Intro to machine learning

Greg Landrum, ETH Zurich

SCS Spring School on Digital Chemistry

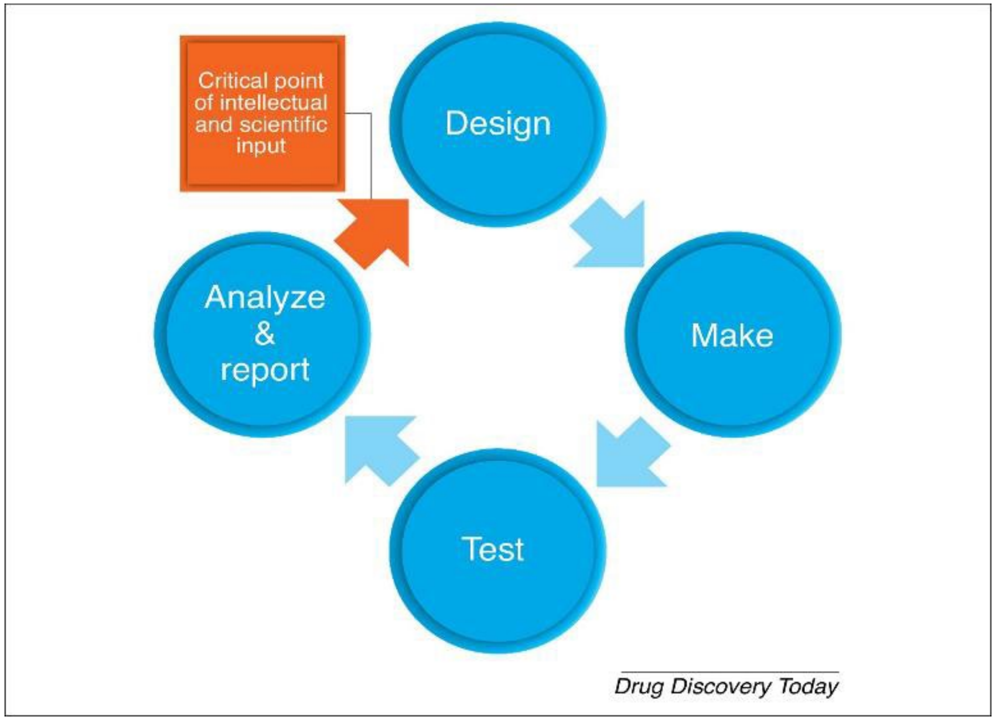
April 2023

Outline

- Introduction
- QSAR
- ML methods (linear models, decision trees, random forest) + validation

How does this fit into drug discovery?

- Drug discovery teams typically follow an iterative approach as they try to discover and optimize drug candidates
- A useful way of thinking about these iterations is the "Design-Make-Test-Analyze" cycle.
- Computer-aided drug design (CADD) is primarily focused on the Design and Analyze stages
- Chemical Data Science runs throughout



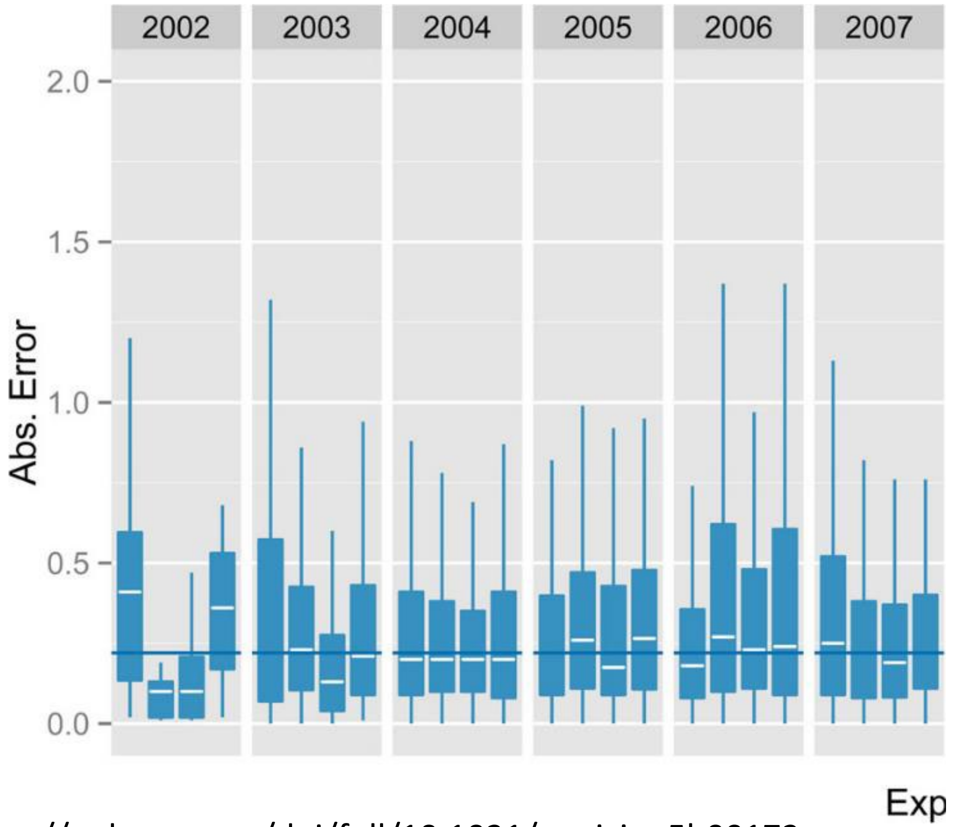
https://doi.org/10.1016/j.drudis.2018.09.016

Chasing that last digit of accuracy

- There are a lot of papers introducing new descriptors/algorithms which distinguish themselves by showing small improvements in accuracy on some dataset (or set of datasets)
- Be *very* careful when selecting methods based on things like this as real-world performance may be completely different.

Chasing that last digit of accuracy

- There are a lot of papers introducing new descriptors/algorithms which distinguish themselves by showing small improvements in accuracy on some dataset (or set of datasets)
- Be *very* careful when selecting methods based on things like this as real-world performance may be completely different.

