

Bagging Gradient-Boosted Trees for High Precision, Low Variance Ranking Models

Yasser Ganjisaffar

UC Irvine

Rich Caruana

Microsoft Research

Cristina Lopes

UC Irvine

Goals

- Learning-to-Rank
 - Approach is general; can be applied to other problems
- Very high (state-of-the-art) precision
 - Combine multiple learning methods
 - Very careful parameter tuning
- Reduce variance
 - Wrap bagging around learning process

Outline

- Three Data Sets
- Combine multiple learning methods
 - LambdaMART boosting
 - Stochastic gradient boosting (internal bagging subsampling)
 - Random forest-like feature sampling
 - External Bagging (higher accuracy, much lower variance)
- Importance of parameter tuning
- Risk of excessive parameter tuning
- Bagging to the rescue!
- Sometimes a little overfitting is good!
- Empirical results
- Importance of low variance

Three Data Sets

Data set	Queries	Pairs	Features	Labels
TD2004	75	74,146	64	{0, 1}
MQ2007	1,692	69,623	46	{0, 1, 2}
MSLR-WEB10K	10,000	1,200,192	136	{0, 1, 2, 3, 4}

- Different sizes:
 - # of queries
 - docs per query
- Different labels
- All data sets pre-folded into 5 folds

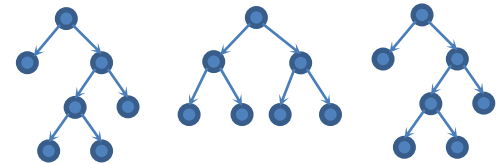
Yahoo! Learning to Rank Challenge

Team	Approach
C. Burges et al. [Microsoft Research]	Weighted Average of several Ensembles of Gradient Boosting
E Gottschalk et al. [Activision Blizzard & Data Mining Solutions]	Weighted Average of several Ensembles of Gradient Boosting and Random Forests
D. Pavlov et al. [Yandex Labs]	Bagging the Gradient Boosting
D. Sorokina [Yandex Labs]	Additive Groves (Bagging around a special form of Additive Regression)

- Top 4 teams all use ensembles of trees
- Winner used:
 - **LambdaMART** boosted tree algorithm
 - Unweighted ensemble of multiple models

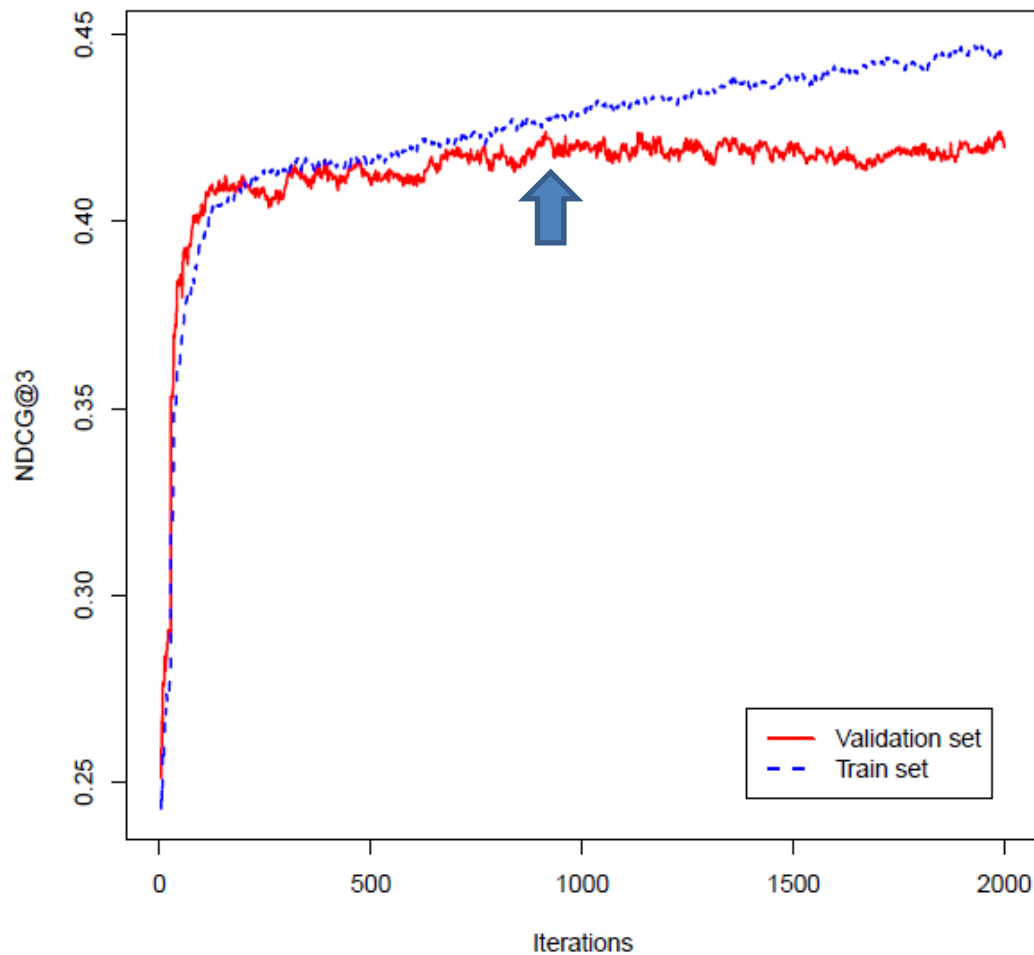
LambdaMART Parameters

- Number of Trees (iterations of boosting)
- Tree Size:
 - Maximum number of leaves
 - Min observations per leaf
- Shrinkage (learning rate)
- Diversity
 - Query subsampling (bagging-like subsampling)
 - Feature subsampling (like random forests)
- Sigmoid Parameter of Cost Function
 - Per Burges et al, set equal to learning rate

