energy-y2	768	8	3	49.9
acute-nephritis	120	6	2	58.3	fertility	100	9	2	88.0
adult	48842	14	2	75.9	flags	194	28	8	30.9
annealing	798	38	6	76.2	glass	214	9	6	35.5
arrhythmia	452	262	13	54.2	haberman-survival	306	3	2	73.5
audiology-std	226	59	18	26.3	hayes-roth	132	3	3	38.6
balance-scale	625	4	3	46.1	heart-cleveland	303	13	5	54.1
balloons	16	4	2	56.2	heart-hungarian	294	12	2	63.9
bank	45211	17	2	88.5	heart-switzerland	123	12	2	39.0
blood	748	4	2	76.2	heart-va	200	12	5	28.0
breast-cancer	286	9	2	70.3	hepatitis	155	19	2	79.3
bc-wisc	699	9	2	65.5	hill-valley	606	100	2	50.7
bc-wisc-diag	569	30	2	62.7	horse-colic	300	25	2	63.7
bc-wisc-prog	198	33	2	76.3	ilpd-indian-liver	583	9	2	71.4
breast-tissue	106	9	6	20.7	image-segmentation	210	19	7	14.3
car	1728	6	4	70.0	ionosphere	351	33	2	64.1
ctg-10classes	2126	21	10	27.2	iris	150	4	3	33.3
ctg-3classes	2126	21	3	77.8	led-display	1000	7	10	11.1
chess-krvk	28056	6	18	16.2	lenses	24	4	3	62.5
chess-krvkp	3196	36	2	52.2	letter	20000	16	26	4.1
congress-voting	435	16	2	61.4	libras	360	90	15	6.7
conn-bench-sonar	208	60	2	53.4	low-res-spect	531	100	9	51.9
conn-bench-vowel	528	11	11	9.1	lung-cancer	32	56	3	40.6
connect-4	67557	42	2	75.4	lymphography	148	18	4	54.7
contrac	1473	9	3	42.7	magic	19020	10	2	64.8
credit-approval	690	15	2	55.5	mammographic	961	5	2	53.7
cylinder-bands	512	35	2	60.9	miniboone	130064	50	2	71.9
dermatology	366	34	6	30.6	molec-biol-promoter	106	57	2	50.0
echocardiogram	131	10	2	67.2	molec-biol-splice	3190	60	3	51.9
ecoli	336	7	8	42.6	monks-1	124	6	2	50.0

Data set	#pat.	#inp.	#cl.	%Maj.	Data set	#pat.	#inp.	#cl.	%Maj.
monks-2	169	6	2	62.1	soybean	307	35	18	13.0
monks-3	3190	6	2	50.8	spambase	4601	57	2	60.6
mushroom	8124	21	2	51.8	spect	80	22	2	67.1
musk-1	476	166	2	56.5	spectf	80	44	2	50.0
musk-2	6598	166	2	84.6	st-australian-credit	690	14	2	67.8
nursery	12960	8	5	33.3	st-german-credit	1000	24	2	70.0
oocMerl2F	1022	25	3	67.0	st-heart	270	13	2	55.6
oocMerl4D	1022	41	2	68.7	st-image	2310	18	7	14.3
oocTris2F	912	25	2	57.8	st-landsat	4435	36	6	24.2
oocTris5B	912	32	3	57.6	st-shuttle	43500	9	7	78.4
optical	3823	62	10	10.2	st-vehicle	846	18	4	25.8
ozone	2536	72	2	97.1	steel-plates	1941	27	7	34.7
page-blocks	5473	10	5	89.8	synthetic-control	600	60	6	16.7
parkinsons	195	22	2	75.4	teaching	151	5	3	34.4
pendigits	7494	16	10	10.4	thyroid	3772	21	3	92.5
pima	768	8	2	65.1	tic-tac-toe	958	9	2	65.3
pb-MATERIAL	106	4	3	74.5	titanic	2201	3	2	67.7
pb-REL-L	103	4	3	51.5	trains	10	28	2	50.0
pb-SPAN	92	4	3	52.2	twonorm	7400	20	2	50.0
pb-T-OR-D	102	4	2	86.3	vc-2classes	310	6	2	67.7
pb-TYPE	105	4	6	41.9	vc-3classes	310	6	3	48.4
planning	182	12	2	71.4	wall-following	5456	24	4	40.4
plant-margin	1600	64	100	1.0	waveform	5000	21	3	33.9
plant-shape	1600	64	100	1.0	waveform-noise	5000	40	3	33.8
plant-texture	1600	64	100	1.0	wine	179	13	3	39.9
post-operative	90	8	3	71.1	wine-quality-red	1599	11	6	42.6
primary-tumor	330	17	15	25.4	wine-quality-white	4898	11	7	44.9
ringnorm	7400	20	2	50.5	yeast	1484	8	10	31.2
seeds	210	7	3	33.3	zoo	101	16	7	40.6
semeion	1593	256	10	10.2					

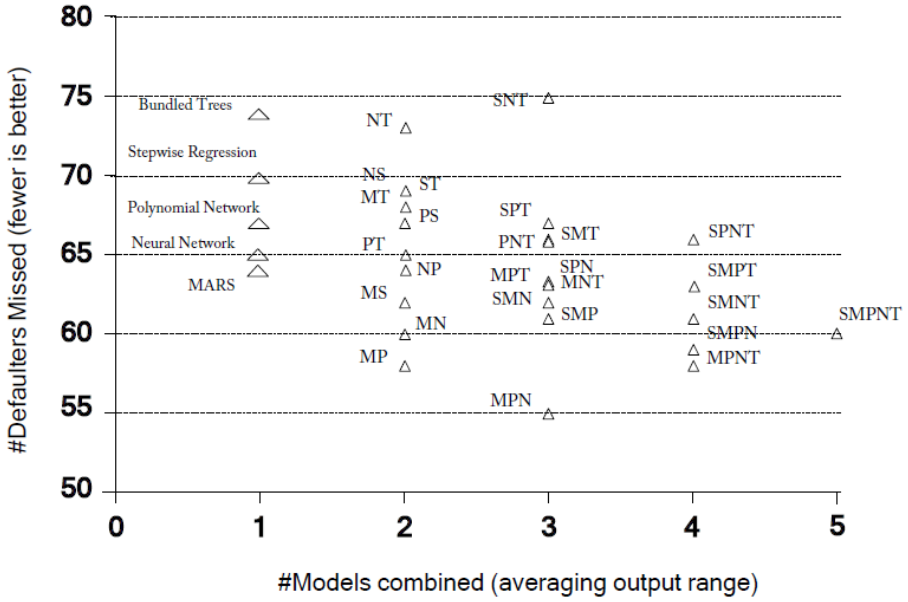
Empirical Evidence

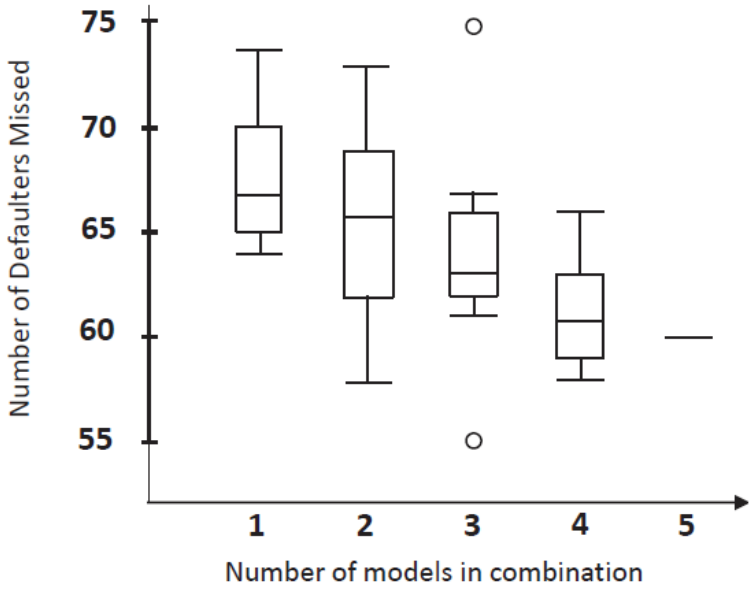
- Empirical study 3: 179 algorithms on 121 datasets

Rank	Acc.	κ	Classifier	Rank	Acc.	κ	Classifier
32.9	82.0	63.5	parRF_t (RF)	67.3	77.7	55.6	pda_t (DA)
33.1	**82.3**	**63.6**	rf_t (RF)	67.6	78.7	55.2	elm_m (NNET)
36.8	81.8	62.2	svm_C (SVM)	67.6	77.8	54.2	SimpleLogistic_w (LMR)
38.0	81.2	60.1	svmPoly_t (SVM)	69.2	78.3	57.4	MAB_J48_w (BST)
39.4	81.9	62.5	rforest_R (RF)	69.8	78.8	56.7	BG_REPTree_w (BAG)
39.6	82.0	62.0	elm_kernel_m (NNET)	69.8	78.1	55.4	SMO_w (SVM)
40.3	81.4	61.1	svmRadialCost_t (SVM)	70.6	78.3	58.0	MLP_w (NNET)
42.5	81.0	60.0	svmRadial_t (SVM)	71.0	78.8	58.23	BG_RandomTree_w (BAG)
42.9	80.6	61.0	C5.0_t (BST)	71.0	77.1	55.1	mlm_R (GLM)
44.1	79.4	60.5	avNNet_t (NNET)	71.0	77.8	56.2	BG_J48_w (BAG)
45.5	79.5	61.0	nnet_t (NNET)	72.0	75.7	52.6	rbf_t (NNET)
47.0	78.7	59.4	pcaNNet_t (NNET)	72.1	77.1	54.8	fda_R (DA)
47.1	80.8	53.0	BG_LibSVM_w (BAG)	72.4	77.0	54.7	lda_R (DA)
47.3	80.3	62.0	mlp_t (NNET)	72.4	79.1	55.6	svmlight_C (NNET)
47.6	80.6	60.0	RotationForest_w (RF)	72.6	78.4	57.9	AdaBoostM1_J48_w (BST)
50.1	80.9	61.6	RRF_t (RF)	72.7	78.4	56.2	BG_IBk_w (BAG)
51.6	80.7	61.4	RRFglobal_t (RF)	72.9	77.1	54.6	ldaBag_R (BAG)
52.5	80.6	58.0	MAB_LibSVM_w (BST)	73.2	78.3	56.2	BG_LWL_w (BAG)
52.6	79.9	56.9	LibSVM_w (SVM)	73.7	77.9	56.0	MAB_REPTree_w (BST)
57.6	79.1	59.3	adaboost_R (BST)	74.0	77.4	52.6	RandomSubSpace_w (DT)
58.5	79.7	57.2	pnn_m (NNET)	74.4	76.9	54.2	lda2_t (DA)
58.9	78.5	54.7	cforest_t (RF)	74.6	74.1	51.8	svmBag_R (BAG)
59.9	79.7	42.6	dkp_C (NNET)	74.6	77.5	55.2	LibLINEAR_w (SVM)
60.4	80.1	55.8	gausssprRadial_R (OM)	75.9	77.2	55.6	rbfDDA_t (NNET)
60.5	80.0	57.4	RandomForest_w (RF)	76.5	76.9	53.8	sda_t (DA)
62.1	78.7	56.0	svmLinear_t (SVM)	76.6	78.1	56.5	END_w (OEN)
62.5	78.4	57.5	fda_t (DA)	76.6	77.3	54.8	LogitBoost_w (BST)
62.6	78.6	56.0	knn_t (NN)	76.6	78.2	57.3	MAB_RandomTree_w (BST)
62.8	78.5	58.1	mlp_C (NNET)	77.1	78.4	54.0	BG_RandomForest_w (BAG)
63.0	79.9	59.4	RandomCommittee_w (OEN)	78.5	76.5	53.7	Logistic_w (LMR)
63.4	78.7	58.4	Decorate_w (OEN)	78.7	76.6	50.5	ctreeBag_R (BAG)
63.6	76.9	56.0	mlpWeightDecay_t (NNET)	79.0	76.8	53.5	BG_Logistic_w (BAG)
63.8	78.7	56.7	rda_R (DA)	79.1	77.4	53.0	lvq_t (NNET)
64.0	79.0	58.6	MAB_MLP_w (BST)	79.1	74.4	50.7	pls_t (PLSR)
64.1	79.9	56.9	MAB_RandomForest_w (BST)	79.8	76.9	54.7	hdda_R (DA)
65.0	79.0	56.8	knn_R (NN)	80.6	75.9	53.3	MCC_w (OEN)
65.2	77.9	56.2	multinom_t (LMR)	80.9	76.9	54.5	mda_R (DA)
65.5	77.4	56.6	gcvEarth_t (MARS)	81.4	76.7	55.2	C5.0Rules_t (RL)
65.5	77.8	55.7	glmnet_R (GLM)	81.6	78.3	55.8	lssvmRadial_t (SVM)
65.6	78.6	58.4	MAB_PART_w (BST)	81.7	75.6	50.9	JRip_t (RL)
66.0	78.5	56.5	CVR_w (OM)	82.0	76.1	53.3	MAB_Logistic_w (BST)
66.4	79.2	58.9	treebag_t (BAG)	84.2	75.8	53.9	C5.0Tree_t (DT)
66.6	78.2	56.8	BG_PART_w (BAG)	84.6	75.7	50.8	BG_DecisionTable_w (BAG)
66.7	75.5	55.2	mda_t (DA)	84.9	76.5	53.4	NBTree_w (DT)

Real-world Examples

- Credit card scoring
  - ✓ Mean error reduces with increasing degree of combination





- Netflix competition
  - ✓ The final edge was obtained by weighting contributions from the models of up to 30 competitors

Real-world Examples

- The 10 main takeaways from MLConf SF (2016)
  - ✓ It's (still) not all about Deep Learning
  - ✓ Choose the right problem to solve, with the right metric
  - ✓ Fine tuning your models is 5% of a project
  - ✓ Ensembles almost always work better
  - ✓ The trend towards personalization
  - ✓ Manual curation of content is still used in practice
  - ✓ Avoid the curse of complexity
  - ✓ Learn the best practices from established players
  - ✓ Everybody is using open source
  - ✓ Make sure you have support from the executives

https://tryolabs.com/blog/2016/11/18/10-main-takeaways-from-mlconf/

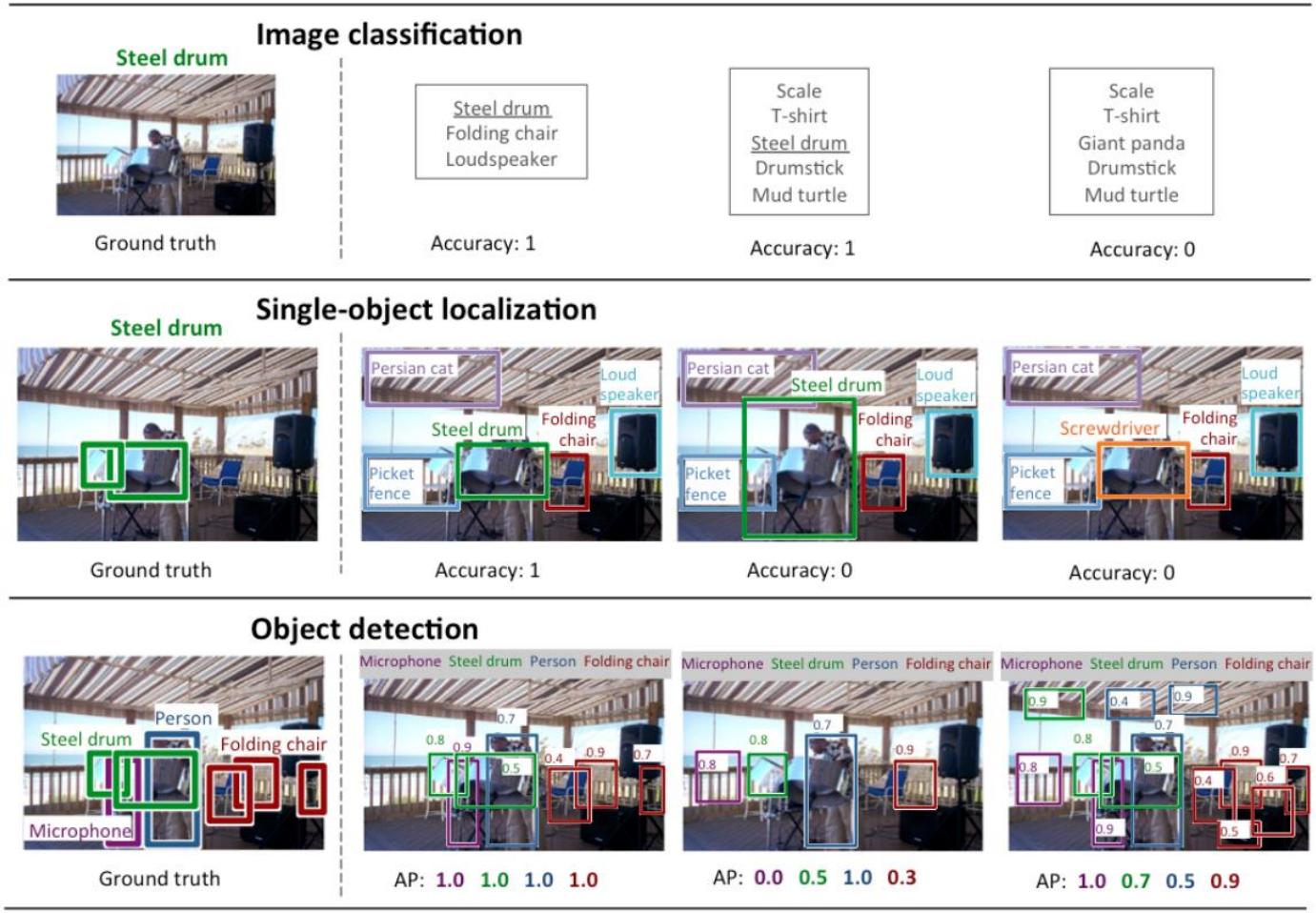
Real-world Examples

- Large Scale Visual Recognition Challenge
  - ✓ With given these images…

Real-world Examples

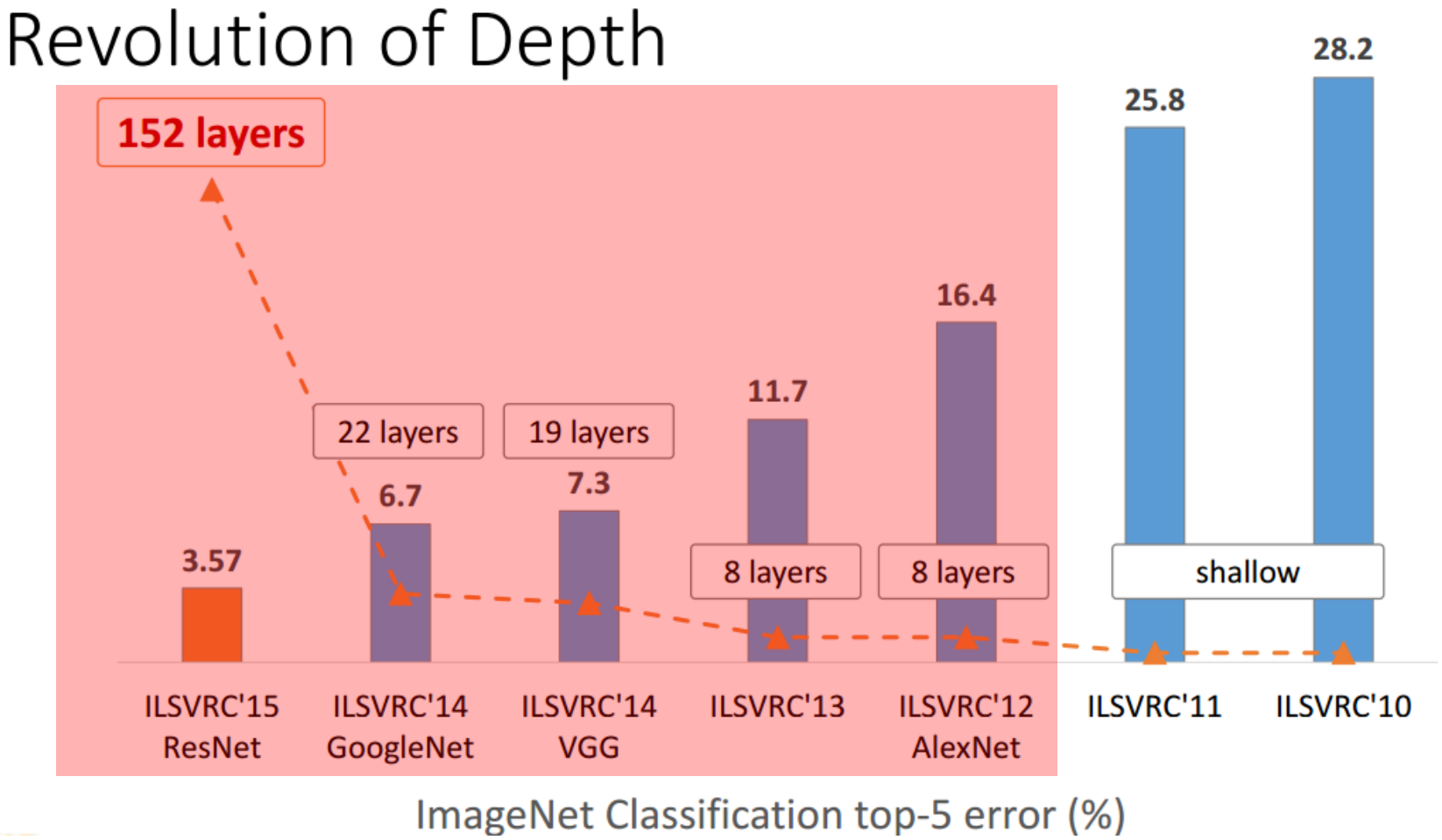
Russakovsky et al. (2015)

- Large Scale Visual Recognition Challenge
  - ✓ Tasks



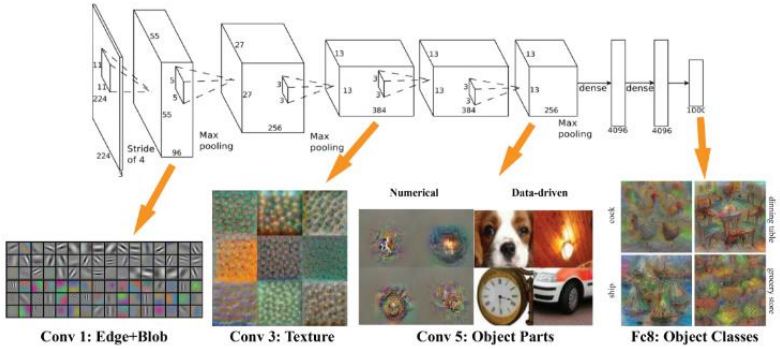
Real-world Examples

- Large Scale Visual Recognition Challenge (~ ILSVRC2015)

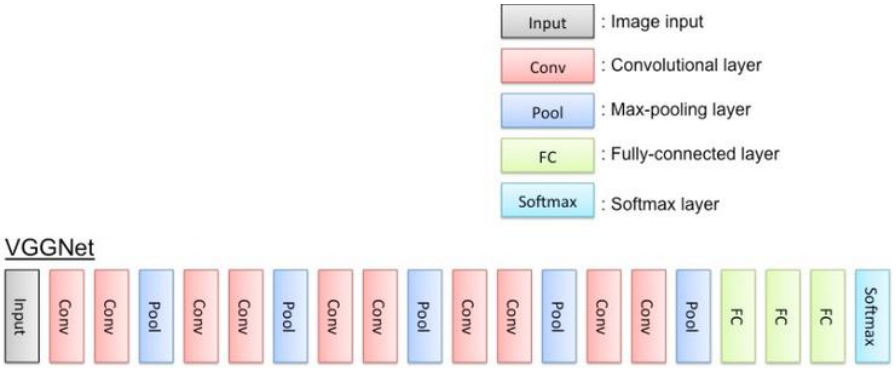


Real-world Examples

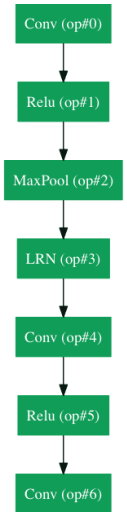
- Large Scale Visual Recognition Challenge (~ ILSVRC2015)



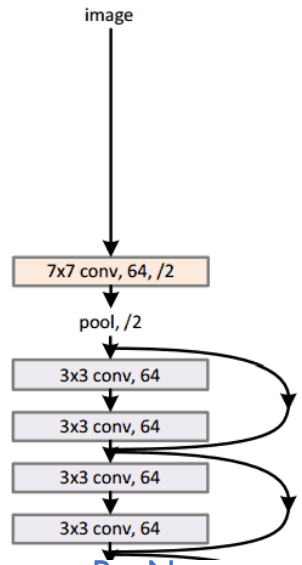
Alexnet



VGGNet



GoogLeNet



ResNet

Real-world Examples

- Large Scale Visual Recognition Challenge (ILSVRC2016 ~)

✓2016

Object detection (DET)[top]

Task 1a: Object detection with provided training data

Ordered by number of categories won

Team name	Entry description	Number of object categories won	mean AP
CUImage	Ensemble of 6 models using provided data	109	0.662751
Hikvision	Ensemble A of 3 RPN and 6 FRCN models, mAP is 67 on val2	30	0.652704
Hikvision	Ensemble B of 3 RPN and 5 FRCN models, mean AP is 66.9, median AP is 69.3 on val2	18	0.652003

Object localization (LOC)[top]

Task 2a: Classification+localization with provided training data

Ordered by localization error

Team name	Entry description	Localization error	Classification error
Trimps-Soushen	Ensemble 3	0.077087	0.02991
Trimps-Soushen	Ensemble 4	0.077429	0.02991
Trimps-Soushen	Ensemble 2	0.077668	0.02991
Trimps-Soushen	Ensemble 1	0.079068	0.03144

✓2017

Object detection (DET)[top]

Task 1a: Object detection with provided training data

Ordered by number of categories won

Team name	Entry description	Number of object categories won	mean AP
BDAT	submission4	85	0.731392
BDAT	submission3	65	0.732227
BDAT	submission2	30	0.723712
DeepView(ETRI)	Ensemble_A	10	0.593084
NUS-Qihoo_DPNs (DET)	Ensemble of DPN models	9	0.656932
KAISTNIA_ETRI	Ensemble Model5	1	0.61022
KAISTNIA_ETRI	Ensemble Model4	0	0.609402
KAISTNIA_ETRI	Ensemble Model2	0	0.608299
KAISTNIA_ETRI	Ensemble Model1	0	0.608278
KAISTNIA_ETRI	Ensemble Model3	0	0.60631

Object localization (LOC)[top]

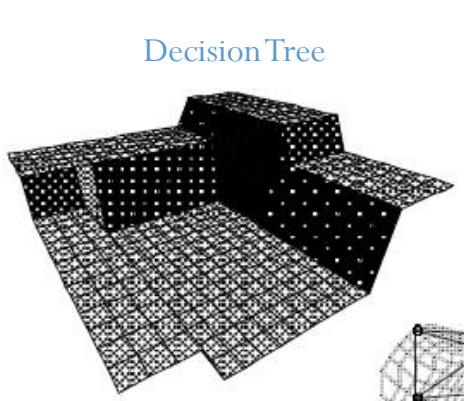
Task 2a: Classification+localization with provided training data

Ordered by localization error

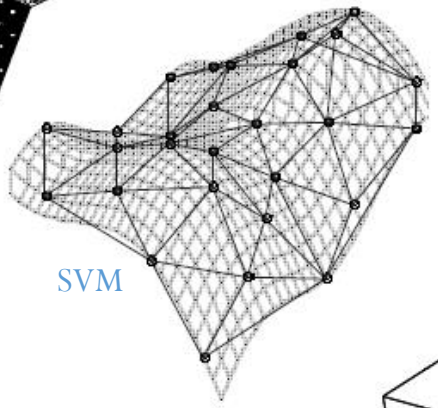
Team name	Entry description	Localization error	Classification error
NUS-Qihoo_DPNs (CLS-LOC)	[E3] LOC:: Dual Path Networks + Basic Ensemble	0.062263	0.03413
Trimps-Soushen	Result-3	0.064991	0.02481
Trimps-Soushen	Result-2	0.06525	0.02481
Trimps-Soushen	Result-4	0.065261	0.02481
Trimps-Soushen	Result-5	0.065302	0.02481
Trimps-Soushen	Result-1	0.067698	0.02481

Theoretical Backgrounds: Model Space

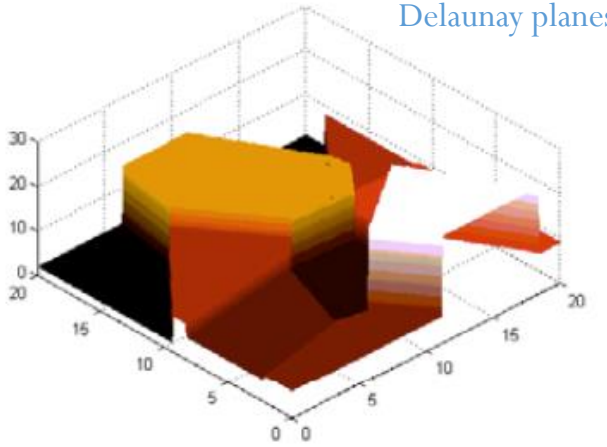
- Different model produce different class boundaries or fitted functions



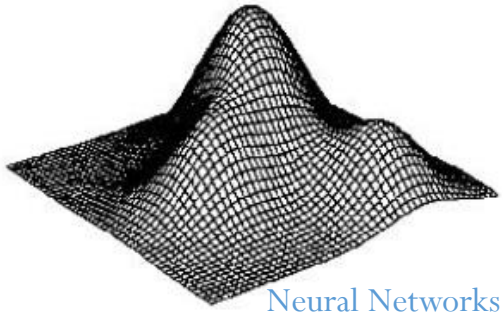
Decision Tree



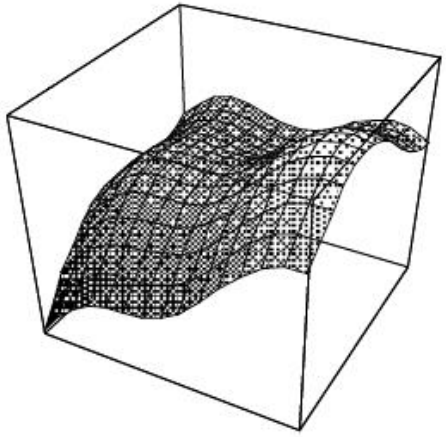
SVM



Delaunay planes



Neural Networks



k-NN

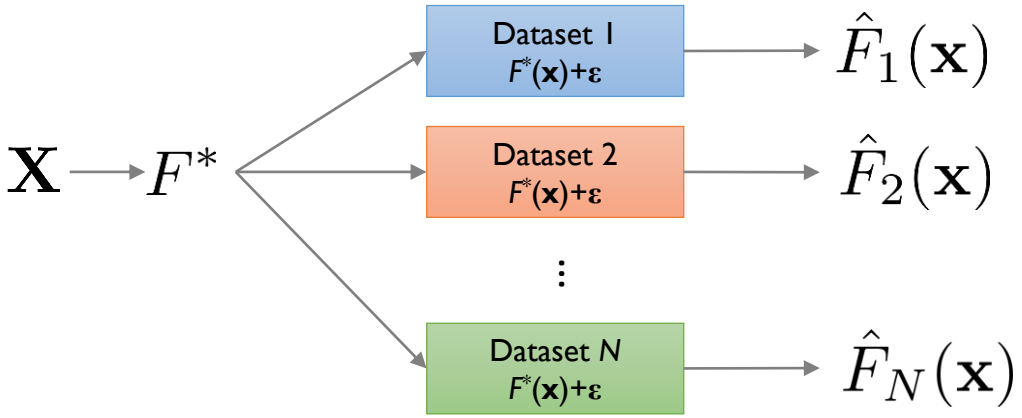
Theoretical Backgrounds: Bias-Variance Decomposition

- Suppose the data comes from the "additive error" model

$$y = F^*(\mathbf{x}) + \epsilon, \quad \epsilon \sim N(0, \sigma^2)$$

  - ✓ $F^*(\mathbf{x})$ is the target function that we are trying to learn, but do not really know
  - ✓ The errors are independent and identically distributed

- Consider the estimation process



  - ✓ The average fit over all possible datasets:

$$\bar{F}(\mathbf{x}) = E[\hat{F}_D(\mathbf{x})]$$

Theoretical Backgrounds: Bias-Variance Decomposition

- The MSE for a particular data point

$$Err(\mathbf{x}_0) = E\Big[y - \hat{F}(\mathbf{x})|\mathbf{x} = \mathbf{x}_0\Big]^2 \qquad (y = F^*(\mathbf{x}) + \epsilon)$$

$$= E\Big[\hat{F}^*(\mathbf{x}_0) + \epsilon - \hat{F}(\mathbf{x}_0)\Big]^2$$

$$= E\Big[\hat{F}^*(\mathbf{x}_0) - \hat{F}(\mathbf{x}_0)\Big]^2 + \sigma^2$$

$$= E\Big[\hat{F}^*(\mathbf{x}_0) - \bar{F}(\mathbf{x}_0) + \bar{F}(\mathbf{x}_0) - \hat{F}(\mathbf{x}_0)\Big]^2 + \sigma^2$$

Theoretical Backgrounds: Bias-Variance Decomposition

- The MSE for a particular data point

$$= E\left[\hat{F}^*(\mathbf{x}_0) - \bar{F}(\mathbf{x}_0) + \bar{F}(\mathbf{x}_0) - \hat{F}(\mathbf{x}_0)\right]^2 + \sigma^2$$

  ✓ By the properties of the expectation operator

$$= E\left[\hat{F}^*(\mathbf{x}_0) - \bar{F}(\mathbf{x}_0)\right]^2 + E\left[\bar{F}(\mathbf{x}_0) - \hat{F}(\mathbf{x}_0)\right]^2 + \sigma^2$$

$$= \left[\hat{F}^*(\mathbf{x}_0) - \bar{F}(\mathbf{x}_0)\right]^2 + E\left[\bar{F}(\mathbf{x}_0) - \hat{F}(\mathbf{x}_0)\right]^2 + \sigma^2$$

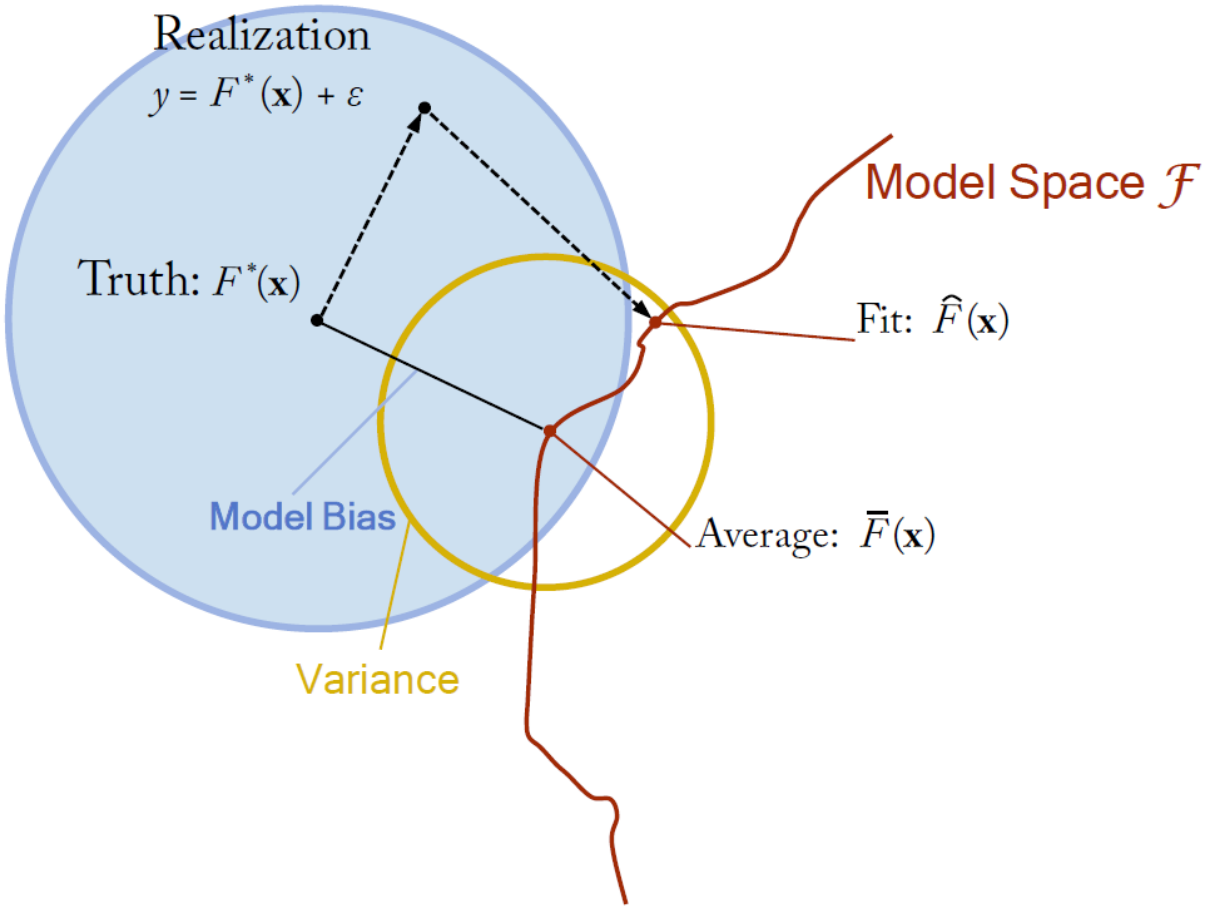
$$= Bias^2\left(\hat{F}(\mathbf{x}_0)\right) + Var\left(\hat{F}(\mathbf{x}_0)\right) + \sigma^2$$

Theoretical Backgrounds: Bias-Variance Decomposition

- Properties of Bias and Variance
  - ✓ Bias^2: the amount by which the average estimator differs from the truth
    - Low bias: on average, we will accurately estimate the function from the dataset
    - High bias implies a **poor** match
  - ✓ Variance: spread of the individual estimations around their mean
    - Low variance: estimated function does not change much with different datasets
    - High variance implies a **weak** match
  - ✓ Irreducible error: the error that was present in the original data
  - ✓ Bias and variance are not independent of each other

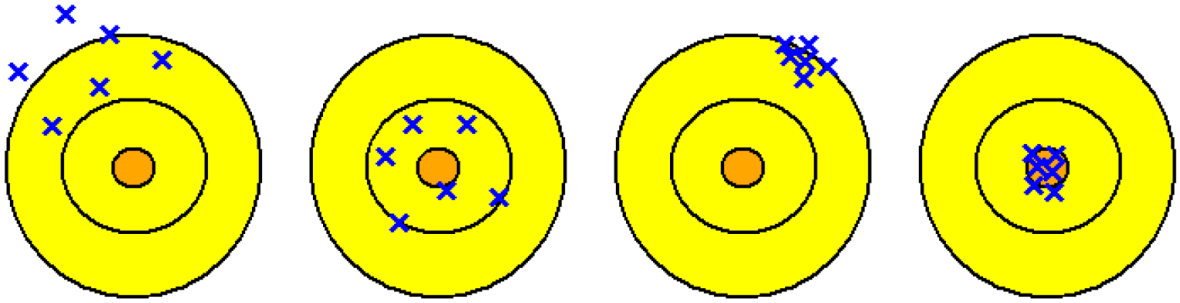
Theoretical Backgrounds: Bias-Variance Decomposition

- Graphical representation of Bias-Variance decomposition



Theoretical Backgrounds: Bias-Variance Decomposition

- Graphical representation of Bias-Variance decomposition



Bias	High	Low	High	Low
Variance	High	High	Low	Low

✓ Lower model complexity: high bias & low variance

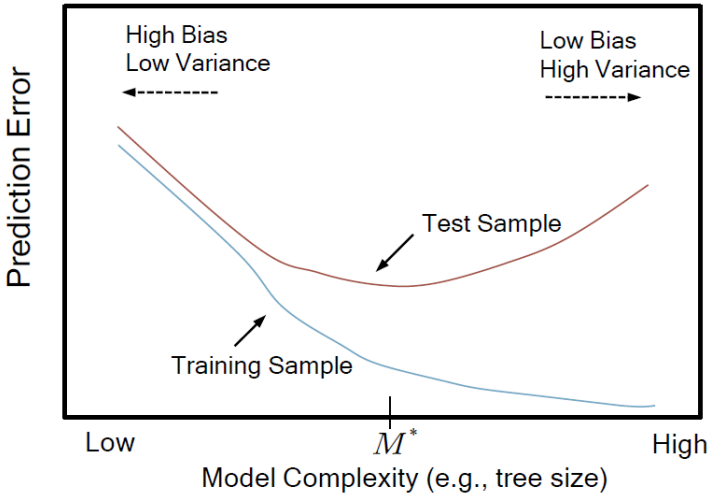
- Logistic regression, LDA, k-NN with large k, etc.

✓ Higher model complexity: low bias & high variance

- DT, ANN, SVM, k-NN with small k, etc.

Bias-Variance Dilemma

The more complex (flexible) we make the model, the lower the bias but the higher the variance it is subjected to.

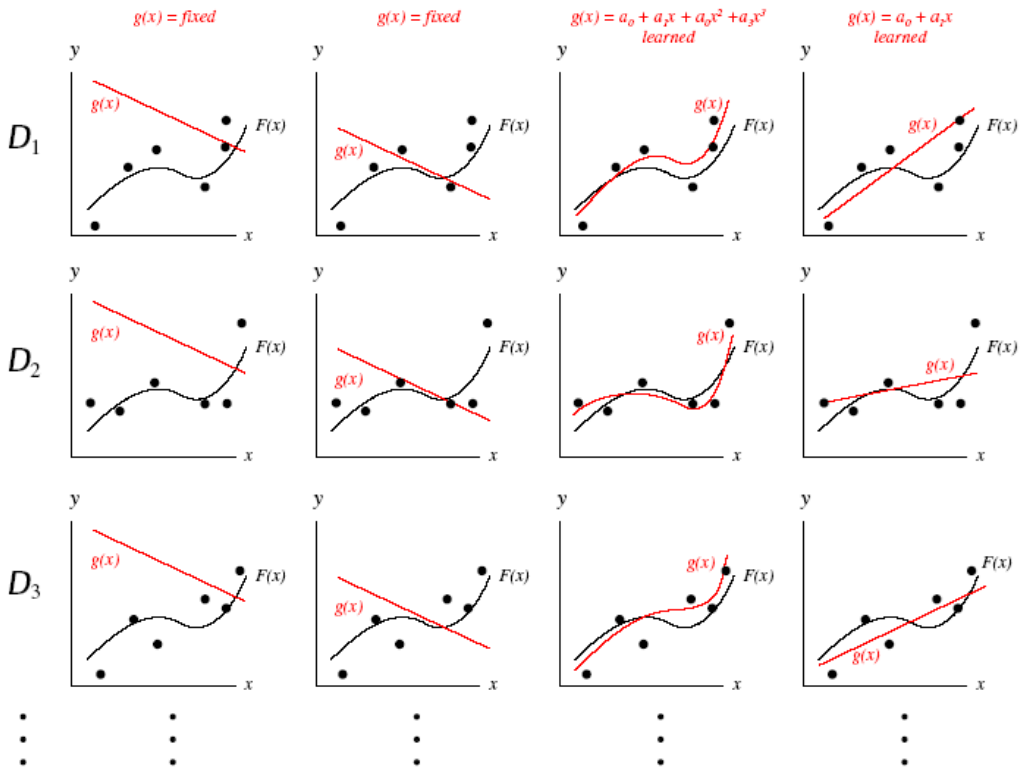


Theoretical Backgrounds: Bias-Variance Decomposition

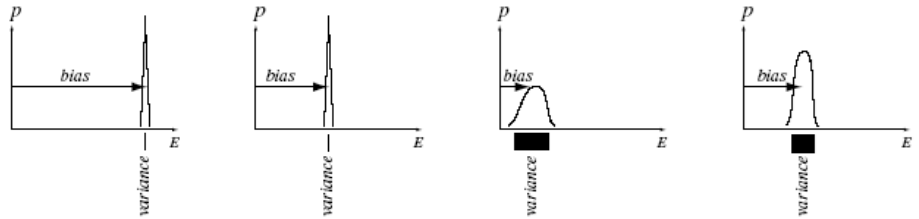
- Bias-Variance example

Each column is a different model.

Each row is a different dataset of 6 points.



Histograms of mean-squared error of the fit.