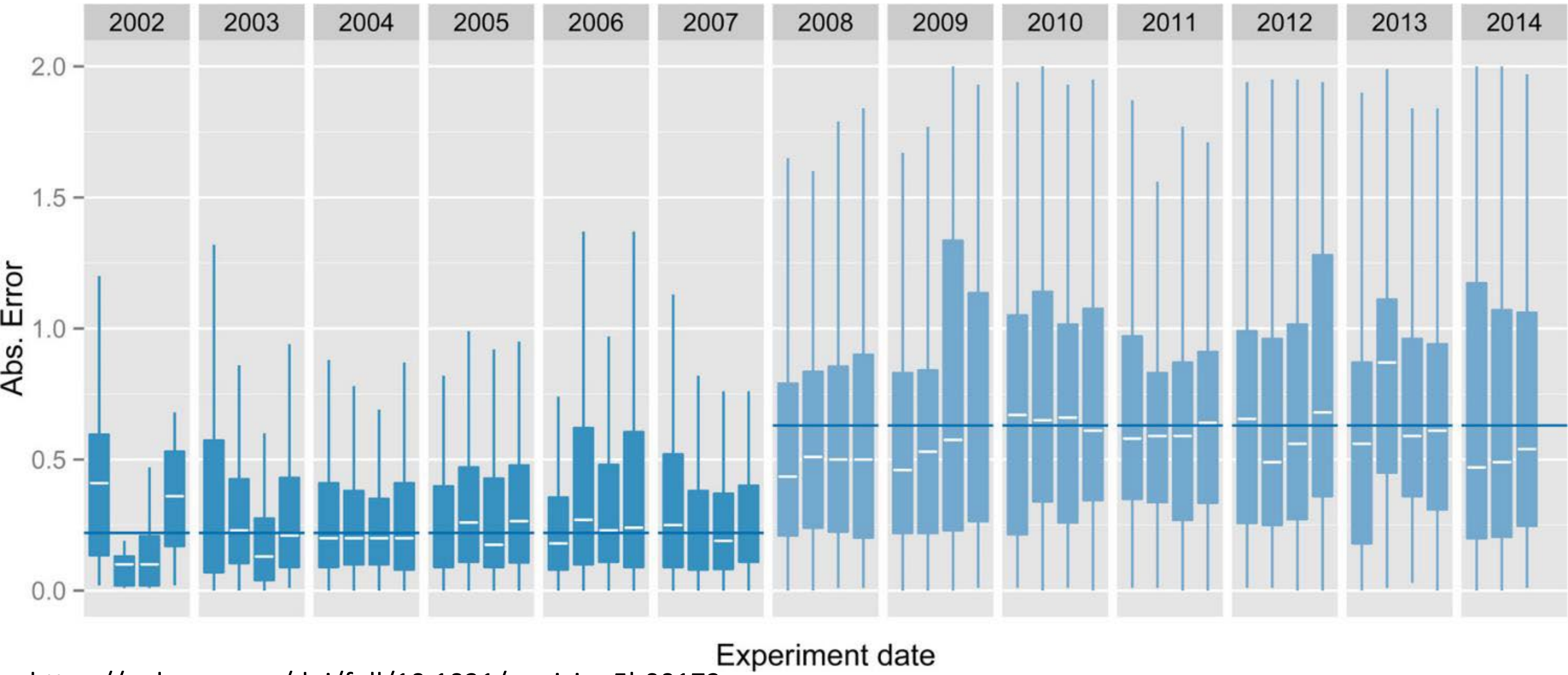


Errors for a pKa model built on Novartis internal data.

https://pubs.acs.org/doi/full/10.1021/acs.jcim.5b00172

Chasing that last digit of accuracy

- There are a lot of papers introducing new descriptors/algorithms which distinguish themselves by showing small improvements in accuracy on some dataset (or set of datasets)
- Be *very* careful when selecting methods based on things like this as real-world performance may be completely different.



https://pubs.acs.org/doi/full/10.1021/acs.jcim.5b00172

What’s the idea?

- We’d like to be able to use computational models that can (relatively) quickly predict properties which are slow(er) or expensive to measure
- Some examples of things we may want to model:
  - Physicochemical properties: solubility, pKa, stability, etc.
  - Bioactivity, on-target or off-target (selectivity)
  - Absorption, Distribution, Metabolism and Excretion (ADME) properties: membrane permeability, bioavailability, half-life, etc.
  - Safety, toxicity

Classes of models

- Global
  - Trained on a broad chemical space
  - Large amounts of training data
  - Generally expected to work for “any” new compound
- Local
  - Trained on a specific region of chemical space (typically a few scaffolds)
  - Small amounts of training data
  - Unlikely to work for arbitrary compounds

This is a simplified view, and it’s not always easy to distinguish between the categories, but it’s a useful way of thinking about things.

Some uses of models

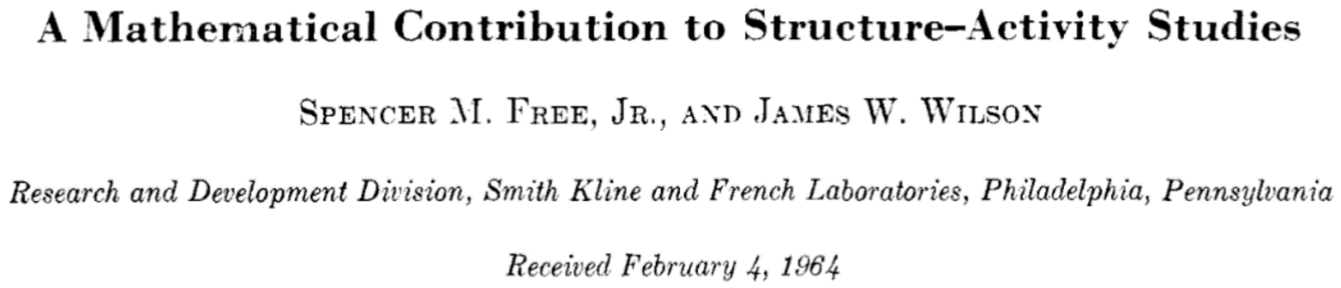
- What should I buy/make next for the next iteration in my project?
- Should I send this compound to this assay?
- We're starting a new project and have some data from the literature or from related projects and want to figure out which compounds from our collection we should screen first
- We're designing a new screening library and only want to include compounds with good properties, which are likely to be stable in solution, and which cover as much chemical space as possible