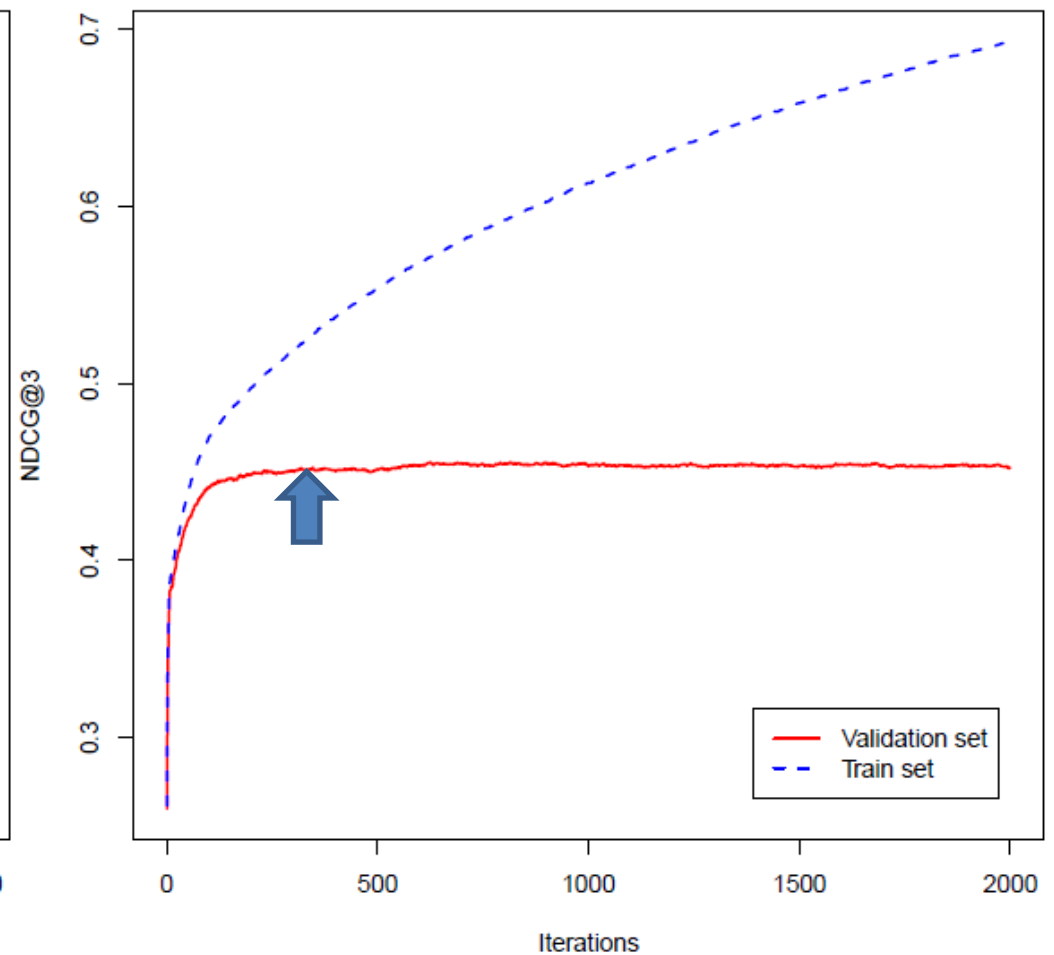


Trees: Picked on Validation Set

MQ2007 data set



MSLR-WEB10K data set



Parameter tuning for LambdaMART

(a) TD2004 and MQ2007 data sets

Parameter	Values
Max Number of Leaves	2, 4, 7, 10, 15, 20, 25
Min Percentage of Obs. per Leaf	0.12, 0.25, 0.50
Learning rate	0.05, 0.1, 0.2, 0.3
Sub-sampling rate	0.3, 0.5, 1.0
Feature Sampling rate	0.1, 0.3, 0.5, 1.0

(b) MSLR-WEB10K data set

Parameter	Values
Max Number of Leaves	10, 40, 70
Min Percentage of Obs. per Leaf	0.12, 0.25, 0.50
Learning rate	0.05, 0.1, 0.2
Sub-sampling rate	0.5, 1.0
Feature Sampling rate	0.3, 0.5, 1.0

$$1,008 \times 5 \times 3 = 15,120$$

$$162 \times 5 \times 3 = 2,430$$

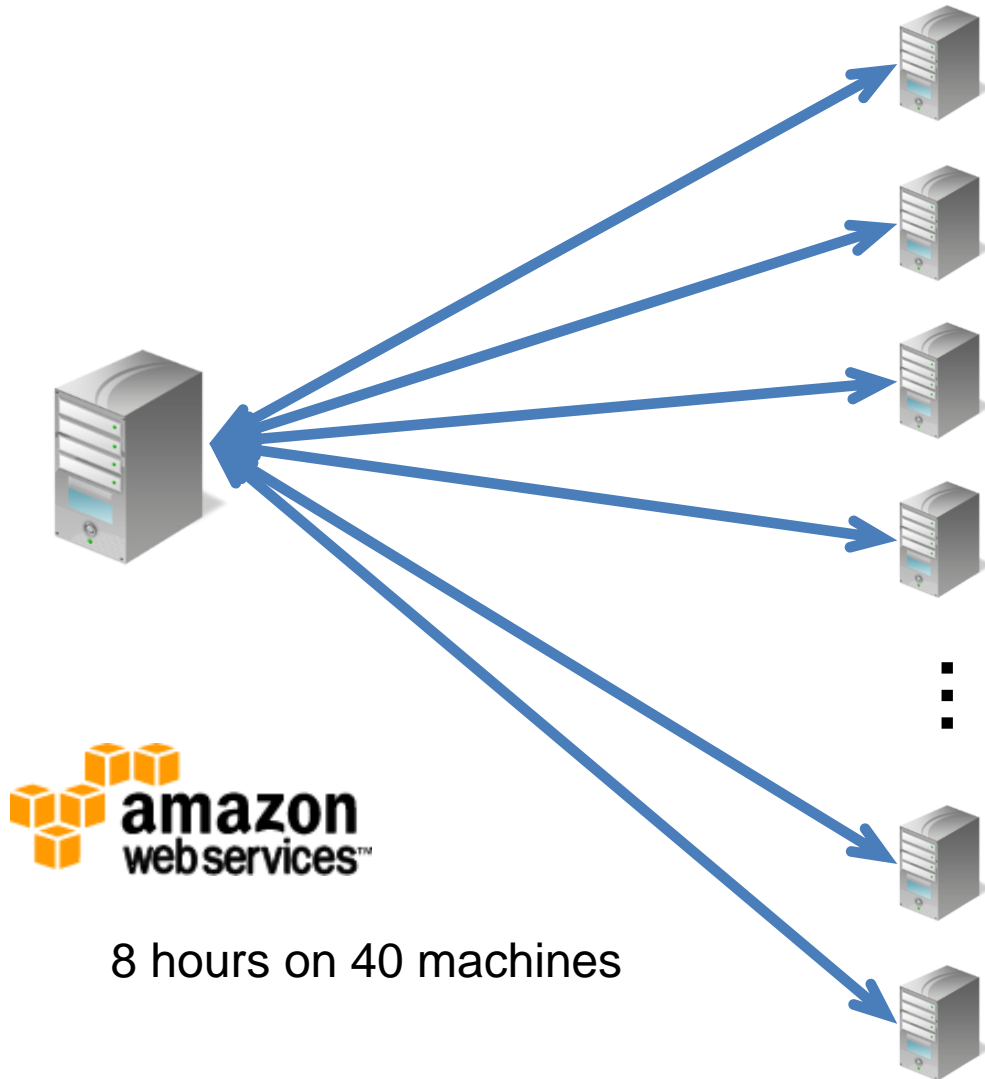
$$\text{Total Number of Experiments: } 2 \times 15,120 + 2,430 = \mathbf{32,670}$$

Trees: **5.2 million**

Splits: **120 million**

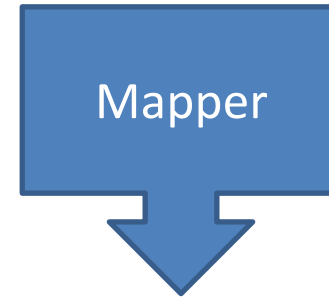
Splits/Threshold: **46 trillion**

Parameter Tuning on a MapReduce Cluster



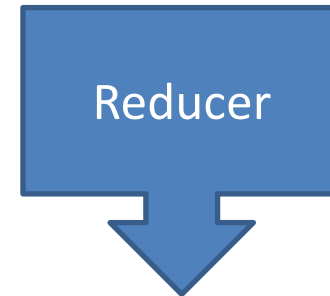
Config:1, Fold:4: Seed: 2
Config:1, Fold:4: Seed: 3

...



Config:1, Valid: 0.4567, Test: 0.4324
Config:1, Valid: 0.4534, Test: 0.4311

...