Col 1:
Poor fixed linear model;
High bias, zero variance

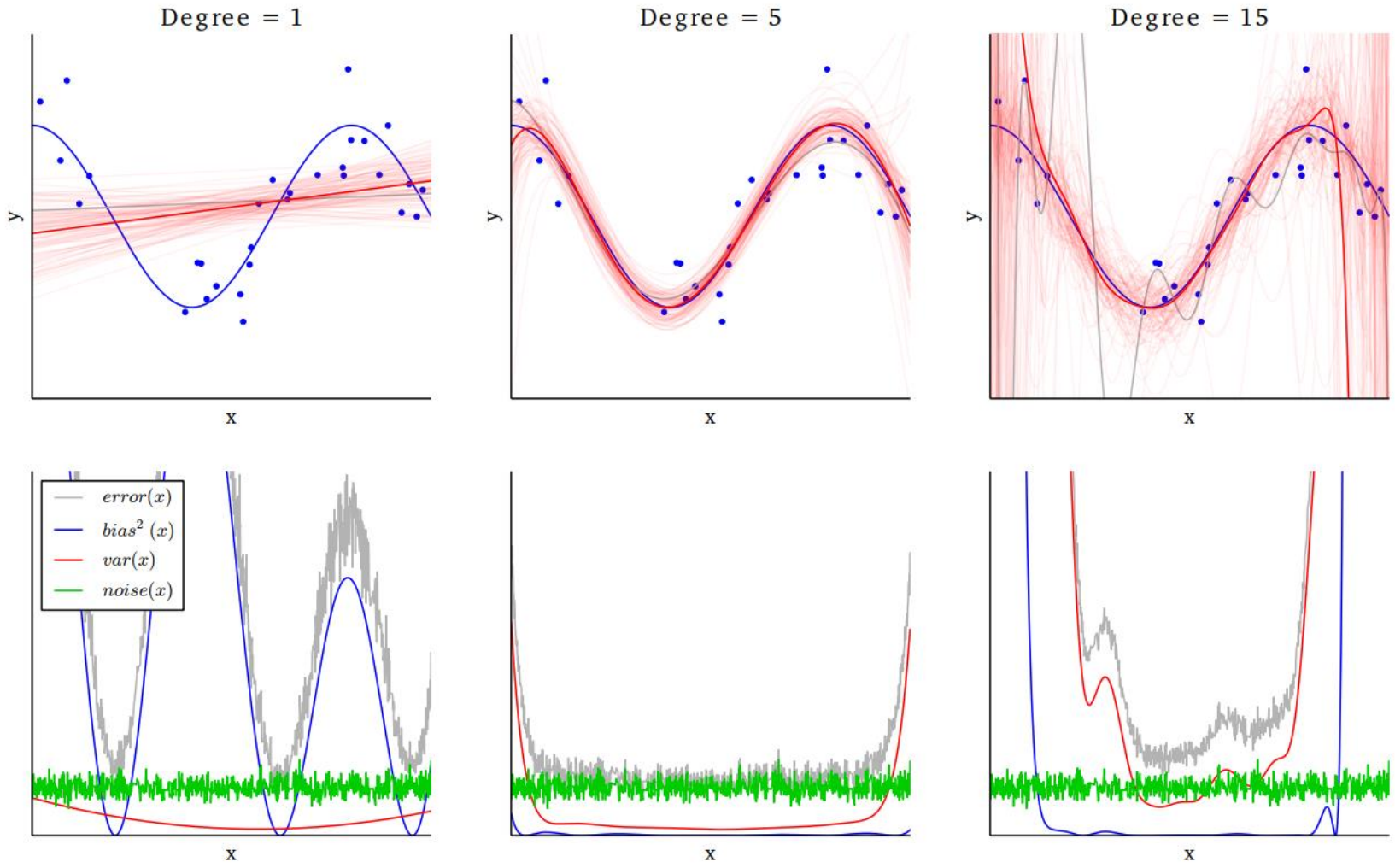
Col 2:
Slightly better fixed linear model;
Lower (but high) bias, zero variance.

Col 3:
Learned cubic model;
Low bias, moderate variance.

Col 4:
Learned linear model;
Intermediate bias and variance.

Theoretical Backgrounds: Bias-Variance Decomposition

- Bias-Variance example



Purpose of Ensemble

- Goal: Reduce the error through constructing multiple learners to
  - ✓ Reduce the variance: Bagging, Random Forests
  - ✓ Reduce the bias: AdaBoost
  - ✓ Both: Mixture of experts

- Two key questions on the ensemble construction
  - ✓ Q1: How to generate individual components of the ensemble systems (base classifiers) to achieve sufficient degree of diversity?
  - ✓ Q2: How to combine the outputs of individual classifiers?

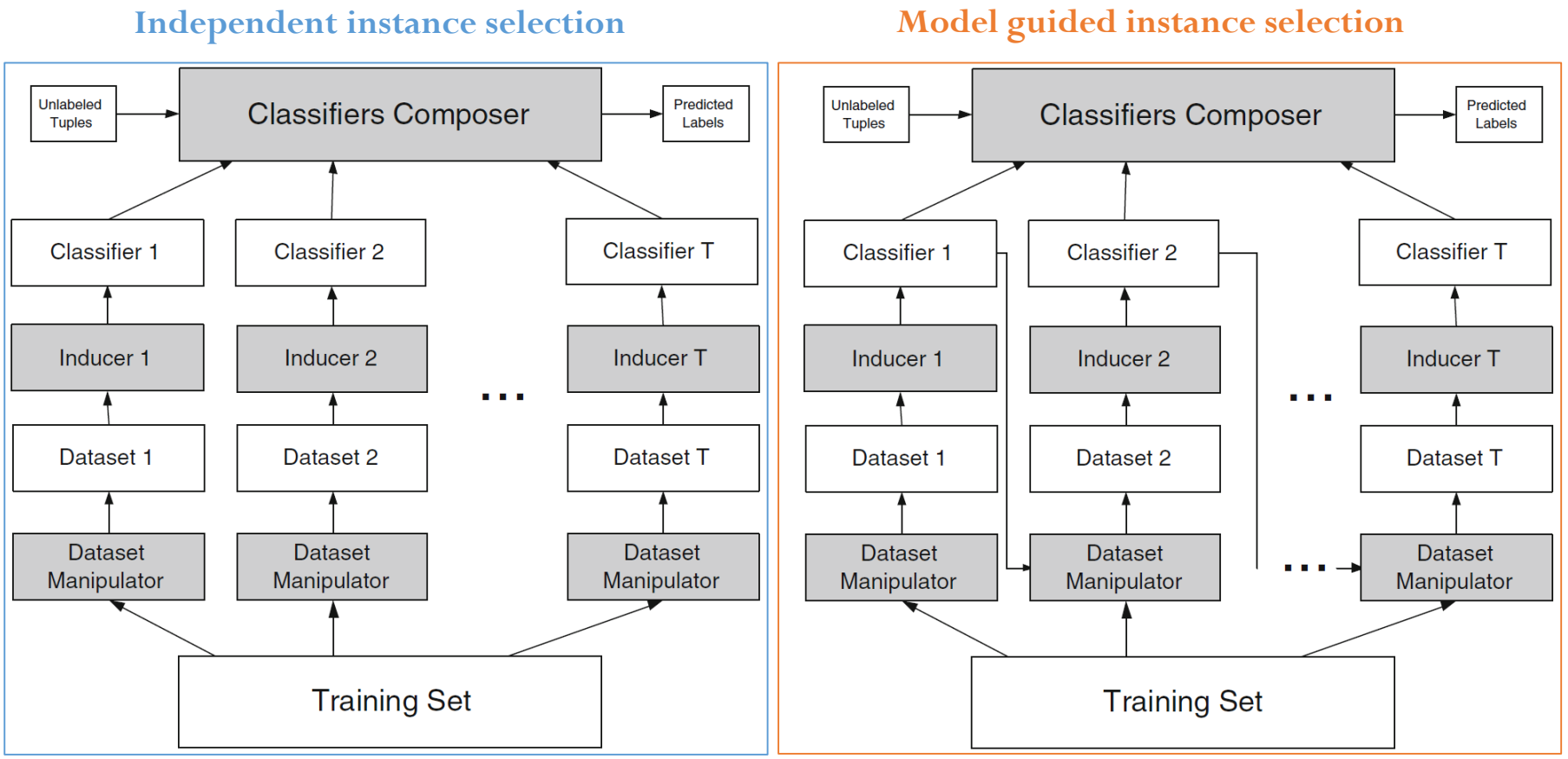
Ensemble Diversity

- Ensemble will have no gain from combining a set of identical models
  - ✓ Need base learners whose fitted functions are adequately different from those of others
  - ✓ Wish models to exhibit a certain element of diversity in their group behavior, though still retaining good performance individually.

Diversity	Implicit	Explicit
Description	Provide different random subset of the training data to each learner	Use some measurement ensuring it is substantially different from the other members
Ensemble Algorithms	Instance: Bagging Variables: Random Subspaces, Rotation Forests Both: Random Forests	Boosting, Negative Correlation Learning

Ensemble Diversity

- Independent (implicit) vs. Model guided (explicit) instance selection



Why Ensemble?

- Why Ensemble works?
  - ✓ True functions, estimations, and the expected error

$$y_m(\mathbf{x}) = f(\mathbf{x}) + \epsilon_m(\mathbf{x}). \quad \mathbb{E}_{\mathbf{x}}\left[\{y_m(\mathbf{x}) - f(\mathbf{x})\}^2\right] = \mathbb{E}_{\mathbf{x}}\left[\epsilon_m(\mathbf{x})^2\right]$$

  - ✓ The average error made by M individual models vs. Expected error of the ensemble

$$E_{Avg} = \frac{1}{M}\sum_{m=1}^{M}\mathbb{E}_{\mathbf{x}}\left[\epsilon_m(\mathbf{x})^2\right]$$

$$E_{Ensemble} = \mathbb{E}_{\mathbf{x}}\left[\left\{\frac{1}{M}\sum_{m=1}^{M} y_m(\mathbf{x}) - f(\mathbf{x})\right\}^2\right]$$

$$= \mathbb{E}_{\mathbf{x}}\left[\left\{\frac{1}{M}\sum_{m=1}^{M} \epsilon_m(\mathbf{x})\right\}^2\right]$$

Why Ensemble?

- Why Ensemble works?
  - ✓ Assume that the errors have zero mean and are uncorrelated,

$$\mathbb{E}_{\mathbf{x}}[\epsilon_m(\mathbf{x})] = 0, \qquad \mathbb{E}_{\mathbf{x}}[\epsilon_m(\mathbf{x})\epsilon_l(\mathbf{x})] = 0 \ (m \neq l)$$

  - ✓ The average error made by M individual models vs. Expected error of the ensemble

$$E_{Ensemble} = \frac{1}{M} E_{Avg}$$

  - ✓ In reality (errors are correlated), by the Cauchy's inequality

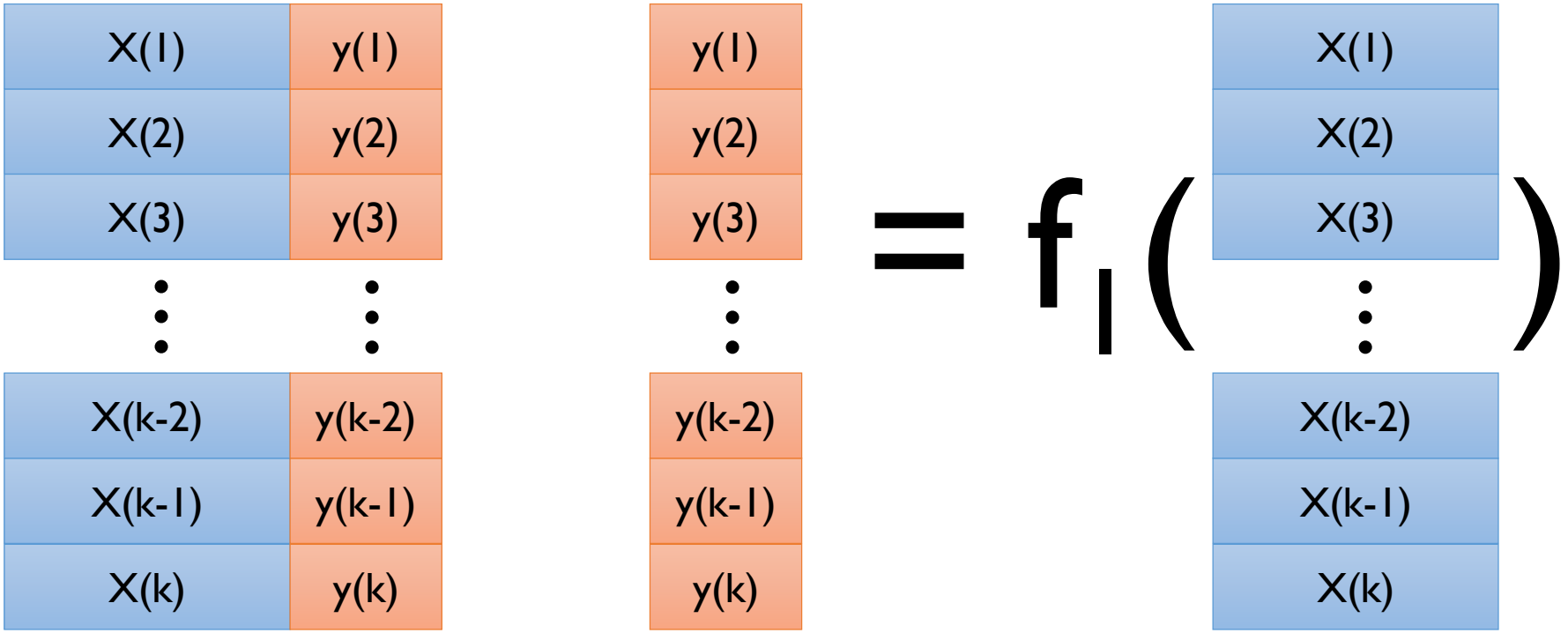
$$\Big[\sum_{m=1}^{M} \epsilon_m(\mathbf{x})\Big]^2 \leq M \sum_{m=1}^{M} \epsilon_m(\mathbf{x})^2 \Rightarrow \Big[\frac{1}{M}\sum_{m=1}^{M} \epsilon_m(\mathbf{x})\Big]^2 \leq \frac{1}{M}\sum_{m=1}^{M} \epsilon_m(\mathbf{x})^2$$

$$E_{Ensemble} \leq E_{Avg}$$

AGENDA

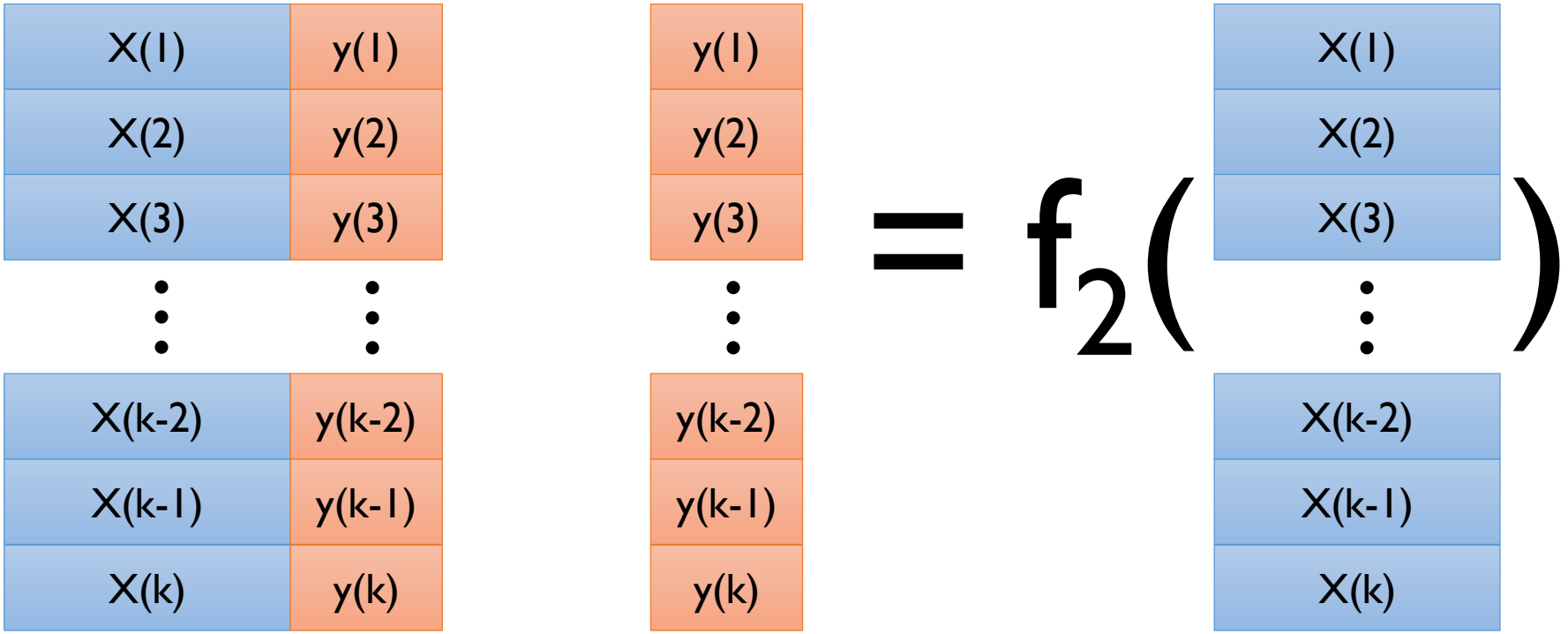
Sampling without Replacement

- K-fold data split
  - ✓ Entire data is split into k blocks; each classifier is trained only on different subset of (k-1) blocks



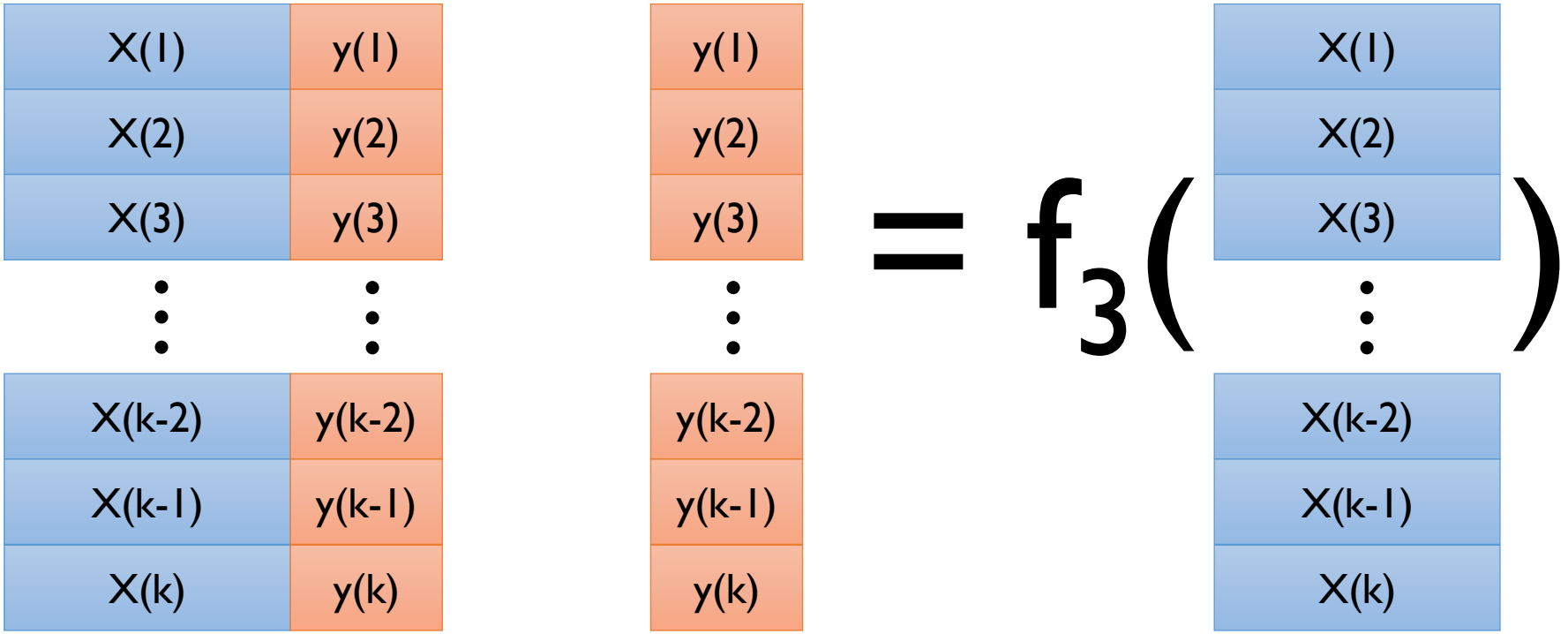
Sampling without Replacement

- K-fold data split
  - ✓ Entire data is split into k blocks; each classifier is trained only on different subset of (k-1) blocks



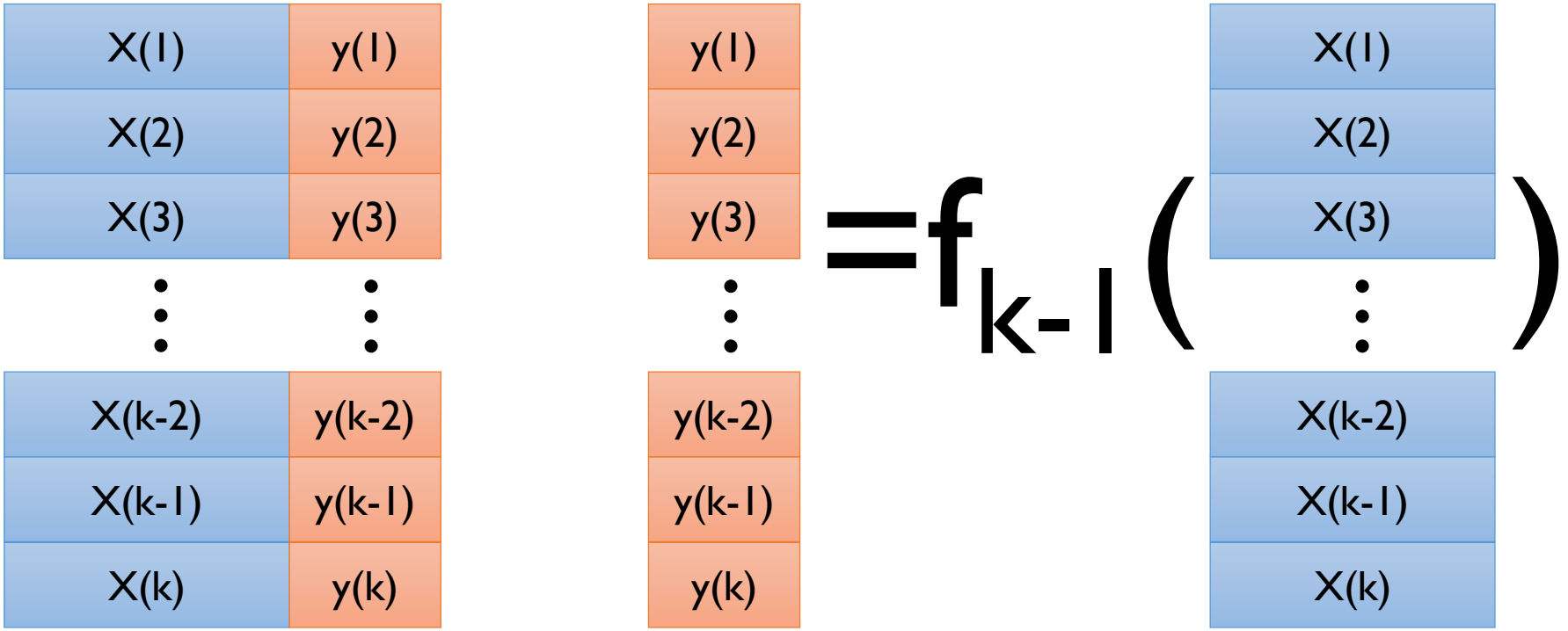
Sampling without Replacement

- K-fold data split
  - ✓ Entire data is split into k blocks; each classifier is trained only on different subset of (k-1) blocks



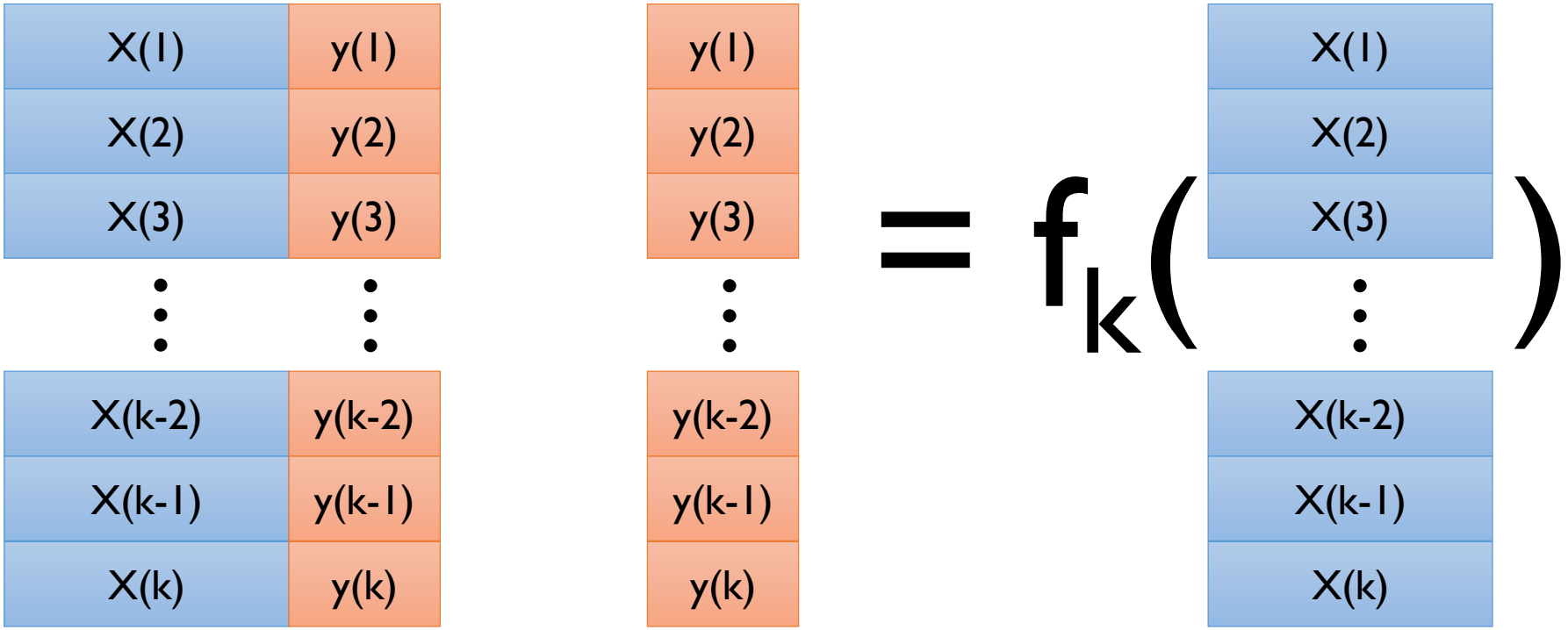
Sampling without Replacement

- K-fold data split
  - ✓ Entire data is split into k blocks; each classifier is trained only on different subset of (k-1) blocks



Sampling without Replacement

- K-fold data split
  - ✓ Entire data is split into k blocks; each classifier is trained only on different subset of (k-1) blocks



Sampling without Replacement

- K-fold data split
  - ✓ Entire data is split into k blocks; each classifier is trained only on different subset of (k-1) blocks
- Final output

$$\hat{y} = \delta\Big(f_1(\mathbf{x}), f_2(\mathbf{x}), \cdots, f_{k-1}(\mathbf{x}), f_k(\mathbf{x})\Big)$$

  - ✓ $\delta(\cdot)$: An aggregation function of individual outputs (ex: simple average)

Bootstrap Aggregating: Bagging

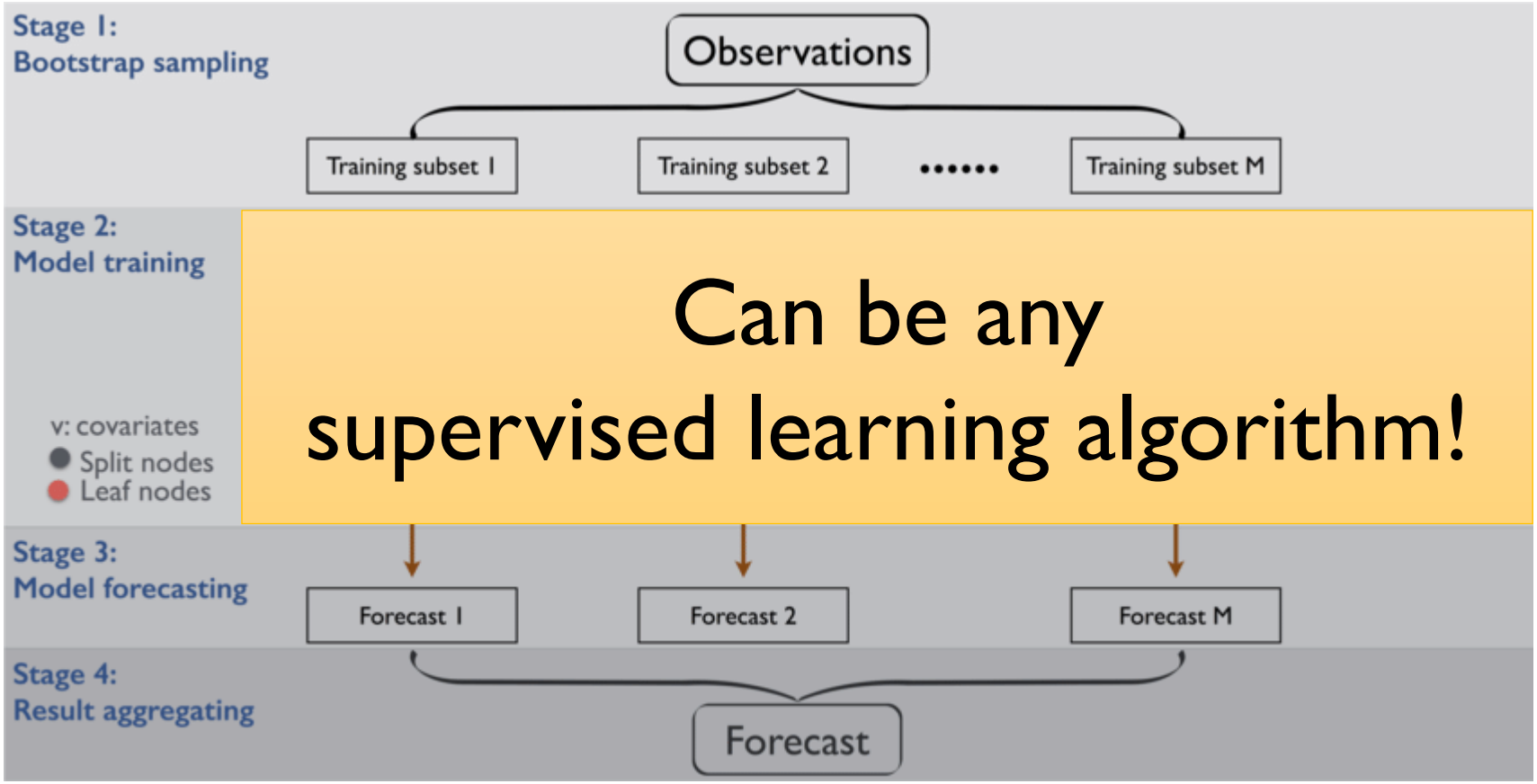
Breiman (1996)

- Main Idea
  - ✓ Each member of the ensemble is constructed from a different training dataset
  - ✓ Each dataset is generated by sampling from the total *N* data examples, choosing *N* items uniformly at random with replacement
  - ✓ Each dataset sample is known as a bootstrap

Original Dataset		Bootstrap 1		Bootstrap 2			Bootstrap B	
x^1	y^1	x^3	y^3	x^7	y^7		x^9	y^9
x^2	y^2	x^6	y^6	x^1	y^1		x^5	y^5
x^3	y^3	x^2	y^2	x^{10}	y^{10}		x^2	y^2
x^4	y^4	x^{10}	y^{10}	x^1	y^1		x^4	y^4
x^5	y^5	x^8	y^8	x^8	y^8		x^7	y^7
x^6	y^6	x^7	y^7	x^6	y^6		x^2	y^2
x^7	y^7	x^7	y^7	x^2	y^2		x^5	y^5
x^8	y^8	x^3	y^3	x^6	y^6		x^{10}	y^{10}
x^9	y^9	x^2	y^2	x^4	y^4		x^8	y^8
x^{10}	y^{10}	x^7	y^7	x^9	y^9		x^2	y^2

Bootstrap Aggregating: Bagging

- Bagging with Decision Tree



Bootstrap Aggregating: Bagging

- Result Aggregating
  - ✓ For classification problem
    - Majority voting

$$\hat{y}_{Ensemble} = arg\max_i \left(\sum_{j=1}^{n} \delta(\hat{y}_j = i), \quad i \in \{0, 1\} \right)$$

Training Accuracy	Ensemble population	P(y=1) for a test instance	Predicted class label
0.80	Model 1	0.90	1
0.75	Model 2	0.92	1
0.88	Model 3	0.87	1
0.91	Model 4	0.34	0
0.77	Model 5	0.41	0
0.65	Model 6	0.84	1
0.95	Model 7	0.14	0
0.82	Model 8	0.32	0
0.78	Model 9	0.98	1
0.83	Model 10	0.57	1

$$\sum_{j=1}^{n} \delta(\hat{y}_j = 0) = 4$$

$$\sum_{j=1}^{n} \delta(\hat{y}_j = 1) = 6$$

$$\hat{y}_{Ensemble} = 1$$

Bootstrap Aggregating: Bagging

- Result Aggregating
  - ✓ For classification problem
    - Weighted voting (weight = training accuracy of individual models)

$$\hat{y}_{Ensemble} = arg\max_{i} \left(\frac{\sum_{j=1}^{n} (TrnAcc_j) \cdot \delta(\hat{y}_j = i)}{\sum_{j=1}^{n} (TrnAcc_j)}, \quad i \in \{0, 1\} \right)$$

Training Accuracy	Ensemble population	P(y=1) for a test instance	Predicted class label
0.80	Model 1	0.90	1
0.75	Model 2	0.92	1
0.88	Model 3	0.87	1
0.91	Model 4	0.34	0
0.77	Model 5	0.41	0
0.65	Model 6	0.84	1
0.95	Model 7	0.14	0
0.82	Model 8	0.32	0
0.78	Model 9	0.98	1
0.83	Model 10	0.57	1

$$\frac{\sum_{j=1}^{n} (TrnAcc_j) \cdot \delta(\hat{y}_j = 0)}{\sum_{j=1}^{n} (TrnAcc_j)} = 0.424$$

$$\frac{\sum_{j=1}^{n} (TrnAcc_j) \cdot \delta(\hat{y}_j = 1)}{\sum_{j=1}^{n} (TrnAcc_j)} = 0.576$$

$$\hat{y}_{Ensemble} = 1$$

Bootstrap Aggregating: Bagging

- Result Aggregating
  - ✓ For classification problem
    - Weighted voting (weight = predicted probability for each class)

$$\hat{y}_{Ensemble} = arg\max_{i} \left(\frac{1}{n} \sum_{j=1}^{n} P(y = i), \quad i \in \{0, 1\} \right)$$

Training Accuracy	Ensemble population	P(y=1) for a test instance	Predicted class label
0.80	Model 1	0.90	1
0.75	Model 2	0.92	1
0.88	Model 3	0.87	1
0.91	Model 4	0.34	0
0.77	Model 5	0.41	0
0.65	Model 6	0.84	1
0.95	Model 7	0.14	0
0.82	Model 8	0.32	0
0.78	Model 9	0.98	1
0.83	Model 10	0.57	1

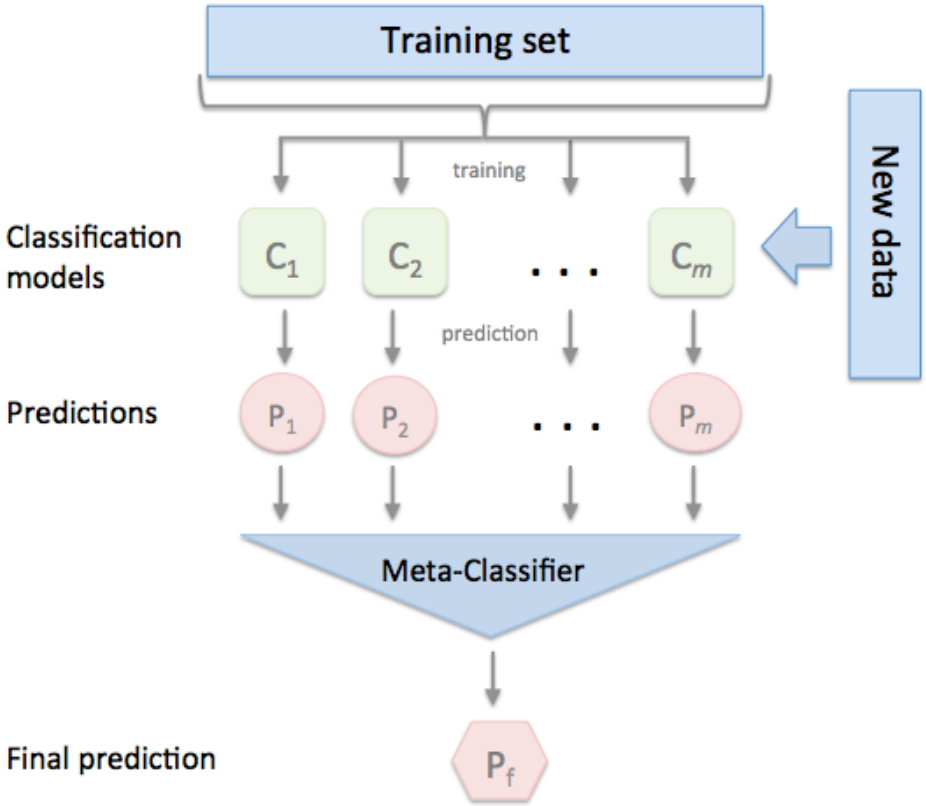
$$\frac{1}{n} \sum_{j=1}^{n} P(y = 0) = 0.375$$

$$\frac{1}{n} \sum_{j=1}^{n} P(y = 1) = 0.625$$

$$\hat{y}_{Ensemble} = 1$$

Bootstrap Aggregating: Bagging

- Result Aggregating: Stacking
  - ✓ Use another prediction model to aggregate the results
    - Input: Predictions made by ensemble members
    - Target: Actual true label



https://rasbt.github.io/mlxtend/user_guide/classifier/StackingClassifier/

Bootstrap Aggregating: Bagging

- Result Aggregating: Stacking
  - ✓ The winner of KDD-cup 2015
    - MOOC dropout prediction

•Jeong-Yoon Lee, Winning Data Science Competitions

Bootstrap Aggregating: Bagging

- Bagging: Algorithm

Algorithm 1 Bagging

Input: Required ensemble size T
Input: Training set $S = \{(x_1, y_1), (x_2, y_2), ..., (x_N, y_N)\}$
for $t = 1$ to T **do**
  Build a dataset S_t, by sampling N items, randomly *with replacement* from S.
  Train a model h_t using S_t, and add it to the ensemble.
end for
For a new testing point (x', y'),
If model outputs are continuous, combine them by averaging.
If model outputs are class labels, combine them by voting.

Bootstrap Aggregating: Bagging

- Bagging: Illustration

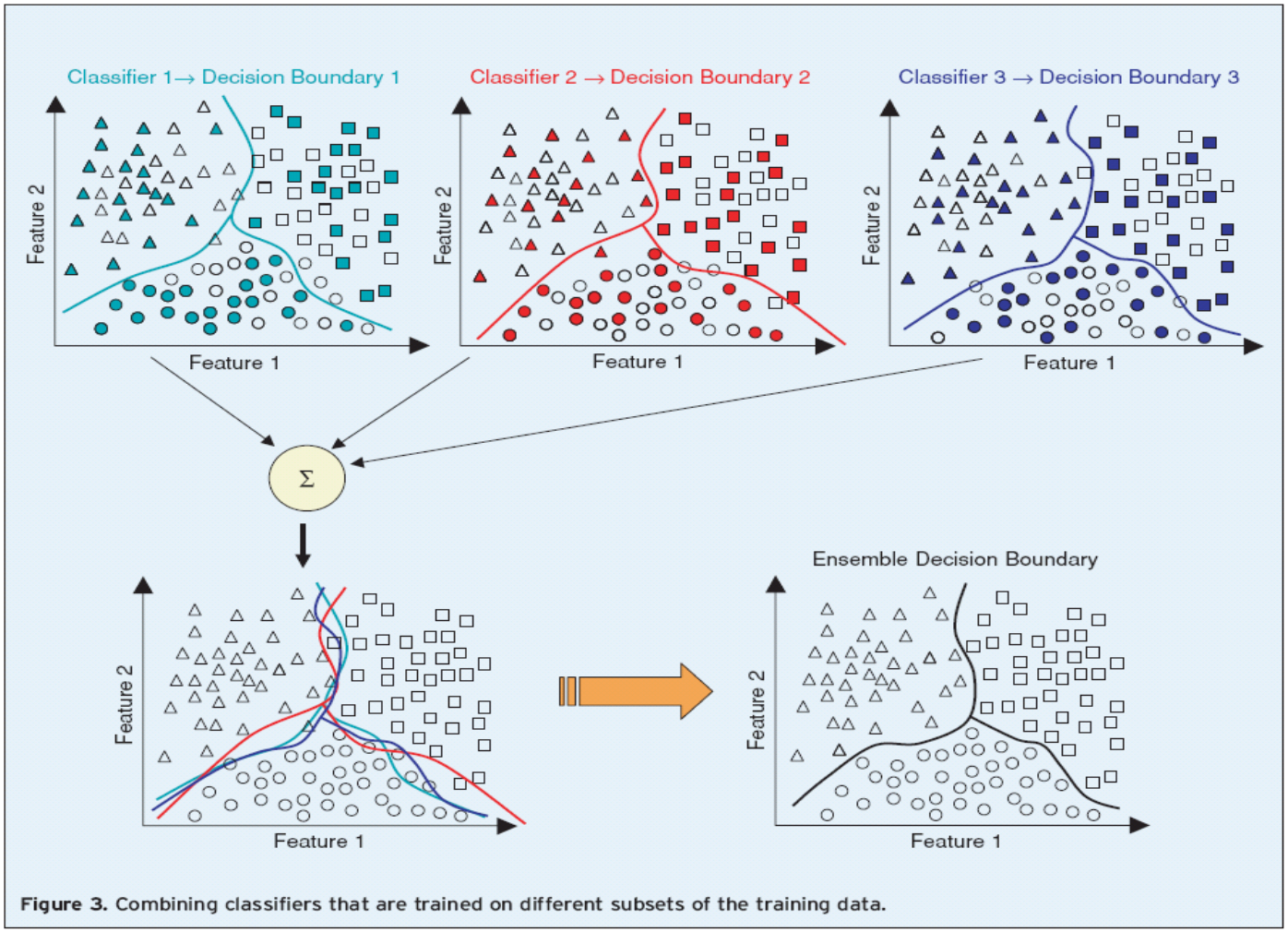
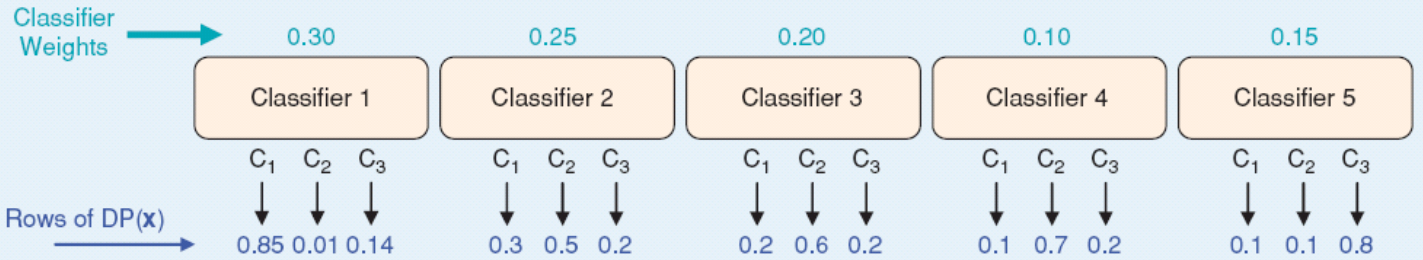


Figure 3. Combining classifiers that are trained on different subsets of the training data.