QSAR

- Quantitative Structure–Activity Relationship: a mathematical model connecting chemical structure to some activity value

- A couple seminal papers:

https://doi.org/10.1021/jm00334a001

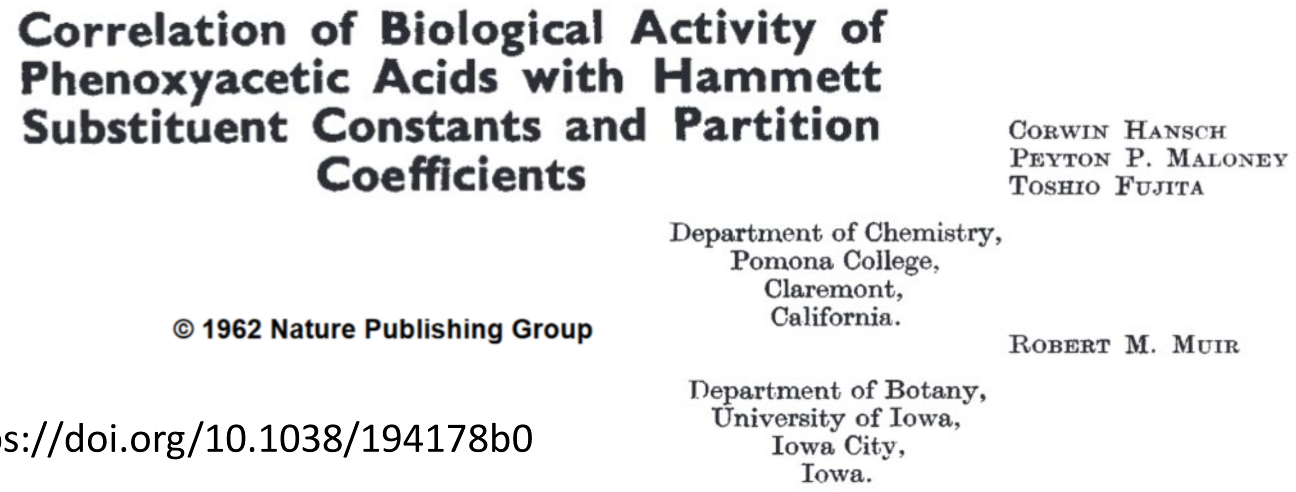


A Mathematical Contribution to Structure–Activity Studies

SPENCER M. FREE, JR., AND JAMES W. WILSON

Research and Development Division, Smith Kline and French Laboratories, Philadelphia, Pennsylvania

Received February 4, 1964



Correlation of Biological Activity of Phenoxyacetic Acids with Hammett Substituent Constants and Partition Coefficients

CORWIN HANSCH
PEYTON P. MALONEY
TOSHIO FUJITA

Department of Chemistry,
Pomona College,
Claremont,
California.

ROBERT M. MUIR

Department of Botany,
University of Iowa,
Iowa City,
Iowa.

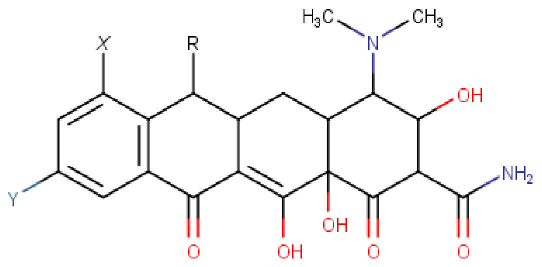
© 1962 Nature Publishing Group

https://doi.org/10.1038/194178b0

Early QSAR

- Strong focus on interpretable models
- The chemical features used tend to be groups : side-chains, aromatic substituents, etc.
- The model was often based on some form of linear regression to derive the contributions of the chemical features to the property of interest
- Part of the goal is often to learn something from the model itself

Example: Free-Wilson analysis



	R		X			Y			
	H	CH3	NO2	Cl	Br	NO2	NH2	CH3CONH	activity
III	1		1			1			60
IV	1			1		1			21
V	1				1	1			15
VI	1			1			1		525
VII	1				1		1		320
VIII	1		1				1		275
IX		1	1				1		160
X		1	1					1	15
XI		1			1		1		140
XII		1			1			1	75

From: https://doi.org/10.1021/jm00334a001

Example: Free-Wilson analysis

- Results:

Group contributions

R		**X**		**Y**	
H	75	Cl	84	NH2	123
CH3	-112	Br	-16	CH3CONH	18
		NO2	-26	NO2	-218

Overall average: 161

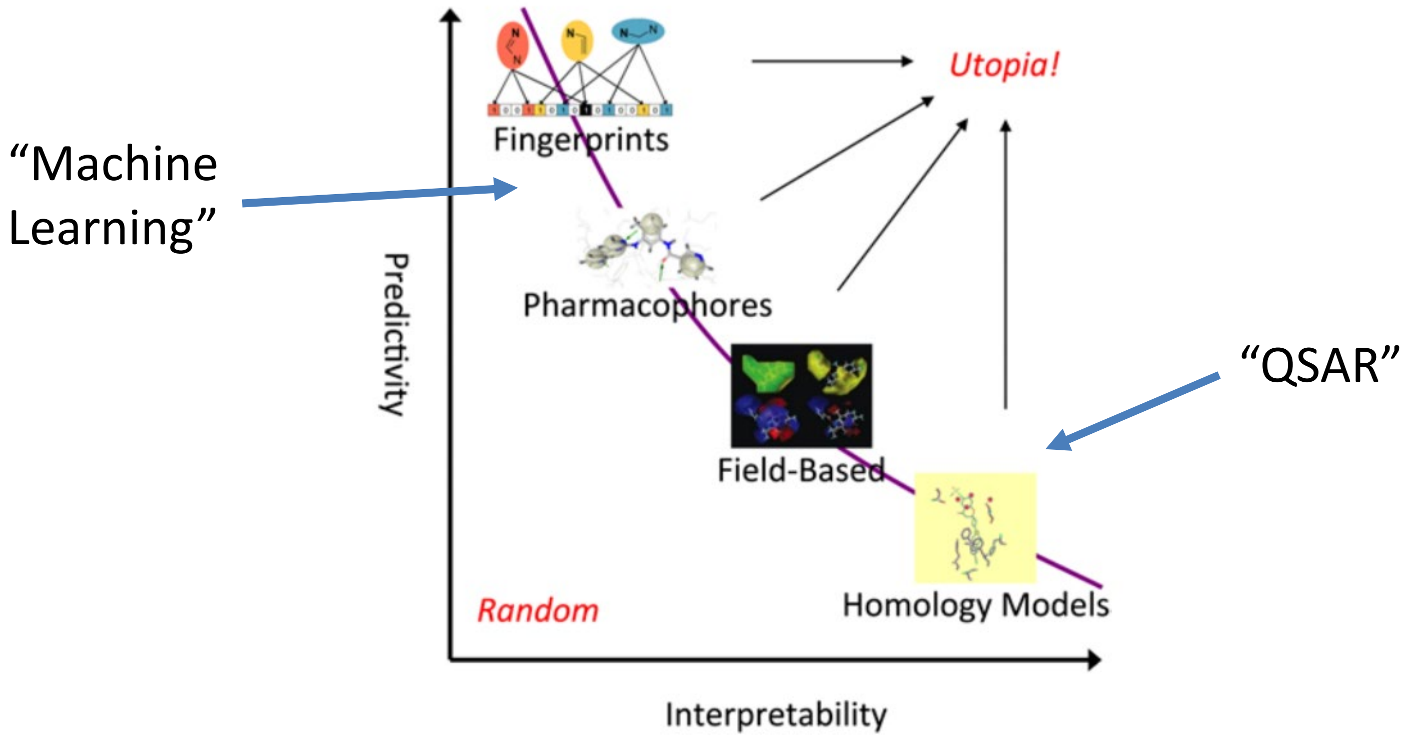
The model accounts for 90.6% of the variance in the data

- Making predictions:
  - Compound VI: R=H, X=Cl, Y=NH2: pred = 161 + 75 + 84 + 123 = 443 (actual = 525)
  - Compound VII: R=H, X=Br, Y=NH2: pred = 161 + 75 - 16 + 123 = 343 (actual = 275)
  - New compound: R=H, X=Cl, Y=CH3CONH: pred = 161 + 75 + 84 + 18 = 338

From: https://doi.org/10.1021/jm00334a001

“Modern” QSAR

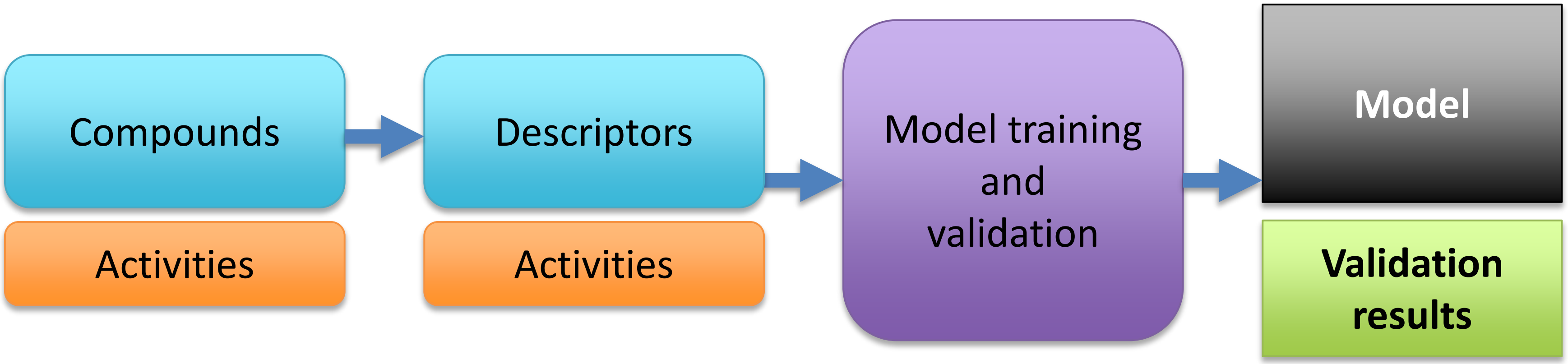
- Using data + chemical features to build a mathematical/statistical model to predict activity is equivalent to machine learning
- (opinion): these days things labeled “QSAR” tend to have more of an emphasis on interpretability and/or learning from the model, otherwise there’s not really a difference



From N. Brown “In Silico Medicinal Chemistry: Computational Methods to Support Drug Design” 2016
https://doi.org/10.1039/9781782622604

Building and validating a model

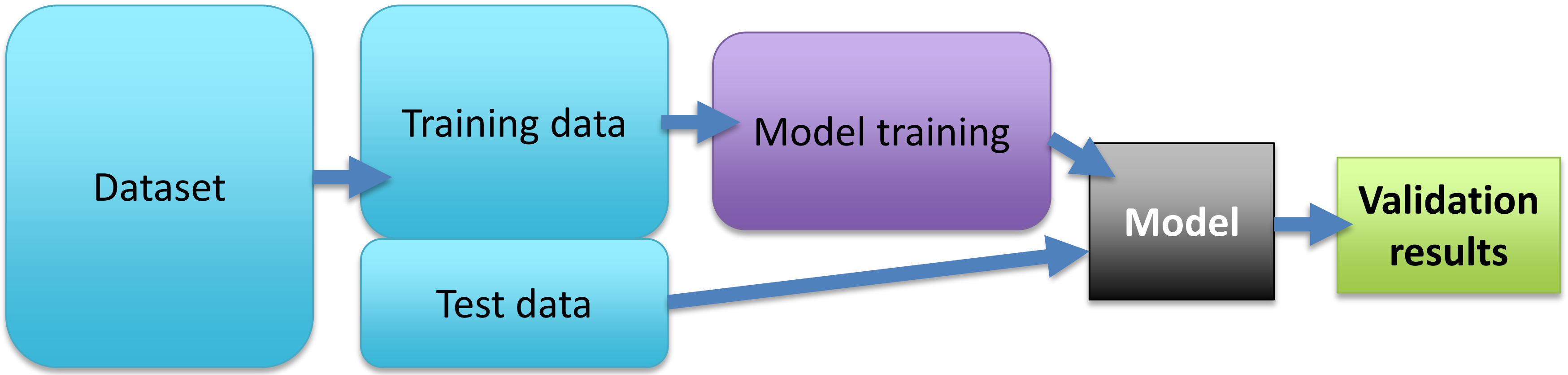
- General flow:



- We already talked about descriptors (fingerprints, physicochemical properties, topological properties, etc.)
- Once we have a model we're happy with we can use it to make predictions for new compounds

Model training and validation

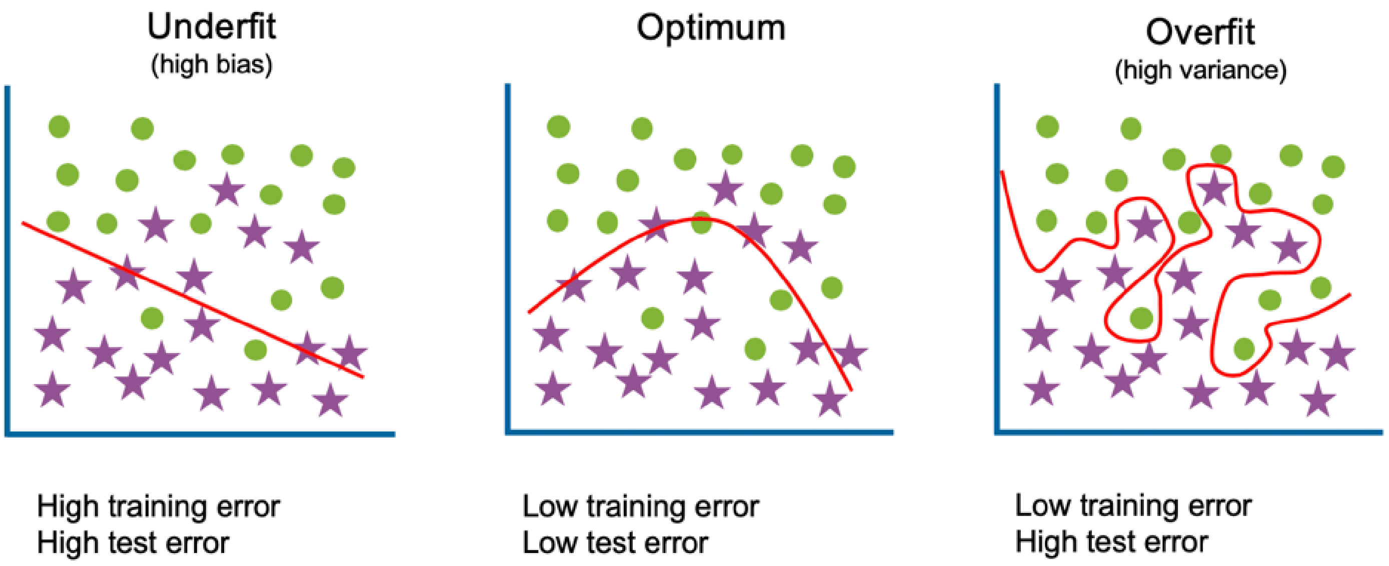
- General flow:



1. Split data into training and test sets
2. Train the model and optimize hyperparameters
3. Generate predictions on the test set and use those to generate validation results

Why all the fuss about validation?

- We're often using amazingly powerful and flexible algorithms to fit our data.
- These can, and do, overfit (memorize) the training data
- This overfitting leads to good training results, but poor performance on new data points



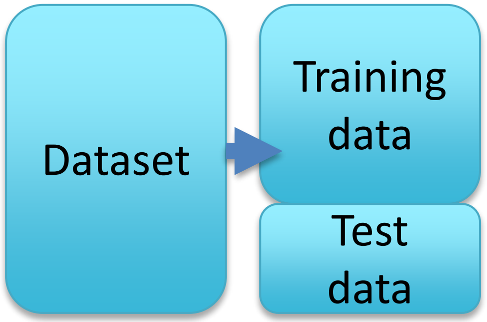
https://www.ibm.com/cloud/learn/overfitting

Splitting the data

- We need the test set to see how the model does on data it hasn't seen before.
- Ideally the test set should resemble the data that the model will actually be used for.
- Standard approaches from machine learning can be surprisingly misleading

- Standard statistical/ML approaches assume that the model will be used to predict new data which is drawn from the same distribution as the training data

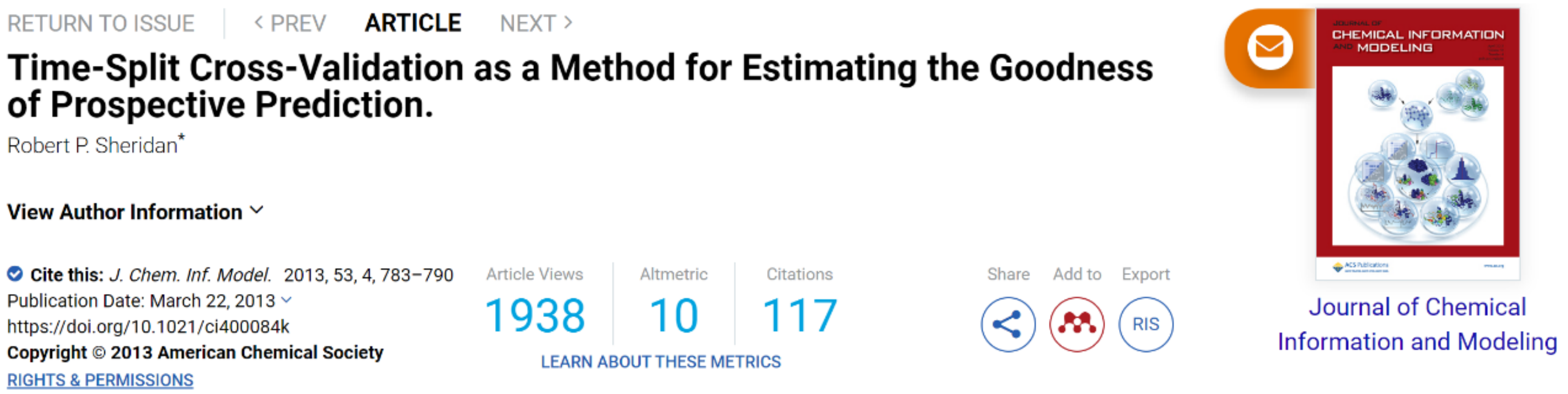
- This can be a *really* poor assumption for models used on drug-discovery projects: the whole goal is often to explore new chemical space



Time-split validation

- The gold standard for many model use cases
- Idea:
  - ➢ Sort the dataset by time
  - ➢ Train using the first X% (often something like 80%)
  - ➢ Validate using the remaining data
- This is testing the model exactly how it will be used

- Requires data points with attached dates
- Unfortunately, this is a big problem with public data



RETURN TO ISSUE | < PREV ARTICLE NEXT >

Time-Split Cross-Validation as a Method for Estimating the Goodness of Prospective Prediction.