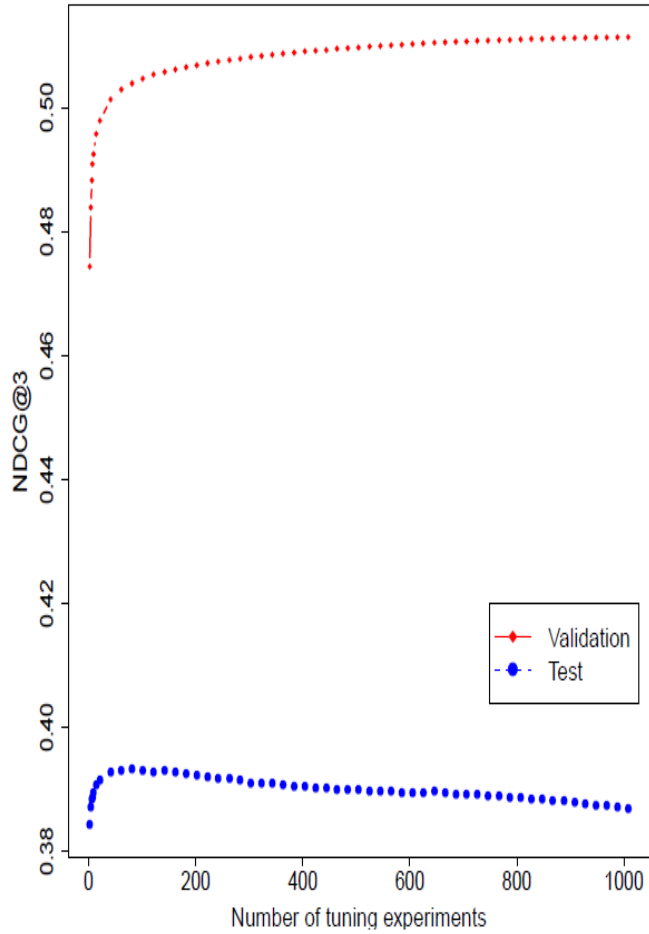


Config:1, Avg Valid: 0.4544, Avg Test: 0.4318

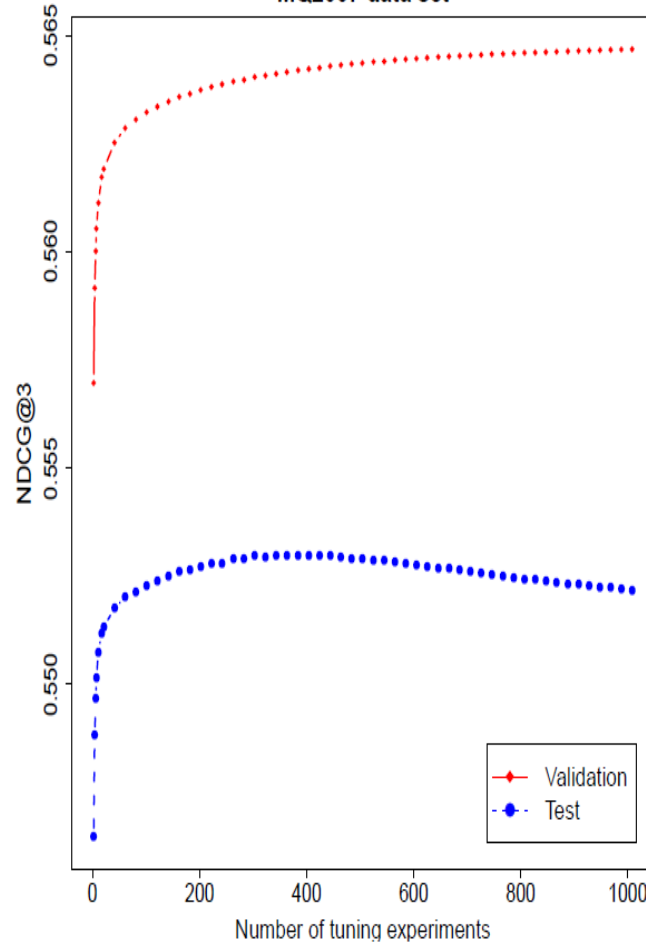
Do We Need this Many Experiments?

Do We Need this Many Experiments?