Bootstrap Aggregating: Bagging

- Aggregation examples

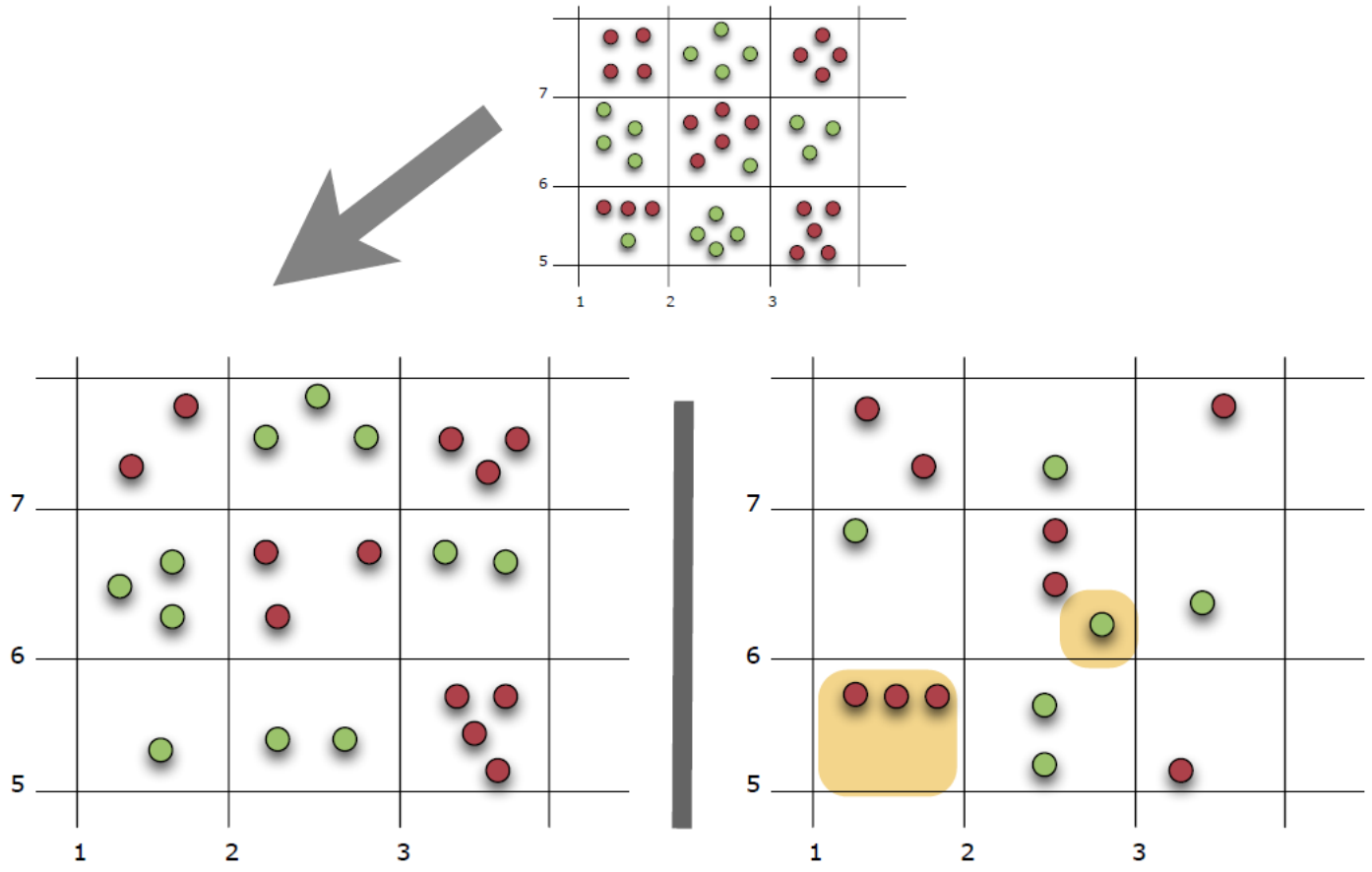


Product Rule:	**C_1: 0.0179,**	C_2: 0.00021,	C_3: 0.0009
Sum Rule:	C_1: 1.55,	**C_2: 1.91,**	C_3: 1.54
Mean Rule:	C_1: 0.310,	**C_2: 0.382,**	C_3: 0.308
Max Rule:	**C_1: 0.85,**	C_2: 0.7,	C_3: 0.8
Median Rule:	C_1: 0.2,	**C_2: 0.5,**	C_3: 0.2
Minimum Rule:	C_1: 0.1,	C_2: 0.01,	**C_3: 0.14**
Weighted Average	**C_1: 0.395,**	C_2: 0.333,	C_3: 0.272
Majority Voting:	C_1: 1,	**C_2: 3,**	C_3: 1 Vote
Weighted Majority Voting:	C_1: 0.3,	**C_2: 0.55,**	C_3: 0.15 Votes
Borda Count:	C_1: 5	**C_2: 6,**	C_3: 4 Votes

Figure 15. Example on various combination rules.

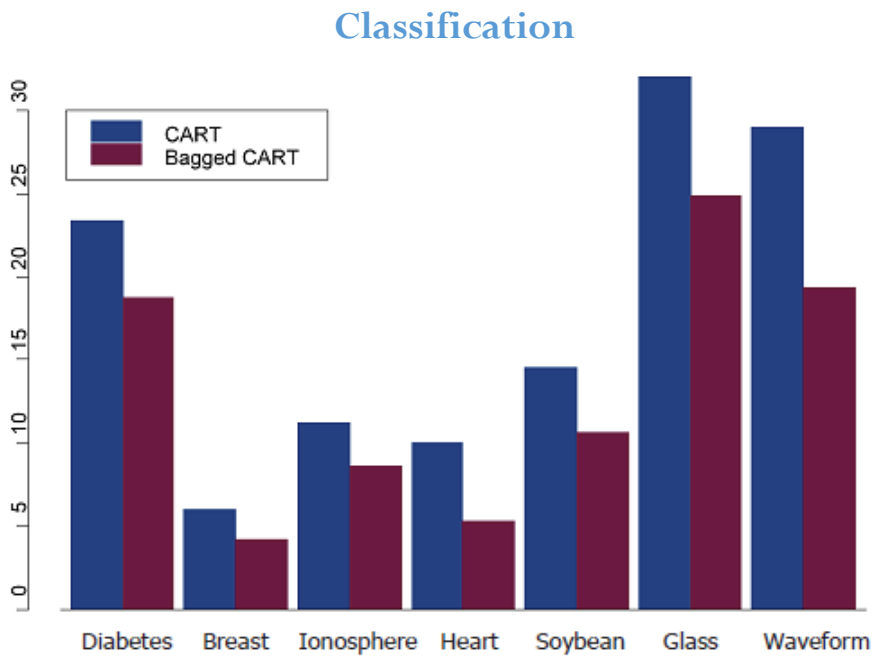
Bootstrap Aggregating: Bagging

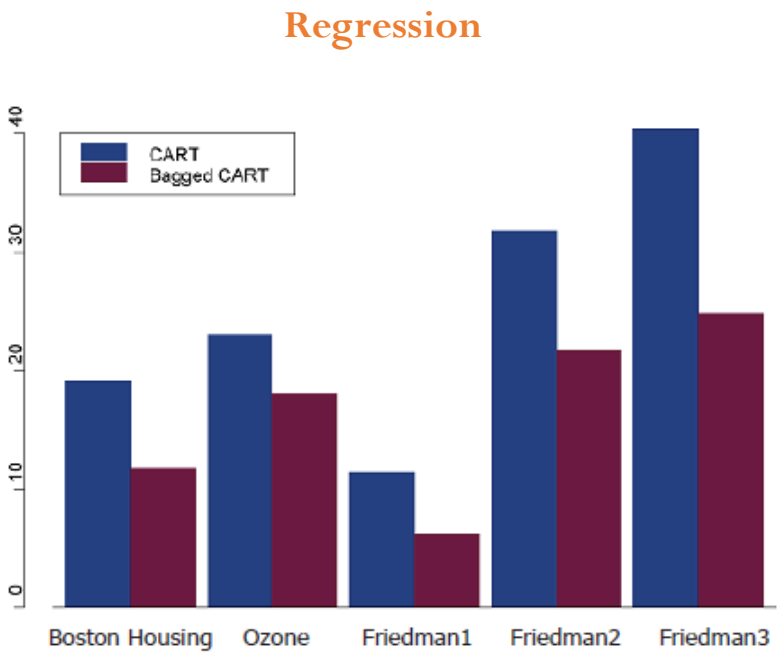
- Out of bag error (OOB Error)
  - ✓ Use the training instances that are not sampled for validation



Bootstrap Aggregating: Bagging

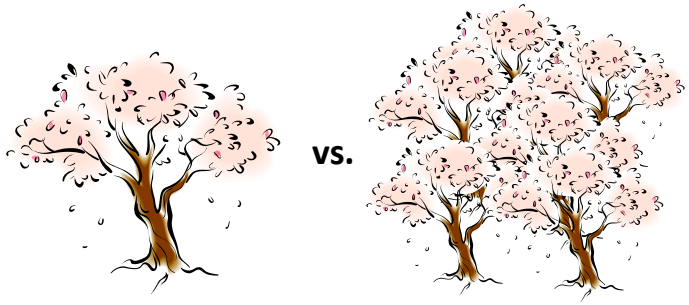
- Bagged Trees vs. Single Tree

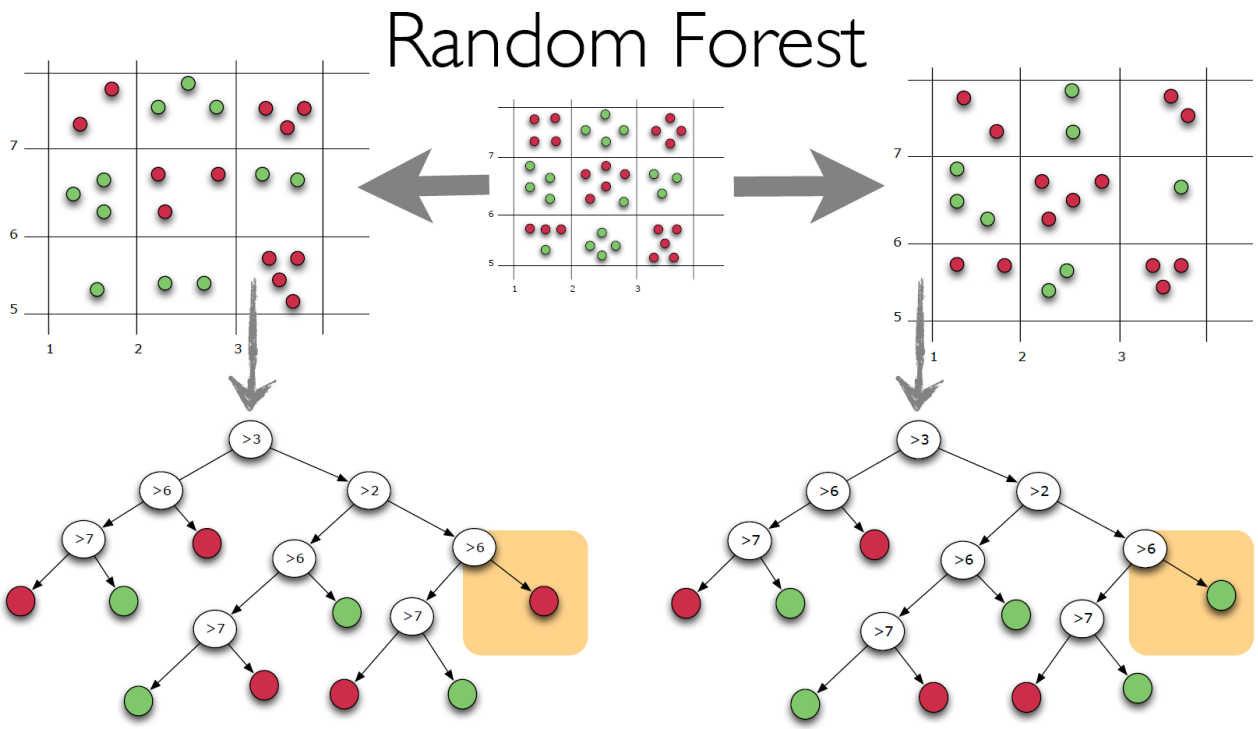




Random Forests

- A specialized bagging for decision tree algorithms
- Two ways to increase the diversity of ensemble
  - ✓ Bagging
  - ✓ Randomly chosen predictor variables





Random Forests

- Random Forests: Algorithm

1. For $b = 1$ to B:

   (a) Draw a bootstrap sample $\mathbf{Z}^*$ of size N from the training data.

   (b) Grow a random-forest tree T_b to the bootstrapped data, by recursively repeating the following steps for each terminal node of the tree, until the minimum node size n_{min} is reached.

      i. Select m variables at random from the p variables.

      ii. Pick the best variable/split-point among the m.

      iii. Split the node into two daughter nodes.

2. Output the ensemble of trees $\{T_b\}_1^B$.

To make a prediction at a new point x:

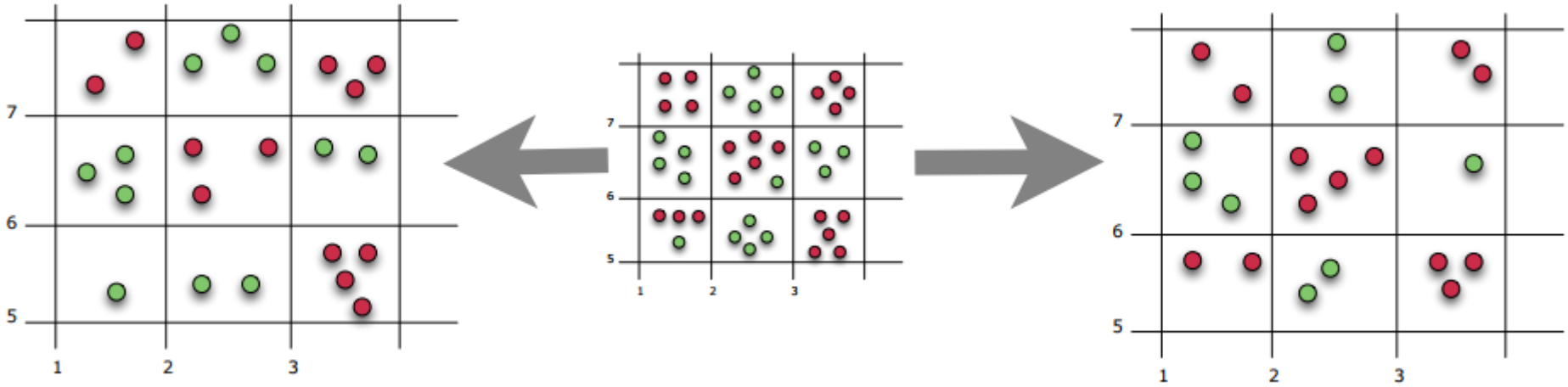
Regression: $\hat{f}_{\text{rf}}^B(x) = \frac{1}{B}\sum_{b=1}^{B} T_b(x)$.

Classification: Let $\hat{C}_b(x)$ be the class prediction of the bth random-forest tree. Then $\hat{C}_{\text{rf}}^B(x) = \textit{majority vote}\ \{\hat{C}_b(x)\}_1^B$.

https://stat.ethz.ch/education/semesters/ss2012/ams/slides/v10.2.pdf

Random Forests

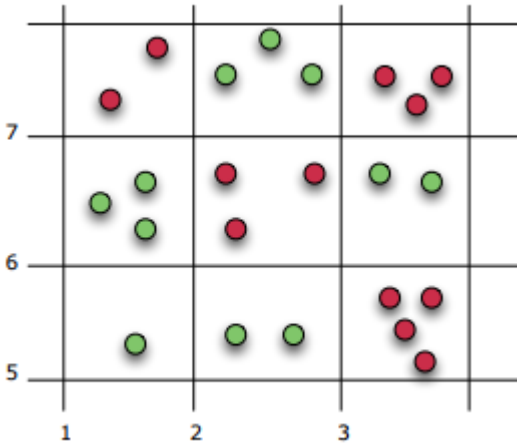
- Bagging
  - ✓ Sampling with replacement



Random Forests

- Bagging
  - ✓ Randomly selected variable

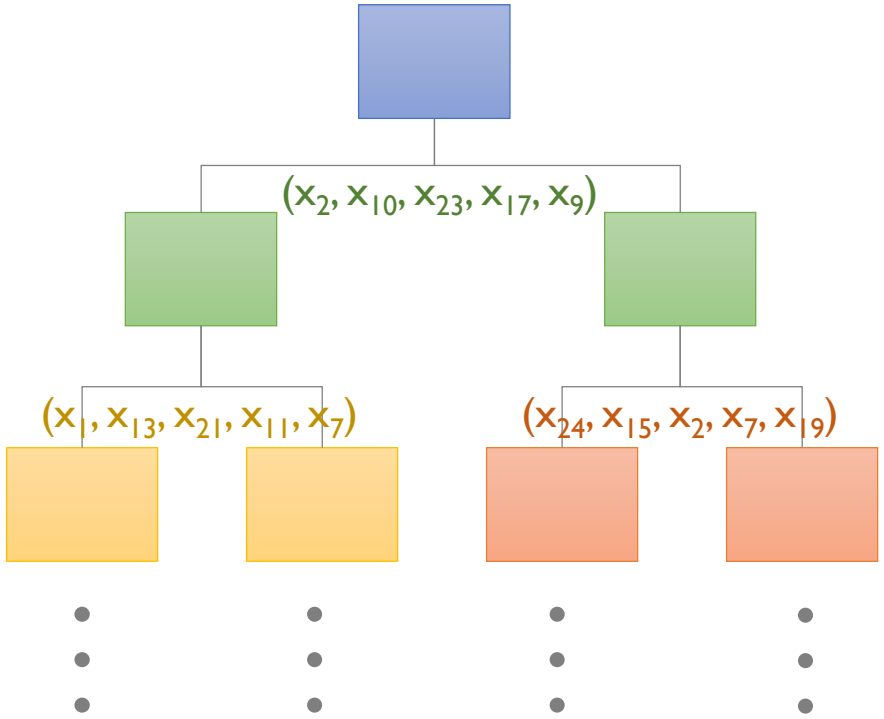
Bootstrap i (X in R^{25})



$(x_2, x_{10}, x_{23}, x_{17}, x_9)$

$(x_1, x_{13}, x_{21}, x_{11}, x_7)$

$(x_{24}, x_{15}, x_2, x_7, x_{19})$



Random Forests

- Generalization Error
  - ✓ Each tree in random forests may over-fit the data because pruning is not conducted.
  - ✓ If the population size is large enough, then the generalization error of random forests bounded by

$$Generalization\ Error \leq \frac{\bar{\rho}(1-s^2)}{s^2}$$

    - $\bar{\rho}$ is the mean value of the correlation coefficients between individual trees
    - s^2 is the margin function (for binary classification, it is simply the average difference proportions between the correct and incorrect trees over all training data.
  - ✓ The more accurate the individual classifiers, the larger the s^2 and the lower the generalization error
  - ✓ The less correlated among the classifiers, the lower the generalization error.

Random Forests

- Variable Importance
  - ✓ Step 1: Compute the OOB error for the original dataset (e_i)
  - ✓ Step 2: Compute the OOB error for the dataset in which the variable x_i is permuted (p_i)
  - ✓ Step 3: Compute the variable importance based on the mean and standard deviation of (p_i-e_i) over all trees in the population

Random Forests

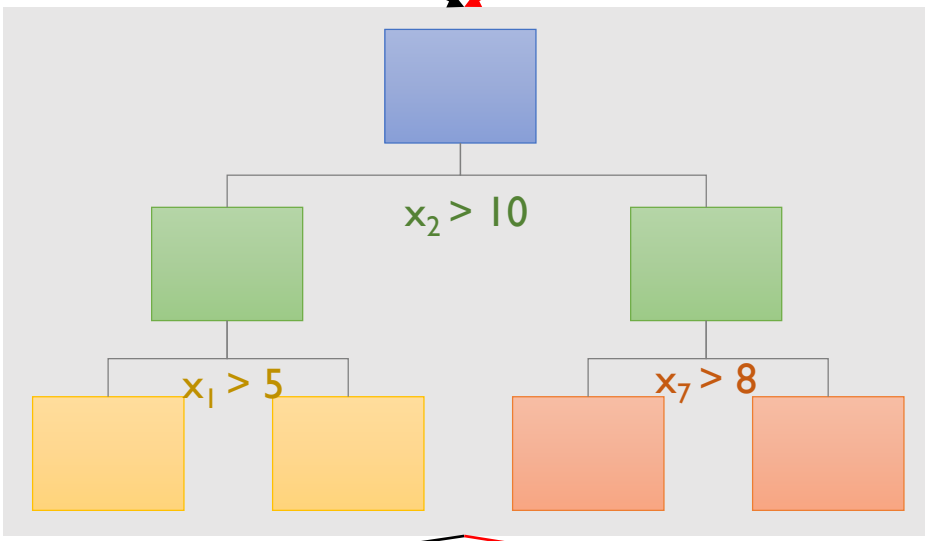
Original OOB Data

ID	X1	...	Xi	...	Xd	Y
1			0.1			
2			0.5			
3			1.1			
4			1.2			
5			0.4			
6			0.2			
7			0.7			
8			0.8			
9			1.4			
10			1.6			

변수 i가 Tree를 split하는데
한번도 사용되지 않았다면

i번째 변수에 대한 random permutation이 수행된 OOB Data

ID	X1	...	Xi	...	Xd	Y
1			1.1			
2			0.2			
3			0.1			
4			1.4			
5			1.2			
6			0.5			
7			1.6			
8			0.8			
9			0.7			
10			0.4			



OOB Error of the Original Data e_i = OOB Error of the Permuted Data p_i

Random Forests

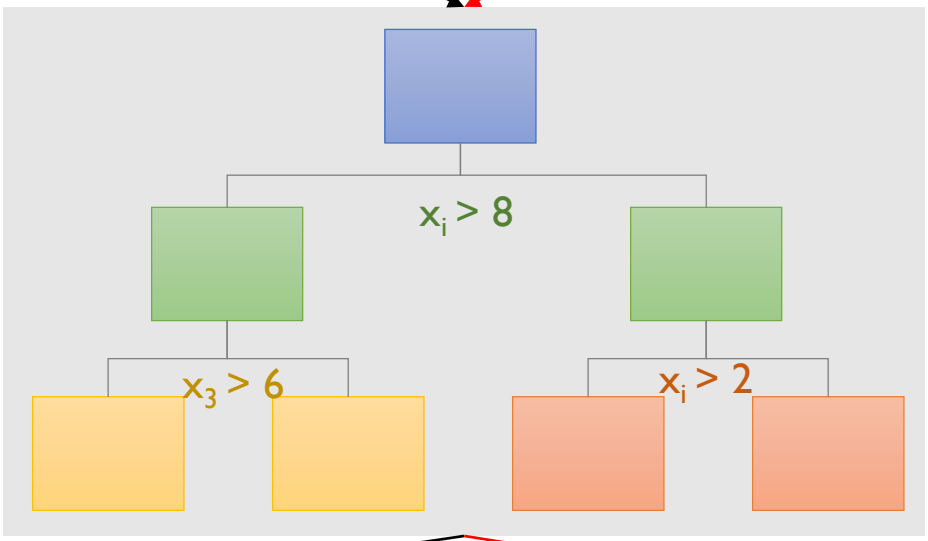
Original OOB Data

ID	X1	...	Xi	...	Xd	Y
1			0.1			
2			0.5			
3			1.1			
4			1.2			
5			0.4			
6			0.2			
7			0.7			
8			0.8			
9			1.4			
10			1.6			

변수 i가 Tree를 split하는데
중요하게 사용되었다면

i번째 변수에 대한 random permutation이 수행된 OOB Data

ID	X1	...	Xi	...	Xd	Y
1			1.1			
2			0.2			
3			0.1			
4			1.4			
5			1.2			
6			0.5			
7			1.6			
8			0.8			
9			0.7			
10			0.4			



OOB Error of the Original Data e_i

OOB Error of the Permuted Data p_i

Random Forests

- 변수의 중요도
  - ✓랜덤 포레스트에서 변수의 중요도가 높다면
    - 1) Random permutation 전-후의 OOB Error 차이가 크게 나타나야 하며,
    - 2) 그 차이의 편차가 적어야 함

    - m번째 tree에서 변수 i에 대한 Random permutation 전후 OOB error의 차이

$$d_i^m = p_i^m - e_i^m$$

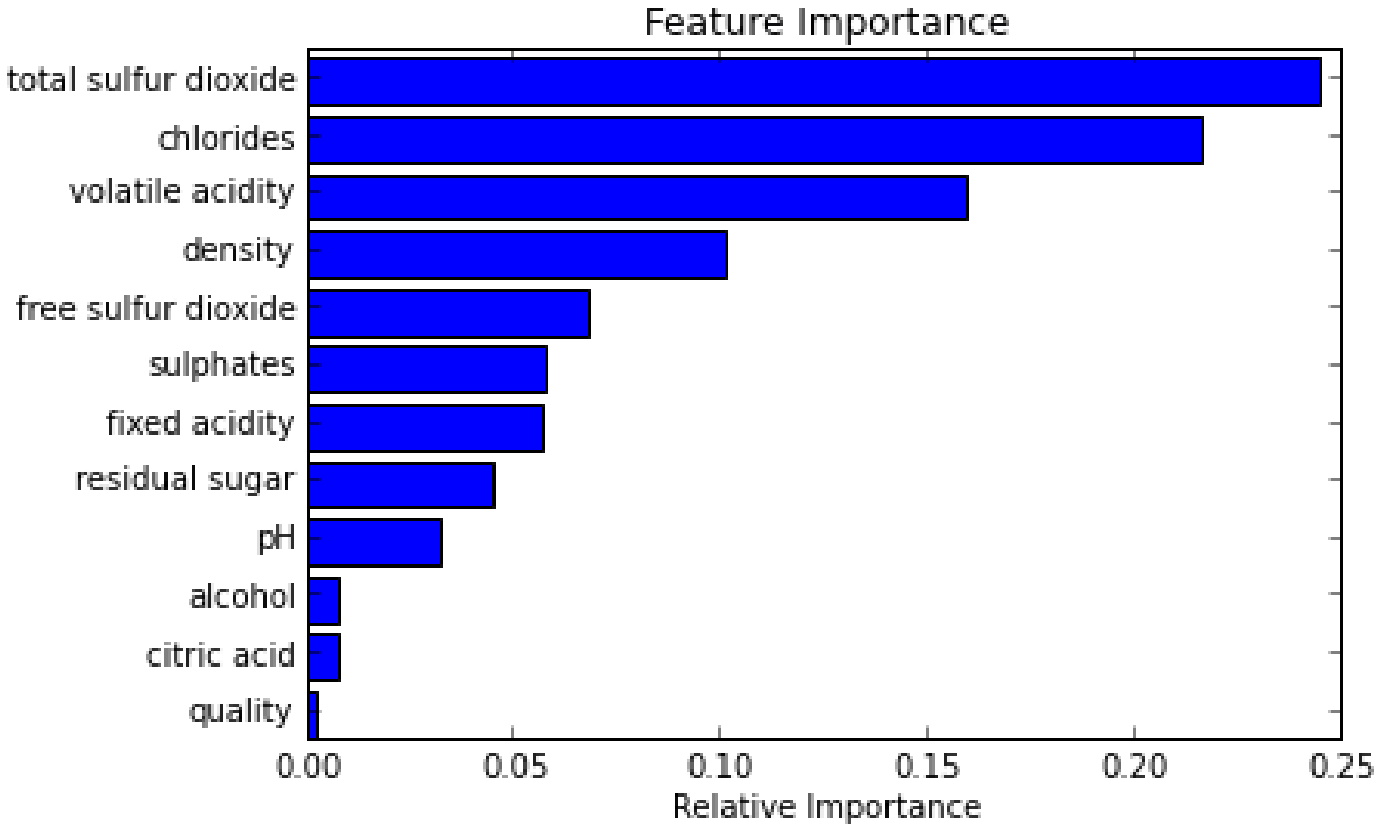
    - 전체 Tree들에 대한 OOB error 차이의 평균 및 분산

$$\bar{d}_i = \frac{1}{m}\sum_{i=1}^{m} d_i^m, \quad s_i^2 = \frac{1}{m-1}\sum_{i=1}^{m}(d_i^m - \bar{d}_i)^2$$

    - i번째 변수의 중요도: $v_i = \dfrac{\bar{d}_i}{s_i}$

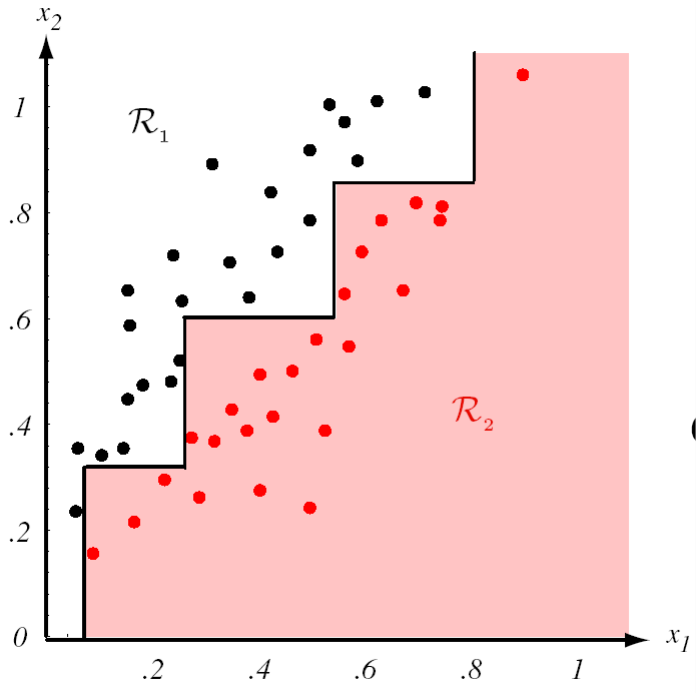
Random Forests

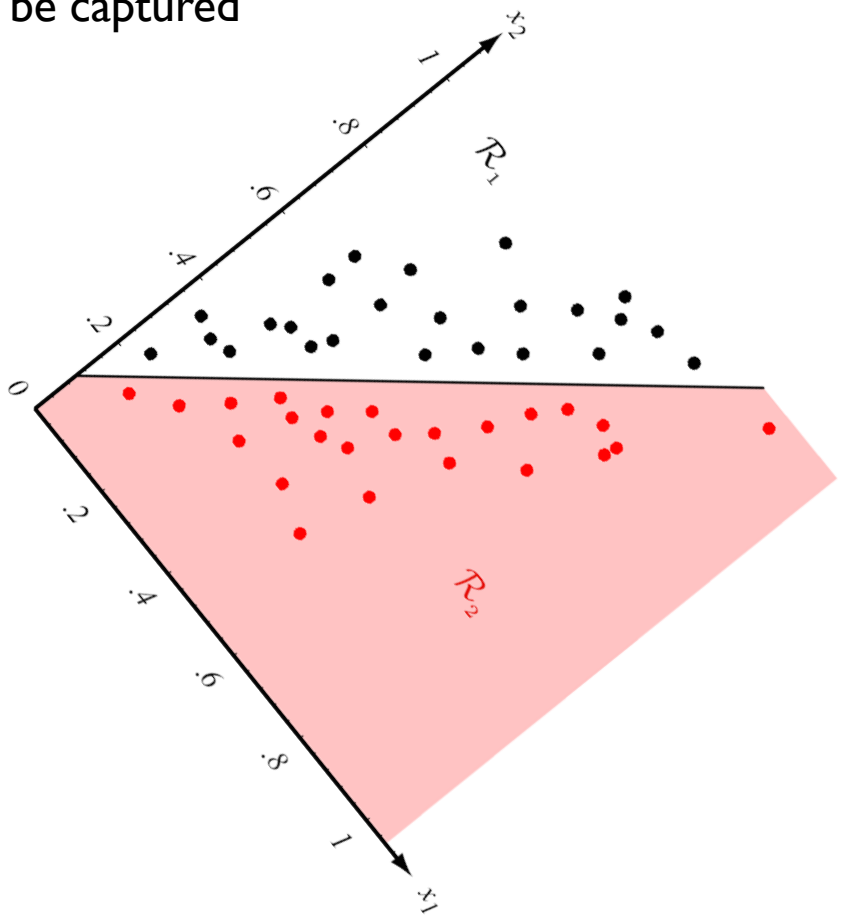
- 변수 중요도 산출 결과



Rotation Forests

- Limitation of decision tree
  - ✓ Only one variable is used for a split
  - ✓ Relationship between variables cannot be captured





Rotation Forests

Rodriguez et al. (2006)

- Rotation Forest Algorithm

Training Phase

Given

- X: the objects in the training data set (an $N \times n$ matrix)
- Y: the labels of the training set (an $N \times 1$ matrix)
- L: the number of classifiers in the ensemble
- K: the number of subsets
- $\{\omega_1, \ldots, \omega_c\}$: the set of class labels

For $i = 1 \ldots L$

- Prepare the rotation matrix R_i^a:
  - Split $\mathbf{F}$ (the feature set) into K subsets: $\mathbf{F}_{i,j}$ (for $j = 1 \ldots K$)
  - For $j = 1 \ldots K$
    - Let $X_{i,j}$ be the data set X for the features in $\mathbf{F}_{i,j}$
    - Eliminate from $X_{i,j}$ a random subset of classes
    - Select a bootstrap sample from $X_{i,j}$ of size 75% of the number of objects in $X_{i,j}$. Denote the new set by $X'_{i,j}$
    - Apply PCA on $X'_{i,j}$ to obtain the coefficients in a matrix $C_{i,j}$
  - Arrange the $C_{i,j}$, for $j = 1 \ldots K$ in a rotation matrix R_i as in equation (1)
  - Construct R_i^a by rearranging the the columns of R_i so as to match the order of features in $\mathbf{F}$.
- Build classifier D_i using (XR_i^a, Y) as the training set

Classification Phase

- For a given $\mathbf{x}$, let $d_{i,j}(\mathbf{x}R_i^a)$ be the probability assigned by the classifier D_i to the hypothesis that $\mathbf{x}$ comes from class ω_j. Calculate the confidence for each class, ω_j, by the average combination method:

$$\mu_j(\mathbf{x}) = \frac{1}{L} \sum_{i=1}^{L} d_{i,j}(\mathbf{x}R_i^a), \quad j = 1, \ldots, c.$$

- Assign $\mathbf{x}$ to the class with the largest confidence.

Rotation Forests

Rodriguez et al. (2006)

- Rotation Forest Algorithm

$$R_i = \begin{bmatrix} \mathbf{a}_{i,1}^{(1)},\mathbf{a}_{i,1}^{(2)},\ldots,\mathbf{a}_{i,1}^{(M_1)}, & [\mathbf{0}] & \cdots & [\mathbf{0}] \\ [\mathbf{0}] & \mathbf{a}_{i,2}^{(1)},\mathbf{a}_{i,2}^{(2)},\ldots,\mathbf{a}_{i,2}^{(M_2)}, & \cdots & [\mathbf{0}] \\ \vdots & \vdots & \ddots & \vdots \\ [\mathbf{0}] & [\mathbf{0}] & \cdots & \mathbf{a}_{i,K}^{(1)},\mathbf{a}_{i,K}^{(2)},\ldots,\mathbf{a}_{i,K}^{(M_K)} \end{bmatrix}$$

Rotation Forests

Rodriguez et al. (2006)

- Rotation Forest Algorithm

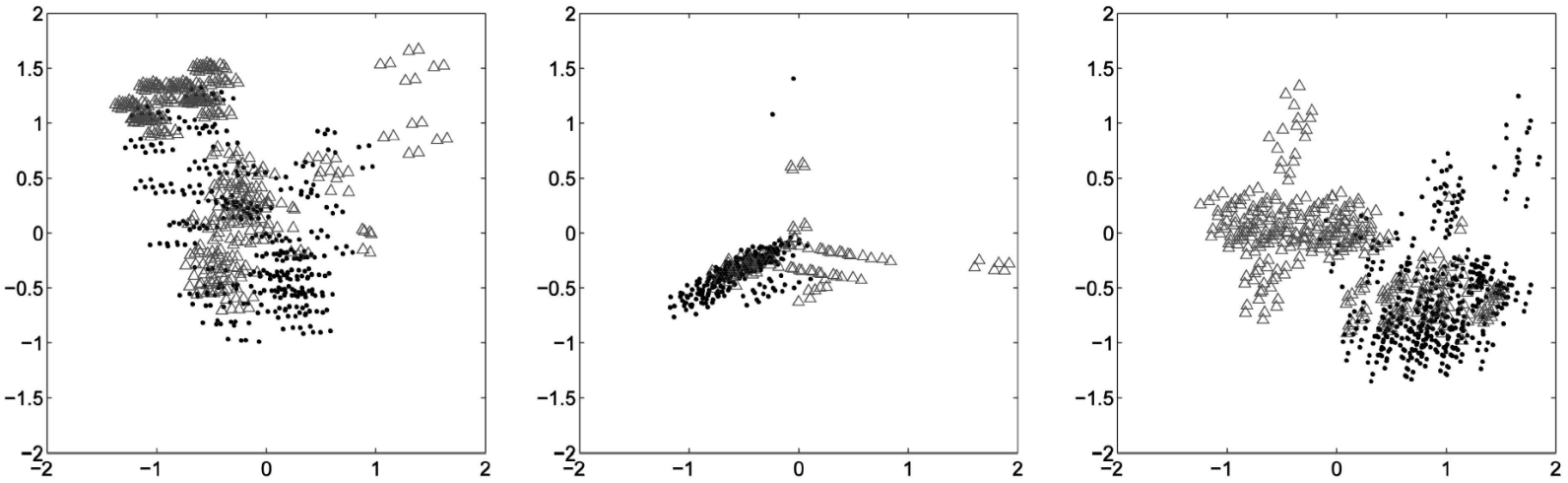
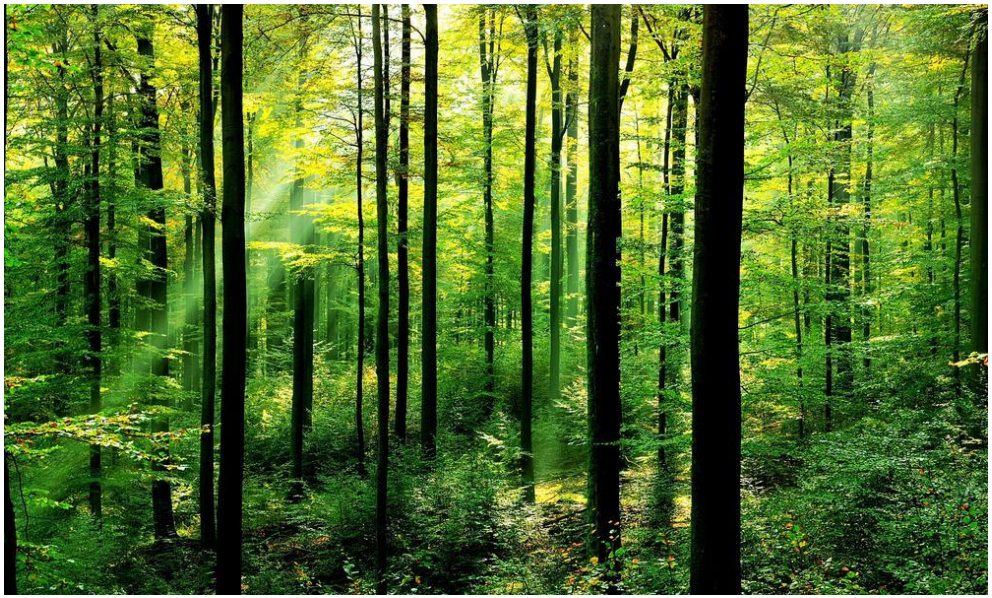


Fig. 2. Scatterplot of the "mushroom" data (22 nominal features). The binary representation of the features (121 in total) was randomly split into three subsets and PCA was applied on each subset. The scatterplots show the two classes in the space defined by the first two principal components.

Decision Jungle

- Forest vs. Jungle

Forest

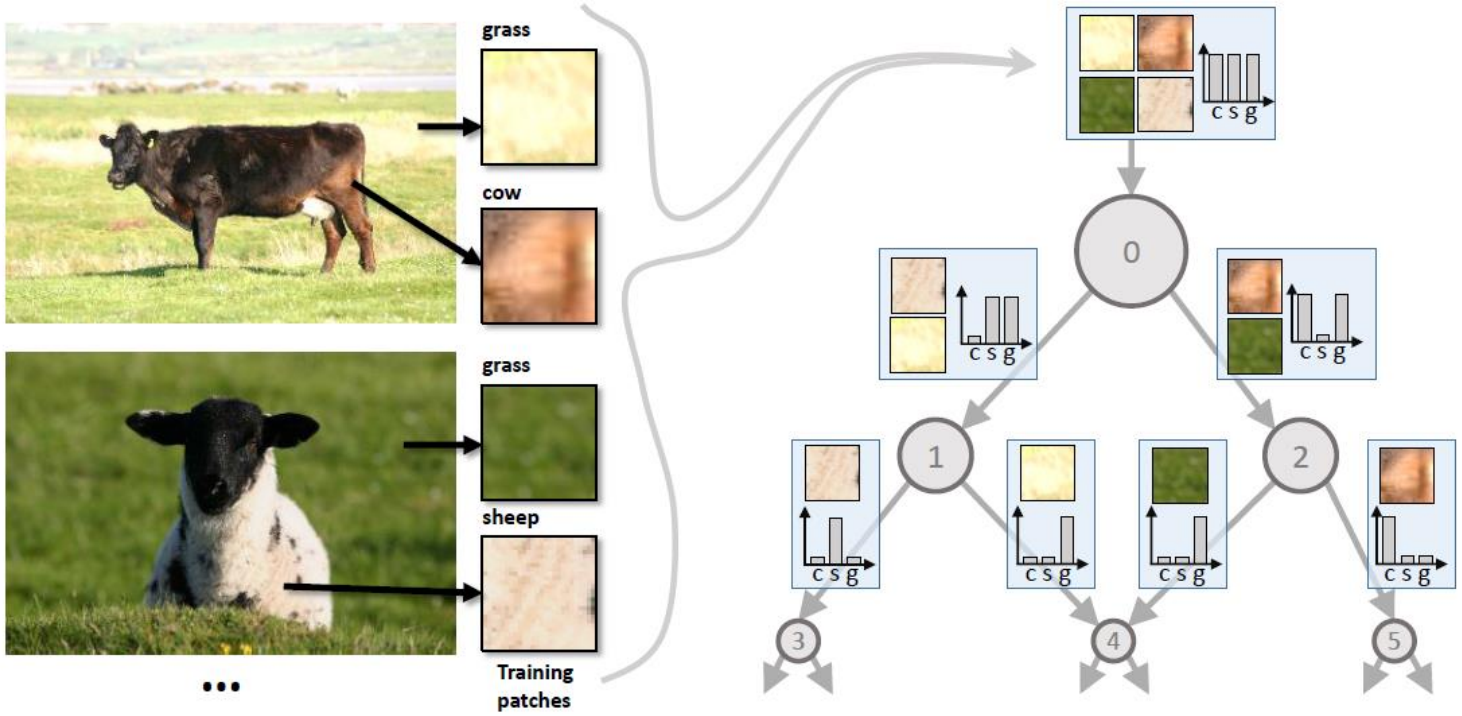


Jungle

Decision Jungle

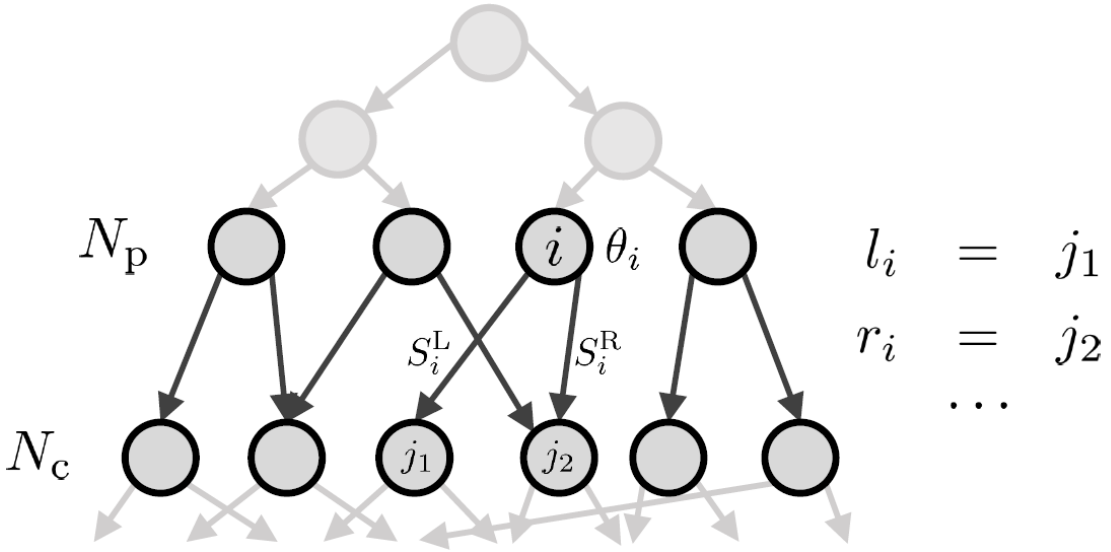
Shotton et al. (2013)

- Motivation
  - ✓ Limitation of decision tree (random forests): the number of nodes in decision trees will grow exponentially with depth given enough data.
- Main Contribution
  - ✓ Allows multiple paths from the root to each leaf.