Robert P. Sheridan*

View Author Information

Cite this: *J. Chem. Inf. Model.* 2013, 53, 4, 783–790
Publication Date: March 22, 2013
https://doi.org/10.1021/ci400084k
Copyright © 2013 American Chemical Society
RIGHTS & PERMISSIONS

Article Views	Altmetric	Citations
1938	10	117

LEARN ABOUT THESE METRICS

Journal of Chemical Information and Modeling

https://doi.org/10.1021/ci400084k

Some other data splitting strategies

- X-fold cross validation:
  1. Divide the dataset into X equally sized pieces
  2. Pick one piece to be the test set, train with the rest
  3. Repeat step 2 for each of the X pieces and combine the results (typically using an average or a median)
- Random splitting:
  1. Randomly split the dataset into a train and test set.
  2. Repeat step 1 multiple times to control for random variations
- Leave-cluster-out validation:
  1. Cluster the dataset
  2. Select some clusters to be the test set and train on the rest
  3. (sometimes) repeat multiple times

  Here there are different strategies for how to select the clusters to use as the test set.
- Leave-scaffold-out validation:

  Same idea as leave-cluster-out, but assign the compounds to scaffolds instead of clusters

Data splits and activity distributions

A subtlety here:

The splitting strategies frequently also take the distribution of activity values into account. This is usually called "stratified" sampling

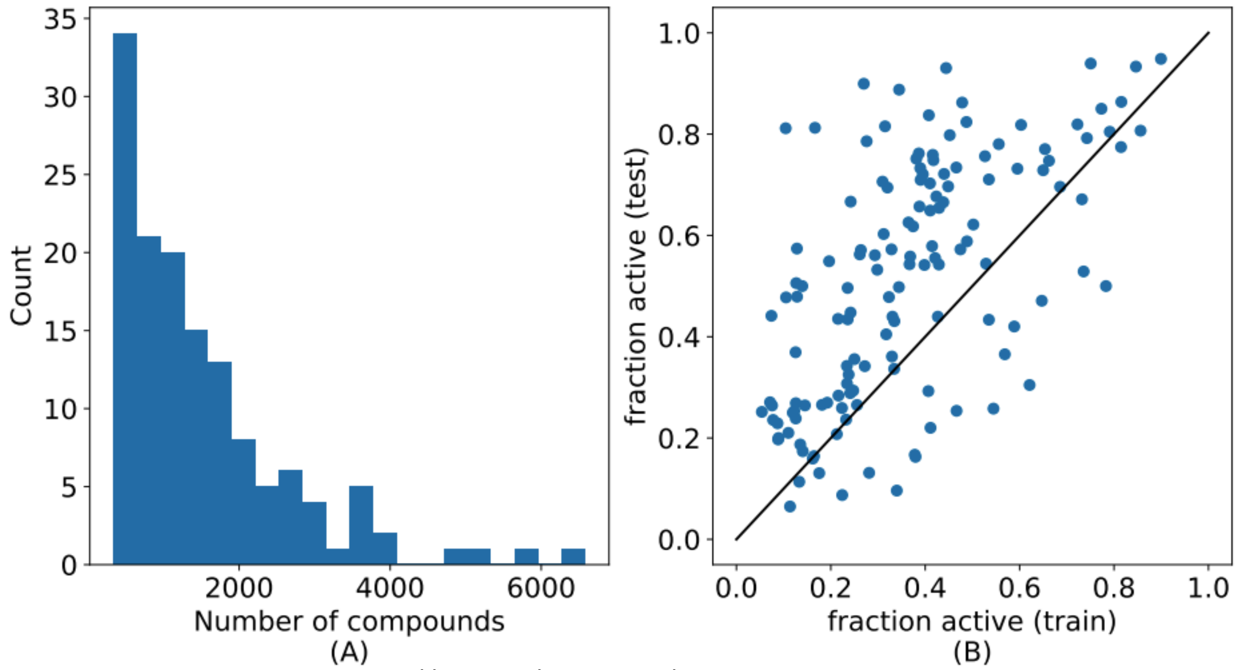
Data splits and activity distributions

A subtlety here:

The splitting strategies frequently also take the distribution of activity values into account. This is usually called "stratified" sampling

It's never easy:

In real world datasets, the distribution of activity values often changes over time.



From: https://doi.org/10.26434/chemrxiv-2023-x9pjf

Which data splitting strategy should I use?

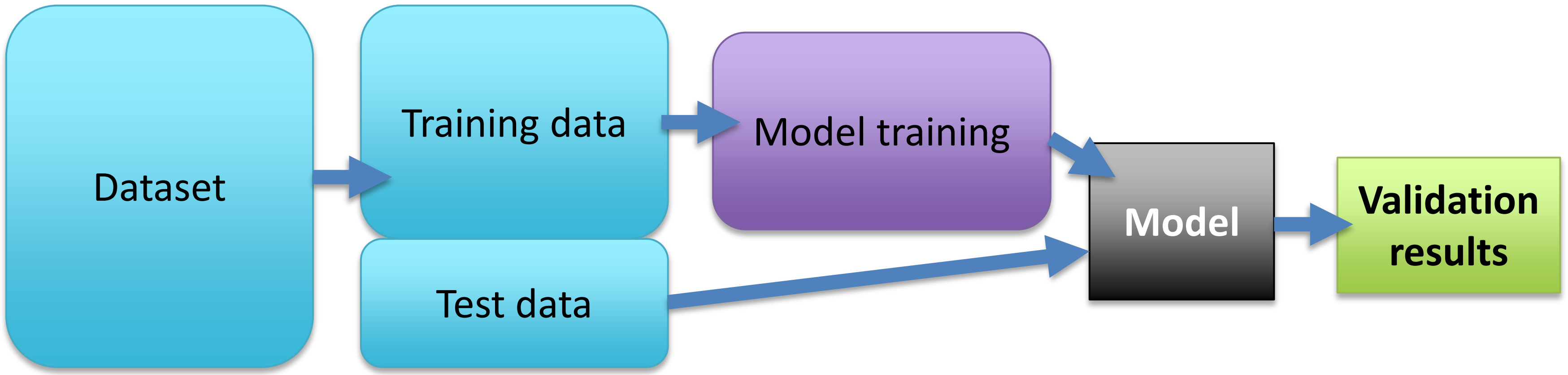
- Time-split validation
- X-fold cross validation / Random splitting
- Leave-cluster-out / Leave-scaffold-out