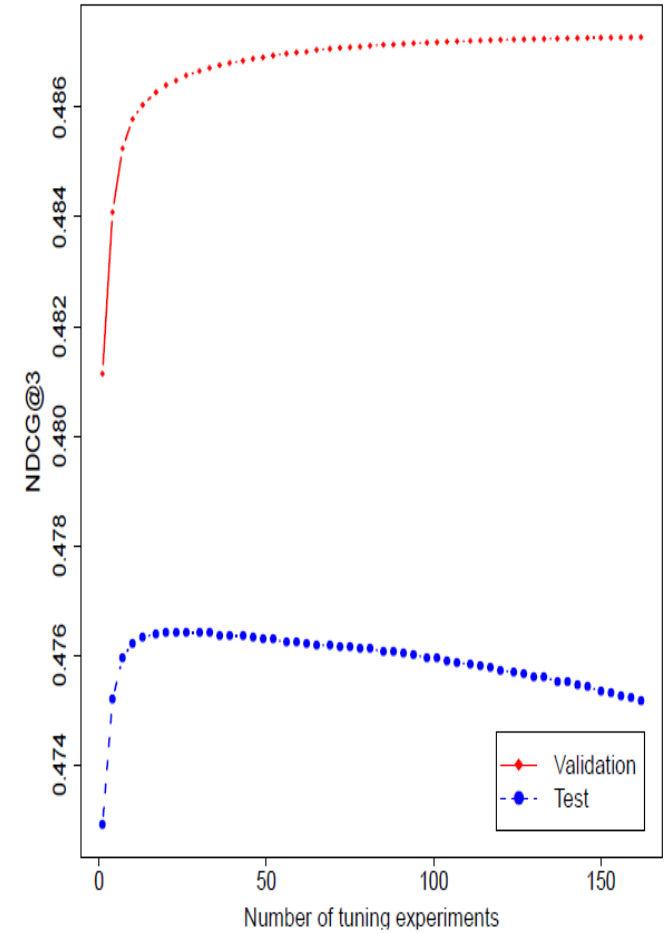
TD2004 data set



MQ2007 data set

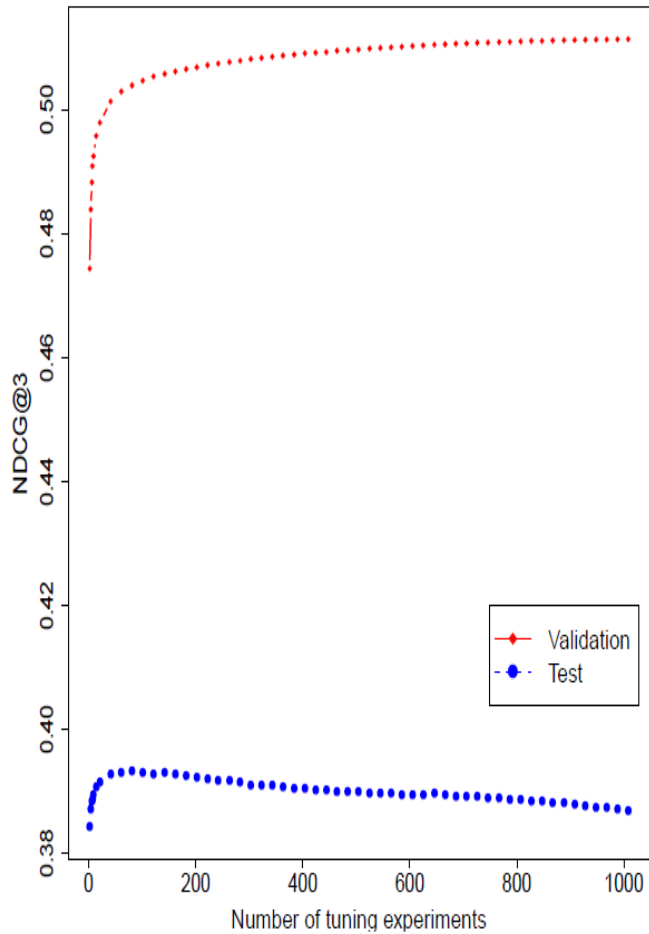


MSLR-WEB10K data set

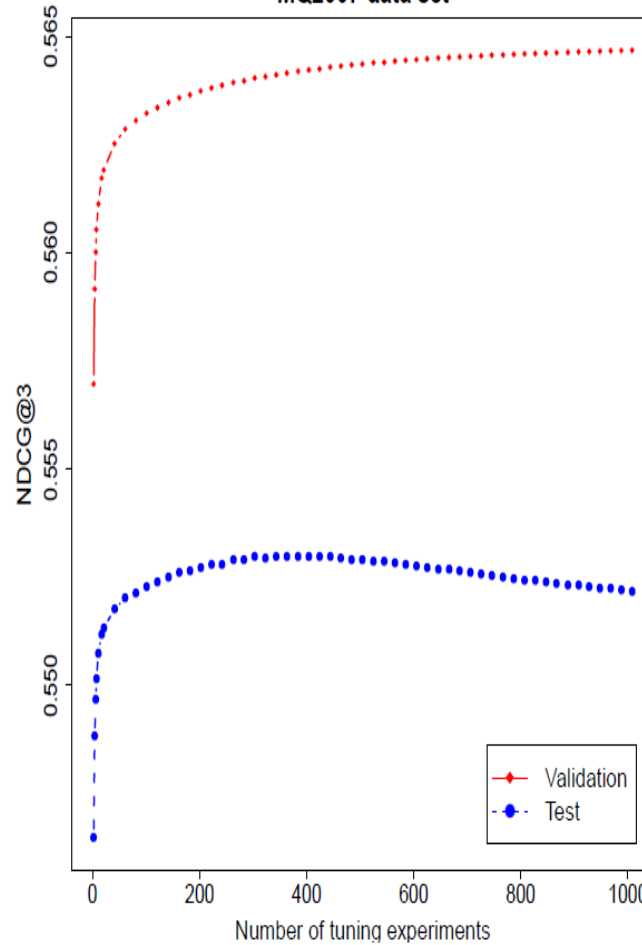


Do We Need this Many Experiments?

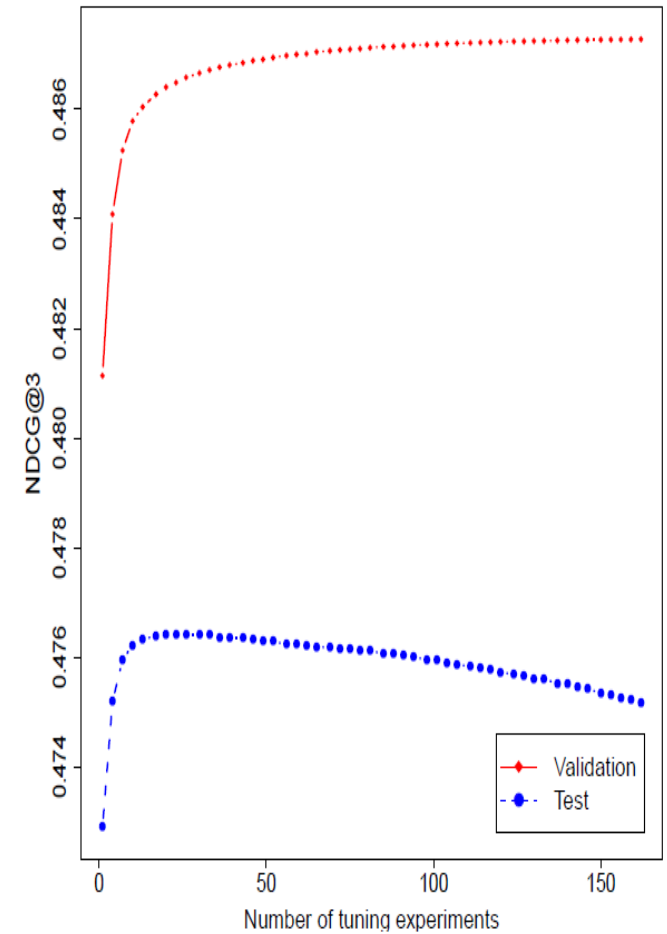
TD2004 data set



MQ2007 data set

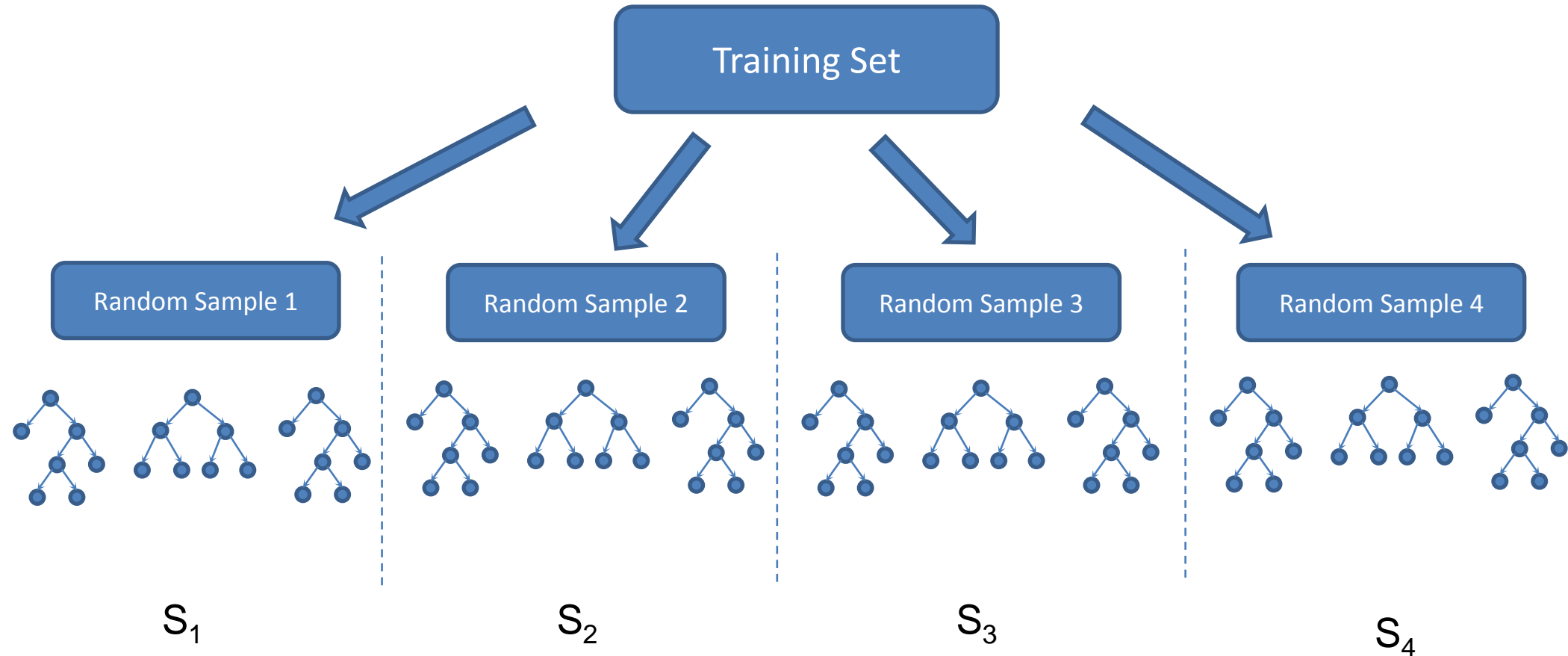


MSLR-WEB10K data set



Too much horsepower can be bad if you don't know how to handle it!