Decision Jungle

- Notation

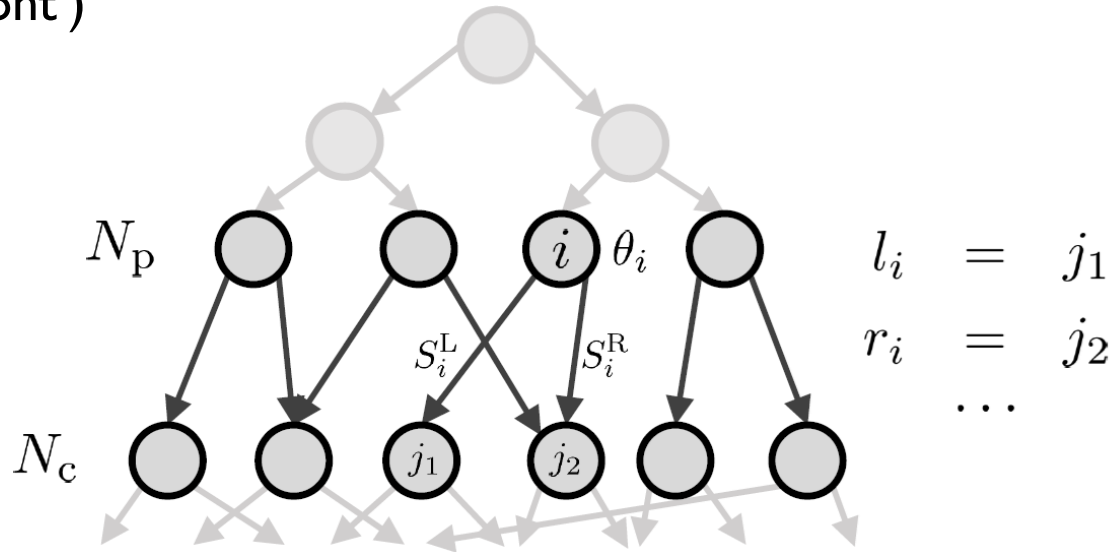


- ✓ N_p: a set of parent nodes
- ✓ N_c: a set of child nodes ($M = |N_c|$ is a parameter of decision jungle)
- ✓ θ_i: the parameters of the split feature function f for parent node i in N_p
- ✓ S_i: the set of labelled training instances (x, y) that reach node i

$$S_i^L(\theta_i) = \{(x, y) \in S_i \mid f(\theta_i, x) \leq 0\} \quad S_i^R(\theta_i) = S_i \setminus S_i^L(\theta_i)$$

Decision Jungle

- Notation (cont')



- ✓ l_i in N_c: the current assignment of the left outwards edge from parent node i
- ✓ r_i in N_c: the current assignment of the right outwards edge from parent node i
- ✓ The set of instances that reach any child node j in N_c

$$S_j(\{\theta_i\},\{l_i\},\{r_i\}) = \left[\bigcup_{i\in N_{\mathrm{p}}\ \mathrm{s.t.}\ l_i=j} S_i^{\mathrm{L}}(\theta_i)\right] \cup \left[\bigcup_{i\in N_{\mathrm{p}}\ \mathrm{s.t.}\ r_i=j} S_i^{\mathrm{R}}(\theta_i)\right]$$

Decision Jungle

- Training
  - ✓ Goal: joint minimization of the objective over the split parameters $\{\theta_i\}$ and the child assignment $\{l_i\}$ and $\{r_i\}$.
  - ✓ Task of learning the current level of a DAG

$$\min_{\{\theta_i\},\{l_i\},\{r_i\}} E(\{\theta_i\},\{l_i\},\{r_i\})$$

  - ✓ Objective function: total weighted entropy of instances

$$E(\{\theta_i\},\{l_i\},\{r_i\}) = \sum_{j \in N_c} |S_j|\, H(S_j)$$

  - ✓ The minimization problem is hard to solve exactly

Decision Jungle

- Lsearch
  - ✓ Starts from a feasible assignment of the parameters, and alternate between two coordinate descent step
    - Split optimization step

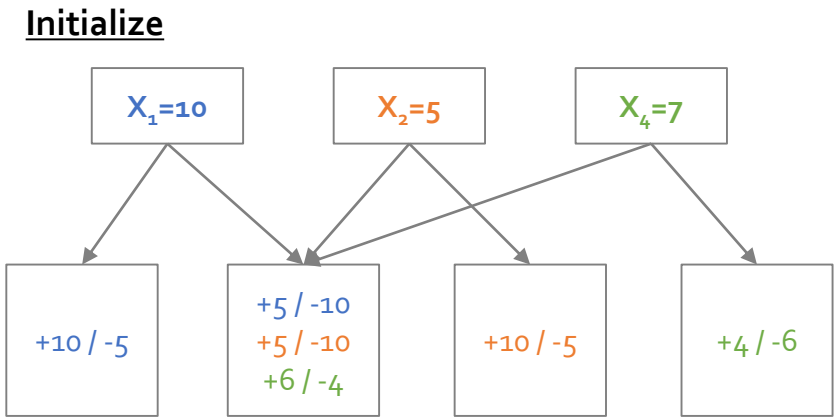
$$\text{for } k \in N_{\mathrm{p}}$$
$$\theta_k \leftarrow \underset{\theta'_k}{\operatorname{argmin}}\, E(\theta'_k \cup \{\theta_i\}_{i \in N_{\mathrm{p}} \setminus \{k\}}, \{l_i\}, \{r_i\})$$

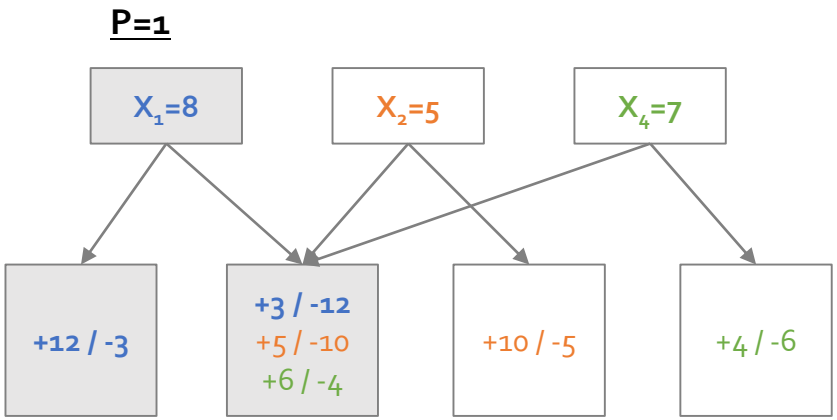
    - Branch optimization step

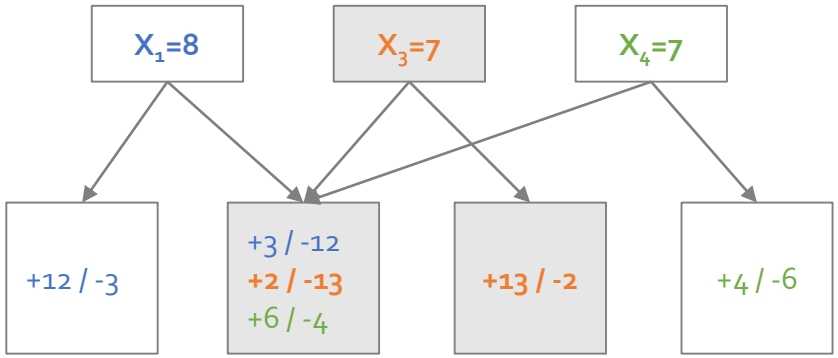
$$\text{for } k \in N_{\mathrm{p}}$$
$$l_k \leftarrow \underset{l'_k \in N_{\mathrm{c}}}{\operatorname{argmin}}\, E(\{\theta_i\}, l'_k \cup \{l_i\}_{i \in N_{\mathrm{p}} \setminus \{k\}}, \{r_i\})$$
$$r_k \leftarrow \underset{r'_k \in N_{\mathrm{c}}}{\operatorname{argmin}}\, E(\{\theta_i\}, \{l_i\}, r'_k \cup \{r_i\}_{i \in N_{\mathrm{p}} \setminus \{k\}})$$

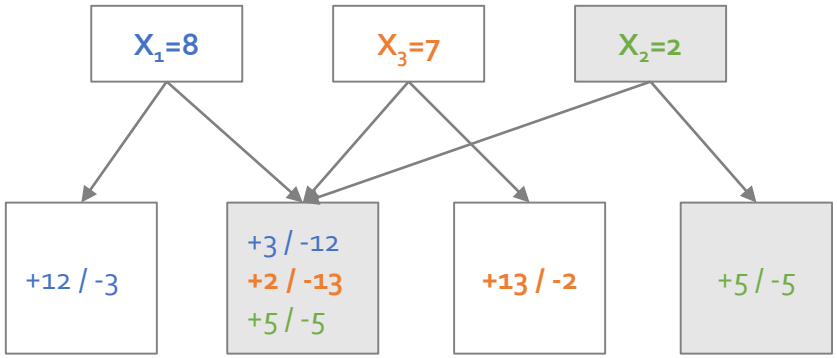
Decision Jungle

- Split Optimization Example



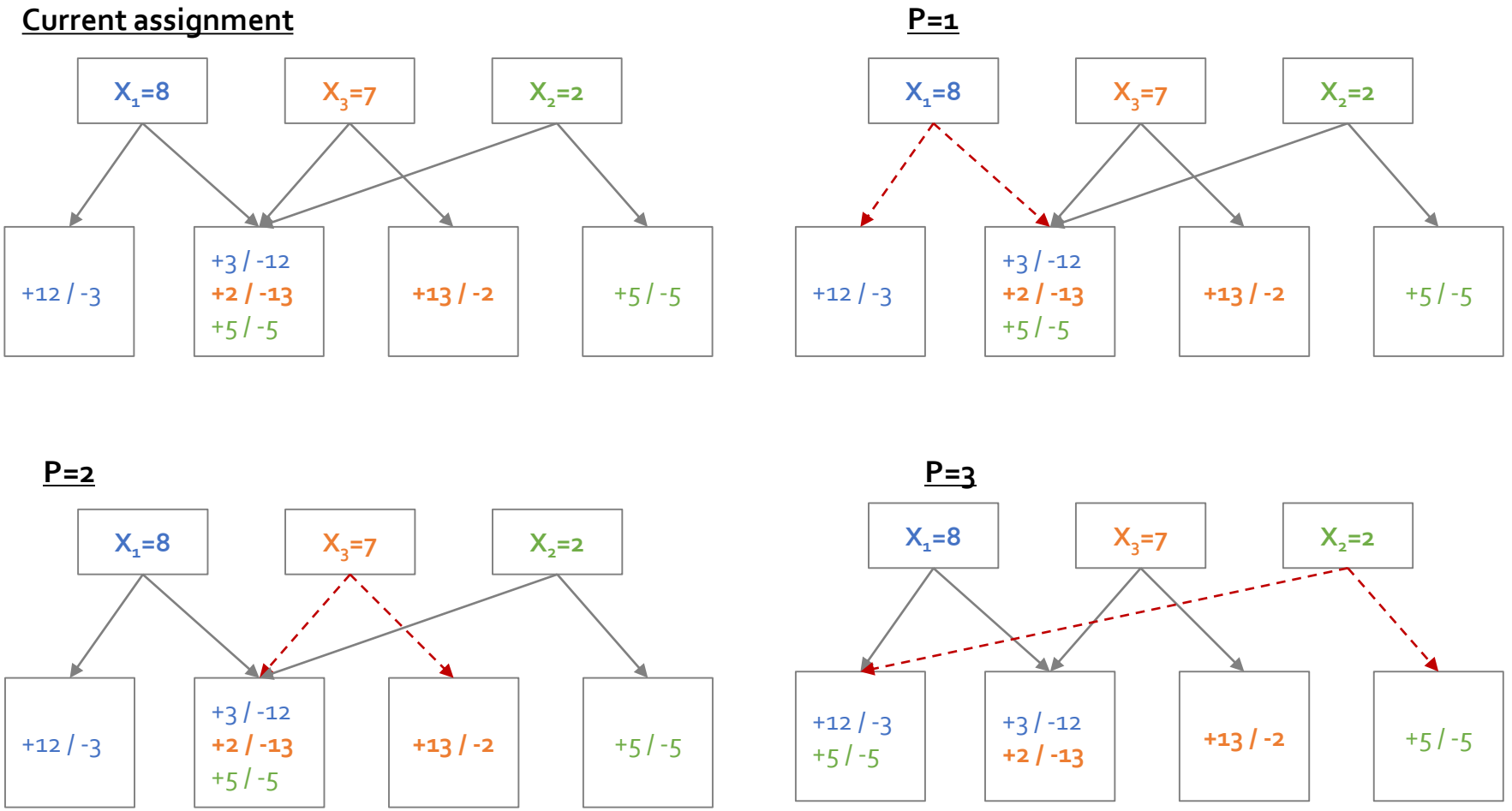






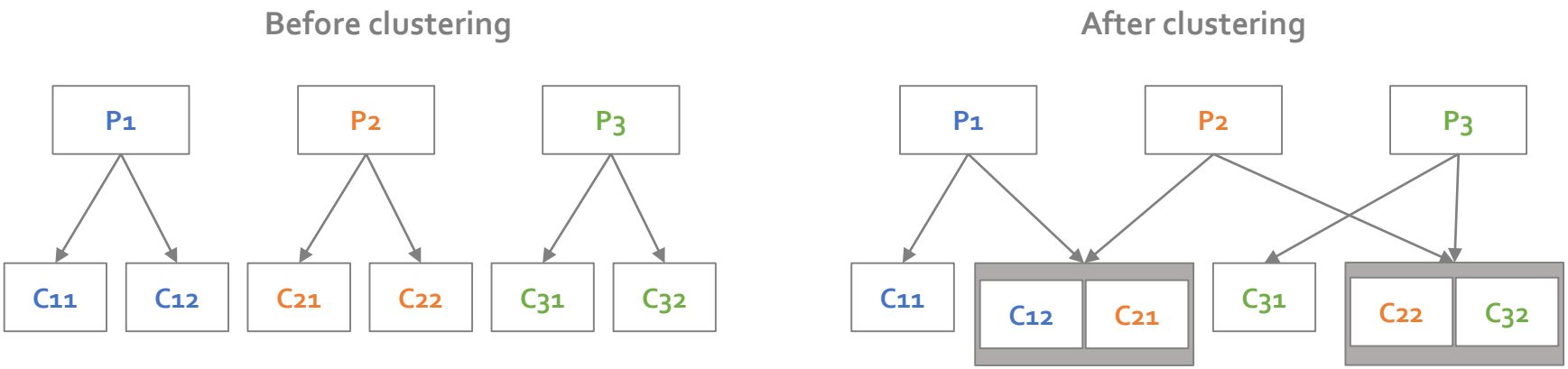
Decision Jungle

- Branch Optimization Example



Decision Jungle

- Cluster Search
  - ✓ Branching variables more globally
    - $2|N_p|$ temporary child nodes are built via conventional tree-based, training-objective minimization procedures
    - The temporary nodes are clustered into $M = |N_c|$ groups

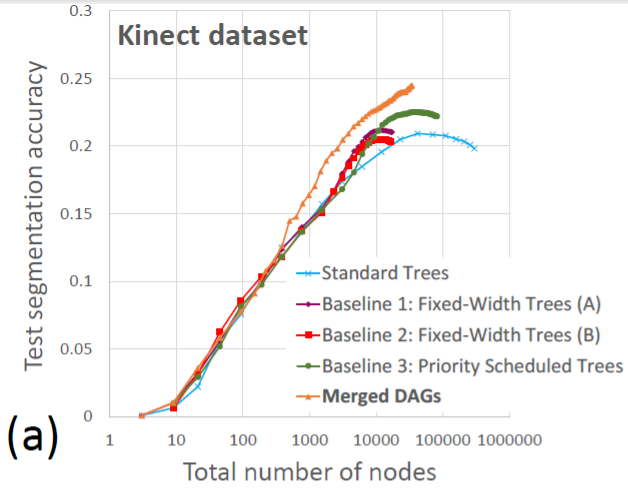


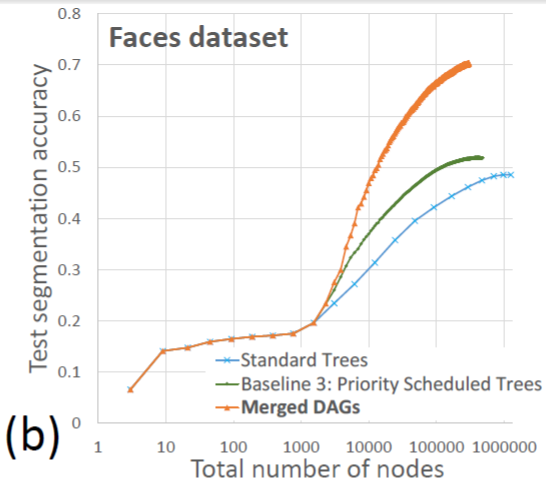
Decision Jungle

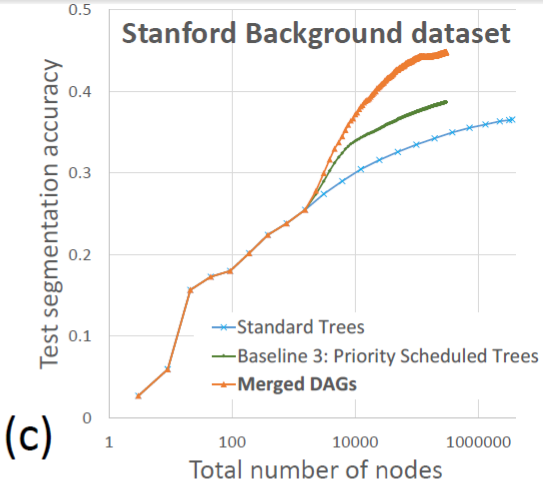
- Experiments
  - ✓ Datasets
    - **Kinect body part classification**: 31 classes, 1,000 training images with 250 example pixels randomly sampled per image, 1,000 test datasets
    - **Facial features segmentation**: 8 classes, 1,000 training images using every pixel
    - **Stanford background dataset**: 8 classes with 715 training images
    - **UCI data sets**: 28 classification datasets
  - ✓ Benchmark algorithms
    - Standard Forests
    - Fixed-width Tree: (A) split M/2 parent nodes with highest information gain, (B) select M child nodes that most reduce the objective
    - Priority scheduled trees

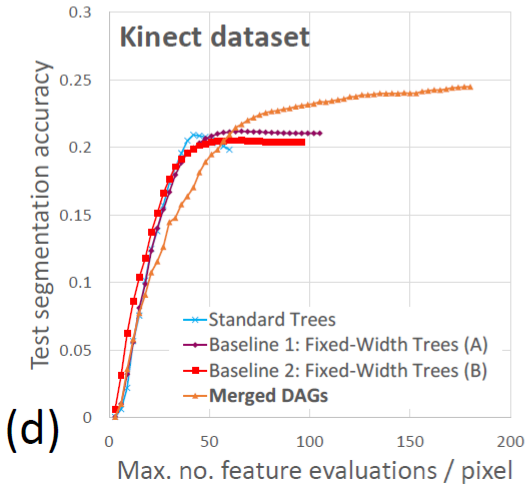
Decision Jungle

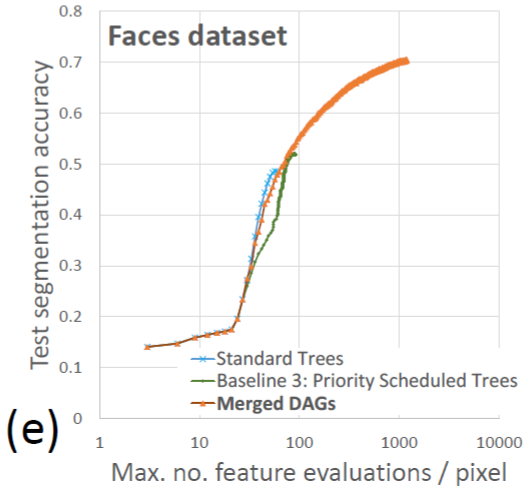
- Experimental Results: Image Classification

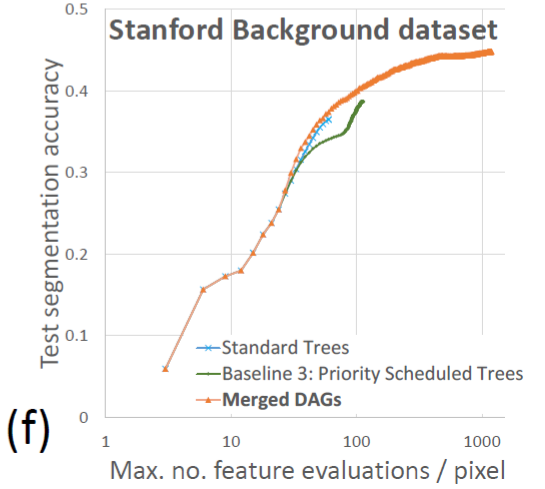






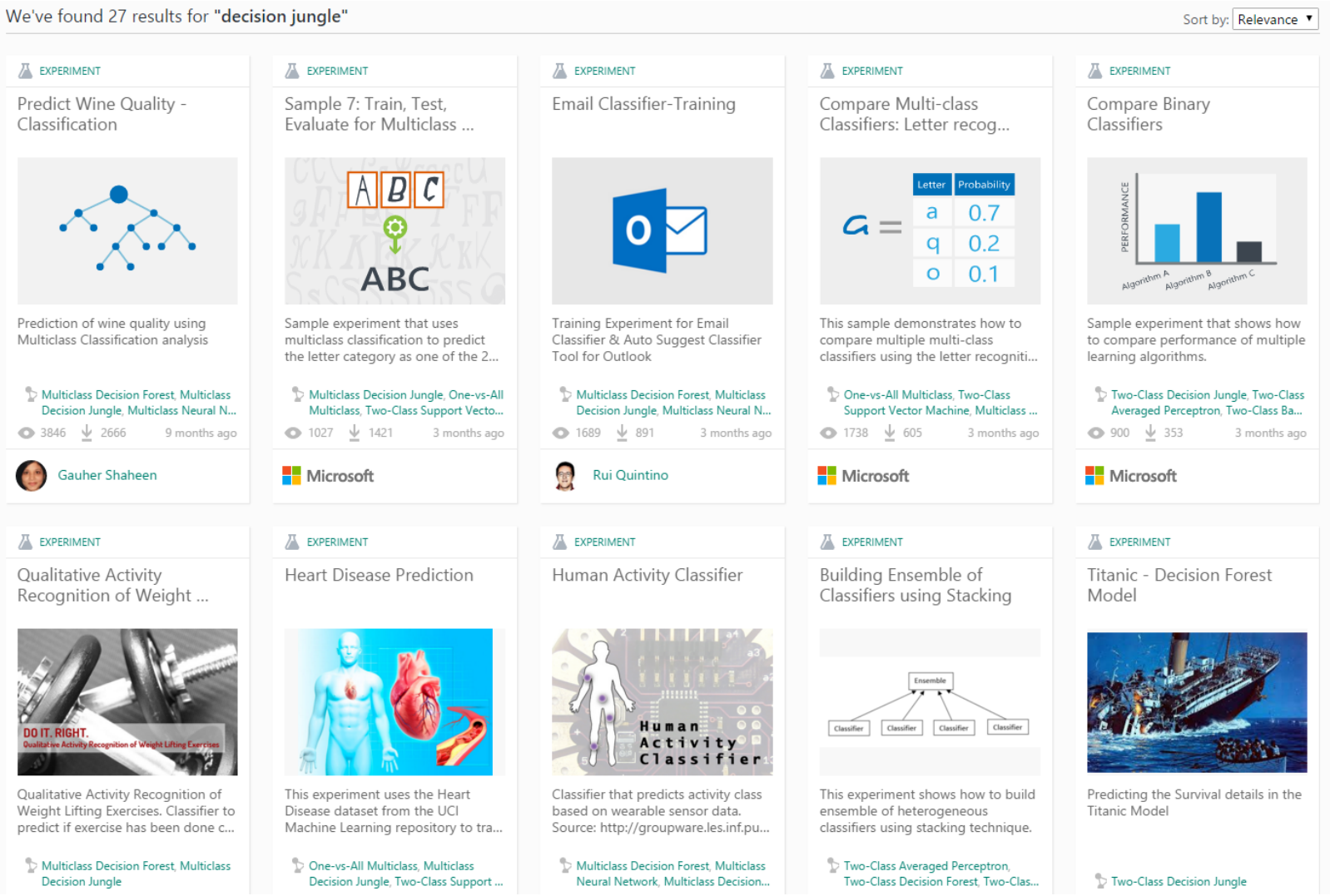






Decision Jungle

- Implementation



http://gallery.cortanaanalytics.com/browse?s=%22decision%20jungle%22

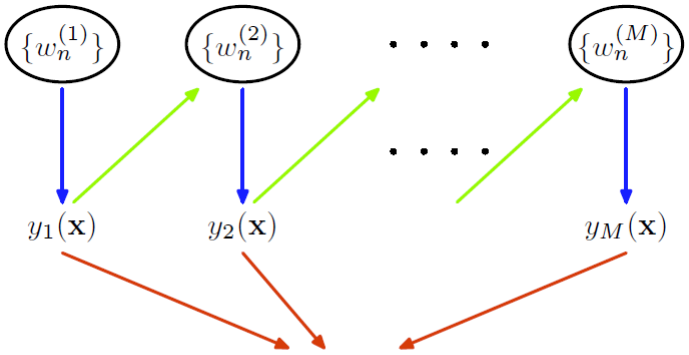
AGENDA

Boosting: AdaBoost

- AdaBoosting: Idea
  - ✓ Strong model vs. Weak model
    - A weak model, performing only slightly better than random guessing, could be boosted in to arbitrarily accurate strong model
  - ✓ New classifiers should focus on difficult cases
    - Examine the learning set
    - Get some rule of thumb
    - Reweight the examples of the training set, concentrate on hard cases for the previous rule
    - Derive the next rule of thumb
    - …
    - Build a single, accurate predictor by combining the rules of thumb

Boosting: AdaBoost

- AdaBoosting: Idea
  - ✓ Strong model vs. Weak model
    - A weak model, performing only slightly better than random guessing, could be boosted in to arbitrarily accurate strong model
  - ✓ Train models sequentially, with a new model training at each round
  - ✓ At the end of each round, misclassified examples are identified and have their emphasis increased in a new training set which is then fed back into the next round
  - ✓ Large errors made by earlier models can be compensated by the subsequent models



$$Y_M(\mathbf{x}) = \text{sign}\left(\sum_m^M \alpha_m y_m(\mathbf{x})\right)$$

Boosting: AdaBoost

- AdaBoosting: Algorithm

Algorithm 2 Adaboost

Input: Required ensemble size T

Input: Training set $S = \{(x_1, y_1), (x_2, y_2), ..., (x_N, y_N)\}$, where $y_i \in \{-1, +1\}$

Define a uniform distribution $D_1(i)$ over elements of S.

for $t = 1$ to T **do**

Train a model h_t using distribution D_t.

Calculate $\epsilon_t = P_{D_t}(h_t(x) \neq y)$

If $\epsilon_t \geq 0.5$ break

Set $\alpha_t = \frac{1}{2} \ln \left(\frac{1-\epsilon_t}{\epsilon_t} \right)$

Update $D_{t+1}(i) = \frac{D_t(i) \exp(-\alpha_t y_i h_t(x_i))}{Z_t}$

where Z_t is a normalization factor so that D_{t+1} is a valid distribution.

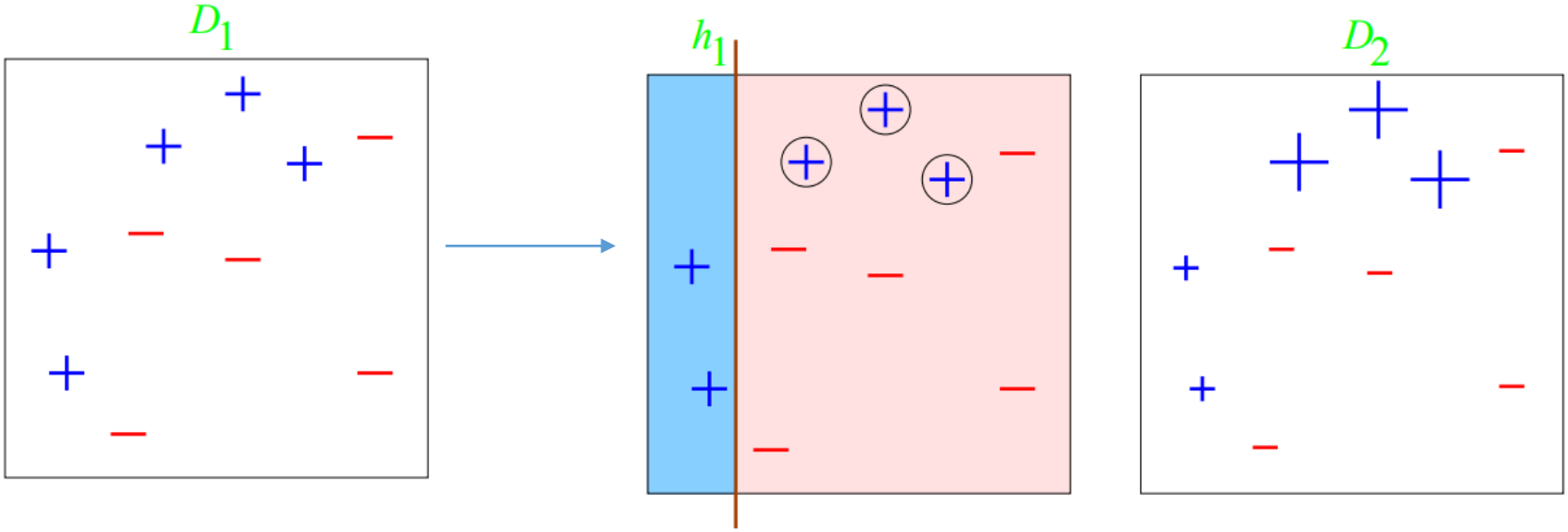
end for

For a new testing point (x', y'),

$H(x') = sign\left(\sum_{t=1}^{T} \alpha_t h_t(x') \right)$

Boosting: AdaBoost

- Illustrative example 1
  - ✓ Round 1



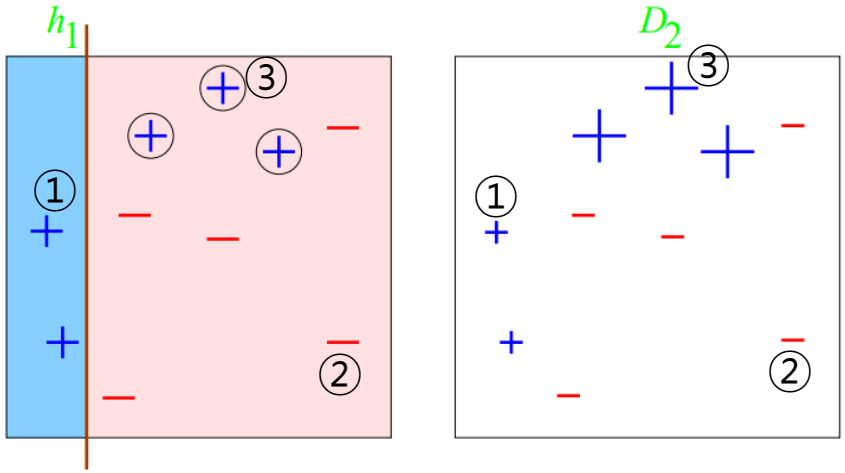
- 3 misclassifications out of 10: $\epsilon_i = 0.30$
- Model confidence: $\alpha_i = \frac{1}{2}\log\left(\frac{1-\epsilon_i}{\epsilon_i}\right) = \frac{1}{2}\log\frac{1-0.3}{0.3} = 0.42$

Boosting: AdaBoost

- AdaBoost Example
  - ✓ The selection probability of x_i for the next training dataset

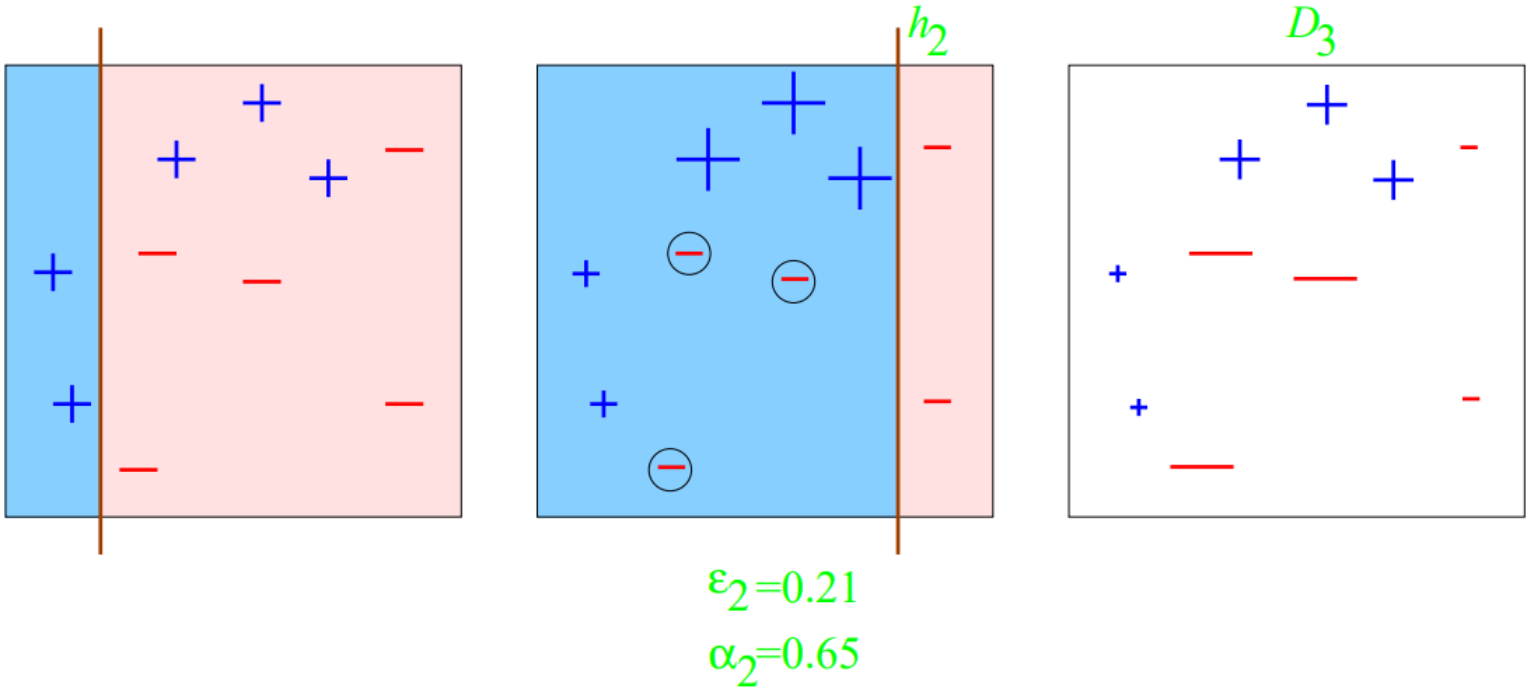
$$D_{t+1}(i) = \frac{D_t(i)\exp(-\alpha_t y_i h_t(x_i))}{Z_t}$$

  - ✓ Case 1: $y_i = 1, h_t(x_i) = 1 \rightarrow y_i h_t(x_i) = 1 \rightarrow -\alpha_t y_i h_t(x_i) < 0 \rightarrow$ increase p
  - ✓ Case 2: $y_i = -1, h_t(x_i) = -1 \rightarrow y_i h_t(x_i) = 1 \rightarrow -\alpha_t y_i h_t(x_i) < 0 \rightarrow$ decrease p
  - ✓ Case 3: $y_i = 1, h_t(x_i) = -1 \rightarrow y_i h_t(x_i) = -1 \rightarrow -\alpha_t y_i h_t(x_i) > 0 \rightarrow$ increase p
  - ✓ α_t is the confidence of the current model that controls the magnitude of change



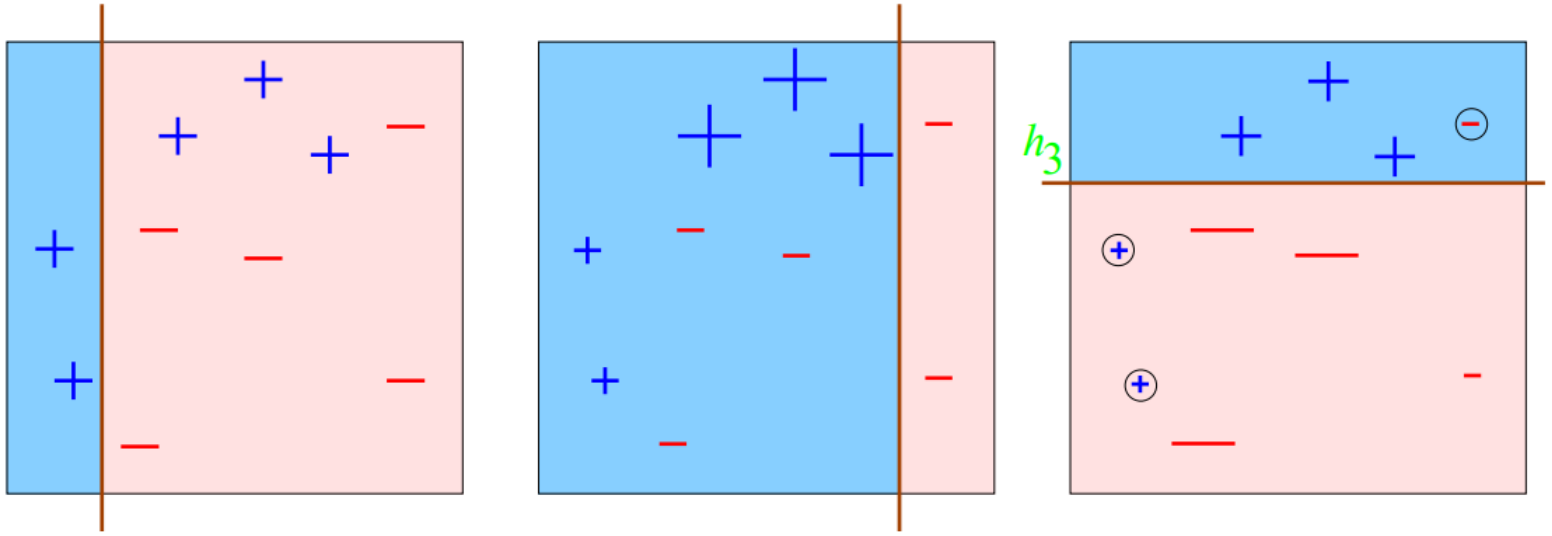
Boosting: AdaBoost

- Illustrative example 1
  - ✓ Round 2



Boosting: AdaBoost

- Illustrative example 1
  - ✓ Round 3

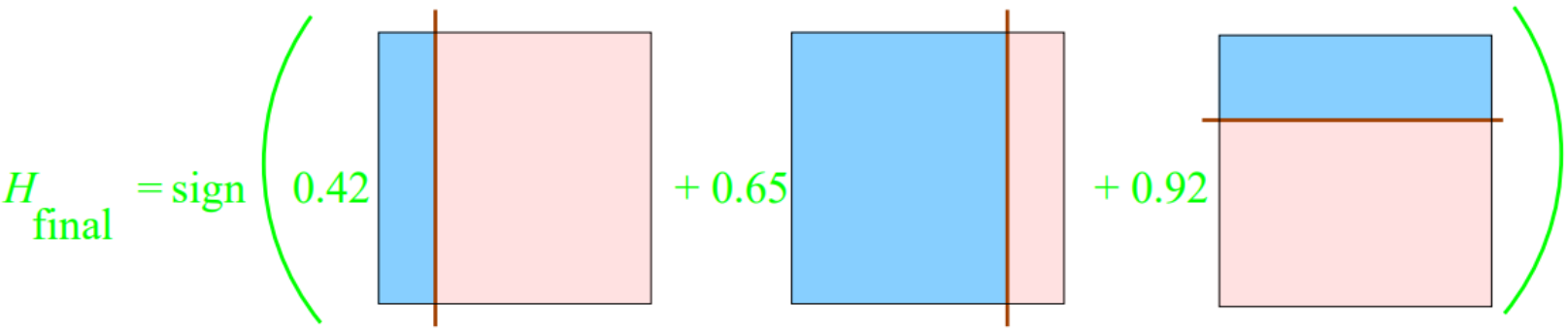


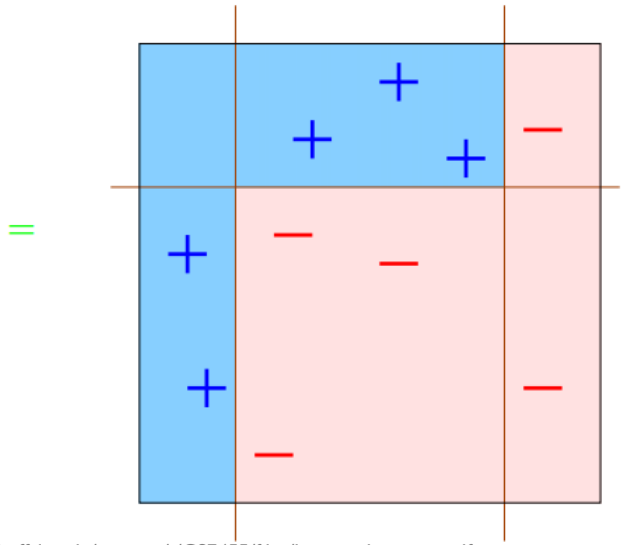
$\varepsilon_3 = 0.14$

$\alpha_3 = 0.92$

Boosting: AdaBoost

- Illustrative example 1
  - ✓ Final classifier

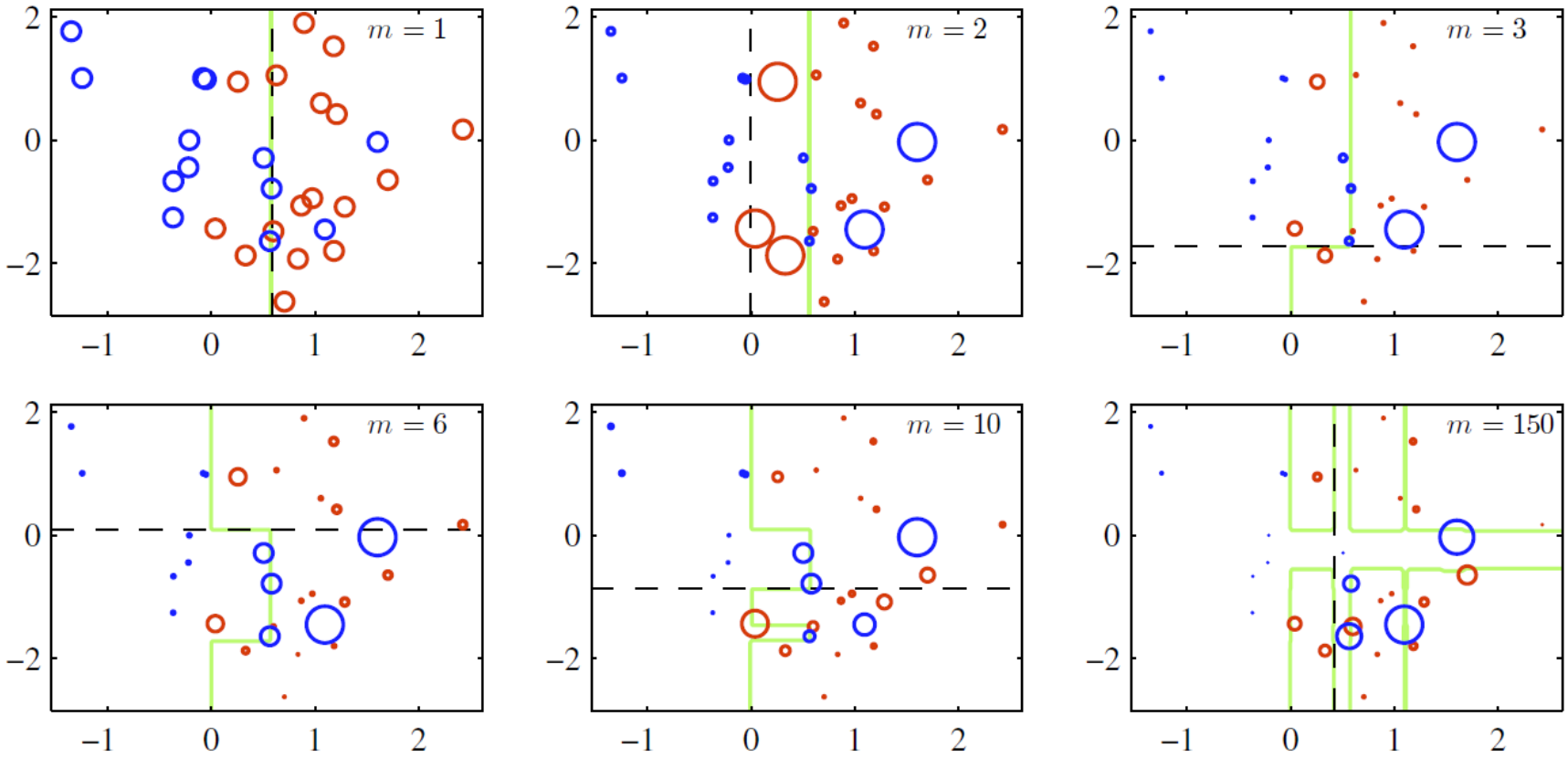




https://www.cse.buffalo.edu/~jcorso/t/CSE455/files/lecture_boosting.pdf

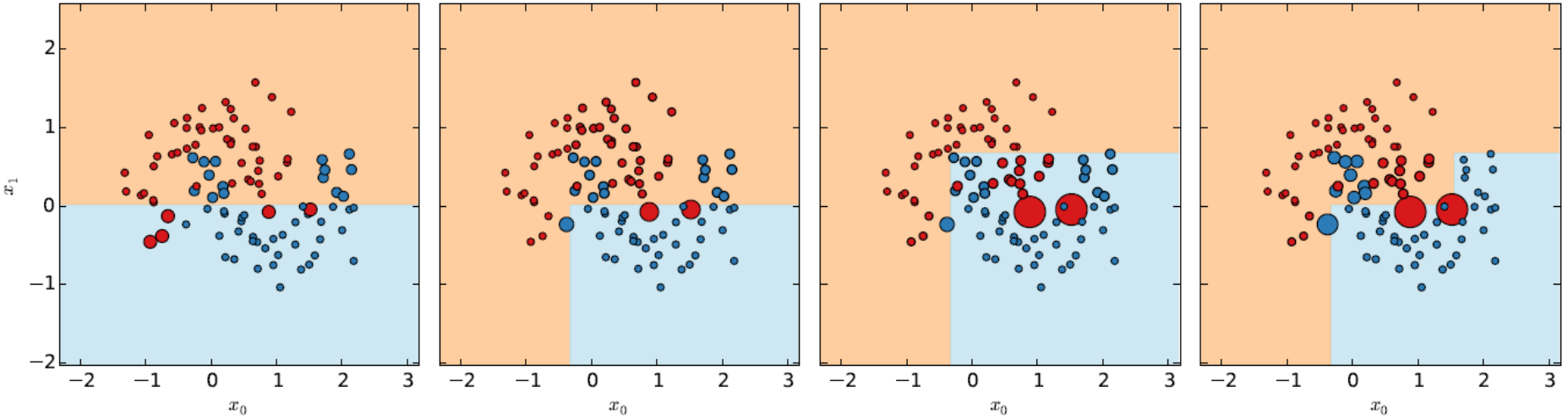
Boosting: AdaBoost

- Illustrative example 2



Boosting: AdaBoost

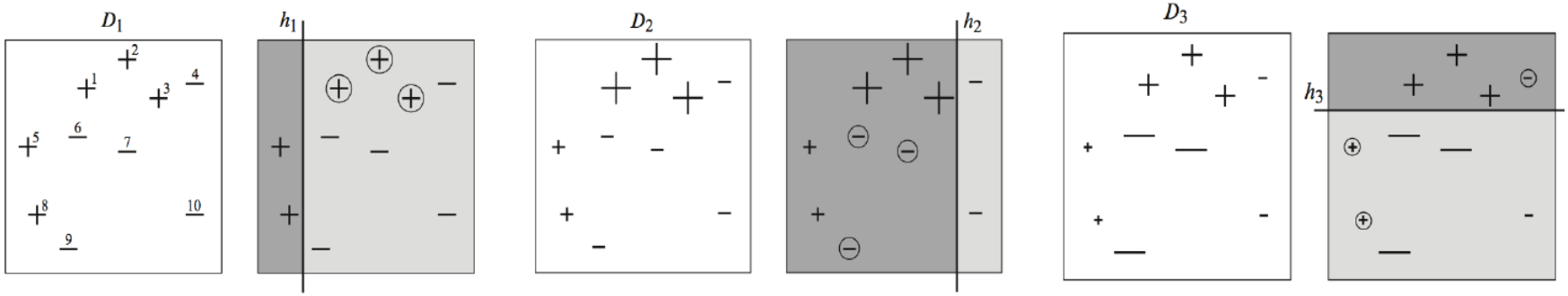
- Illustrative example 3



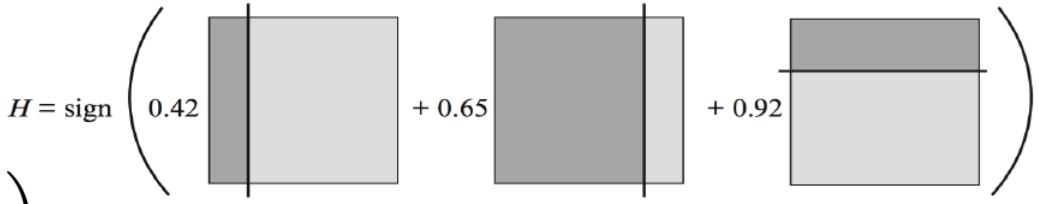
https://www.slideshare.net/DataRobot/gradient-boosted-regression-trees-in-scikitlearn?from_action=save