There's no universal right answer to this question

A general guideline: try to test the model the way that you are going to use it

Model training and validation

- General flow:



Now let’s talk about the models themselves!

Classification vs Regression

- First decision: should we be building a classification model or a regression model?

- Classification: assigns points to classes
  - Examples: active / inactive, mutagen / non-mutagen, soluble / insoluble
  - Can also be non-binary: very active / moderately active / inactive
  - Classes are categorical, not numerical: CDK1 / CDK2 / CDK3 / CDK4

- Regression: generate a numerical prediction
  - Examples: solubility, Ki, logP

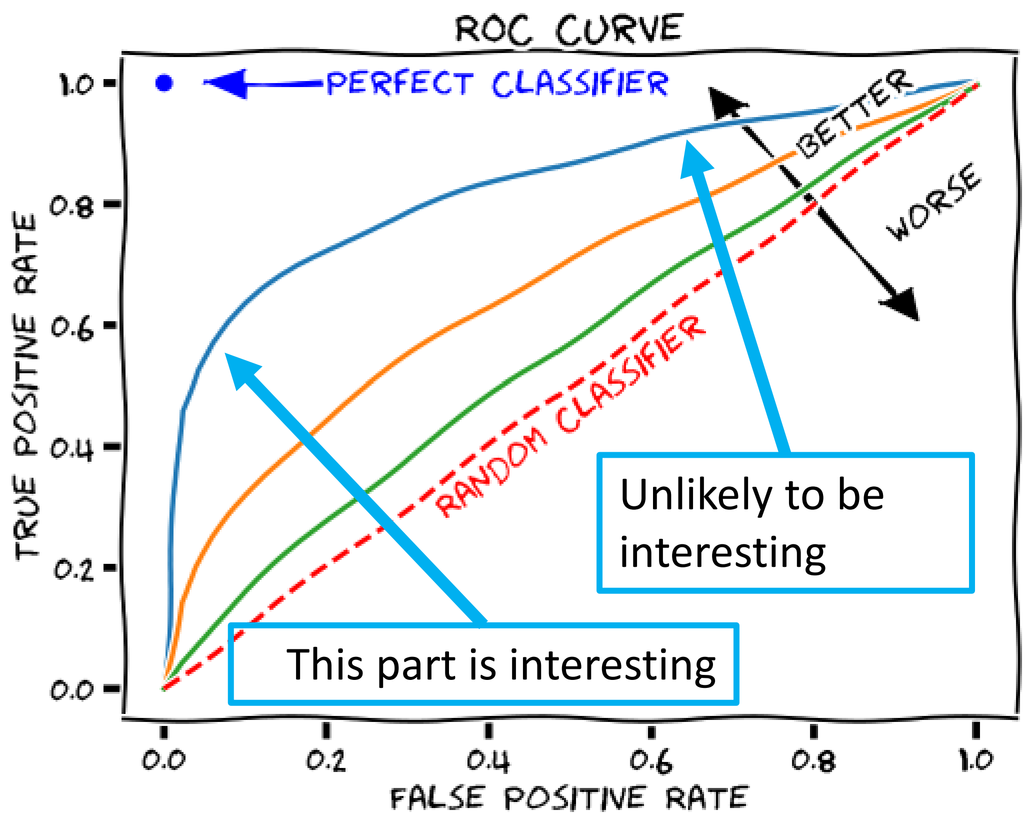
- As you might expect: there’s no general “right” answer for which type of model should be used for a given problem. It depends on the data and how the model is going to be used

- Validation strategies are different for the different model types

Evaluating the performance of a classifier

Evaluating the performance of a classifier

- ROC curve/AUROC and early enrichment methods



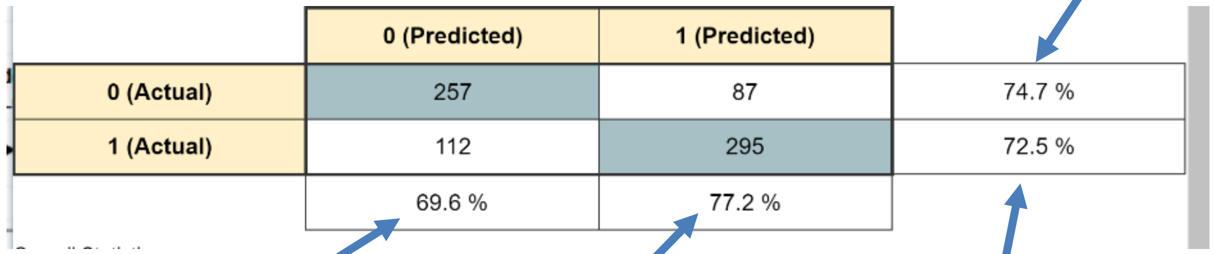
https://commons.wikimedia.org/wiki/File:Roc-draft-xkcd-style.svg

- Area Under the ROC curve (AUROC): measures overall ranking performance
- Early enrichment measures – BEDROC, RIE, EF5, EF1, etc. – measure the performance at the top of the ranking

The confusion matrix

- Summarize predictions and actual values for each class:

Fraction of class 0 predicted correctly



	0 (Predicted)	1 (Predicted)	
0 (Actual)	257	87	74.7 %
1 (Actual)	112	295	72.5 %
	69.6 %	77.2 %	