Bagging Wrapped Around LambdaMART



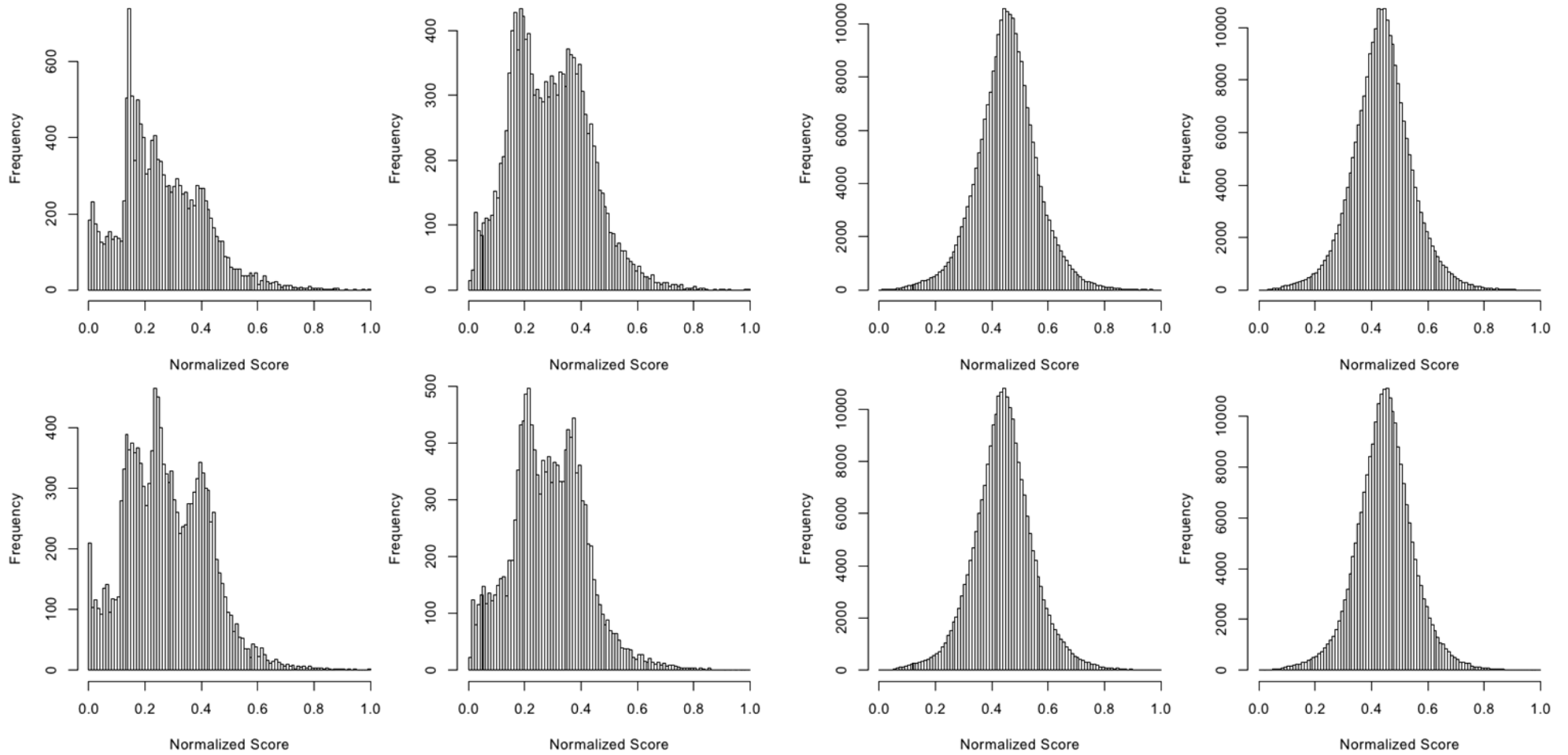
$$S = \frac{S_1 + S_2 + S_3 + S_4}{4}$$

Training time increase?

Combining Scores

- Simple Averaging
 - Different distributions of scores
 - Even on large data sets outputs of individual queries have different scales across different models
- BordaCount
 - Didn't work because of many ties
- Per-model normalization

Distribution of Scores

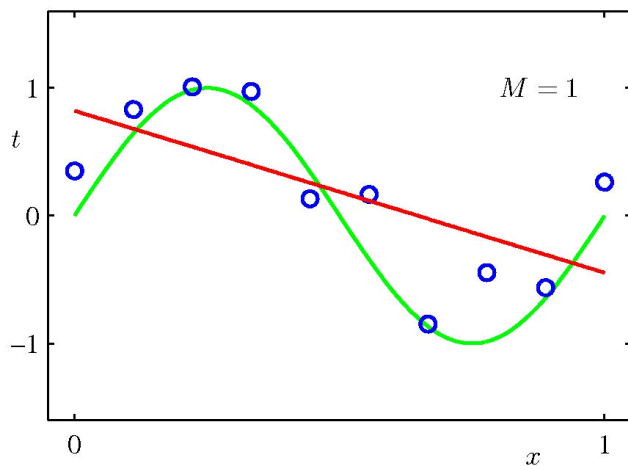


(a) MQ2007 data set

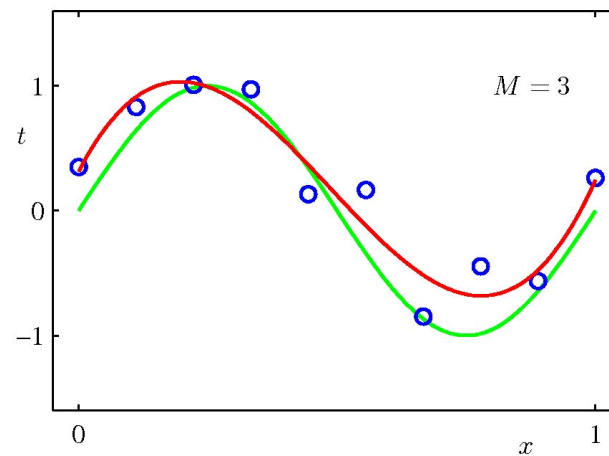
(b) MSLR-WEB10K data set

Why Bagging Increases Accuracy

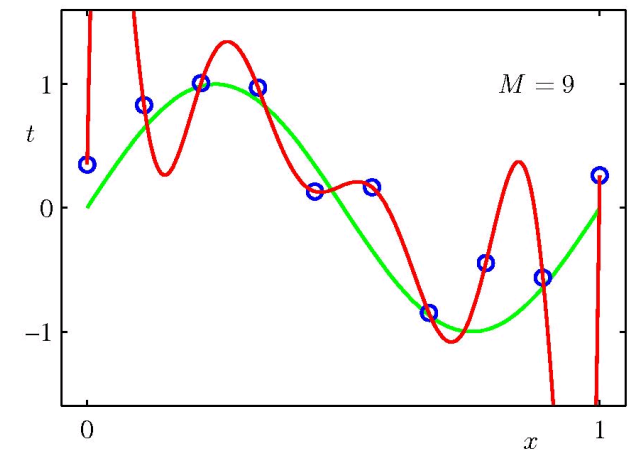
$$\text{expected loss} = (\text{bias})^2 + \text{variance} + \text{noise}$$