Boosting: AdaBoost

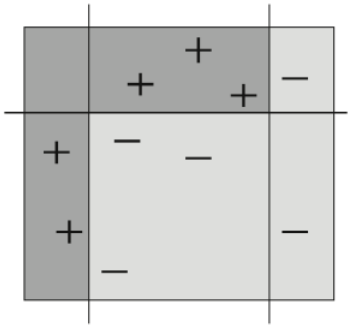
- Illustrative example 4



$$H(x') = sign\left(\sum_{t=1}^{T} \alpha_t h_t(x')\right)$$

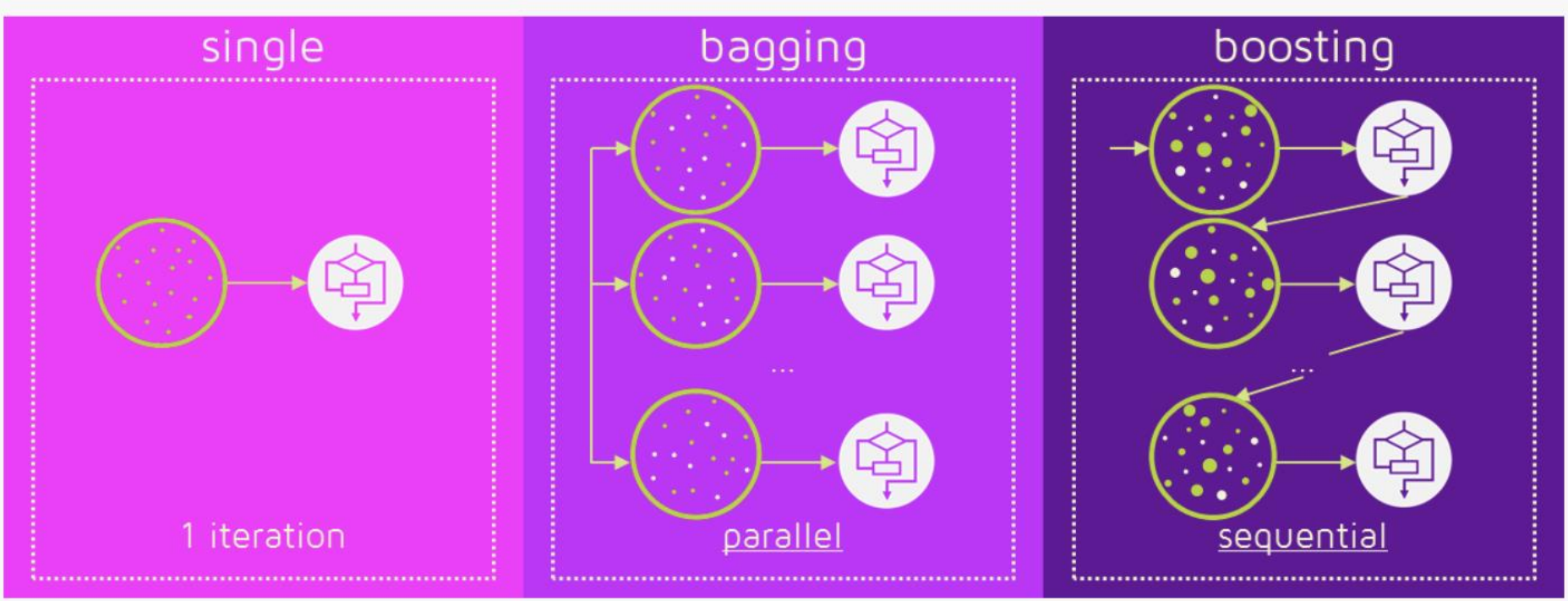


=



Boosting: AdaBoost

- Single model vs. Bagging vs. Boosting



https://quantdare.com/what-is-the-difference-between-bagging-and-boosting/

Boosting: AdaBoost

- AdaBoost in Action

AdaBoost in Action

Kai O. Arras
Social Robotics Lab, University of Freiburg

Nov 2009 Social Robotics Laboratory

https://www.youtube.com/watch?v=k4G2VCuOMMg

Boosting: AdaBoost

- Bagging vs. Boosting
  - ✓ Selected instances in each training dataset

A sample of a single classifier on an imaginary set of data.	
	(Original) Training Set
Training-set-1:	1, 2, 3, 4, 5, 6, 7, 8

A sample of Bagging on the same data.	
	(Resampled) Training Set
Training-set-1:	2, 7, 8, 3, 7, 6, 3, 1
Training-set-2:	7, 8, 5, 6, 4, 2, 7, 1
Training-set-3:	3, 6, 2, 7, 5, 6, 2, 2
Training-set-4:	4, 5, 1, 4, 6, 4, 3, 8

A sample of Boosting on the same data.	
	(Resampled) Training Set
Training-set-1:	2, 7, 8, 3, 7, 6, 3, 1
Training-set-2:	1, 4, 5, 4, 1, 5, 6, 4
Training-set-3:	7, 1, 5, 8, 1, 8, 1, 4
Training-set-4:	1, 1, 6, 1, 1, 3, 1, 5

Boosting: AdaBoost

- Face detection with AdaBoost

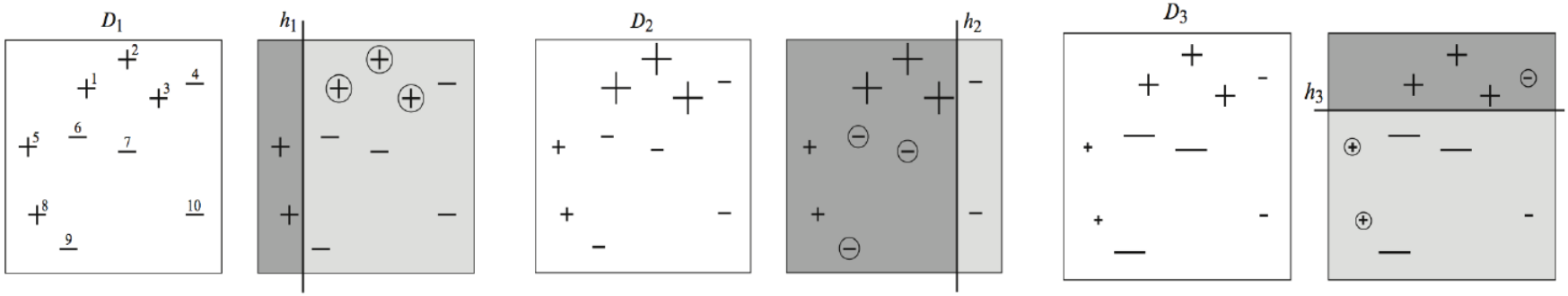
https://www.youtube.com/watch?v=fCnV4MyI9-M

Gradient Boosting Machine: GBM

Friedman (2001), Natekin and Knoll (2013)

Gradient Boosting = Gradient Descent + Boosting

- Adaboost



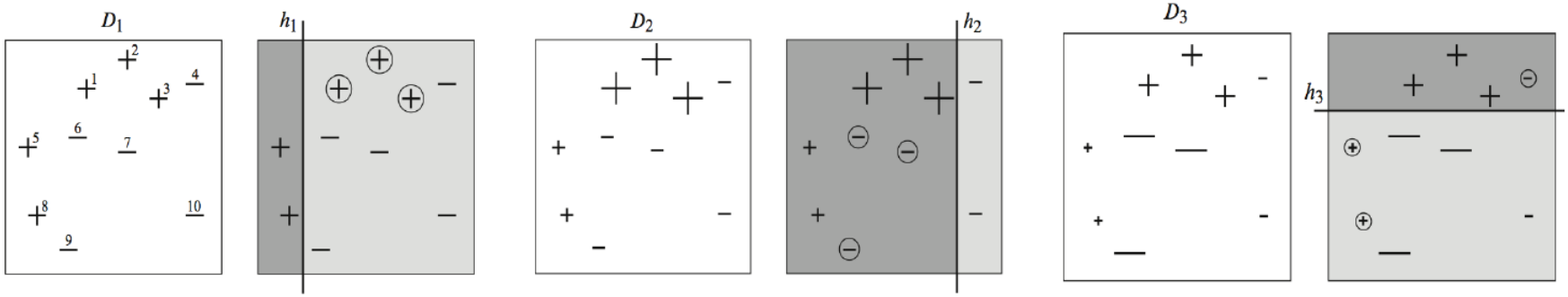
✓ Fit an additive model (ensemble) $\sum_t \rho_t h_t(x)$ in a forward stage-wise manner.

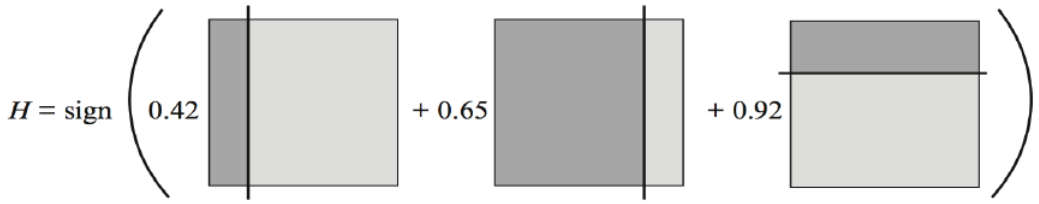
✓ In each stage, introduce a weak leaner to compensate the shortcomings of existing weak leaners.

✓ In Adaboost, "shortcomings" are identified by high-weight data points.

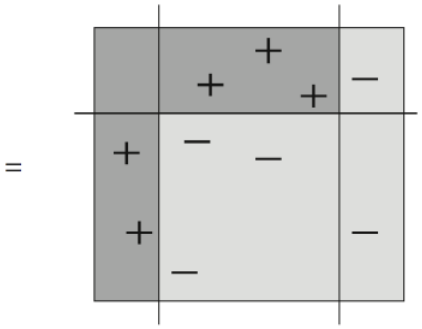
Gradient Boosting Machine: GBM

- Adaboost





$$H(x) = \sum_t \rho_t h_t(x)$$



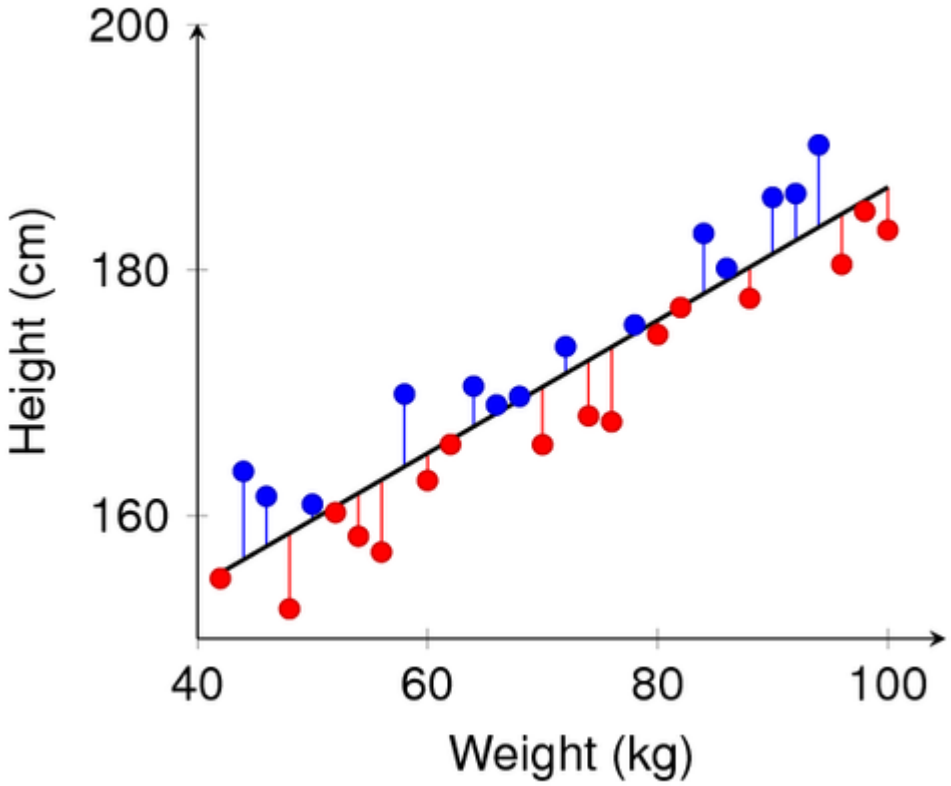
Gradient Boosting Machine: GBM

- Gradient Boosting
  - ✓ Fit an additive model (ensemble) $\sum_t \rho_t h_t(x)$ in a forward stage-wise manner.
  - ✓ In each stage, introduce a weak leaner to compensate the shortcomings of existing weak leaners.
  - ✓ In Gradient Boosting, "shortcomings" are identified by gradients.
  - ✓ Both high-weight data points and gradients tell us how to improve our model.

- Gradient Boosting for Different Problems
  - ✓ Difficulty: Regression < Classification < Ranking
    - Associated with the complexity of the derivative of a loss function

Gradient Boosting Machine: GBM

- Motivation (for regression problem)
  - ✓ What if we attempt to predict the residuals with the additional regression model?



Gradient Boosting Machine: GBM

- Main idea

Original Dataset

x^1	y^1
x^2	y^2
x^3	y^3
x^4	y^4
x^5	y^5
x^6	y^6
x^7	y^7
x^8	y^8
x^9	y^9
x^{10}	y^{10}

Modified Dataset 1

x^1	$y^1-f_1(x^1)$
x^2	$y^2-f_1(x^2)$
x^3	$y^3-f_1(x^3)$
x^4	$y^4-f_1(x^4)$
x^5	$y^5-f_1(x^5)$
x^6	$y^6-f_1(x^6)$
x^7	$y^7-f_1(x^7)$
x^8	$y^8-f_1(x^8)$
x^9	$y^9-f_1(x^9)$
x^{10}	$y^{10}-f_1(x^{10})$

Modified Dataset 2

x^1	$y^1-f_1(x^1)-f_2(x^1)$
x^2	$y^2-f_1(x^2)-f_2(x^2)$
x^3	$y^3-f_1(x^3)-f_2(x^3)$
x^4	$y^4-f_1(x^4)-f_2(x^4)$
x^5	$y^5-f_1(x^5)-f_2(x^5)$
x^6	$y^6-f_1(x^6)-f_2(x^6)$
x^7	$y^7-f_1(x^7)-f_2(x^7)$
x^8	$y^8-f_1(x^8)-f_2(x^8)$
x^9	$y^9-f_1(x^9)-f_2(x^9)$
x^{10}	$y^{10}-f_1(x^{10})-f_2(x^{10})$

$$y = f_1(\mathbf{x}) \qquad y - f_1(\mathbf{x}) = f_2(\mathbf{x}) \qquad y - f_1(\mathbf{x}) - f_2(\mathbf{x}) = f_3(\mathbf{x})$$

Gradient Boosting Machine: GBM

- How is this idea related to the gradient?
  - ✓ Loss function of the ordinary least square (OLS)

$$\min L = \frac{1}{2}\sum_{i=1}^{n}(y_i - f(\mathbf{x}_i))^2$$

  - ✓ Gradient of the Loss function

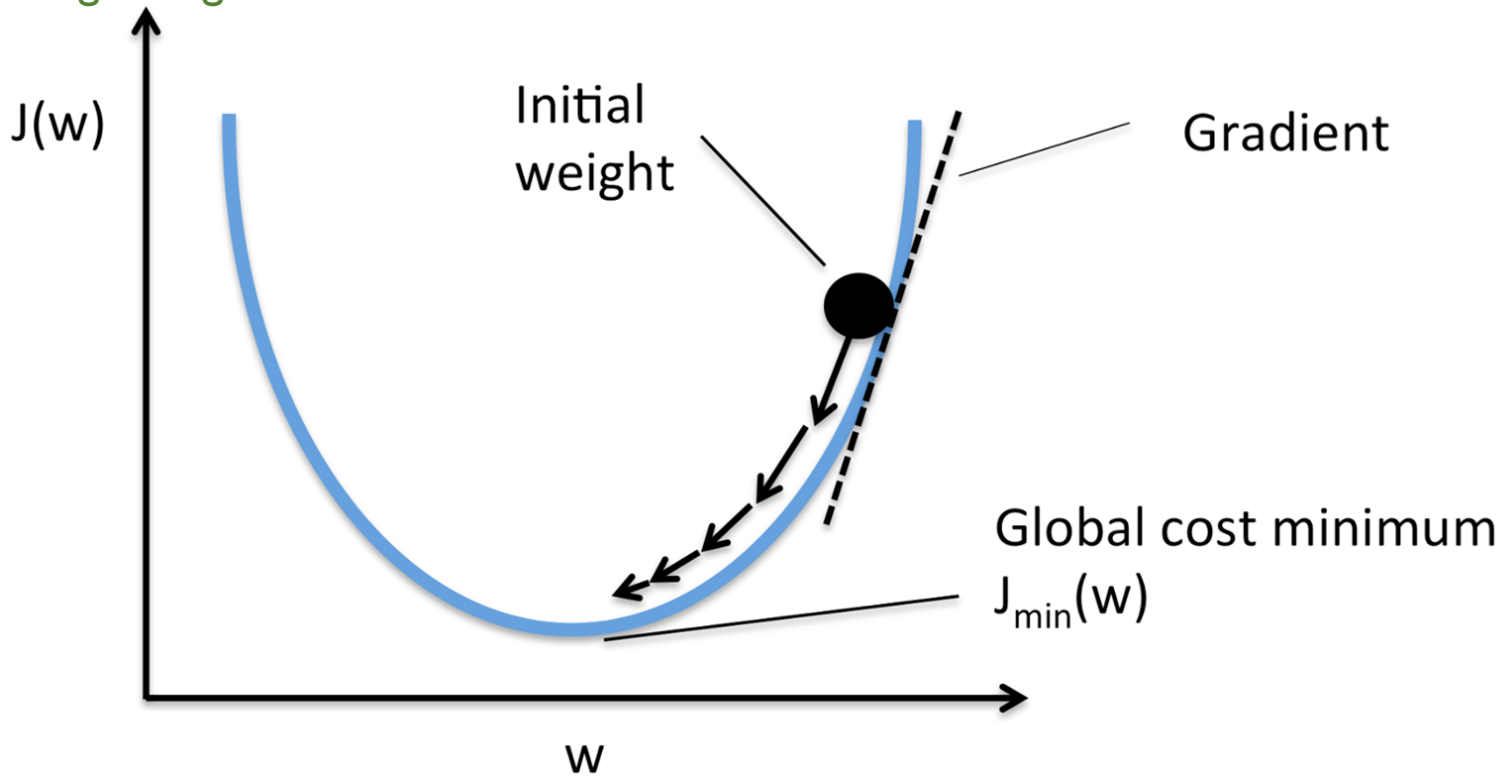
$$\frac{\partial L}{\partial f(\mathbf{x}_i)} = f(\mathbf{x}_i) - y_i$$

  - ✓ Residuals are the negative gradient of the loss function

$$y_i - f(\mathbf{x}_i) = -\frac{\partial L}{\partial f(\mathbf{x}_i)}$$

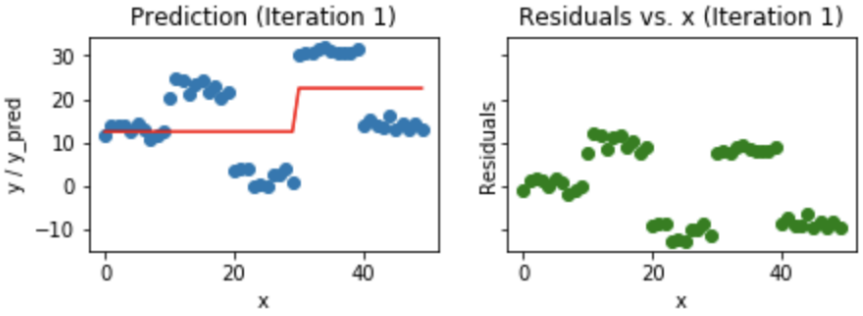
Gradient Boosting Machine: GBM

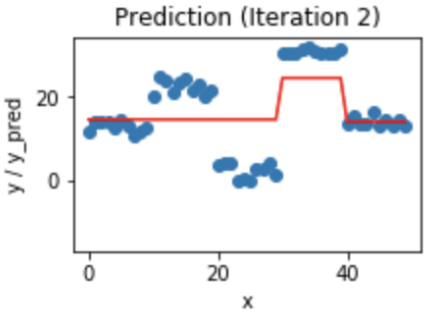
- Gradient Descent Algorithm
  - ✓ Blue line: value of loss function with a given parameter
  - ✓ Black point: current state
  - ✓ Arrows: the direction that the parameter should follow to minimize the loss function
    = negative gradient of the loss function

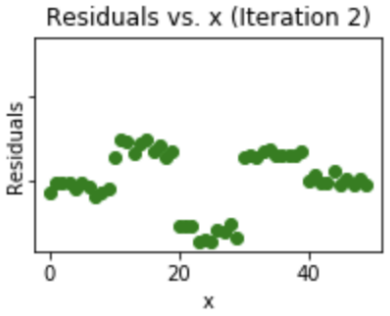


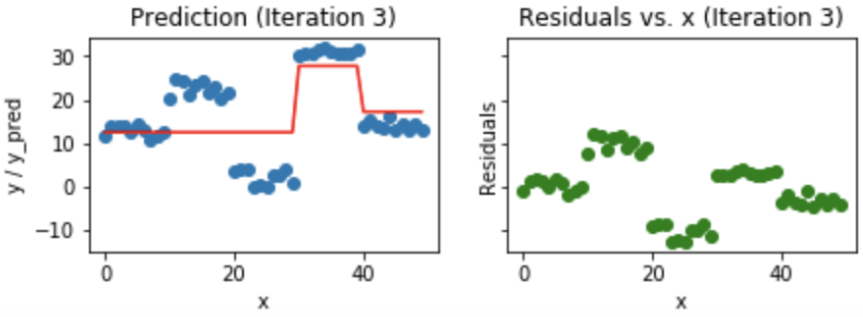
Gradient Boosting Machine: GBM

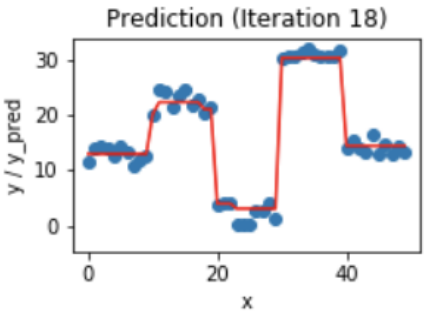
- GBM Regression Example 1

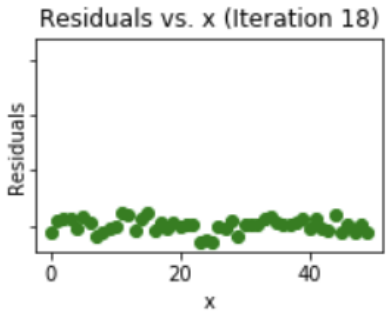


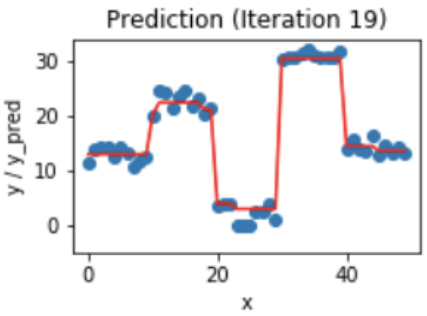


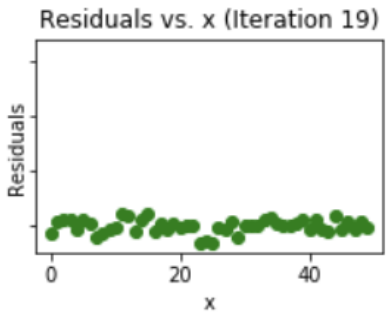


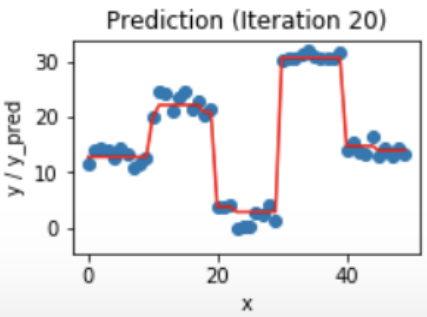


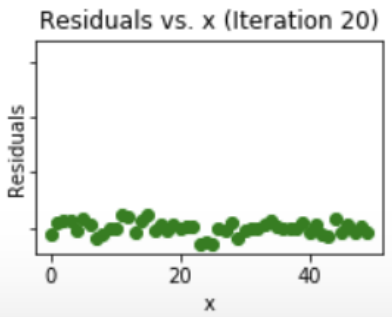








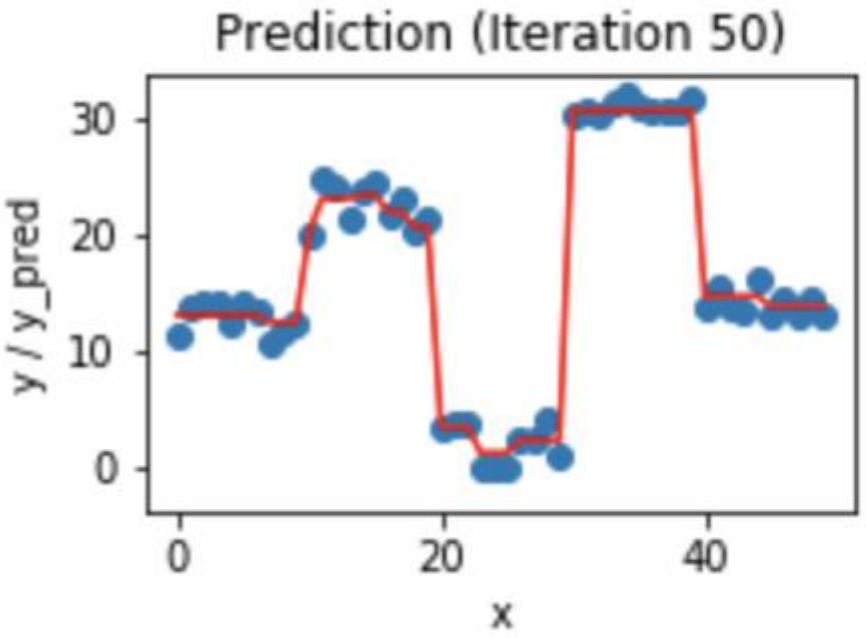


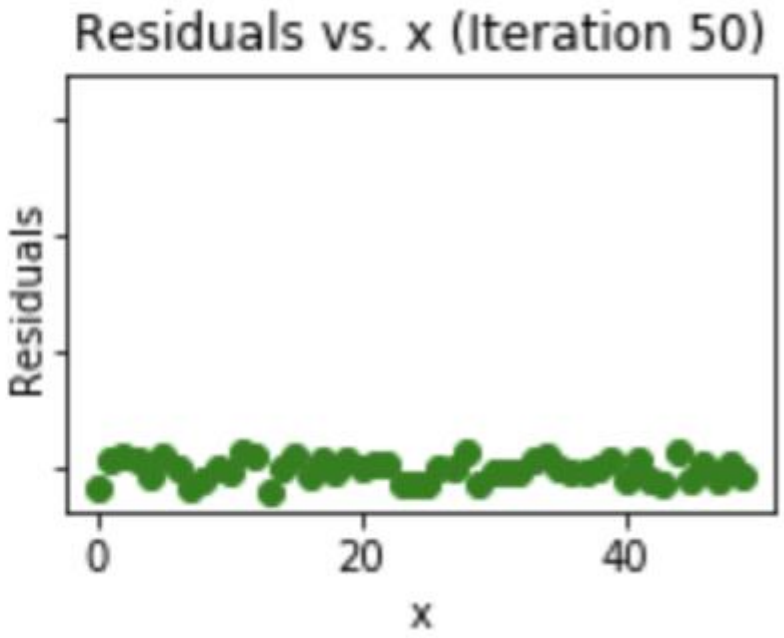


https://medium.com/mlreview/gradient-boosting-from-scratch-1e317ae4587d

Gradient Boosting Machine: GBM

- GBM Regression Example 1

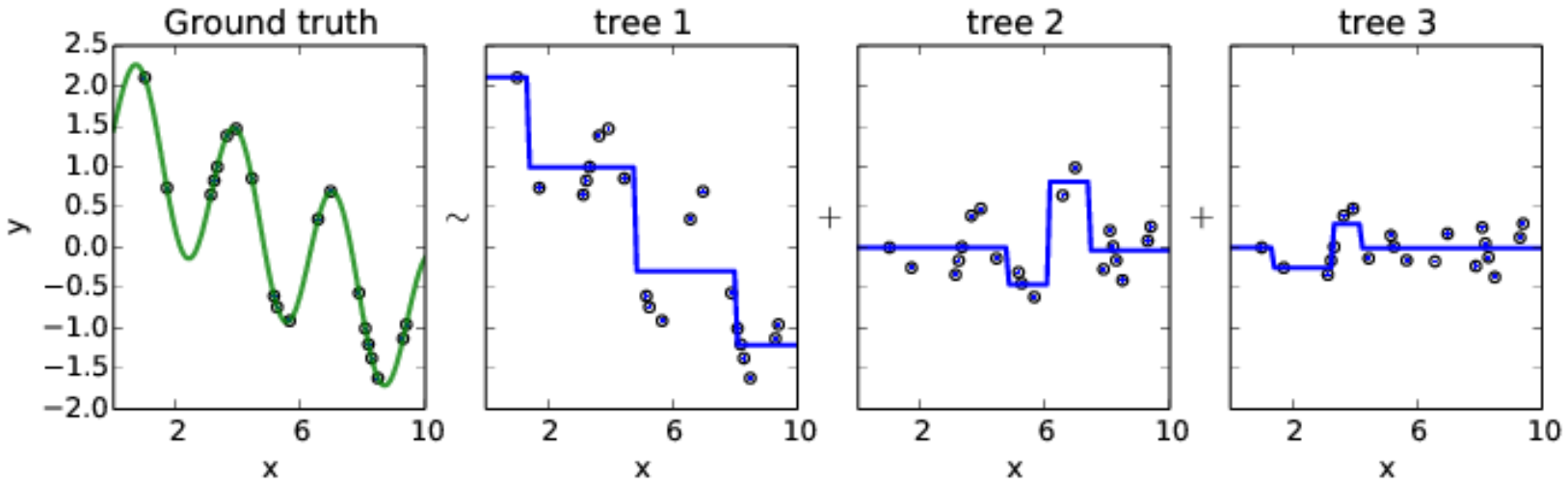




https://medium.com/mlreview/gradient-boosting-from-scratch-1e317ae4587d

Gradient Boosting Machine: GBM

- GBM Regression Example 2



https://www.quora.com/How-would-you-explain-gradient-boosting-machine-learning-technique-in-no-more-than-300-words-to-non-science-major-college-students

Gradient Boosting Machine: GBM

- Gradient Boosting: Algorithm

1. Initialize $f_0(x) = \arg\min_\gamma \sum_{i=1}^{N} L(y_i, \gamma)$.
2. For $m = 1$ to M:
   - 2.1 For $i = 1, \ldots, N$ compute

$$g_{im} = \left[\frac{\partial L(y_i, f(x_i))}{\partial f(x_i)}\right]_{f(x_i)=f_{m-1}(x_i)}$$

   - 2.2 Fit a regression tree to the targets g_{im} giving terminal regions $R_{jm}, j = 1, \ldots, J_m$.
   - 2.3 For $j = 1, \ldots, J_m$ compute

$$\gamma_{jm} = \arg\min_\gamma \sum_{x_i \in R_{jm}} L(y_i, f_{m-1}(x_i) + \gamma)$$

   - 2.4 Update $f_m(x) = f_{m-1}(x) + \sum_{j=1}^{J_m} \gamma_{jm} I(x \in R_{jm})$
3. Output $\hat{f}(x) = f_M(x)$.

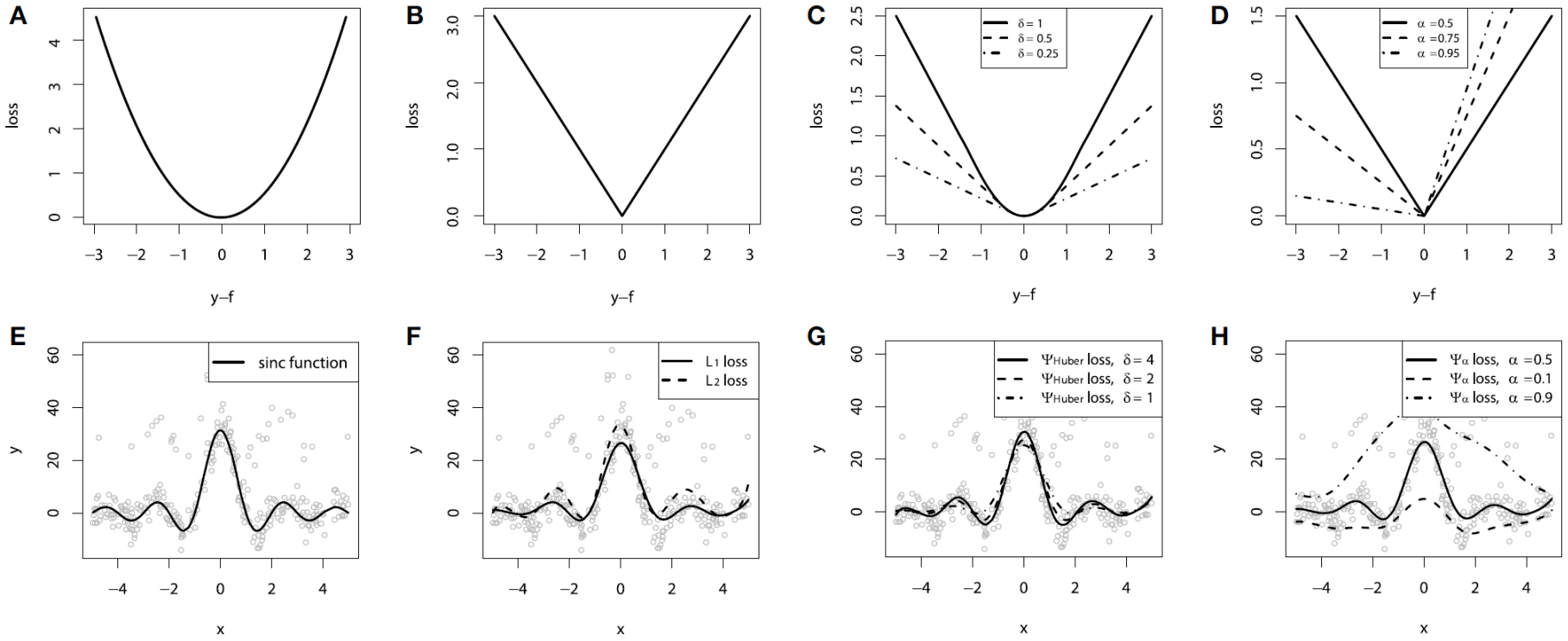
Gradient Boosting Machine: GBM

- Loss Functions for Regression

Loss Function	Formula
Squared loss (L_2)	$\Psi(y,f)_{L_2} = \frac{1}{2}(y-f)^2$
Absolute loss (L_1)	$\Psi(y,f)_{L_1} = \lvert y-f\rvert$
Huber loss	$\Psi(y,f)_{\text{Huber},\,\delta} = \begin{cases} \frac{1}{2}(y-f)^2 & \lvert y-f\rvert \le \delta \\ \delta(\lvert y-f\rvert - \delta/2) & \lvert y-f\rvert > \delta \end{cases}$
Quantile loss	$\Psi(y,f)_{\alpha} = \begin{cases} (1-\alpha)\lvert y-f\rvert & y-f \le 0 \\ \alpha\lvert y-f\rvert & y-f > 0 \end{cases}$

Gradient Boosting Machine: GBM

- Loss Functions for Regression



Gradient Boosting Machine: GBM

- Loss Functions for Classification

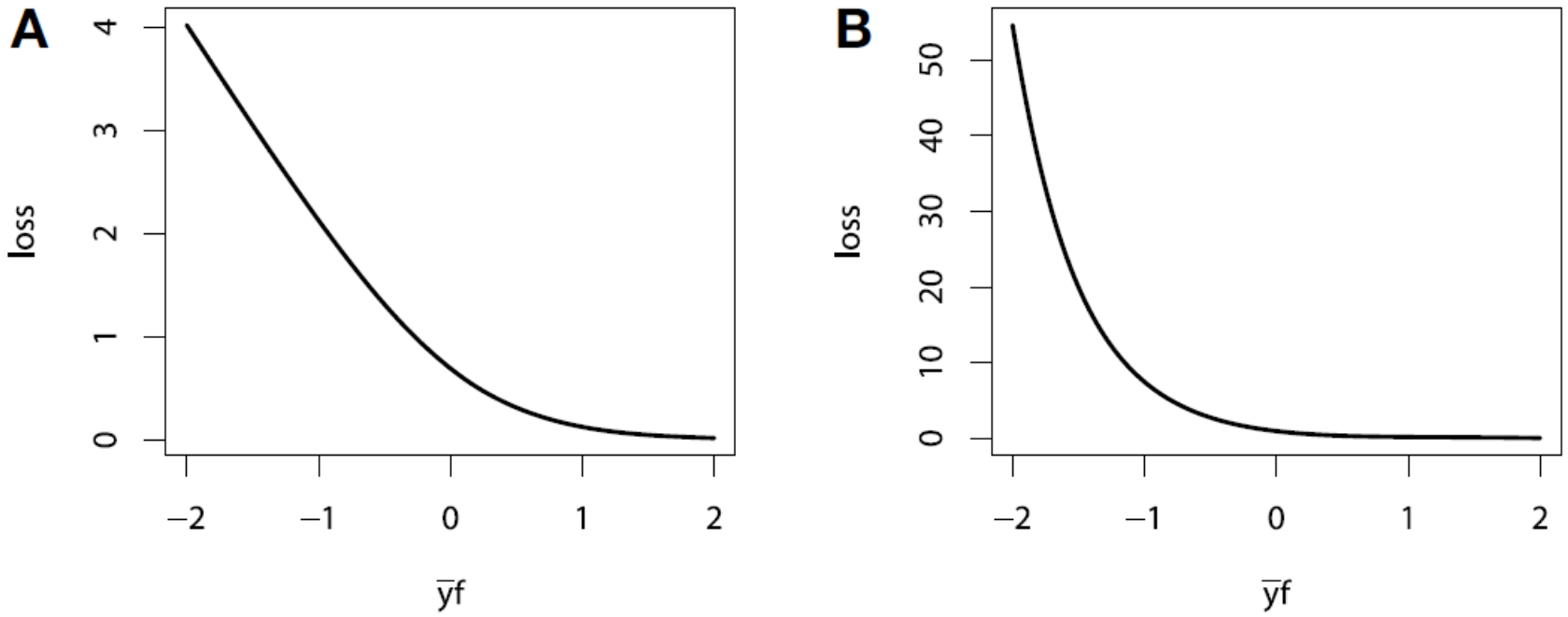
Loss Function	Formula
Bernoulli loss	$\Psi(y,f)_{\text{Bern}} = \log(1+\exp(-2\bar{y}f))$
Adaboost loss	$\Psi(y,f)_{\text{Ada}} = \exp(-\bar{y}f)$

(Note)

In binary classification, the target is usually defined by $y \in \{0,1\}$, but here we define $\bar{y} = 2y - 1$ so that $\bar{y} \in \{-1,1\}$

Gradient Boosting Machine: GBM

- Loss Functions for Classification



(A) Bernoulli loss function. **(B)** Adaboost loss function.

Gradient Boosting Machine: GBM

- Overfitting problem in GBM

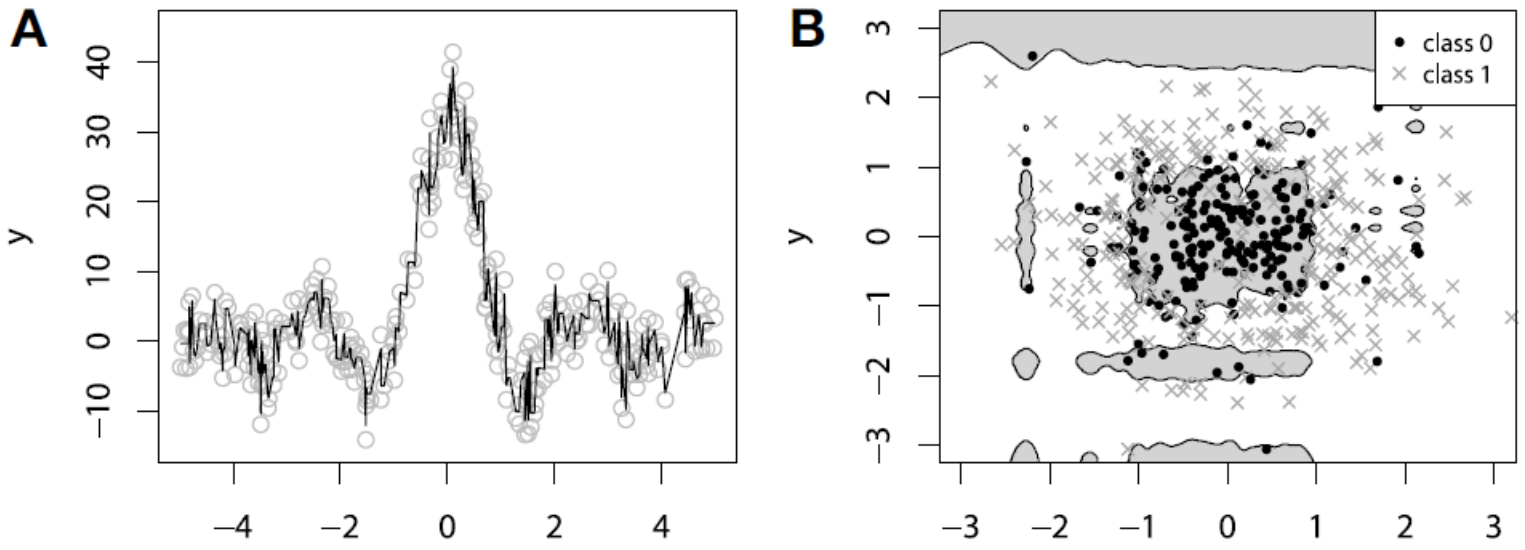


FIGURE 4 | Examples of overfitting in GBMs on: (A) regression task; (B) classification task. Demonstration of fitting a decision-tree GBM to a noisy sinc(x) data: **(C)** $M = 100$, $\lambda = 1$; **(D)** $M = 1000$, $\lambda = 1$; **(E)** $M = 100$, $\lambda = 0.1$; **(F)** $M = 1000$, $\lambda = 0.1$.