High bias
Low Variance



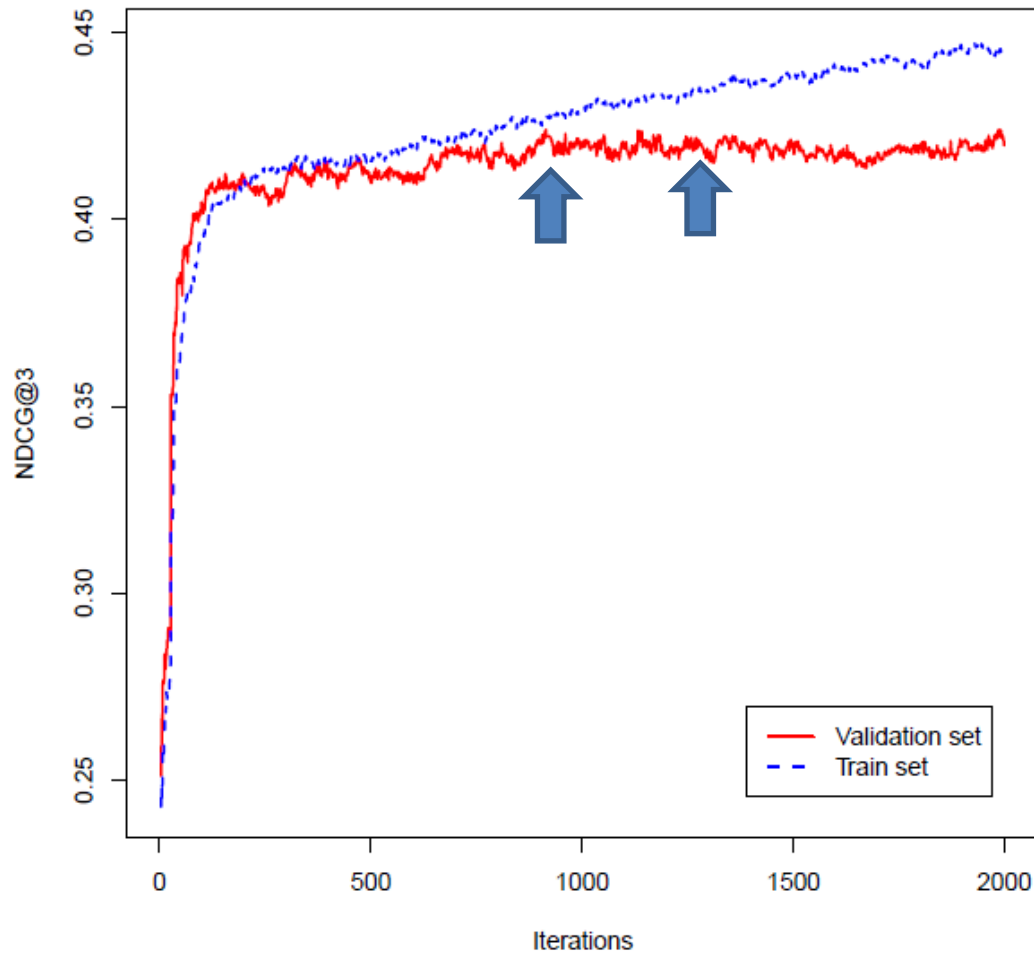
Medium Bias
Medium Variance



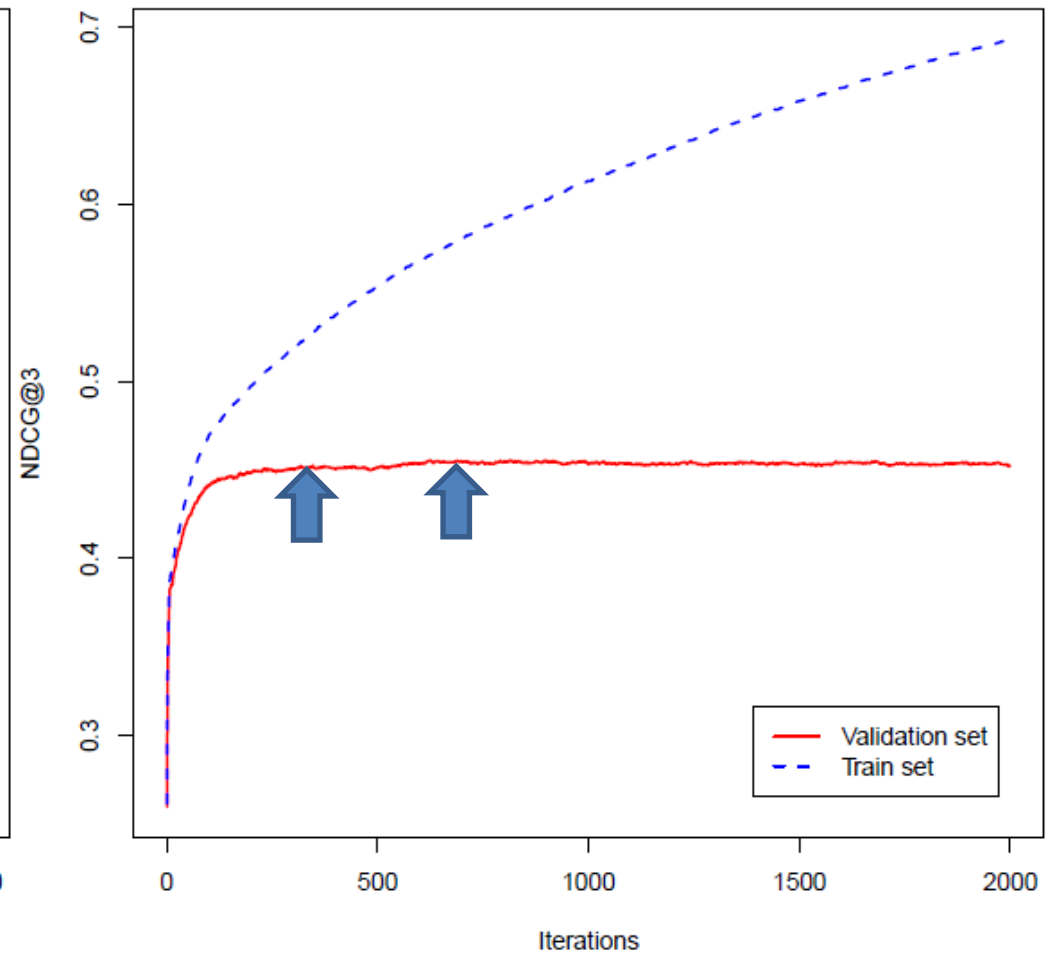
Low Bias
High Variance

Intentional Overfitting?

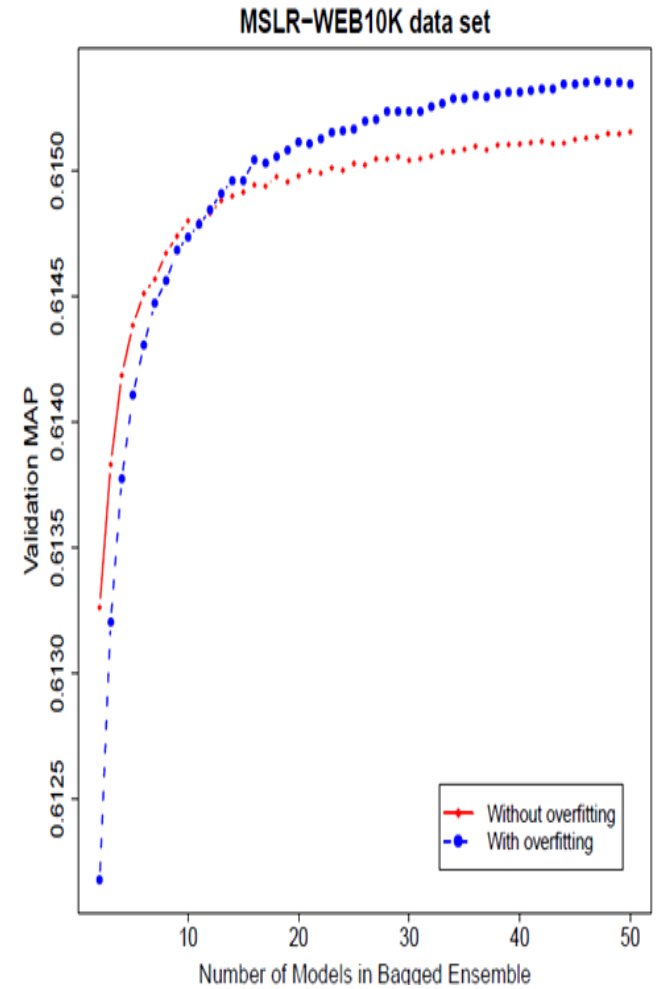
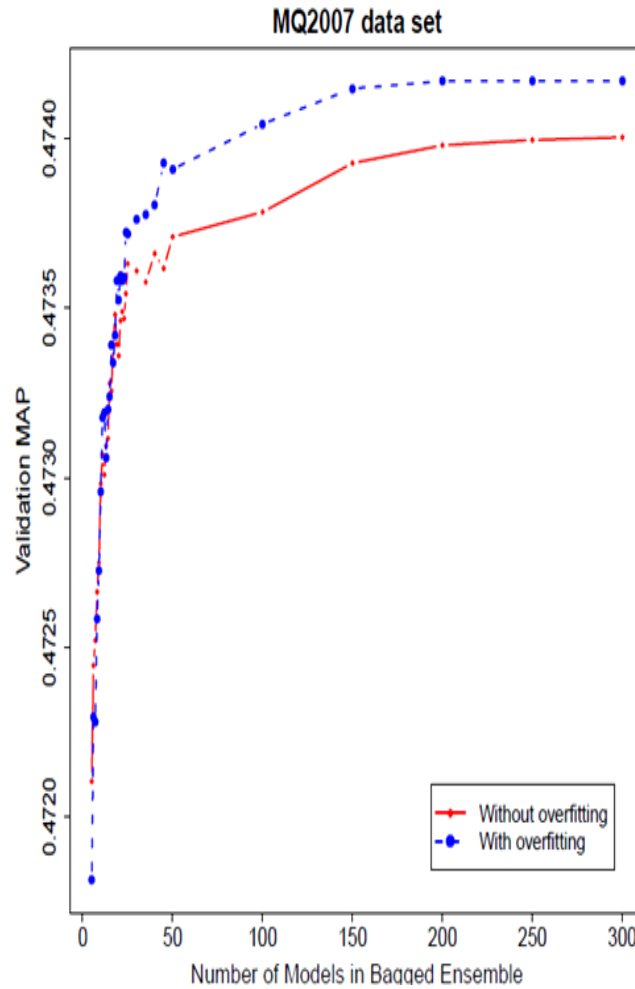
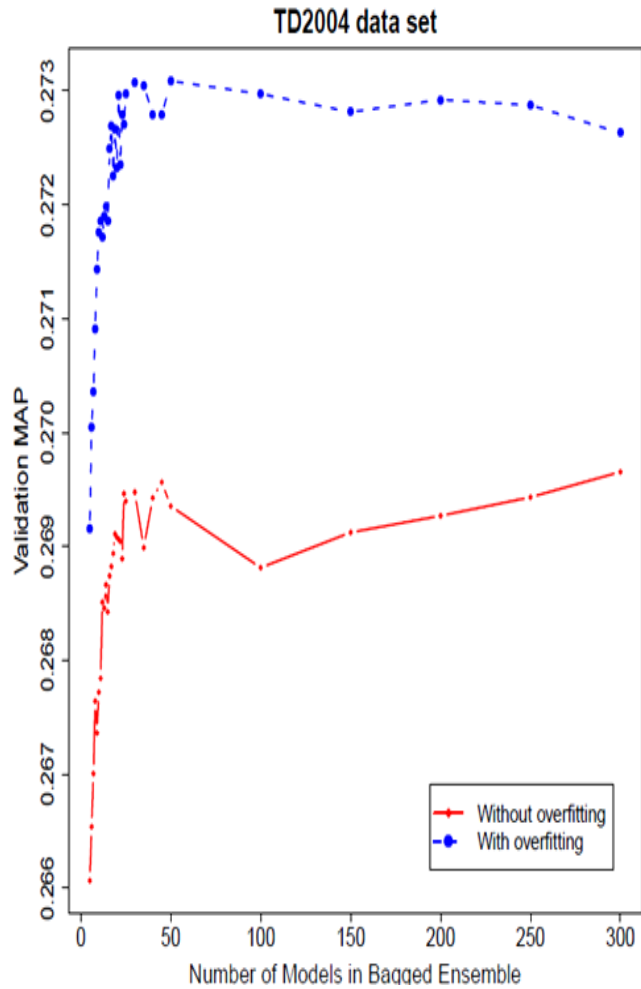
MQ2007 data set



MSLR-WEB10K data set



Intentional Overfitting!



Results: TD2004 Data Set

	NDCG@1	NDCG@3	NDCG@5	MAP
SVMmap [34]	0.2933	0.3035	0.3007	0.2049
RankSVM-Struct [20]	0.3467	0.3371	0.3192	0.2196
ListNet [9]	0.3600	0.3573	0.3325	0.2231
SmoothRank [11]	0.4000	0.3832	0.3555	0.2326
RankSVM [17]	0.4133	0.3467	0.3240	0.2237
AdaRank-MAP [33]	0.4133	0.3757	0.3602	0.2189
AdaRank-NDCG [33]	0.4267	0.3688	0.3514	0.1936
BoltzRank [30]	0.4767	0.3902	0.3635	0.2390
FRank [28]	0.4933	0.3875	0.3629	0.2388
RankBoost [13]	0.5067	0.4295	0.3878	0.2614
BagBoo [23]	0.5067	0.4080	0.3898	0.2499
LambdaMART	0.4267	0.3584	0.3266	0.2378
LambdaMART with randomization	0.4560	0.4033	0.3722	0.2513
BL-MART without overfitting	0.4947	0.4217	0.3886	0.2649
BL-MART with overfitting	0.4947	0.4270	0.3948	0.2684