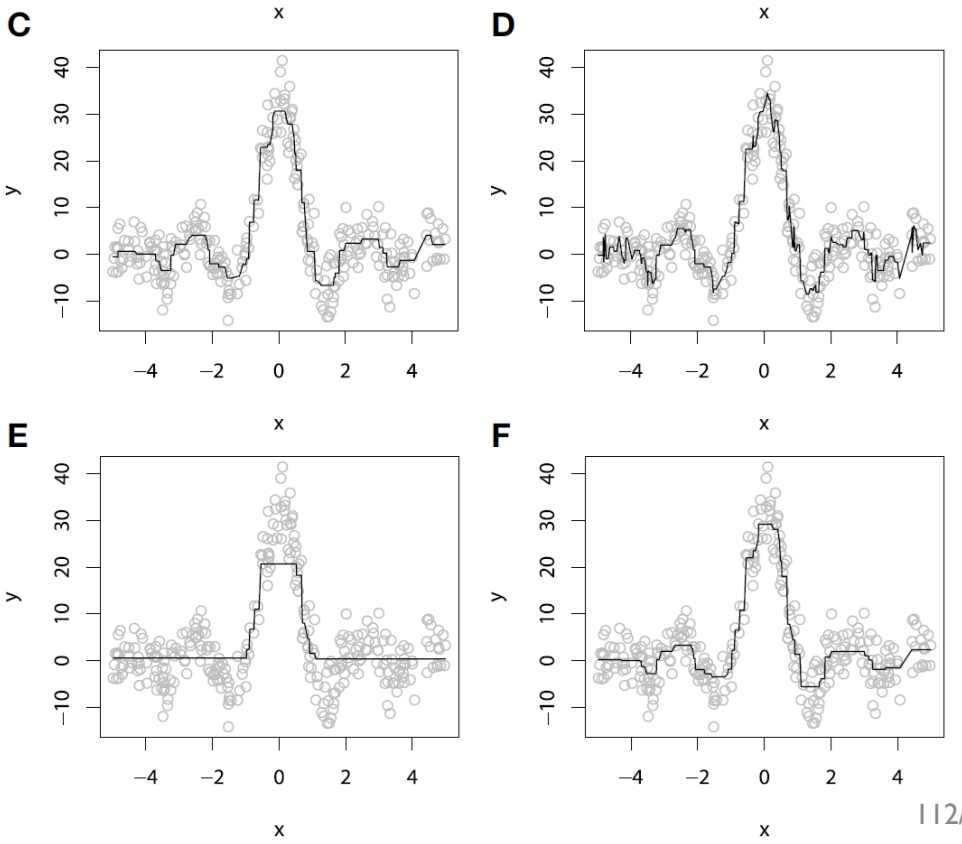
Gradient Boosting Machine: GBM

- Regularization
  - ✓ Subsampling
    - At each learning iteration, only a random part of the training data is used to fit a consecutive base-learner.
    - The training data is typically sampled without replacement, but bagging can be also acceptable.

Gradient Boosting Machine: GBM

- Regularization
  - ✓ Shrinkage
    - Used for reducing/shrinking the impact of each additional fitted base-leaners.
    - Better to improve a model by taking many small steps than by taking fewer large steps.

$$\widehat{f}_t \leftarrow \widehat{f}_{t-1} + \lambda \rho_t h(x, \theta_t)$$



Gradient Boosting Machine: GBM

- Regularization
  - ✓ Early Stopping
    - Use the validation error

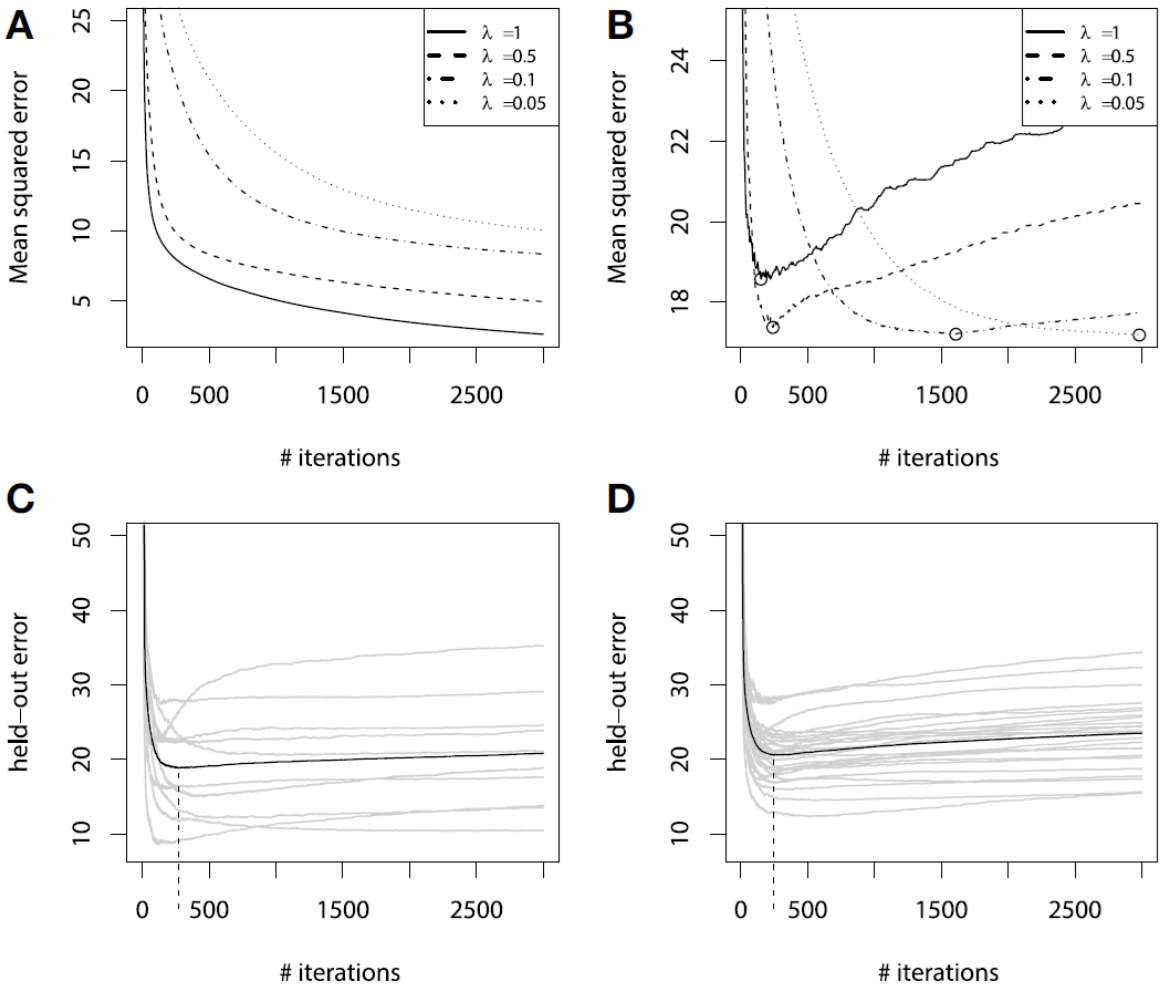


FIGURE 5 | Error curves for GBM fitting on sinc(x) data: (A) training set error; (B) validation set error. Error curves for learning simulations and number of base-learners *M* estimation: **(C)** error curves for cross-validation; **(D)** error curves for bootstrap estimates.

Gradient Boosting Machine: GBM

- Variable Importance in Tree-based Gradient Boosting
  - ✓ $Influence_j(T)$: importance of the variable j in a single tree T.
  - ✓ Assume that there are L terminal nodes → $L-1$ splits.

$$Influence_j(T) = \sum_{i=1}^{L-1} (IG_i \times \mathbf{1}(S_i = j))$$

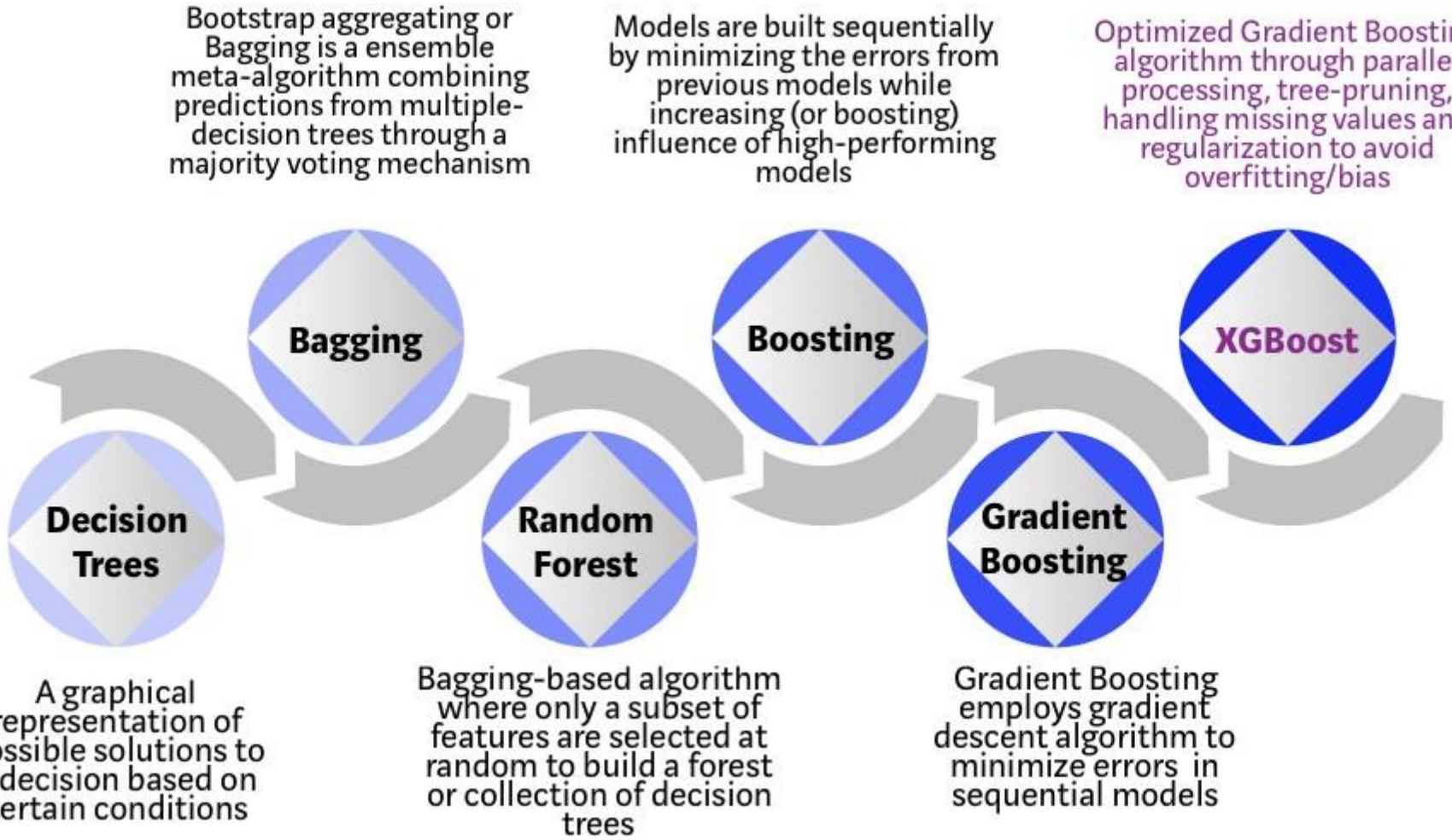
  - ✓ Variable importance of Gradient boosting

$$Influence_j = \frac{1}{M} \sum_{k=1}^{M} Influence_j(T_k)$$

XGBoost

Chen, T., & Guestrin (2016)

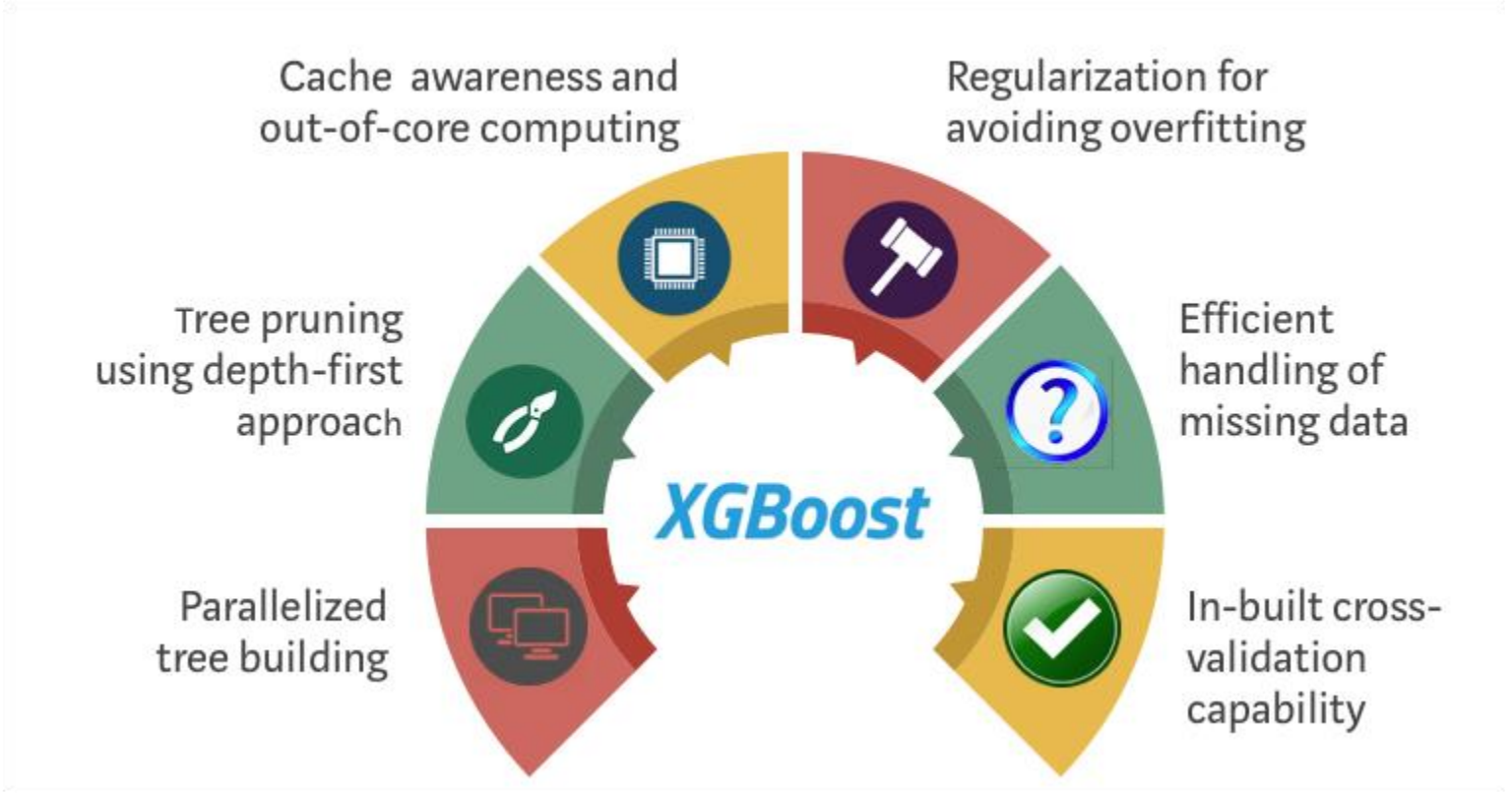
- XGBoost: A Scalable Tree Boosting System



https://towardsdatascience.com/https-medium-com-vishalmorde-xgboost-algorithm-long-she-may-rein-edd9f99be63d

XGBoost

- XGBoost: An optimized version of GBM enabling



https://towardsdatascience.com/https-medium-com-vishalmorde-xgboost-algorithm-long-she-may-rein-edd9f99be63d

XGBoost

- Split Finding Algorithm
  - ✓ Basic exact greedy algorithm vs. Approximate algorithm

Algorithm 1: Exact Greedy Algorithm for Split Finding

Input: I, instance set of current node
Input: d, feature dimension
$gain \leftarrow 0$
$G \leftarrow \sum_{i \in I} g_i$, $H \leftarrow \sum_{i \in I} h_i$
for $k = 1$ ***to*** m **do**
  $G_L \leftarrow 0$, $H_L \leftarrow 0$
  for *j in sorted(I, by* $\mathbf{x}_{jk}$*)* **do**
    $G_L \leftarrow G_L + g_j$, $H_L \leftarrow H_L + h_j$
    $G_R \leftarrow G - G_L$, $H_R \leftarrow H - H_L$
    $score \leftarrow \max(score, \frac{G_L^2}{H_L+\lambda} + \frac{G_R^2}{H_R+\lambda} - \frac{G^2}{H+\lambda})$
  end
end
Output: Split with max score

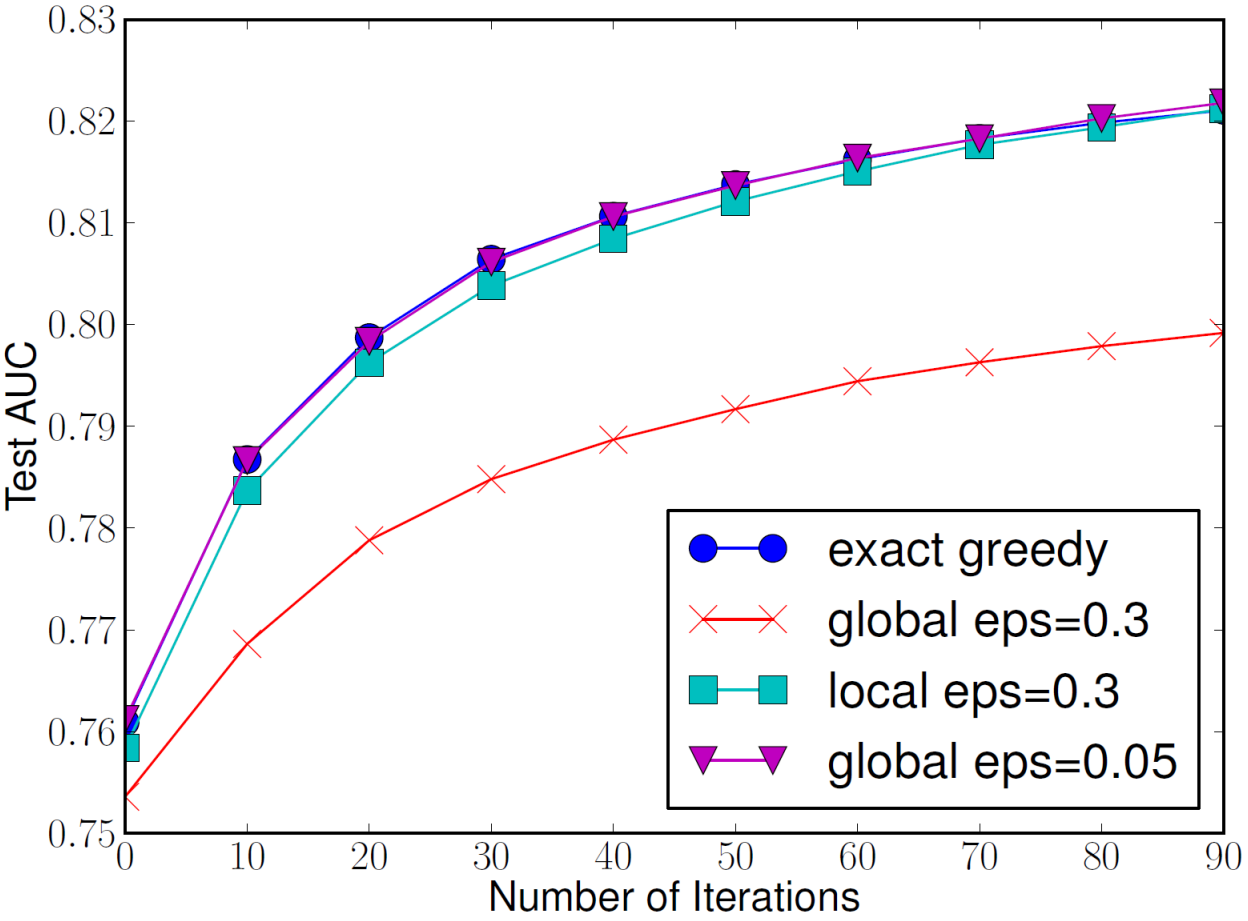
Algorithm 2: Approximate Algorithm for Split Finding

for $k = 1$ ***to*** m **do**
  Propose $S_k = \{s_{k1}, s_{k2}, \cdots s_{kl}\}$ by percentiles on feature k.
  Proposal can be done per tree (global), or per split(local).
end
for $k = 1$ ***to*** m **do**
  $G_{kv} \leftarrow= \sum_{j \in \{j | s_{k,v} \geq \mathbf{x}_{jk} > s_{k,v-1}\}} g_j$
  $H_{kv} \leftarrow= \sum_{j \in \{j | s_{k,v} \geq \mathbf{x}_{jk} > s_{k,v-1}\}} h_j$
end
Follow same step as in previous section to find max score only among proposed splits.

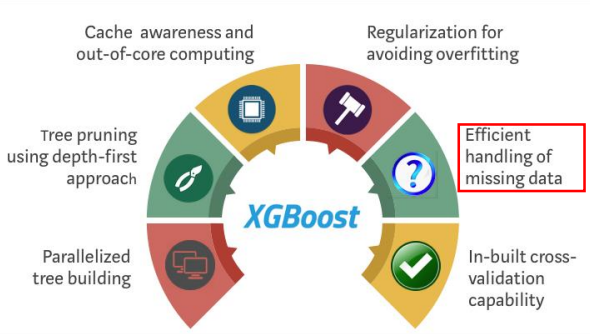
  - ✓ (Note) Read Section 3.3 Weighted Quantile Sketch of the original paper

XGBoost

- Split Finding Algorithm
  - ✓ Approximate algorithm
    - Global proposal vs. Local proposal



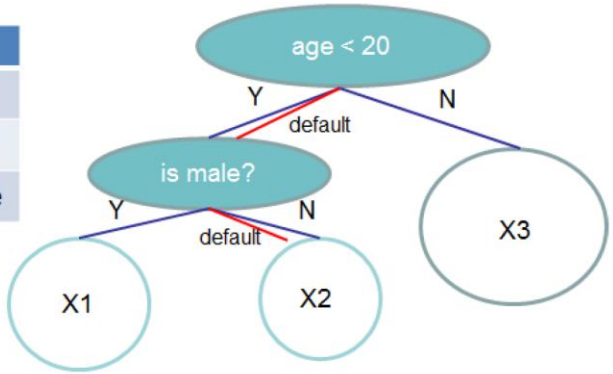
XGBoost

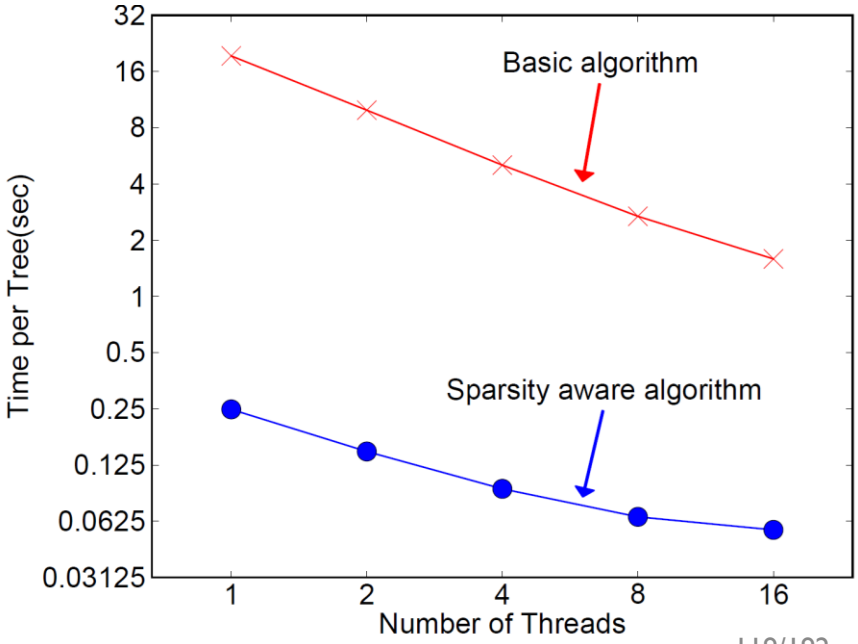


- Sparsity-Aware Split Finding
  - ✓ In many real-world problems, it is quite common for the input x to be sparse
    - presence of missing values in the data
    - frequent zero entries in the statistics
    - artifacts of feature engineering such as one-hot encoding
  - ✓ Solution: Set the default direction that is learned from the data

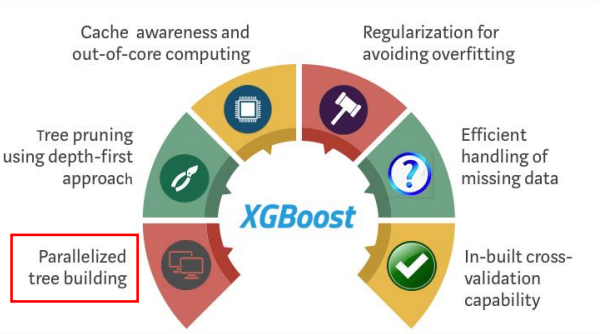
Data

Example	Age	Gender
X1	?	male
X2	15	?
X3	25	female

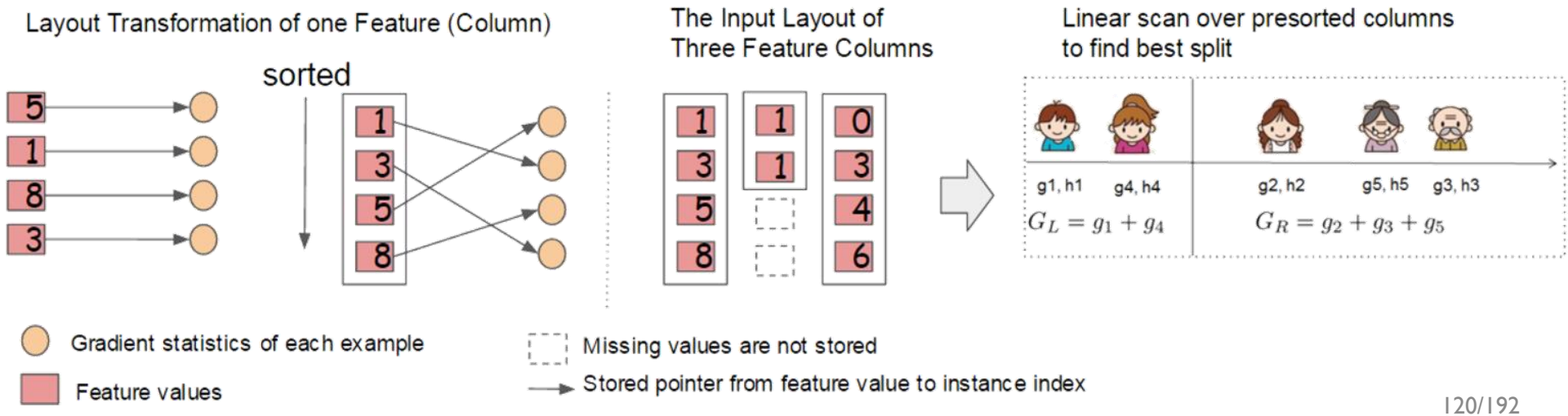




XGBoost

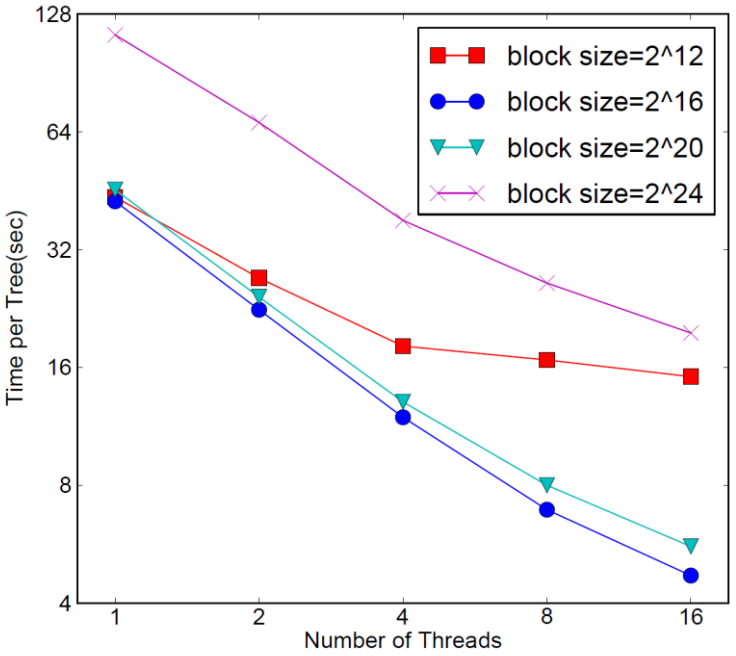


- System Design for Efficient Computing
  - ✓ The most time-consuming part of tree learning
    - to get the data into sorted order
  - ✓ XGBoost propose to store the data in in-memory units called block
    - Data in each block is stored in the compressed column (CSC) format, with each column sorted by the corresponding feature value
    - This input data layout only needs to be computed once before training and can be reused in later iterations.

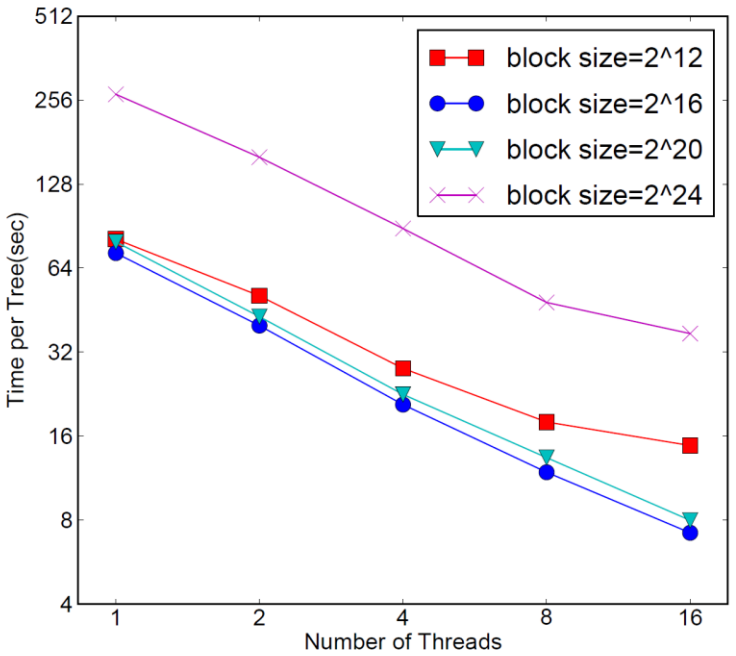


XGBoost

- System Design for Efficient Computing
  - ✓ Cache-aware access
    - For the exact greedy algorithm, we can alleviate the problem by [a cache-aware prefetching algorithm]
    - For approximate algorithms, we solve the problem by [choosing a correct block size]

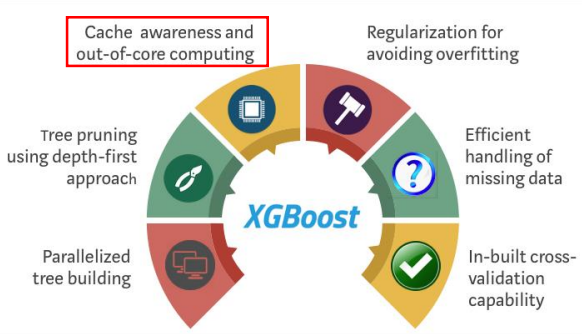


(a) Allstate 10M



(b) Higgs 10M

XGBoost



- System Design for Efficient Computing
  - ✓ Out-of-core computing
    - Besides processors and memory, it is important to utilize disk space to handle data that does not fit into main memory
    - To enable out-of-core computation, the data is divided into multiple blocks and store each block on disk
    - To enable out-of-core computation, we divide the data into multiple blocks and store each block on disk
    - It is important to reduce the overhead and increase the throughput of disk IO
  - ✓ Block Compression
    - The block is compressed by columns and decompressed on the fly by an independent thread when loading into main memory

XGBoost

- System Design for Efficient Computing
  - ✓ Block Compression
    - The block is compressed by columns and decompressed on the fly by an independent thread when loading into main memory
    - This helps to trade some of the computation in decompression with the disk reading cost
  - ✓ Block Sharding
    - A pre-fetcher thread is assigned to each disk and fetches the data into an in-memory buffer
    - The training thread then alternatively reads the data from each buffer
    - This helps to increase the throughput of disk reading when multiple disks are available

Table 1: Comparison of major tree boosting systems.

System	exact greedy	approximate global	approximate local	out-of-core	sparsity aware	parallel
XGBoost	yes	yes	yes	yes	yes	yes
pGBRT	no	no	yes	no	no	yes
Spark MLLib	no	yes	no	no	partially	yes
H2O	no	yes	no	no	partially	yes
scikit-learn	yes	no	no	no	no	no
R GBM	yes	no	no	no	partially	no

XGBoost

- Experiments

Table 2: Dataset used in the Experiments.

Dataset	n	m	Task
Allstate	10 M	4227	Insurance claim classification
Higgs Boson	10 M	28	Event classification
Yahoo LTRC	473K	700	Learning to Rank
Criteo	1.7 B	67	Click through rate prediction

Table 3: Comparison of Exact Greedy Methods with 500 trees on Higgs-1M data.

Method	Time per Tree (sec)	Test AUC
XGBoost	0.6841	0.8304
XGBoost (colsample=0.5)	0.6401	0.8245
scikit-learn	28.51	0.8302
R.gbm	1.032	0.6224

Table 4: Comparison of Learning to Rank with 500 trees on Yahoo! LTRC Dataset

Method	Time per Tree (sec)	NDCG@10
XGBoost	0.826	0.7892
XGBoost (colsample=0.5)	0.506	0.7913
pGBRT [22]	2.576	0.7915

XGBoost

- Experiments

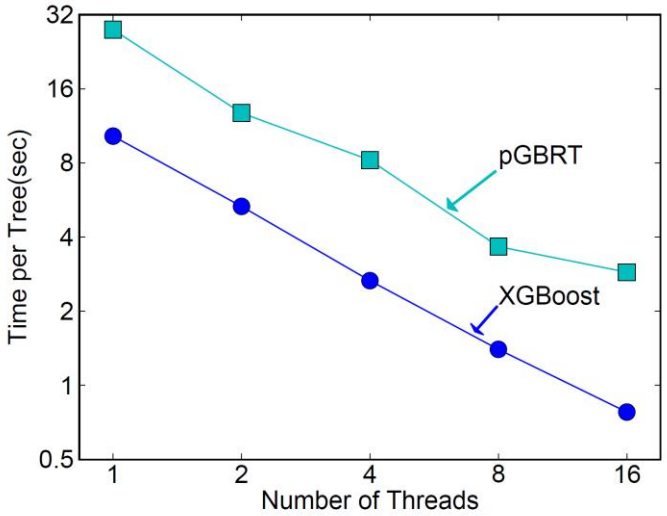


Figure 10: Comparison between XGBoost and pGBRT on Yahoo LTRC dataset.

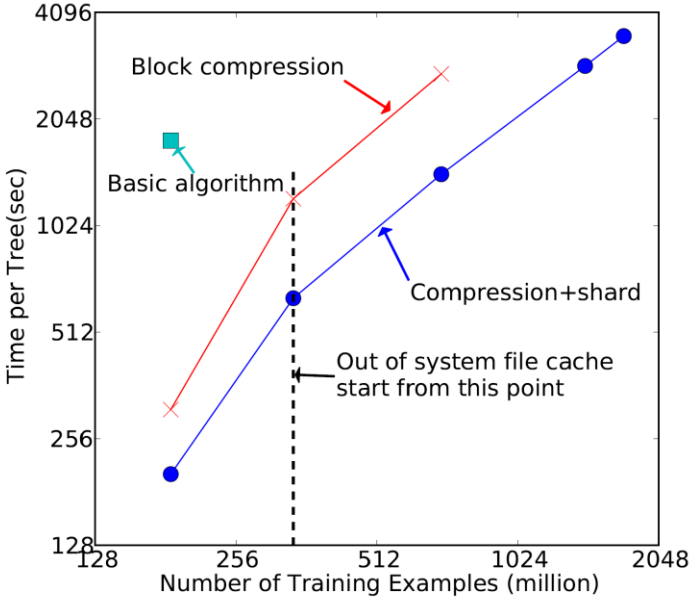
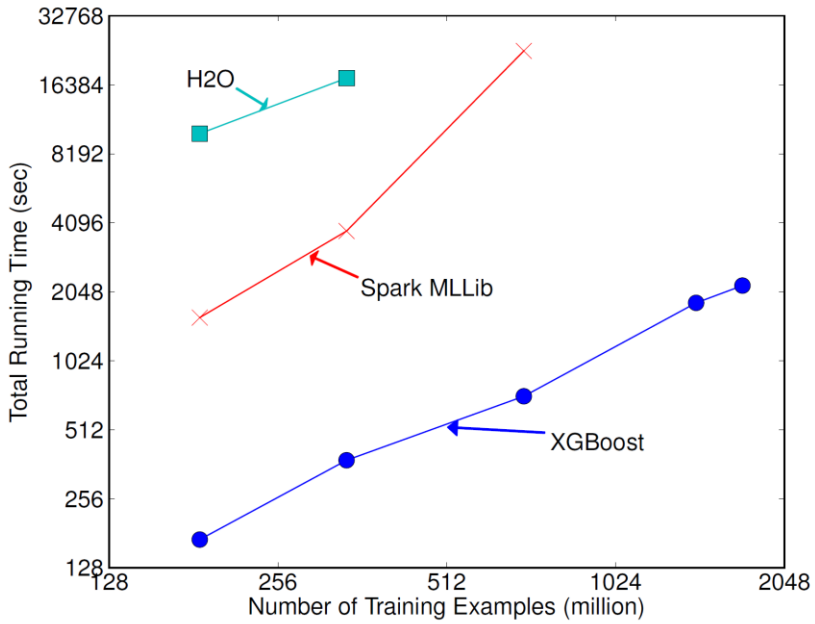


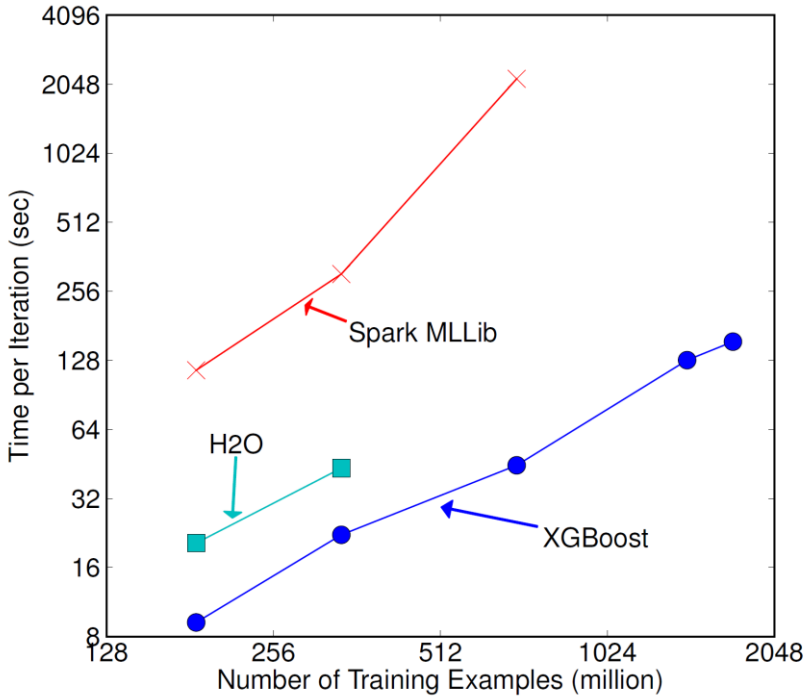
Figure 11: Comparison of out-of-core methods

XGBoost

- Experiments



(a) End-to-end time cost include data loading



(b) Per iteration cost exclude data loading

Light GBM

Ke et al. (2017)

- Motivation
  - ✓ Conventional GBM need to, for every feature, scan all the data instances to estimate the information gain of all the possible split points
- Idea
  - ✓ To reduce the number of data instances and the number of features
    - Gradient-based One-Side Sampling (GOSS)
      - Data instances with different gradients play different roles in the computation of information gain
      - Keep instances with large gradients and randomly drop instances with small gradients
    - Exclusive Feature Bundling (EFB)
      - In a sparse feature space, many features are (almost) exclusive, i.e., they rarely take nonzero values simultaneously (ex: one-hot encoding)
      - Bundling these exclusive features does not degenerate the performance

Light GBM

- Gradient-based One-sided Sampling (GOSS)

XGBoost

Algorithm 1: Histogram-based Algorithm

Input: I: training data, d: max depth
Input: m: feature dimension
$nodeSet \leftarrow \{0\}$ ▷ tree nodes in current level
$rowSet \leftarrow \{\{0, 1, 2, ...\}\}$ ▷ data indices in tree nodes
for $i = 1$ ***to*** d **do**
  for $node$ ***in*** $nodeSet$ **do**
    usedRows $\leftarrow rowSet[node]$
    for $k = 1$ ***to*** m **do**
      $H \leftarrow$ new Histogram()
      ▷ Build histogram
      for j ***in*** *usedRows* **do**
        bin $\leftarrow I$.f[k][j].bin
        H[bin].y $\leftarrow H$[bin].y + I.y[j]
        H[bin].n $\leftarrow H$[bin].n + 1
      Find the best split on histogram H.
      ...
  Update $rowSet$ and $nodeSet$ according to the best split points.
  ...

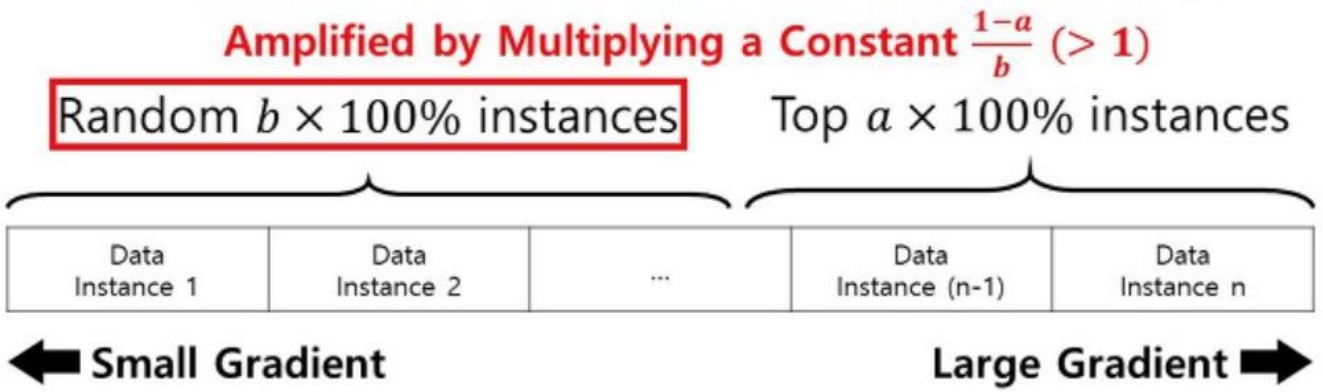
LightGBM

Algorithm 2: Gradient-based One-Side Sampling

Input: I: training data, d: iterations
Input: a: sampling ratio of large gradient data
Input: b: sampling ratio of small gradient data
Input: $loss$: loss function, L: weak learner
models $\leftarrow$ { }, fact $\leftarrow \frac{1-a}{b}$
topN $\leftarrow$ a $\times$ len(I) , randN $\leftarrow$ b $\times$ len(I)
for $i = 1$ ***to*** d **do**
  preds $\leftarrow$ models.predict(I)
  g $\leftarrow loss(I,$ preds), w $\leftarrow$ {1,1,...}
  sorted $\leftarrow$ GetSortedIndices(abs(g))
  topSet $\leftarrow$ sorted[1:topN]
  randSet $\leftarrow$ RandomPick(sorted[topN:len(I)], randN)
  usedSet $\leftarrow$ topSet + randSet
  w[randSet] $\times =$ fact ▷ Assign weight $fact$ to the small gradient data.
  newModel $\leftarrow$ L(I[usedSet], $-$ g[usedSet], w[usedSet])
  models.append(newModel)

Light GBM

- Gradient-based One-sided Sampling (GOSS)



https://cdm98.tistory.com/m/31

Light GBM

- Exclusive Feature Bundling (EFB)

Algorithm 3: Greedy Bundling

Input: F: features, K: max conflict count
Construct graph G
searchOrder $\leftarrow G$.sortByDegree()
bundles $\leftarrow \{\}$, bundlesConflict $\leftarrow \{\}$
for i ***in*** *searchOrder* **do**
    needNew $\leftarrow$ True
    for $j = 1$ ***to*** *len(bundles)* **do**
        cnt $\leftarrow$ ConflictCnt(bundles[j],F[i])
        if *cnt* + *bundlesConflict[i]* $\leq K$ **then**
            bundles[j].add(F[i]), needNew $\leftarrow$ False
            break
    if *needNew* **then**
        Add $F[i]$ as a new bundle to $bundles$
Output: $bundles$

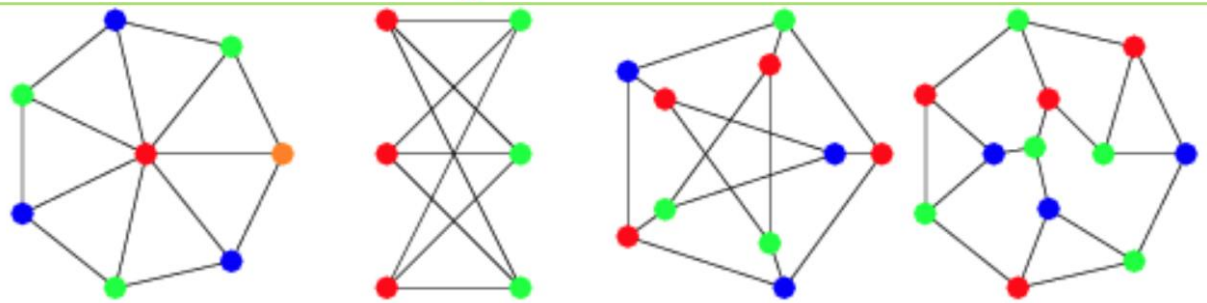
Algorithm 4: Merge Exclusive Features

Input: $numData$: number of data
Input: F: One bundle of exclusive features
binRanges $\leftarrow \{0\}$, totalBin $\leftarrow 0$
for f ***in*** F **do**
    totalBin += f.numBin
    binRanges.append(totalBin)
newBin $\leftarrow$ new Bin(numData)
for $i = 1$ ***to*** $numData$ **do**
    newBin[i] $\leftarrow 0$
    for $j = 1$ ***to*** *len*(F) **do**
        if F*[j].bin[i]* $\neq 0$ **then**
            newBin[i] $\leftarrow$ F[j].bin[i] + binRanges[j]
Output: $newBin, binRanges$

Light GBM

- Exclusive Feature Bundling (EFB)
  - ✓ Can be formulated as a Graph coloring problem
    - Construct a Graph (V, E)
      - V: feature
      - E: total conflicts between features

Minimum Vertex Coloring



Light GBM

- Exclusive Feature Bundling (EFB)
  - ✓ Greedy bundling example

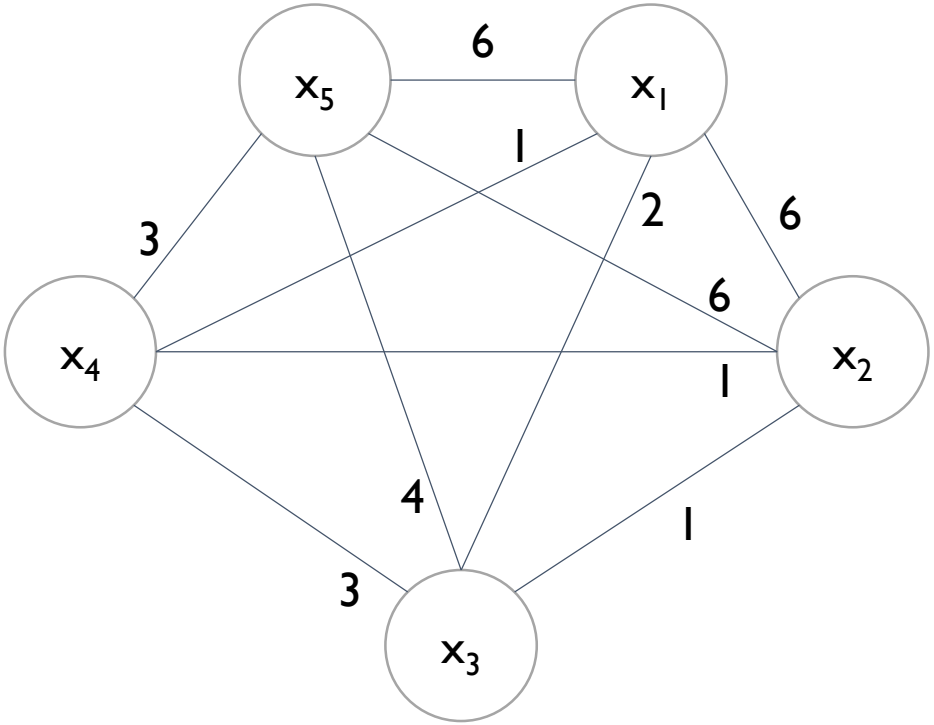
	x_1	x_2	x_3	x_4	x_5
I_1	1	1	0	0	1
I_2	0	0	1	1	1
I_3	1	2	0	0	2
I_4	0	0	2	3	1
I_5	2	1	0	0	3
I_6	3	3	0	0	1
I_7	0	0	3	0	2
I_8	1	2	3	4	3
I_9	1	0	1	0	0
I_{10}	2	3	0	0	2

	x_1	x_2	x_3	x_4	x_5
x_1	-	6	2	1	6
x_2	6	-	1	1	6
x_3	2	1	-	3	4
x_4	1	1	3	-	3
x_5	6	6	4	3	-

	x_5	x_1	x_2	x_3	x_4
d	19	15	14	10	8

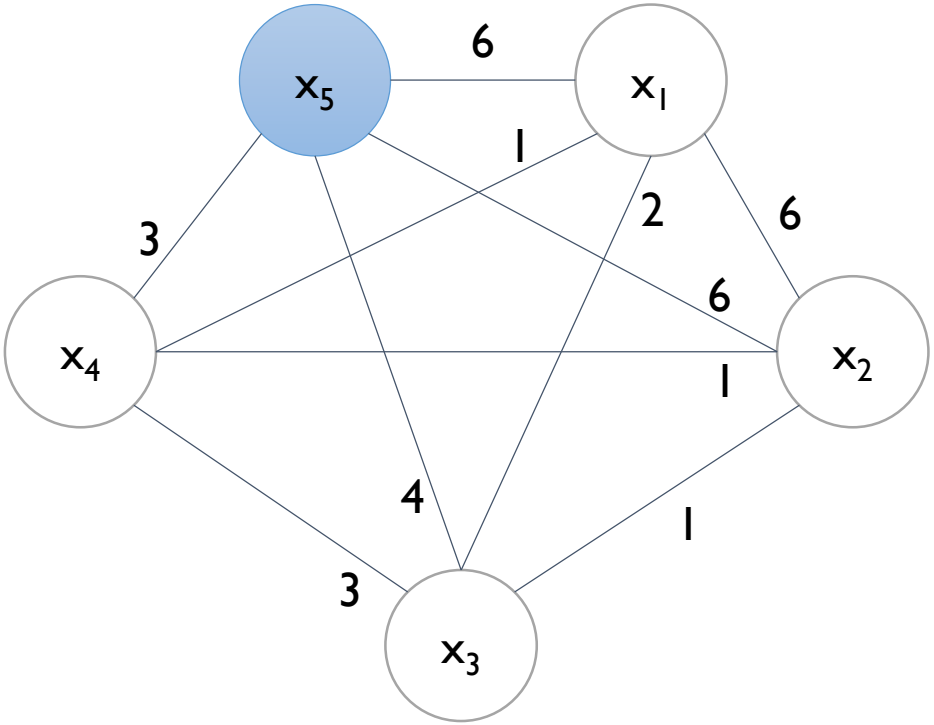
Light GBM

- Exclusive Feature Bundling (EFB)
  - ✓ Greedy bundling example (cut-off = 0.2)



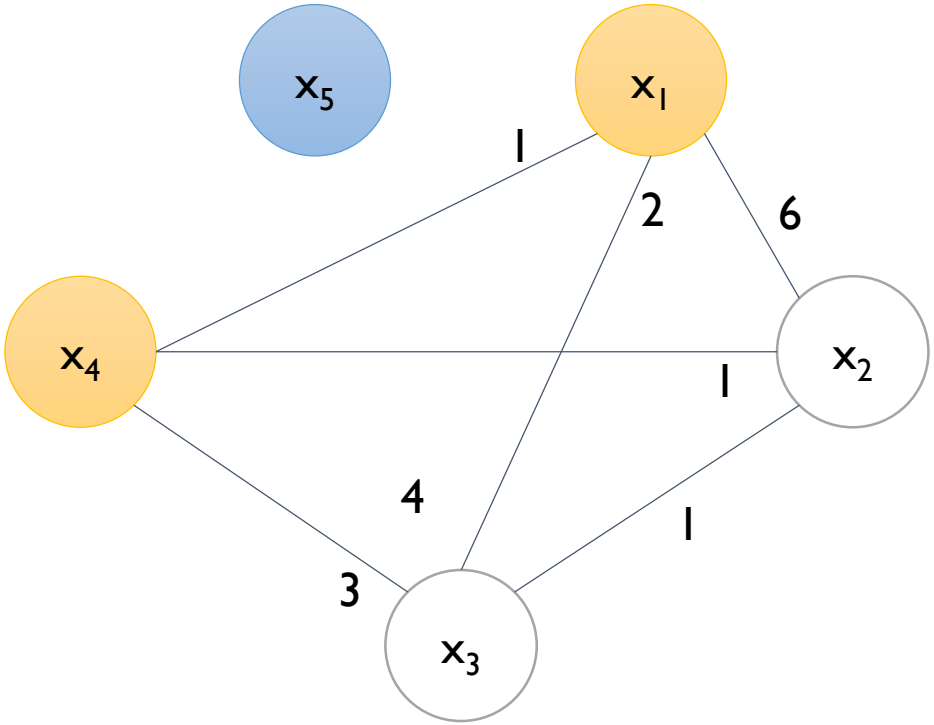
Light GBM

- Exclusive Feature Bundling (EFB)
  - ✓ Greedy bundling example (cut-off = 0.2)



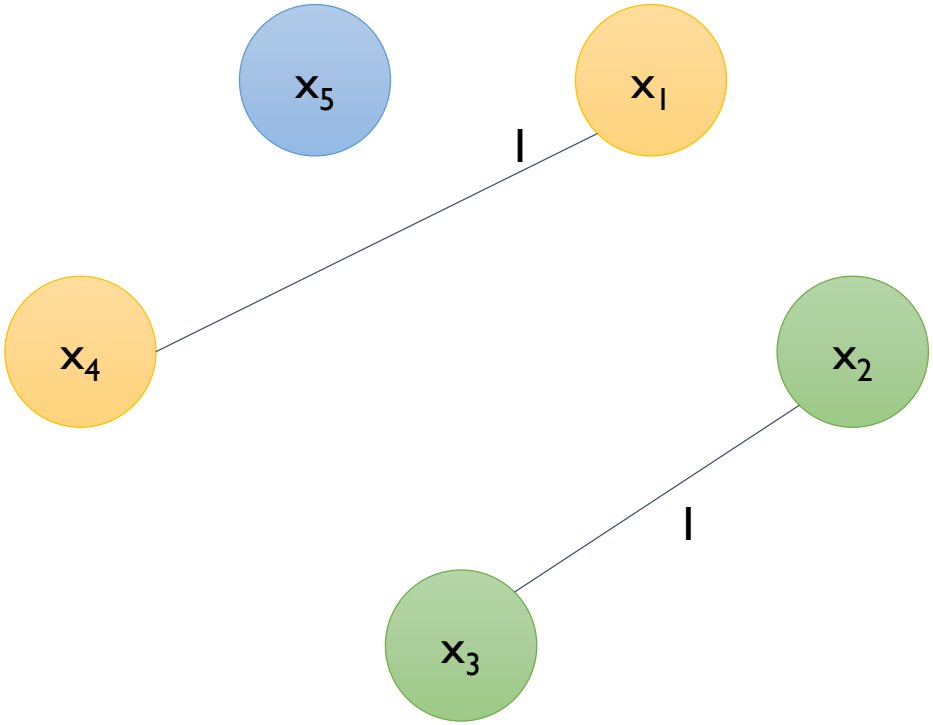
Light GBM

- Exclusive Feature Bundling (EFB)
  - ✓ Greedy bundling example (cut-off = 0.2)



Light GBM

- Exclusive Feature Bundling (EFB)
  - ✓ Greedy bundling example (cut-off = 0.2)



Light GBM

- Exclusive Feature Bundling (EFB)
  - ✓ Greedy bundling example (cut-off = 0.2)

	x_1	x_2	x_3	x_4	x_5
I_1	1	1	0	0	1
I_2	0	0	1	1	1
I_3	1	2	0	0	2
I_4	0	0	2	3	1
I_5	2	1	0	0	3
I_6	3	3	0	0	1
I_7	0	0	3	0	2
I_8	1	2	3	4	3
I_9	1	0	1	0	0
I_{10}	2	3	0	0	2

	x_5	x_1	x_4	x_2	x_3
I_1	1	1	0	1	0
I_2	1	0	1	0	1
I_3	2	1	0	2	0
I_4	1	0	3	0	2
I_5	3	2	0	1	0
I_6	1	3	0	3	0
I_7	2	0	0	0	3
I_8	3	1	4	2	3
I_9	0	1	0	0	1
I_{10}	2	2	0	3	0

Light GBM

- Exclusive Feature Bundling (EFB)
  - ✓ Exclusive feature merging
    - Add offsets to the original values of the features

	x_5	x_1	x_4	x_2	x_3
I_1	1	1	0	1	0
I_2	1	0	1	0	1
I_3	2	1	0	2	0
I_4	1	0	3	0	2
I_5	3	2	0	1	0
I_6	1	3	0	3	0
I_7	2	0	0	0	3
I_8	3	1	4	2	3
I_9	0	1	0	0	1
I_{10}	2	2	0	3	0

	x_5	x_{14}	x_{23}
I_1	1	1	1
I_2	1	4	4
I_3	2	1	2
I_4	1	6	5
I_5	3	2	1
I_6	1	3	3
I_7	2	0	6
I_8	3	1	2
I_9	0	1	4
I_{10}	2	2	3

Add the offset 3 to the nonzero values of x_4

Add the offset 3 to the nonzero values of x_3

Conflict: Use the value of x_1

Conflict: Use the value of x_2

Light GBM

- Experiments
  - ✓ Dataset description

Table 1: Datasets used in the experiments.

Name	$\#data$	$\#feature$	Description	Task	Metric
Allstate	12 M	4228	Sparse	Binary classification	AUC
Flight Delay	10 M	700	Sparse	Binary classification	AUC
LETOR	2M	136	Dense	Ranking	NDCG [4]
KDD10	19M	29M	Sparse	Binary classification	AUC
KDD12	119M	54M	Sparse	Binary classification	AUC

  - ✓ Training time

	xgb_exa	xgb_his	lgb_baseline	EFB_only	**LightGBM**
Allstate	10.85	2.63	6.07	0.71	**0.28**
Flight Delay	5.94	1.05	1.39	0.27	**0.22**
LETOR	5.55	0.63	0.49	0.46	**0.31**
KDD10	108.27	OOM	39.85	6.33	**2.85**
KDD12	191.99	OOM	168.26	20.23	**12.67**

Light GBM

- Experiments
  - ✓ Overall accuracy

	xgb_exa	xgb_his	lgb_baseline	SGB	**LightGBM**
Allstate	0.6070	0.6089	0.6093	0.6064 ± 7e-4	**0.6093 ± 9e-5**
Flight Delay	0.7601	0.7840	0.7847	0.7780 ± 8e-4	**0.7846 ± 4e-5**
LETOR	0.4977	0.4982	0.5277	0.5239 ± 6e-4	**0.5275 ± 5e-4**
KDD10	0.7796	OOM	0.78735	0.7759 ± 3e-4	**0.78732 ± 1e-4**
KDD12	0.7029	OOM	0.7049	0.6989 ± 8e-4	**0.7051 ± 5e-5**

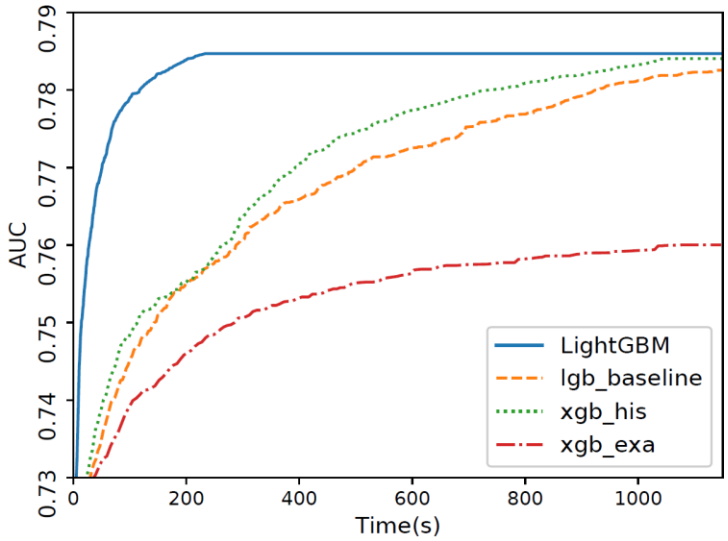


Figure 1: Time-AUC curve on Flight Delay.

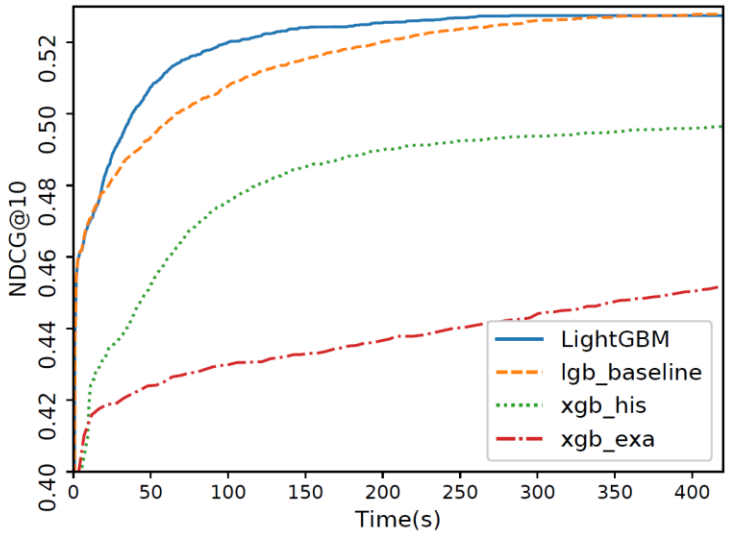


Figure 2: Time-NDCG curve on LETOR.

CatBoost

Prokhorenkova et al. (2018)

- Background in brief
  - ✓ Assume we observe a dataset of examples $\mathcal{D} = \{(\mathbf{x}_k, y_k)\}_{k=1,...,n}$
    - where $\mathbf{x}_k = (x_k^1, ..., x_k^m)$ is a random vector of m features and $y_k \in \mathbb{R}$ is a target
  - ✓ Gradient boosting builds iteratively a sequence of approximations $F^t : \mathbb{R}^m \to \mathbb{R}$ in a greedy fashion
    - $F^t : F^{t-1} + \alpha h^t$ (alpha is a step size and h is a base predictor)
    - h^t is chosen from a family of functions *H* in order to minimize the expected loss

$$h^t = \arg\min_{h \in H} \mathcal{L}(F^{t-1} + h) = \arg\min_{h \in H} \mathbb{E}L(y, F^{t-1}(\mathbf{x}) + h(\mathbf{x}))$$

    - The gradient step h^t is chosen in such a way that $h^t(\mathbf{x})$ approximates $-g^t(\mathbf{x}, y)$, where $g^t(\mathbf{x}, y) := \left.\frac{\partial L(y, s)}{\partial s}\right|_{s=F^{t-1}(\mathbf{x})}$
    - Usually the least squares approximation is used

$$h^t = \arg\min_{h \in H} \mathbb{E}\big(- g^t(\mathbf{x}, y) - h(\mathbf{x})\big)^2$$

CatBoost

March, 2014

XGBoost initially started as research project by Tianqi Chen but it actually became famous in 2016

Jan, 2017

Microsoft released first stable version of LightGBM

April, 2017

Yandex, one of Russia's leading tech companies open sources CatBoost

https://towardsdatascience.com/lightgbm-vs-xgboost-which-algorithm-win-the-race-1ff7dd4917d

Features

1  **Great quality without parameter tuning**

Reduce time spent on parameter tuning, because CatBoost provides great results with default parameters

2 

Categorical features support

Improve your training results with CatBoost that allows you to use non-numeric factors, instead of having to pre-process your data or spend time and effort turning it to numbers.

https://catboost.ai/

3  **Fast and scalable GPU version**

Train your model on a fast implementation of gradient-boosting algorithm for GPU. Use a multi-card configuration for large datasets.

4  **Improved accuracy**

Reduce overfitting when constructing your models with a novel gradient-boosting scheme.

5  **Fast prediction**

Apply your trained model quickly and efficiently even to latency-critical tasks using CatBoost's model applier

CatBoost

CatBoost

- Categorical features in boosted tree
  - ✓ Basic approach: one-hot encoding

	…	x^i	…
I_1	…	A	…
I_2	…	B	…
I_3	…	C	…
I_4	…	A	…
I_5	…	B	…
I_6	…	C	…
I_7	…	B	…
I_8	…	C	…
I_9	…	C	…
I_{10}	…	C	…

	…	$x^i(A)$	$x^i(B)$	$x^i(C)$	…
I_1	…	1	0	0	…
I_2	…	0	1	0	…
I_3	…	0	0	1	…
I_4	…	1	0	0	…
I_5	…	0	1	0	…
I_6	…	0	0	1	…
I_7	…	0	1	0	…
I_8	…	0	0	1	…
I_9	…	0	0	1	…
I_{10}	…	0	0	1	…

CatBoost

- Categorical features in boosted tree
  - ✓ Another popular method: to group categories by target statistics (TS)
    - an estimate expected target value in each category
    - Greedy TS without smoothing

$$\hat{x}_k^i \approx \mathbb{E}(y \mid x^i = x_k^i)$$

    - i is a categorical feature while k is training example index

	…	x^i	…	y
I_1	…	A	…	1
I_2	…	B	…	1
I_3	…	C	…	1
I_4	…	A	…	0
I_5	…	B	…	1
I_6	…	C	…	1
I_7	…	B	…	0
I_8	…	C	…	1
I_9	…	C	…	1
I_{10}	…	C	…	0

	…	x^i(TS)	…	y
I_1	…	0.50	…	1
I_2	…	0.67	…	1
I_3	…	0.80	…	1
I_4	…	0.50	…	0
I_5	…	0.67	…	1
I_6	…	0.80	…	1
I_7	…	0.67	…	0
I_8	…	0.80	…	1
I_9	…	0.80	…	1
I_{10}	…	0.80	…	0

CatBoost

- Categorical features in boosted tree
  - ✓ Another popular method: to group categories by target statistics (TS)
    - Greedy TS with smoothing

$$\hat{x}_k^i = \frac{\sum_{j=1}^{n} \mathbb{1}_{\{x_j^i = x_k^i\}} \cdot y_j + ap}{\sum_{j=1}^{n} \mathbb{1}_{\{x_j^i = x_k^i\}} + a}$$

      - a > 0 is a parameter
      - A common setting for p is the average target value in the dataset

CatBoost

- Greedy TS with smoothing example
  - ✓ a = 0.1 (parameter) , p = 0.7 (computed from the training dataset)
  - ✓ For category A

	…	x^i	…	y
I_1	…	A	…	1
I_2	…	B	…	1
I_3	…	C	…	1
I_4	…	A	…	0
I_5	…	B	…	1
I_6	…	C	…	1
I_7	…	B	…	0
I_8	…	C	…	1
I_9	…	C	…	0
I_{10}	…	C	…	1

$$\hat{x}_k^A = \frac{\sum_{j=1}^{n} \mathbb{1}_{\{x_j^A = x_k^A\}} \cdot y_j + ap}{\sum_{j=1}^{n} \mathbb{1}_{\{x_j^A = x_k^A\}} + a}$$

$$= \frac{1 + 0.1 \times 0.7}{2 + 0.1} = 0.5095$$

CatBoost

- Greedy TS with smoothing example
  - ✓ a = 0.1 (parameter) , p = 0.7 (computed from the training dataset)
  - ✓ For category B

	…	x^i	…	y
I_1	…	A	…	1
I_2	…	B	…	1
I_3	…	C	…	1
I_4	…	A	…	0
I_5	…	B	…	1
I_6	…	C	…	1
I_7	…	B	…	0
I_8	…	C	…	1
I_9	…	C	…	0
I_{10}	…	C	…	1

$$\hat{x}_k^B = \frac{\sum_{j=1}^{n} \mathbb{1}_{\{x_j^B = x_k^B\}} \cdot y_j + ap}{\sum_{j=1}^{n} \mathbb{1}_{\{x_j^B = x_k^B\}} + a}$$

$$= \frac{2 + 0.1 \times 0.7}{3 + 0.1} = 0.6677$$

CatBoost

- Greedy TS with smoothing example
  - ✓ a = 0.1 (parameter) , p = 0.7 (computed from the training dataset)
  - ✓ For category C

	…	x^i	…	y
I_1	…	A	…	I
I_2	…	B	…	I
I_3	…	C	…	I
I_4	…	A	…	0
I_5	…	B	…	I
I_6	…	C	…	I
I_7	…	B	…	0
I_8	…	C	…	I
I_9	…	C	…	0
I_{10}	…	C	…	I

$$\hat{x}_k^C = \frac{\sum_{j=1}^{n} \mathbb{1}_{\{x_j^C = x_k^C\}} \cdot y_j + ap}{\sum_{j=1}^{n} \mathbb{1}_{\{x_j^C = x_k^C\}} + a}$$

$$= \frac{4 + 0.1 \times 0.7}{5 + 0.1} = 0.7980$$

CatBoost

- Greedy TS with smoothing example
  - ✓ a = 0.1 (parameter) , p = 0.7 (computed from the training dataset)

	…	x^i	…	y
I_1	…	A	…	1
I_2	…	B	…	1
I_3	…	C	…	1
I_4	…	A	…	0
I_5	…	B	…	1
I_6	…	C	…	1
I_7	…	B	…	0
I_8	…	C	…	1
I_9	…	C	…	0
I_{10}	…	C	…	1

	…	x^i(TS)	…	y
I_1	…	0.5095	…	1
I_2	…	0.6677	…	1
I_3	…	0.7980	…	1
I_4	…	0.5095	…	0
I_5	…	0.6677	…	1
I_6	…	0.7980	…	1
I_7	…	0.6677	…	0
I_8	…	0.7980	…	1
I_9	…	0.7980	…	0
I_{10}	…	0.7980	…	1

CatBoost

- Target leakage
  - ✓ feature $\hat{x}_k^i$ is computed using y_k, the target of $\mathbf{x}_k$
  - ✓ leads to a conditional shift: the distribution of $\hat{x}^i|y$ differs for training and test examples
    - Assume that i^{th} feature is categorical, all its values are unique,

	…	x^i	…	y
I_1	…	A	…	I
I_2	…	B	…	0
I_3	…	C	…	I
I_4	…	D	…	0
I_5	…	E	…	I

	…	x^i	…	y
I_1	…	$\frac{1+ap}{1+a}$	…	I
I_2	…	$\frac{0+ap}{1+a}$	…	0
I_3	…	$\frac{1+ap}{1+a}$	…	I
I_4	…	$\frac{0+ap}{1+a}$	…	0
I_5	…	$\frac{1+ap}{1+a}$	…	I

    - Training data can be perfectly classified when the split criterion is set to $x^i = \frac{0.5+ap}{1+a}$

CatBoost

- Target leakage

	…	x^i	…	y
I_1	…	A	…	I
I_2	…	B	…	0
I_3	…	C	…	I
I_4	…	D	…	0
I_5	…	E	…	I
I_6	…	F	…	I
I_7	…	G	…	0
I_8	…	H	…	I
I_9	…	I	…	0

Training Set

Test Set

	…	x^i	…	y
I_1	…	$\frac{1+ap}{1+a}$	…	I
I_2	…	$\frac{0+ap}{1+a}$	…	0
I_3	…	$\frac{1+ap}{1+a}$	…	I
I_4	…	$\frac{0+ap}{1+a}$	…	0
I_5	…	$\frac{1+ap}{1+a}$	…	I
I_6	…	p	…	I
I_7	…	p	…	0
I_8	…	p	…	I
I_9	…	p	…	0

No discrimination ability

CatBoost

- Desired property for TS
  - ✓ Property 1:

$$\mathbb{E}(\hat{x}^i \mid y = v) = \mathbb{E}(\hat{x}^i \mid y_k = v)$$

$$where\ (\mathbf{x}_k, y_k)\ is\ the\ k-th\ training\ example$$

  - ✓ Property 2:
    - Effective usage of all training data for calculating TS features and for learning a model

CatBoost

- Ways to avoid conditional shift
  - ✓ General idea: compute the TS for $\mathbf{x}_k$ on a subset of examples $\mathcal{D}_k \subset \mathcal{D} \backslash \{\mathbf{x}_k\}$ excluding $\mathbf{x}_k$

$$\hat{x}_k^i = \frac{\sum_{\mathbf{x}_j \in \mathcal{D}_k} \mathbb{1}_{\{x_j^i = x_k^i\}} \cdot y_j + ap}{\sum_{\mathbf{x}_j \in \mathcal{D}_k} \mathbb{1}_{\{x_j^i = x_k^i\}} + a}$$

  - ✓ Holdout TS
    - To partition the training dataset into two part $\mathcal{D} = \hat{\mathcal{D}}_0 \cup \hat{\mathcal{D}}_1$
      - Use the former to compute the TS
      - Use the latter for training
      - Violates the Property 2

CatBoost

- Ways to avoid conditional shift
  - ✓ Leave-one-out TS
    - Take $\mathcal{D}_k = \mathcal{D} \backslash \mathbf{x}_k$ for training examples
    - Take $\mathcal{D}_k = \mathcal{D}$ for test ones
    - Does not prevent target leakage

$$\hat{x}_k^i = \frac{\sum_{\mathbf{x}_j \in \mathcal{D}_k} \mathbb{1}_{\{x_j^i = x_k^i\}} \cdot y_j \; + ap}{\sum_{\mathbf{x}_j \in \mathcal{D}_k} \mathbb{1}_{\{x_j^i = x_k^i\}} \; + \; a}$$

	…	x^i	…	y
I_1	…	A	…	I
I_2	…	A	…	0
I_3	…	A	…	I
I_4	…	A	…	0
I_5	…	A	…	I

	…	x^i	…	y
I_1	…	$\frac{2+ap}{4+a}$	…	I
I_2	…	$\frac{3+ap}{4+a}$	…	0
I_3	…	$\frac{2+ap}{4+a}$	…	I
I_4	…	$\frac{3+ap}{4+a}$	…	0
I_5	…	$\frac{2+ap}{4+a}$	…	I

- We can perfectly classify this training dataset by setting the cut-off value $\frac{2.5+ap}{4+a}$

CatBoost

- Ways to avoid conditional shift
  - ✓ Ordered TS: introduce an artificial time
    - a random permutation σ of the training examples

$$\mathcal{D}_k = \{\mathbf{x}_j : \sigma(j) < \sigma(k)\}$$

    - a = 0.1 (parameter) , p = 0.7 (computed from the training dataset)

	…	x^i	…	TS	y
I_1	…	A	…	0.700	I
I_2	…	B	…		I
I_3	…	C	…		I
I_4	…	A	…		0
I_5	…	B	…		I
I_6	…	C	…		I
I_7	…	B	…		0
I_8	…	C	…		I
I_9	…	C	…		0
I_{10}	…	C	…		I

$$\hat{x}_k^i = \frac{\sum_{\mathbf{x}_j \in \mathcal{D}_k} \mathbb{1}_{\{x_j^i = x_k^i\}} \cdot y_j + ap}{\sum_{\mathbf{x}_j \in \mathcal{D}_k} \mathbb{1}_{\{x_j^i = x_k^i\}} + a}$$

$$= \frac{0 + 0.1 \times 0.7}{0 + 0.1} = 0.7$$

CatBoost

- Ways to avoid conditional shift
  - ✓ Ordered TS: introduce an artificial time
    - a random permutation σ of the training examples

$$\mathcal{D}_k = \{\mathbf{x}_j : \sigma(j) < \sigma(k)\}$$

    - a = 0.1 (parameter) , p = 0.7 (computed from the training dataset)

	…	x^i	…	TS	y
I_1	…	A	…	0.700	I
I_2	…	B	…	0.700	I
I_3	…	C	…		I
I_4	…	A	…		0
I_5	…	B	…		I
I_6	…	C	…		I
I_7	…	B	…		0
I_8	…	C	…		I
I_9	…	C	…		0
I_{10}	…	C	…		I

$$\hat{x}_k^i = \frac{\sum_{\mathbf{x}_j \in \mathcal{D}_k} \mathbb{1}_{\{x_j^i = x_k^i\}} \cdot y_j + ap}{\sum_{\mathbf{x}_j \in \mathcal{D}_k} \mathbb{1}_{\{x_j^i = x_k^i\}} + a}$$

$$= \frac{0 + 0.1 \times 0.7}{0 + 0.1} = 0.700$$

CatBoost

- Ways to avoid conditional shift
  - ✓ Ordered TS: introduce an artificial time
    - a random permutation σ of the training examples

$$\mathcal{D}_k = \{\mathbf{x}_j : \sigma(j) < \sigma(k)\}$$

    - a = 0.1 (parameter) , p = 0.7 (computed from the training dataset)

	…	x^i	…	TS	y
I_1	…	A	…	0.700	I
I_2	…	B	…	0.700	I
I_3	…	C	…	0.700	I
I_4	…	A	…		0
I_5	…	B	…		I
I_6	…	C	…		I
I_7	…	B	…		0
I_8	…	C	…		I
I_9	…	C	…		0
I_{10}	…	C	…		I

$$\hat{x}_k^i = \frac{\sum_{\mathbf{x}_j \in \mathcal{D}_k} \mathbb{1}_{\{x_j^i = x_k^i\}} \cdot y_j + ap}{\sum_{\mathbf{x}_j \in \mathcal{D}_k} \mathbb{1}_{\{x_j^i = x_k^i\}} + a}$$

$$= \frac{0 + 0.1 \times 0.7}{0 + 0.1} = 0.700$$

CatBoost

- Ways to avoid conditional shift
  - ✓ Ordered TS: introduce an artificial time
    - a random permutation σ of the training examples

$$\mathcal{D}_k = \{\mathbf{x}_j : \sigma(j) < \sigma(k)\}$$

    - a = 0.1 (parameter) , p = 0.7 (computed from the training dataset)

	…	x^i	…	TS	y
I_1	…	A	…	0.700	1
I_2	…	B	…	0.700	1
I_3	…	C	…	0.700	1
I_4	…	A	…	0.973	0
I_5	…	B	…		1
I_6	…	C	…		1
I_7	…	B	…		0
I_8	…	C	…		1
I_9	…	C	…		0
I_{10}	…	C	…		1

$$\hat{x}_k^i = \frac{\sum_{\mathbf{x}_j \in \mathcal{D}_k} \mathbb{1}_{\{x_j^i = x_k^i\}} \cdot y_j + ap}{\sum_{\mathbf{x}_j \in \mathcal{D}_k} \mathbb{1}_{\{x_j^i = x_k^i\}} + a}$$

$$= \frac{1 + 0.1 \times 0.7}{1 + 0.1} = 0.973$$

CatBoost

- Ways to avoid conditional shift
  - ✓ Ordered TS: introduce an artificial time
    - a random permutation σ of the training examples

$$\mathcal{D}_k = \{\mathbf{x}_j : \sigma(j) < \sigma(k)\}$$

    - a = 0.1 (parameter) , p = 0.7 (computed from the training dataset)

	…	x^i	…	TS	y
I_1	…	A	…	0.700	1
I_2	…	B	…	0.700	1
I_3	…	C	…	0.700	1
I_4	…	A	…	0.973	0
I_5	…	B	…	0.973	1
I_6	…	C	…		1
I_7	…	B	…		0
I_8	…	C	…		1
I_9	…	C	…		0
I_{10}	…	C	…		1

$$\hat{x}_k^i = \frac{\sum_{\mathbf{x}_j \in \mathcal{D}_k} \mathbb{1}_{\{x_j^i = x_k^i\}} \cdot y_j + ap}{\sum_{\mathbf{x}_j \in \mathcal{D}_k} \mathbb{1}_{\{x_j^i = x_k^i\}} + a}$$

$$= \frac{1 + 0.1 \times 0.7}{1 + 0.1} = 0.973$$

CatBoost

- Ways to avoid conditional shift
  - ✓ Ordered TS: introduce an artificial time
    - a random permutation σ of the training examples

$$\mathcal{D}_k = \{\mathbf{x}_j : \sigma(j) < \sigma(k)\}$$

    - a = 0.1 (parameter) , p = 0.7 (computed from the training dataset)

	…	x^i	…	TS	y
I_1	…	A	…	0.700	1
I_2	…	B	…	0.700	1
I_3	…	C	…	0.700	1
I_4	…	A	…	0.973	0
I_5	…	B	…	0.973	1
I_6	…	C	…	0.973	1
I_7	…	B	…		0
I_8	…	C	…		1
I_9	…	C	…		0
I_{10}	…	C	…		1

$$\hat{x}_k^i = \frac{\sum_{\mathbf{x}_j \in \mathcal{D}_k} \mathbb{1}_{\{x_j^i = x_k^i\}} \cdot y_j + ap}{\sum_{\mathbf{x}_j \in \mathcal{D}_k} \mathbb{1}_{\{x_j^i = x_k^i\}} + a}$$

$$= \frac{1 + 0.1 \times 0.7}{1 + 0.1} = 0.973$$

CatBoost

- Ways to avoid conditional shift
  - ✓ Ordered TS: introduce an artificial time
    - a random permutation σ of the training examples

$$\mathcal{D}_k = \{\mathbf{x}_j : \sigma(j) < \sigma(k)\}$$

    - a = 0.1 (parameter) , p = 0.7 (computed from the training dataset)

	…	x^i	…	TS	y
I_1	…	A	…	0.700	1
I_2	…	B	…	0.700	1
I_3	…	C	…	0.700	1
I_4	…	A	…	0.973	0
I_5	…	B	…	0.973	1
I_6	…	C	…	0.973	1
I_7	…	B	…	0.986	0
I_8	…	C	…		1
I_9	…	C	…		0
I_{10}	…	C	…		1

$$\hat{x}_k^i = \frac{\sum_{\mathbf{x}_j \in \mathcal{D}_k} \mathbb{1}_{\{x_j^i = x_k^i\}} \cdot y_j + ap}{\sum_{\mathbf{x}_j \in \mathcal{D}_k} \mathbb{1}_{\{x_j^i = x_k^i\}} + a}$$

$$= \frac{2 + 0.1 \times 0.7}{2 + 0.1} = 0.986$$

CatBoost

- Ways to avoid conditional shift
  - ✓ Ordered TS: introduce an artificial time
    - a random permutation σ of the training examples

$$\mathcal{D}_k = \{\mathbf{x}_j : \sigma(j) < \sigma(k)\}$$

    - a = 0.1 (parameter) , p = 0.7 (computed from the training dataset)

	…	x^i	…	TS	y
I_1	…	A	…	0.700	1
I_2	…	B	…	0.700	1
I_3	…	C	…	0.700	1
I_4	…	A	…	0.973	0
I_5	…	B	…	0.973	1
I_6	…	C	…	0.973	1
I_7	…	B	…	0.986	0
I_8	…	C	…	0.986	1
I_9	…	C	…		0
I_{10}	…	C	…		1

$$\hat{x}_k^i = \frac{\sum_{\mathbf{x}_j \in \mathcal{D}_k} \mathbb{1}_{\{x_j^i = x_k^i\}} \cdot y_j + ap}{\sum_{\mathbf{x}_j \in \mathcal{D}_k} \mathbb{1}_{\{x_j^i = x_k^i\}} + a}$$

$$= \frac{2 + 0.1 \times 0.7}{2 + 0.1} = 0.986$$

CatBoost

- Ways to avoid conditional shift
  - ✓ Ordered TS: introduce an artificial time
    - a random permutation σ of the training examples

$$\mathcal{D}_k = \{\mathbf{x}_j : \sigma(j) < \sigma(k)\}$$

    - a = 0.1 (parameter) , p = 0.7 (computed from the training dataset)

	…	x^i	…	TS	y
I_1	…	A	…	0.700	1
I_2	…	B	…	0.700	1
I_3	…	C	…	0.700	1
I_4	…	A	…	0.973	0
I_5	…	B	…	0.973	1
I_6	…	C	…	0.973	1
I_7	…	B	…	0.986	0
I_8	…	C	…	0.986	1
I_9	…	C	…	0.990	0
I_{10}	…	C	…		1

$$\hat{x}_k^i = \frac{\sum_{\mathbf{x}_j \in \mathcal{D}_k} \mathbb{1}_{\{x_j^i = x_k^i\}} \cdot y_j + ap}{\sum_{\mathbf{x}_j \in \mathcal{D}_k} \mathbb{1}_{\{x_j^i = x_k^i\}} + a}$$

$$= \frac{3 + 0.1 \times 0.7}{3 + 0.1} = 0.990$$

CatBoost

- Ways to avoid conditional shift
  - ✓ Ordered TS: introduce an artificial time
    - a random permutation σ of the training examples

$$\mathcal{D}_k = \{\mathbf{x}_j : \sigma(j) < \sigma(k)\}$$

    - a = 0.1 (parameter) , p = 0.7 (computed from the training dataset)

	…	x^i	…	TS	y
I_1	…	A	…	0.700	1
I_2	…	B	…	0.700	1
I_3	…	C	…	0.700	1
I_4	…	A	…	0.973	0
I_5	…	B	…	0.973	1
I_6	…	C	…	0.973	1
I_7	…	B	…	0.986	0
I_8	…	C	…	0.986	1
I_9	…	C	…	0.990	0
I_{10}	…	C	…	0.749	1

$$\hat{x}_k^i = \frac{\sum_{\mathbf{x}_j \in \mathcal{D}_k} \mathbb{1}_{\{x_j^i = x_k^i\}} \cdot y_j + ap}{\sum_{\mathbf{x}_j \in \mathcal{D}_k} \mathbb{1}_{\{x_j^i = x_k^i\}} + a}$$

$$= \frac{3 + 0.1 \times 0.7}{4 + 0.1} = 0.749$$

CatBoost

- Prediction Shift

$$h^t = \arg\min_{h \in H} \; \mathbb{E}\big(-g^t(\mathbf{x}, y) - h(\mathbf{x})\big)^2$$

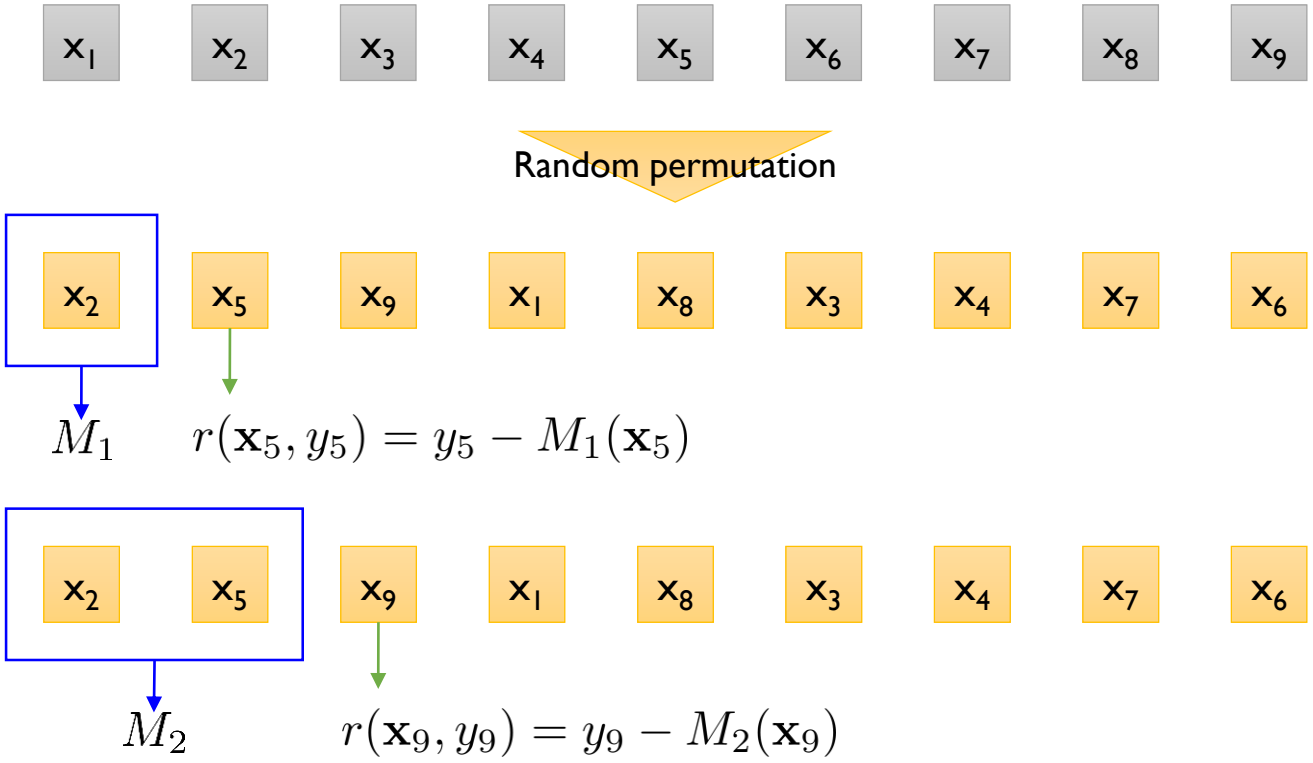
✓ The expectation is unknown and is usually approximated using the same dataset $\mathcal{D}$

$$h^t = \arg\min_{h \in H} \; \frac{1}{n} \sum_{k=1}^{n} \big(-g^t(\mathbf{x}, y) - h(\mathbf{x})\big)^2$$

- The conditional distribution of the gradient $g^t(\mathbf{x}_k, y_k)|\mathbf{x}_k$ is shifted from that distribution on a test example $g^t(\mathbf{x}, y)|\mathbf{x}$
- In turn, base predictor h^t is biased from the solution
- Finally it affects the generalization ability of the trained model F^t

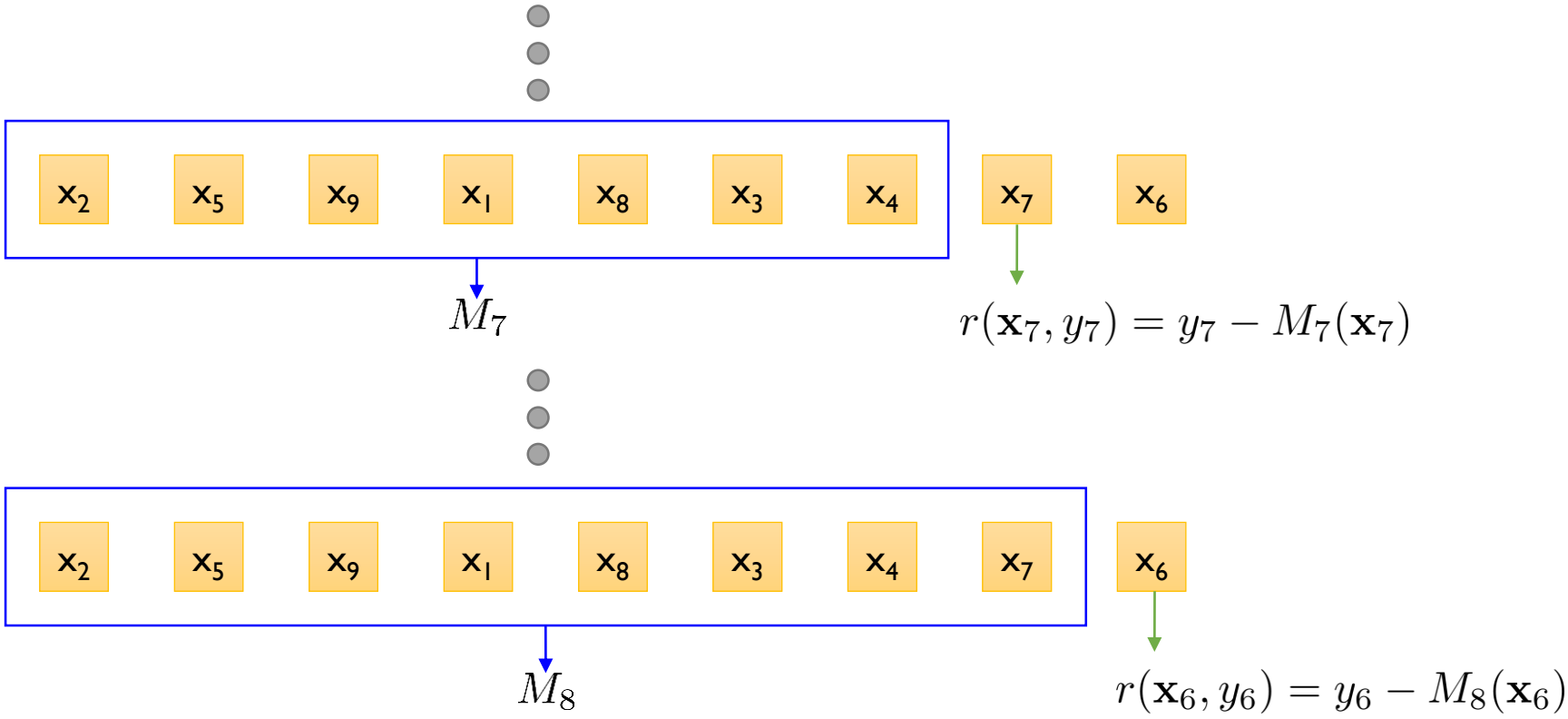
CatBoost

- Ordered Boosting
  - ✓ A boosting algorithm not suffering from the prediction shift problem



CatBoost

- Ordered Boosting
  - ✓ A boosting algorithm not suffering from the prediction shift problem



  - ✓ The same permutation is used for both TS computing and ordered boosting

CatBoost

- Ordered Boosting and Building a Tree in CatBoost

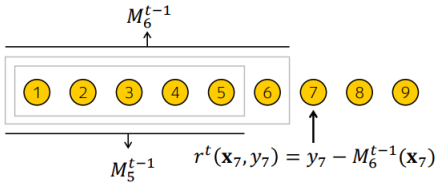


Figure 1: Ordered boosting principle, examples are ordered according to σ.