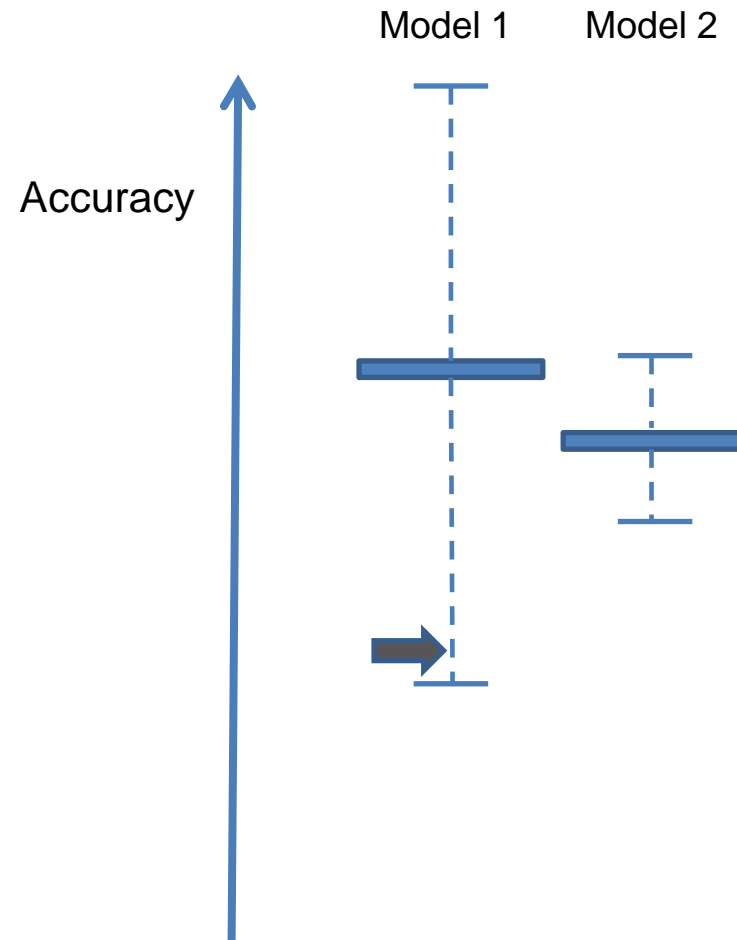
Results: MQ2007 Data Set

	NDCG@1	NDCG@3	Mean NDCG	MAP
RankSVM-Struct [20]	0.4096	0.4063	0.4966	0.4645
ListNet [9]	0.4002	0.4091	0.4988	0.4652
AdaRank-MAP [33]	0.3821	0.3984	0.4891	0.4577
AdaRank-NDCG [33]	0.3876	0.4044	0.4914	0.4602
RankBoost [13]	0.4134	0.4072	0.5003	0.4662
CRR [25]	–	–	0.5000	0.4660
BagBoo [23]	0.4071	0.4176	–	0.4676
LambdaMART	0.4147	0.4119	0.5011	0.4660
LambdaMART with randomization	0.4137	0.4157	0.5035	0.4684
BL-MART without overfitting	0.4197	0.4217	0.5079	0.4726
BL-MART with overfitting	0.4200	0.4224	0.5093	0.4731

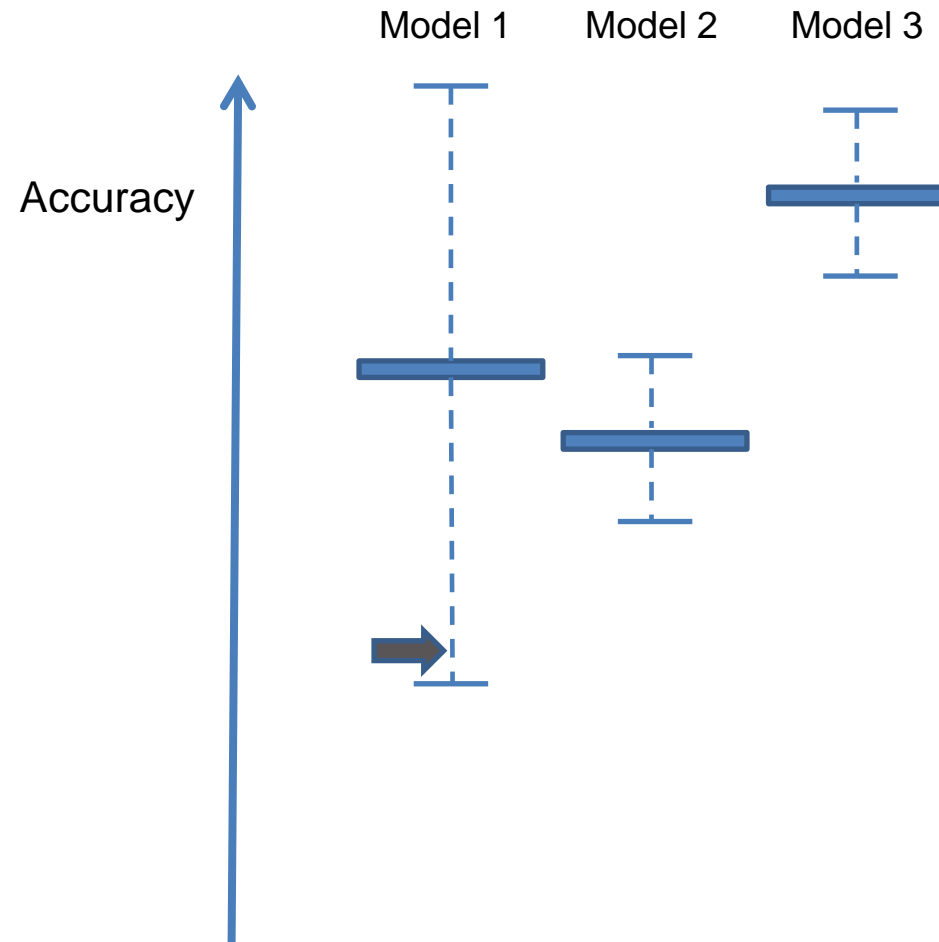
Results: MSLR-WEB10K data set

	NDCG@1	NDCG@3	Mean NDCG	MAP
LambdaMART	0.4580	0.4467	0.5693	0.3670
LambdaMART with randomization	0.4628	0.4487	0.5706	0.3684
BL-MART without overfitting	0.4640	0.4514	0.5720	0.3696
BL-MART with overfitting	0.4642	0.4516	0.5729	0.3705

Model Variance



Model Variance



Importance of Low Variance

- High variance often reduces expected performance:

$$\text{expected loss} = (\text{bias})^2 + \text{variance} + \text{noise}$$

- Easier to judge if one model better than another
 - Less risk when selecting model to ship
 - Easier to judge incremental improvement from new features
- Lower variance leads to more rapid improvement

Variance Analysis

	NDCG@1		NDCG@3		Mean NDCG		MAP	
	Mean	Variance	Mean	Variance	Mean	Variance	Mean	Variance
LambdaMART	0.4484	18×10^{-6}	0.4395	9.1×10^{-6}	0.5640	1.2×10^{-6}	0.3657	0.9×10^{-6}
LambdaMART with randomization	0.4492	22×10^{-6}	0.4421	5.4×10^{-6}	0.5647	1.4×10^{-6}	0.3665	1.3×10^{-6}
BL-MART without overfitting	0.4516	10×10^{-6}	0.4468	7.8×10^{-6}	0.5675	1.5×10^{-6}	0.3690	0.9×10^{-6}
BL-MART with overfitting	0.4528	7×10^{-6}	0.4471	4.4×10^{-6}	0.5686	0.8×10^{-6}	0.3703	0.5×10^{-6}

- Randomization improves accuracy, but also increases variance
- Bagging significantly reduces variance and increases accuracy!
- Bagging rescues us from parameter-level overfitting
- Intentional overfitting improves accuracy further
- Surprisingly, overfitting also reduces variance...

Conclusions

- Combining multiple methods is good
 - Approach is general, not just for learning-to-rank
- Better accuracy (NDCG, MAP, ...) on 10 of 12 comparisons
- Parameter tuning is critical:
 - Too much can hurt accuracy (2nd validation set, bagging, ...)
- Bagging to the rescue:
 - Saves us from overfitting (intentional and unintentional)!
 - Increased average accuracy 2.6%
 - Reduced variance almost 50% across all metrics
- MapReduce makes all of this practical

Thank You!

Questions?