Algorithm 1: Ordered boosting

input : $\{(\mathbf{x}_k, y_k)\}_{k=1}^n$, I;

$\sigma \leftarrow$ random permutation of $[1, n]$;
$M_i \leftarrow 0$ for $i = 1..n$;
for $t \leftarrow 1$ **to** I **do**
  for $i \leftarrow 1$ **to** n **do**
    $r_i \leftarrow y_i - M_{\sigma(i)-1}(\mathbf{x}_i)$;
  for $i \leftarrow 1$ **to** n **do**
    $\Delta M \leftarrow$ $LearnModel((\mathbf{x}_j, r_j) : \sigma(j) \leq i)$;
    $M_i \leftarrow M_i + \Delta M$;
return M_n

Algorithm 2: Building a tree in CatBoost

input : M, $\{(\mathbf{x}_i, y_i)\}_{i=1}^n$, α, L, $\{\sigma_i\}_{i=1}^s$, $Mode$

$grad \leftarrow CalcGradient(L, M, y)$;
$r \leftarrow random(1, s)$;
if $Mode = Plain$ **then**
  $G \leftarrow (grad_r(i)$ for $i = 1..n)$;
if $Mode = Ordered$ **then**
  $G \leftarrow (grad_{r,\sigma_r(i)-1}(i)$ for $i = 1..n)$;
$T \leftarrow$ empty tree;
foreach *step of top-down procedure* **do**
  foreach *candidate split* c **do**
    $T_c \leftarrow$ add split c to T;
    if $Mode = Plain$ **then**
      $\Delta(i) \leftarrow \text{avg}(grad_r(p)$ for $p : leaf_r(p) = leaf_r(i))$ for $i = 1..n$;
    if $Mode = Ordered$ **then**
      $\Delta(i) \leftarrow \text{avg}(grad_{r,\sigma_r(i)-1}(p)$ for $p : leaf_r(p) = leaf_r(i), \sigma_r(p) < \sigma_r(i))$ for $i = 1..n$;
    $loss(T_c) \leftarrow \cos(\Delta, G)$
  $T \leftarrow \arg\min_{T_c}(loss(T_c))$
if $Mode = Plain$ **then**
  $M_{r'}(i) \leftarrow M_{r'}(i) - \alpha\,\text{avg}(grad_{r'}(p)$ for $p : leaf_{r'}(p) = leaf_{r'}(i))$ for $r' = 1..s$, $i = 1..n$;
if $Mode = Ordered$ **then**
  $M_{r',j}(i) \leftarrow M_{r',j}(i) - \alpha\,\text{avg}(grad_{r',j}(p)$ for $p : leaf_{r'}(p) = leaf_{r'}(i), \sigma_{r'}(p) \leq j)$ for $r' = 1..s$, $i = 1..n$, $j \geq \sigma_{r'}(i) - 1$;
return T, M

CatBoost

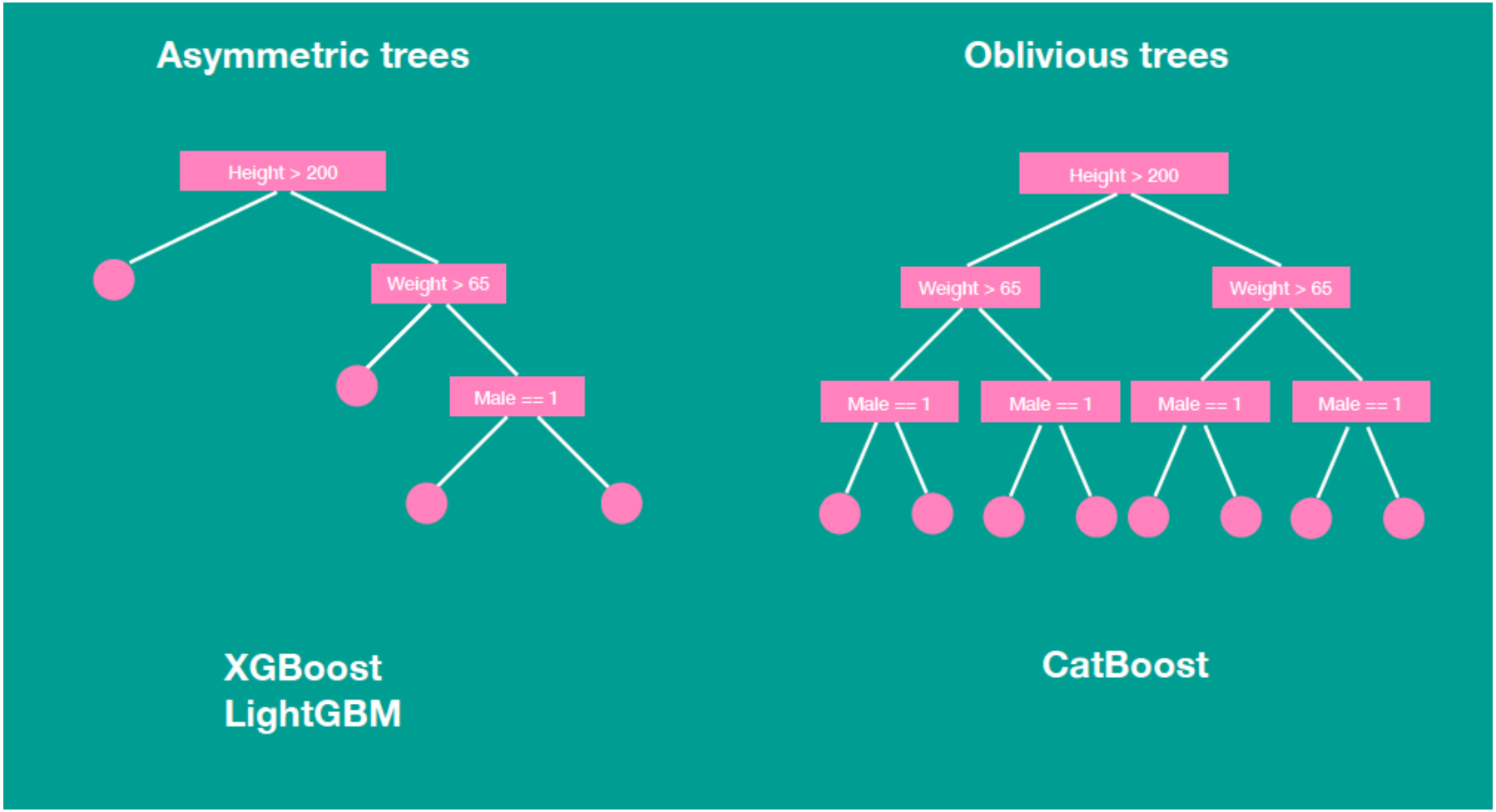
- CatBoost Procedure
  - ✓ Initialization: generate (s+1) independent random permutations of the training dataset
    - s permutations to evaluate the split
    - 1 permutation to compute the leaf value of the obtained trees
  - ✓ Both TS and predictions used by ordered boosting have a high variance
  - ✓ Using only one permutation may increase the variance of the final model predictions, while several permutations can reduce it

CatBoost

- CatBoost Procedure
  - ✓ Building an oblivious tree in the ordered boosting mode
    - Maintain the supporting models $M_{r,j}$
    - $M_{r,j}(i)$: the current prediction for the i[th] example based on the first *j* examples in the permutation σ_r
    - At each iteration t, sample a random permutation σ_r and construct a tree T_t
      - TS are computed according to this permutation
      - The permutation affects the tree learning procedure
    - For each example i, take the gradient $grad_{r,\sigma_r(i)-1}(i)$
    - While constructing a tree, the loss of the candidate split c is computed by the cosine similarity between the leaf value and the gradient of each example
      - At the candidate splits evaluation step, the leaf value $\Delta(i)$ for example i is obtained individually by averaging the gradients $grad_{r,\sigma_r(i)-1}$ of the preceding example p lying in the same leaf node of i

CatBoost

- Oblivious tree

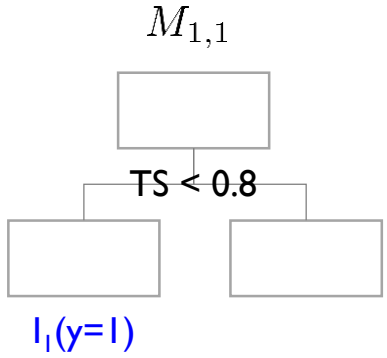


https://towardsdatascience.com/introduction-to-gradient-boosting-on-decision-trees-with-catboost-d511a9ccbd14

CatBoost

- Ordered Boosting Example
  - ✓ Assumption: squared loss (gradient: $f(x) - y$)
  - ✓ random permutation is conducted

	…	x^i	…	TS	y	G	Δ
I_1	…	A	…	0.700	I	0	0
I_2	…	B	…	0.700	I		
I_3	…	C	…	0.700	I		
I_4	…	A	…	0.973	0		
I_5	…	B	…	0.973	I		
I_6	…	C	…	0.973	I		
I_7	…	B	…	0.986	0		
I_8	…	C	…	0.986	I		
I_9	…	C	…	0.990	0		
I_{10}	…	C	…	0.749	I		



$$G(I_1) = 0$$

$$\Delta(I_1) = 0$$

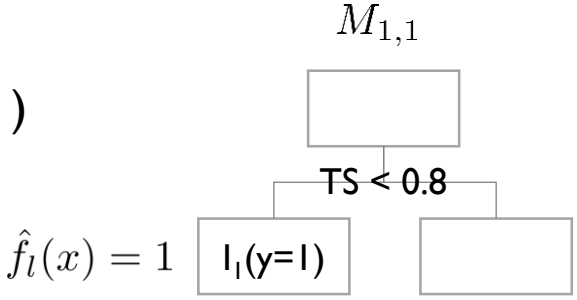
Lecturer's guess

- Initialize both the first gradient and leaf value to 0

CatBoost

- Ordered Boosting Example
  - ✓ Assumption: squared loss (gradient: $f(x) - y$)
  - ✓ random permutation is conducted

	…	x^i	…	TS	y	G	Δ
I_1	…	A	…	0.700	I	0	0
I_2	…	B	…	0.700	I	0	0
I_3	…	C	…	0.700	I		
I_4	…	A	…	0.973	0		
I_5	…	B	…	0.973	I		
I_6	…	C	…	0.973	I		
I_7	…	B	…	0.986	0		
I_8	…	C	…	0.986	I		
I_9	…	C	…	0.990	0		
I_{10}	…	C	…	0.749	I		



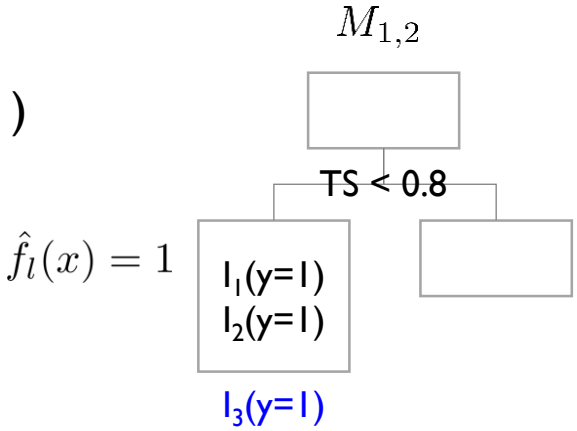
$$G(I_2) = grad_{1,\sigma_1(2)-1}(I_2)$$
$$= \hat{f}_l(x) - y_2 = 1 - 1 = 0$$

$$\Delta(I_2) = grad_{1,\sigma_1(2)-1}(I_1)$$
$$= 0$$

CatBoost

- Ordered Boosting Example
  - ✓ Assumption: squared loss (gradient: $f(x) - y$)
  - ✓ random permutation is conducted

	…	x^i	…	TS	y	G	Δ
I_1	…	A	…	0.700	I	0	0
I_2	…	B	…	0.700	I	0	0
I_3	…	C	…	0.700	I	0	0
I_4	…	A	…	0.973	0		
I_5	…	B	…	0.973	I		
I_6	…	C	…	0.973	I		
I_7	…	B	…	0.986	0		
I_8	…	C	…	0.986	I		
I_9	…	C	…	0.990	0		
I_{10}	…	C	…	0.749	I		



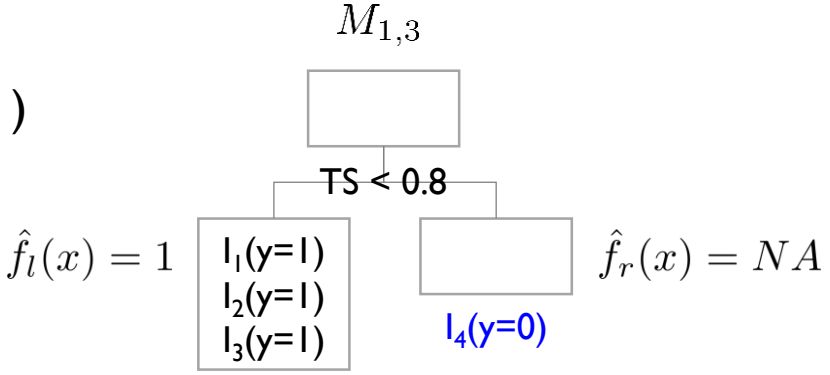
$$G(I_3) = grad_{1,\sigma_1(3)-1}(I_3)$$
$$= \hat{f}_l(x) - y_3 = 1 - 1 = 0$$

$$\Delta(I_3) = \frac{1}{2}(grad_{1,\sigma_1(3)-1}(I_1)$$
$$+ grad_{1,\sigma_1(3)-1}(I_2)) = 0$$
$$= \frac{1}{2}(0+0) = 0$$

CatBoost

- Ordered Boosting Example
  - ✓ Assumption: squared loss (gradient: $f(x) - y$)
  - ✓ random permutation is conducted

	…	x^i	…	TS	y	G	Δ
I_1	…	A	…	0.700	1	0	0
I_2	…	B	…	0.700	1	0	0
I_3	…	C	…	0.700	1	0	0
I_4	…	A	…	0.973	0	0	0
I_5	…	B	…	0.973	1		
I_6	…	C	…	0.973	1		
I_7	…	B	…	0.986	0		
I_8	…	C	…	0.986	1		
I_9	…	C	…	0.990	0		
I_{10}	…	C	…	0.749	1		



$$G(I_4) = grad_{1,\sigma_1(4)-1}(I_4)$$
$$= 0$$

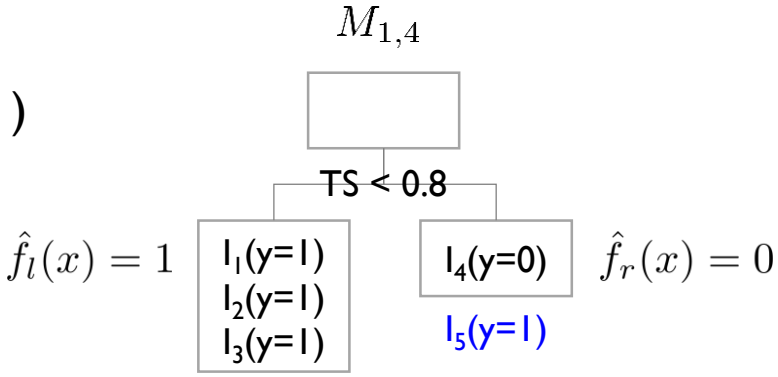
$$\Delta(I_4) = 0$$

Lecturer's guess

CatBoost

- Ordered Boosting Example
  - ✓ Assumption: squared loss (gradient: $f(x) - y$)
  - ✓ random permutation is conducted

	…	x^i	…	TS	y	G	Δ
I_1	…	A	…	0.700	1	0	0
I_2	…	B	…	0.700	1	0	0
I_3	…	C	…	0.700	1	0	0
I_4	…	A	…	0.973	0	0	0
I_5	…	B	…	0.973	1	-1	0
I_6	…	C	…	0.973	1		
I_7	…	B	…	0.986	0		
I_8	…	C	…	0.986	1		
I_9	…	C	…	0.990	0		
I_{10}	…	C	…	0.749	1		



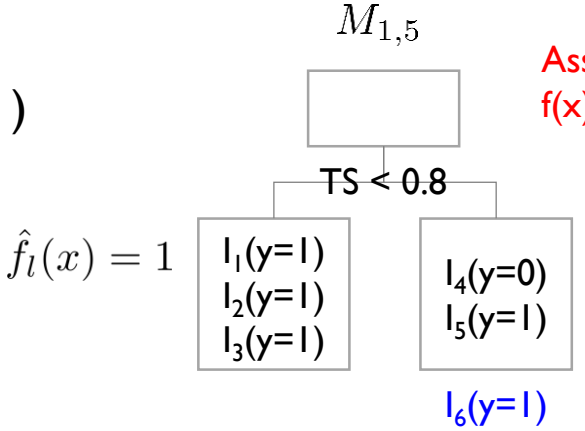
$$G(I_5) = grad_{1,\sigma_1(5)-1}(I_5)$$
$$= \hat{f}_r(x) - y_5 = -1$$

$$\Delta(I_5) = grad_{1,\sigma_1(5)-1}(I_4)$$
$$= 0$$

CatBoost

- Ordered Boosting Example
  - ✓ Assumption: squared loss (gradient: $f(x) - y$)
  - ✓ random permutation is conducted

	…	x^i	…	TS	y	G	Δ
I_1	…	A	…	0.700	1	0	0
I_2	…	B	…	0.700	1	0	0
I_3	…	C	…	0.700	1	0	0
I_4	…	A	…	0.973	0	0	0
I_5	…	B	…	0.973	1	-1	0
I_6	…	C	…	0.973	1	-1	-0.5
I_7	…	B	…	0.986	0		
I_8	…	C	…	0.986	1		
I_9	…	C	…	0.990	0		
I_{10}	…	C	…	0.749	1		



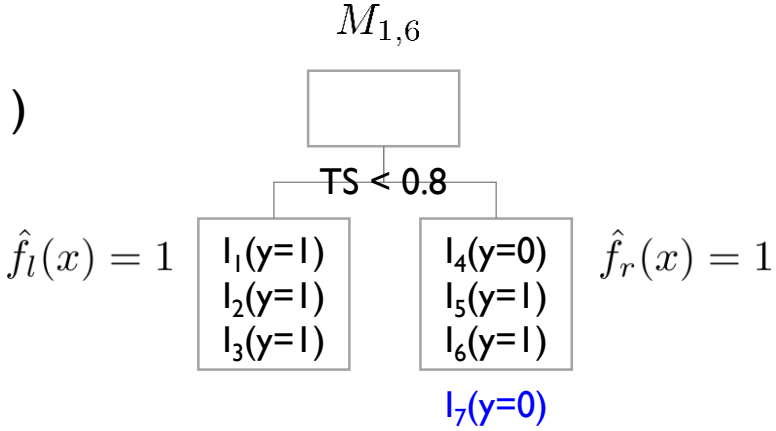
$$G(I_6) = grad_{1,\sigma_1(6)-1}(I_6)$$
$$= \hat{f}_r(x) - y_6 = 0 - 1 = -1$$

$$\Delta(I_6) = \frac{1}{2}\big(grad_{1,\sigma_1(6)-1}(I_4) + grad_{1,\sigma_1(6)-1}(I_5)\big)$$
$$= \frac{1}{2}(0 - 1) = -0.5$$

CatBoost

- Ordered Boosting Example
  - ✓ Assumption: squared loss (gradient: $f(x) - y$)
  - ✓ random permutation is conducted

	…	x^i	…	TS	y	G	Δ
I_1	…	A	…	0.700	1	0	0
I_2	…	B	…	0.700	1	0	0
I_3	…	C	…	0.700	1	0	0
I_4	…	A	…	0.973	0	0	0
I_5	…	B	…	0.973	1	-1	0
I_6	…	C	…	0.973	1	-1	-0.5
I_7	…	B	…	0.986	0	1	-0.67
I_8	…	C	…	0.986	1		
I_9	…	C	…	0.990	0		
I_{10}	…	C	…	0.749	1		



$$G(I_7) = grad_{1,\sigma_1(7)-1}(I_7)$$
$$= \hat{f}_r(x) - y_7 = 1 - 0 = 1$$

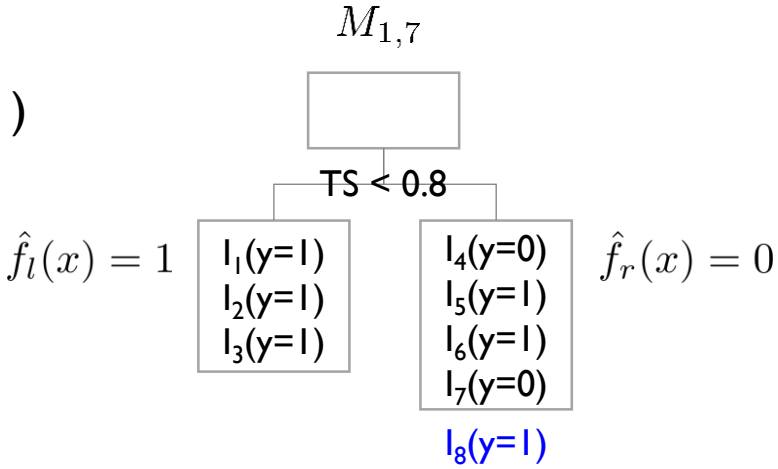
$$\Delta(I_7) = \frac{1}{3}\big(grad_{1,\sigma_1(7)-1}(I_4) + grad_{1,\sigma_1(7)-1}(I_5) + grad_{1,\sigma_1(7)-1}(I_6)\big)$$

$$= \frac{1}{3}(0 - 1 - 1) = -0.67$$

CatBoost

- Ordered Boosting Example
  - ✓ Assumption: squared loss (gradient: $f(x) - y$)
  - ✓ random permutation is conducted

	…	x^i	…	TS	y	G	Δ
I_1	…	A	…	0.700	1	0	0
I_2	…	B	…	0.700	1	0	0
I_3	…	C	…	0.700	1	0	0
I_4	…	A	…	0.973	0	0	0
I_5	…	B	…	0.973	1	-1	0
I_6	…	C	…	0.973	1	-1	-0.5
I_7	…	B	…	0.986	0	1	-0.67
I_8	…	C	…	0.986	1	-1	-0.25
I_9	…	C	…	0.990	0		
I_{10}	…	C	…	0.749	1		



$$G(I_8) = grad_{1,\sigma_1(8)-1}(I_8)$$

$$= \hat{f}_r(x) - y_8 = 0 - 1 = -1$$

$$\Delta(I_8) = \frac{1}{4}\big(grad_{1,\sigma_1(8)-1}(I_4) + grad_{1,\sigma_1(8)-1}(I_5)$$

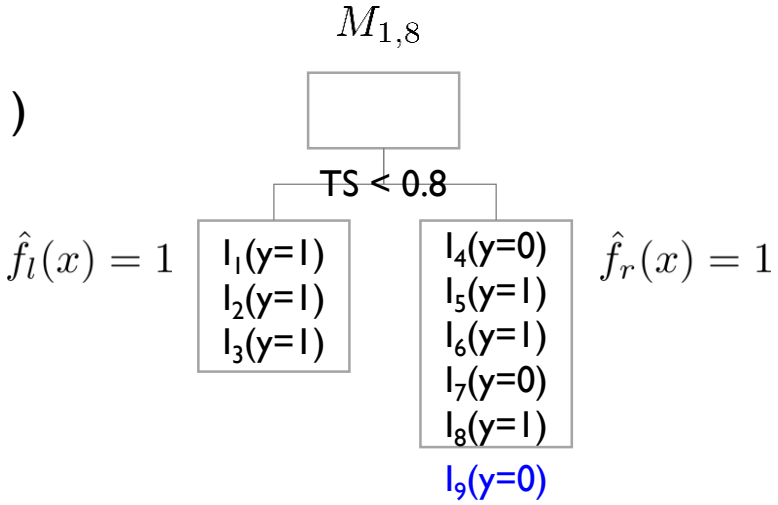
$$+ grad_{1,\sigma_1(8)-1}(I_6) + grad_{1,\sigma_1(8)-1}(I_7)\big)$$

$$= \frac{1}{4}(0 - 1 - 1 + 1) = -0.25$$

CatBoost

- Ordered Boosting Example
  - ✓ Assumption: squared loss (gradient: $f(x) - y$)
  - ✓ random permutation is conducted

	…	x^i	…	TS	y	G	Δ
I_1	…	A	…	0.700	1	0	0
I_2	…	B	…	0.700	1	0	0
I_3	…	C	…	0.700	1	0	0
I_4	…	A	…	0.973	0	0	0
I_5	…	B	…	0.973	1	-1	0
I_6	…	C	…	0.973	1	-1	-0.5
I_7	…	B	…	0.986	0	1	-0.67
I_8	…	C	…	0.986	1	-1	-0.25
I_9	…	C	…	0.990	0	0	-0.4
I_{10}	…	C	…	0.749	1		



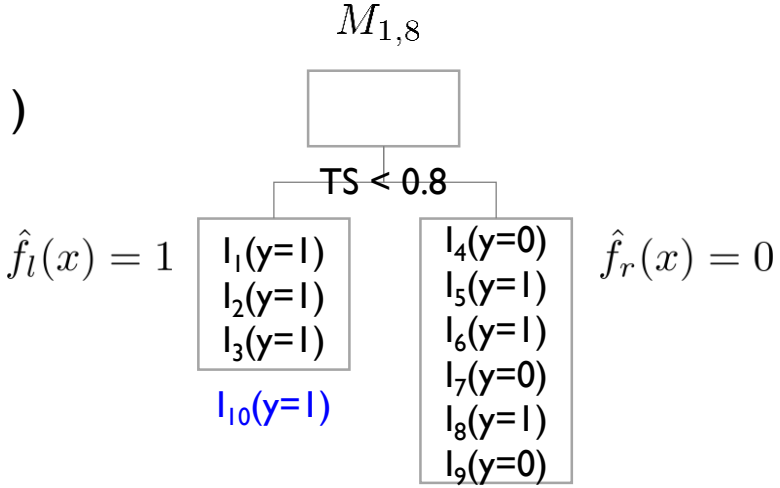
$$G(I_9) = grad_{1,\sigma_1(9)-1}(I_9)$$
$$= \hat{f}_r(x) - y_9 = 1 - 0 = 1$$

$$\Delta(I_9) = \frac{1}{5}\big(grad_{1,\sigma_1(9)-1}(I_4) + grad_{1,\sigma_1(9)-1}(I_5)$$
$$+ grad_{1,\sigma_1(9)-1}(I_6) + grad_{1,\sigma_1(9)-1}(I_7)$$
$$+ grad_{1,\sigma_1(9)-1}(I_8)\big)$$
$$= \frac{1}{5}(0 - 1 - 1 + 1 - 1) = -0.4$$

CatBoost

- Ordered Boosting Example
  - ✓ Assumption: squared loss (gradient: $f(x) - y$)
  - ✓ random permutation is conducted

	…	x^i	…	TS	y	G	Δ
I_1	…	A	…	0.700	1	0	0
I_2	…	B	…	0.700	1	0	0
I_3	…	C	…	0.700	1	0	0
I_4	…	A	…	0.973	0	0	0
I_5	…	B	…	0.973	1	-1	0
I_6	…	C	…	0.973	1	-1	-0.5
I_7	…	B	…	0.986	0	1	-0.67
I_8	…	C	…	0.986	1	-1	-0.25
I_9	…	C	…	0.990	0	0	-0.4
I_{10}	…	C	…	0.749	1	0	0



$$G(I_10) = grad_{1,\sigma_1(10)-1}(I_10)$$
$$= \hat{f}_l(x) - y_{10} = 1 - 1 = 0$$

$$\Delta(I_10) = \frac{1}{3}\big(grad_{1,\sigma_1(10)-1}(I_1) + grad_{1,\sigma_1(10)-1}(I_2) + grad_{1,\sigma_1(10)-1}(I_3)\big)$$
$$= \frac{1}{3}(0+0+0) = 0$$

CatBoost

- Ordered Boosting Example
  - ✓ Assumption: squared loss (gradient: $f(x) - y$)
  - ✓ random permutation is conducted

	…	x^i	…	TS	y	G	Δ
I_1	…	A	…	0.700	I	0	0
I_2	…	B	…	0.700	I	0	0
I_3	…	C	…	0.700	I	0	0
I_4	…	A	…	0.973	0	0	0
I_5	…	B	…	0.973	I	-I	0
I_6	…	C	…	0.973	I	-I	-0.5
I_7	…	B	…	0.986	0	I	-0.67
I_8	…	C	…	0.986	I	-I	-0.25
I_9	…	C	…	0.990	0	0	-0.4
I_{10}	…	C	…	0.749	I	0	0

foreach *step of top-down procedure* **do**
  foreach *candidate split* c **do**
    $T_c \leftarrow$ add split c to T;
    if $Mode = Plain$ **then**
      $\Delta(i) \leftarrow \text{avg}(grad_r(p)$ for $p : leaf_r(p) = leaf_r(i))$ for $i = 1..n$;
    if $Mode = Ordered$ **then**
      $\Delta(i) \leftarrow \text{avg}(grad_{r,\sigma_r(i)-1}(p)$ for $p : leaf_r(p) = leaf_r(i), \sigma_r(p) < \sigma_r(i))$ for $i = 1..n$;
    $loss(T_c) \leftarrow \cos(\Delta, G)$
  $T \leftarrow \arg\min_{T_c}(loss(T_c))$

$$loss(T_c) = cos(\Delta, G) = 0.0417$$

(TS < 0.8)

CatBoost

- Choosing leaf values
  - ✓ Given all the trees constructed, the leaf values of the final model F are calculated by the standard gradient boosting procedure
  - ✓ Training examples i are matched to leaves $leaf_0(i)$ using permutation σ_0 to calculate TS
  - ✓ When the final model F is applied to a new example at testing time, TS is calculated on the whole training data

CatBoost

- Experimental Results

Table 8: Comparison with baselines: logloss / zero-one loss, relative increase is presented in the brackets.

	CatBoost	LightGBM	XGBoost
Adult	**0.2695 / 0.1267**	0.2760 (+2.4%) / 0.1291 (+1.9%)	0.2754 (+2.2%) / 0.1280 (+1.0%)
Amazon	**0.1394 / 0.0442**	0.1636 (+17%) / 0.0533 (+21%)	0.1633 (+17%) / 0.0532 (+21%)
Click	**0.3917 / 0.1561**	0.3963 (+1.2%) / 0.1580 (+1.2%)	0.3962 (+1.2%) / 0.1581 (+1.2%)
Epsilon	**0.2647 / 0.1086**	0.2703 (+1.5%) / 0.114 (+4.1%)	0.2993 (+11%) / 0.1276 (+12%)
Appetency	**0.0715 / 0.01768**	0.0718 (+0.4%) / 0.01772 (+0.2%)	0.0718 (+0.4%) / 0.01780 (+0.7%)
Churn	**0.2319 / 0.0719**	0.2320 (+0.1%) / 0.0723 (+0.6%)	0.2331 (+0.5%) / 0.0730 (+1.6%)
Internet	**0.2089 / 0.0937**	0.2231 (+6.8%) / 0.1017 (+8.6%)	0.2253 (+7.9%) / 0.1012 (+8.0%)
Upselling	**0.1662 / 0.0490**	0.1668 (+0.3%) / 0.0491 (+0.1%)	0.1663 (+0.04%) / 0.0492 (+0.3%)
Kick	**0.2855 / 0.0949**	0.2957 (+3.5%) / 0.0991 (+4.4%)	0.2946 (+3.2%) / 0.0988 (+4.1%)

Summary

- XGBoost vs. LightGBM vs. CatBoost

Function	XGBoost	CatBoost	Light GBM
Important parameters which control overfitting	1. **learning_rate or eta** – optimal values lie between 0.01-0.2 2. **max_depth** 3. **min_child_weight:** similar to min_child leaf; default is 1	1. **Learning_rate** 2. **Depth** - value can be any integer up to 16. Recommended - [1 to 10] 3. No such feature like min_child_weight 4. **l2-leaf-reg**: L2 regularization coefficient. Used for leaf value calculation (any positive integer allowed)	1. **learning_rate** 2. **max_depth**: default is 20. Important to note that tree still grows leaf-wise. Hence it is important to tune **num_leaves** (number of leaves in a tree) which should be smaller than 2^(max_depth). It is a very important parameter for LGBM 3. **min_data_in_leaf**: default=20, alias= min_data, min_child_samples
Parameters for categorical values	Not Available	1. **cat_features**: It denotes the index of categorical features 2. **one_hot_max_size**: Use one-hot encoding for all features with number of different values less than or equal to the given parameter value (max – 255)	1. **categorical_feature:** specify the categorical features we want to use for training our model
Parameters for controlling speed	1. **colsample_bytree**: subsample ratio of columns 2. **subsample**: subsample ratio of the training instance 3. **n_estimators**: maximum number of decision trees; high value can lead to overfitting	1. **rsm:** Random subspace method. The percentage of features to use at each split selection 2. No such parameter to subset data 3. **iterations**: maximum number of trees that can be built; high value can lead to overfitting	1. **feature_fraction:** fraction of features to be taken for each iteration 2. **bagging_fraction:** data to be used for each iteration and is generally used to speed up the training and avoid overfitting 3. **num_iterations:** number of boosting iterations to be performed; default=100

https://towardsdatascience.com/catboost-vs-light-gbm-vs-xgboost-5f93620723db

Summary

Anghel et al. (2019)

- XGBoost vs. LightGBM vs. CatBoost

Table 3: Best test scores across algorithms and datasets.

Dataset	Baseline		XGBoost		LightGBM		Catboost	
	Test	Val	Test	Val	Test	Val	Test	Val
Higgs	0.4996	0.5005	0.8353	0.8512	*0.8573*	0.8577	0.8498	0.8496
Epsilon	0.4976	0.5008	-	-	0.9513	0.9518	*0.9537*	0.9538
Microsoft	0.2251	0.3974	*0.4917*	0.6443	0.4871	0.6473	0.3782	0.5492
Yahoo	0.5802	0.8106	*0.7983*	0.9146	0.7965	0.9142	0.7351	0.8849

Summary

Anghel et al. (2019)

- XGBoost vs. LightGBM vs. CatBoost

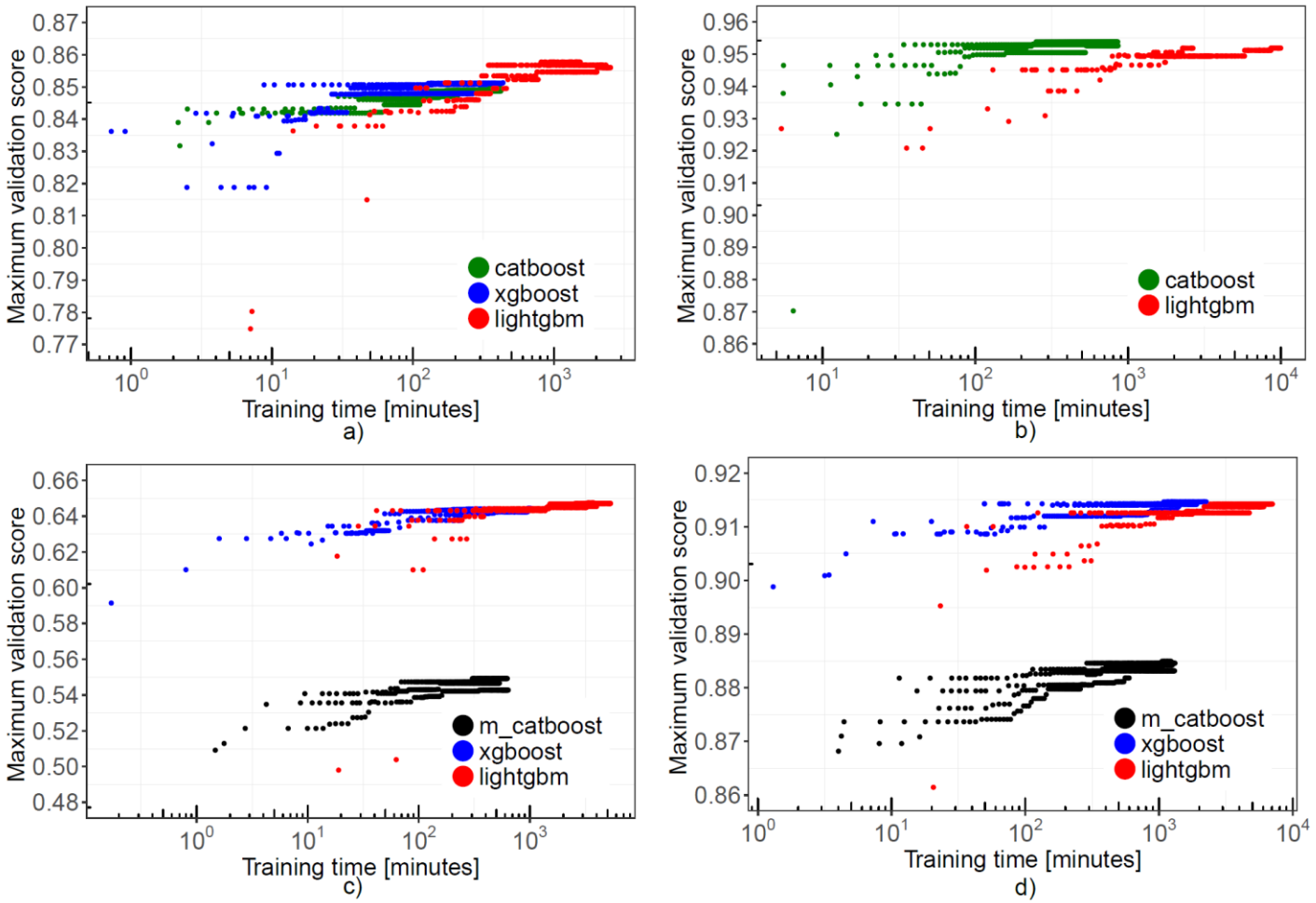


Figure 2: Max. validation score vs. total HPO runtime (a) Higgs (b) Epsilon (c) Microsoft (d) Yahoo.


Q & A

References

Research Papers

- Anghel, A., Papandreou, N., Parnell, T., De Palma, A., & Pozidis, H. (2018). Benchmarking and Optimization of Gradient Boosted Decision Tree Algorithms. arXiv preprint arXiv:1809.04559.
- Breiman, L. (1996). Bagging predictors. Machine learning 24(2): 123-140.
- Breiman, L. (2001). Random forests. Machine learning 45(1): 5-32.
- Caruana, R. and Niculescu-Mizil, A. (2006). An empirical comparison of supervised learning algorithms. In Proceedings of the 23rd international conference on Machine learning: 161-168.
- Chen, T., & Guestrin, C. (2016, August). Xgboost: A scalable tree boosting system. In Proceedings of the 22nd acm sigkdd international conference on knowledge discovery and data mining (pp. 785-794). ACM.
- Fernández-Delgado, M., Cernadas, E., Barro, S., and Amorim, D. (2014). Do we need hundreds of classifiers to solve real world classification problems? The Journal of Machine Learning Research 15(1): 3133-3181.
- Freund, Y., Schapire, R., and Abe, N. (1999). A short introduction to boosting. Journal-Japanese Society For Artificial Intelligence 14:771-780.
- Friedman, J. H. (2001). Greedy function approximation: a gradient boosting machine. Annals of statistics, 1189-1232.
- Kang, P., Cho, S., and MacLachlan, D. L. (2012). Improved response modeling based on clustering, under-sampling, and ensemble. Expert Systems with Applications 39(8): 6738-6753.

References

Research Papers

- Kang, S., Cho, S., and Kang, P. (2015). Multi-class classification via heterogeneous ensemble of one-class classifiers. Engineering Applications of Artificial Intelligence 43: 35-43.
- Ke, G., Meng, Q., Finley, T., Wang, T., Chen, W., Ma, W., ... & Liu, T.Y. (2017). Lightgbm: A highly efficient gradient boosting decision tree. In Advances in Neural Information Processing Systems (pp. 3146-3154).
- Liang, G., Zhu, X., and Zhang, C. (2011). An empirical study of bagging predictors for imbalanced data with different levels of class distribution. In Proceedings of the Advances in Artificial Intelligence: 213-222.
- Natekin, A., & Knoll, A. (2013). Gradient boosting machines, a tutorial. Frontiers in neurorobotics, 7:21
- Opitz, D. & Maclin, R. (1999). Popular ensemble methods: An empirical study. Journal of Artificial Intelligence Research 11: 169-198.
- Prokhorenkova, L., Gusev, G., Vorobev, A., Dorogush, A.V., & Gulin, A. (2018). CatBoost: unbiased boosting with categorical features. In Advances in Neural Information Processing Systems (pp. 6638-6648).
- Rodriguez, J. J., Kuncheva, L. I., and Alonso, C. J. (2006). Rotation forest: A new classifier ensemble method. IEEE Transactions on Pattern Analysis and Machine Intelligence 28(10): 1619-1630.
- Russakovsky, O., Deng, J., Su, H., Krause, J., Satheesh, S., Ma, S., ... & Berg, A. C. (2015). Imagenet large scale visual recognition challenge. International Journal of Computer Vision, 115(3), 211-252.
- Seni, G. and Elder, J. F. (2010). Ensemble methods in data mining: improving accuracy through combining predictions. Synthesis Lectures on Data Mining and Knowledge Discovery 2(1): 1-126.

References

Research Papers

- Shotton, J., Sharp, T., Kohli, P., Nowozin, S., Winn, J., and Criminisi, A. (2013). Decision jungles: Compact and rich models for classification. In Advances in Neural Information Processing Systems (NIPS'13): 234-242.
- Viola, P. and Jones, M. J. (2004). Robust real-time face detection. International journal of computer vision 57(2): 137-154.

References

Other Materials

- Mixture of Experts: http://people.cs.pitt.edu/~milos/courses/cs2750-Spring2011/Lectures/class18.pdf
- Li, C. A gentle introduction to gradient boosting, http://www.ccs.neu.edu/home/vip/teach/MLcourse/4_boosting/slides/gradient_boosting.pdf
- Ihler, A. (2012). Machine Learning and Data Mining: Ensemble of Leaners, http://sli.ics.uci.edu/Classes/2011W-178?action=download&upname=CS178_L10.pdf
- Corso, J. Boosting and AdaBoost: https://www.cse.buffalo.edu/~jcorso/t/CSE455/files/lecture_boosting.pdf