Accuracy of class 0 predictions

Accuracy of class 1 predictions

Fraction of class 1 predicted correctly

Metrics from the confusion matrix

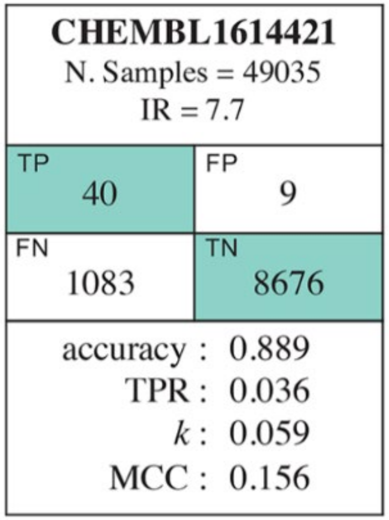
TN	**FP**	**CN** = TN + FP
FN	**TP**	**CP** = TP + FN
PN = TN + FN	**PP** = FP + TP	**N** = CN + CP

- Accuracy: $\frac{TN+TP}{N}$
- Precision: $\frac{TP}{TP+FP}$
- Recall: $\frac{TP}{TP+FN}$
- True positive rate (TPR): $\frac{TP}{TP+FN}$
- True negative rate (TNR): $\frac{TN}{TN+FP}$
- Balanced accuracy: $\frac{TPR+TNR}{2}$
- F1: $\frac{2TP}{2TP+FP+FN}$
- Cohen's kappa: $\frac{N*TP + N*TN - (CN*PN) - (CP*PP)N*N - (CN*PN) - (CP*PP)}{N*N - (CN*PN) - (CP*PP)}$
- Matthews correlation coefficient: $\frac{TP*TN - FP*FN}{\sqrt{PN*PP*CN*CP}}$

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

A complication: imbalanced data

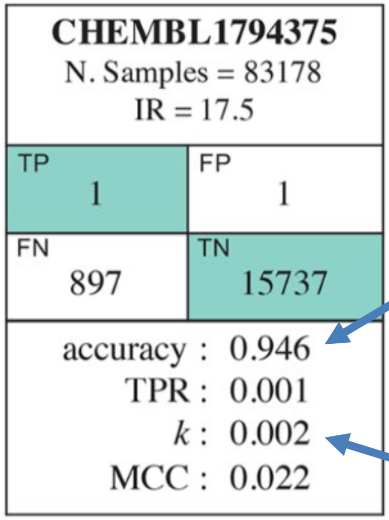
- Many drug-discovery datasets are highly *imbalanced* : one class is much more common than the others.
- This makes accuracy a bad metric. Cohen's kappa is better.[1]
  - Provides a balanced view of overall accuracy for imbalanced data
  - Ranges from [-1,1], 0 is random
  - Values >0.4 are generally considered ok, >0.6 is good, >0.8 is great.
- Two example confusion matrices from real bioactivity datasets:



CHEMBL1614421
N. Samples = 49035
IR = 7.7

TP 40	FP 9
FN 1083	TN 8676

accuracy : 0.889
TPR : 0.036
k : 0.059
MCC : 0.156



CHEMBL1794375
N. Samples = 83178
IR = 17.5

TP 1	FP 1
FN 897	TN 15737

accuracy : 0.946
TPR : 0.001
k : 0.002
MCC : 0.022

https://doi.org/10.1021/acs.jcim.1c00160

Accuracy is really good!

Cohen's kappa is *terrible*

[1] "Count on Kappa" https://doi.org/10.1007/s10822-014-9759-6

Which metric should you use?

- You get tired of this, but: “it depends”.
- Here are some rules of thumb
    - ➢ If the model is going to be used to rank compounds: AUROC and/or one of the early enrichment measures
    - ➢ If you’re going to be using the actual predictions: accuracy or one of the approaches which work with imbalanced datasets (kappa, F1, or balanced accuracy)
- Before I trust a model *at all* I generally want to see: the ROC curve, the confusion matrix, accuracy, and kappa

Looking into a few machine learning methods

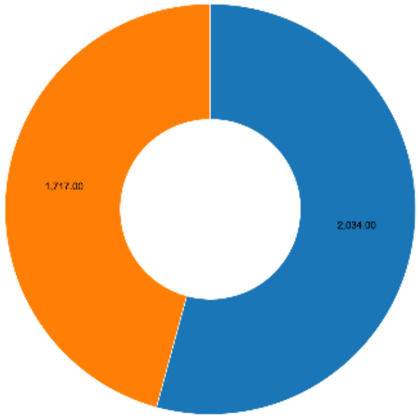
- Logistic regression
- Decision trees
- Bag classifiers
- Random forests
- Boosting
- Gradient boosting

First: the dataset

- I’m using this dataset just to demonstrate the methods we’re learning here. It’s not particularly realistic, but it’s great for demos

- Kaggle bio-response dataset, just the training set
- 3751 rows, 1770 attributes, binary activity measure
- Dataset from Boehringer Ingelheim
- Shockingly balanced
- Preprocessing:

  Do a PCA and take 20 components[1]

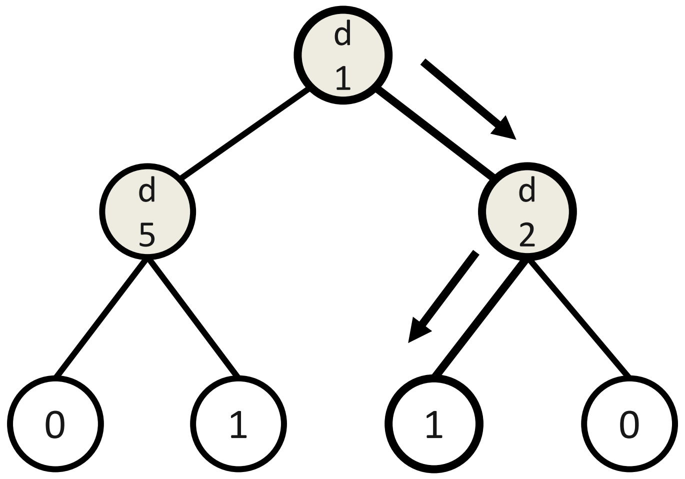


[1]This isn’t what I’d normally do for a real problem

https://www.kaggle.com/c/bioresponse/

Let’s start with decision trees

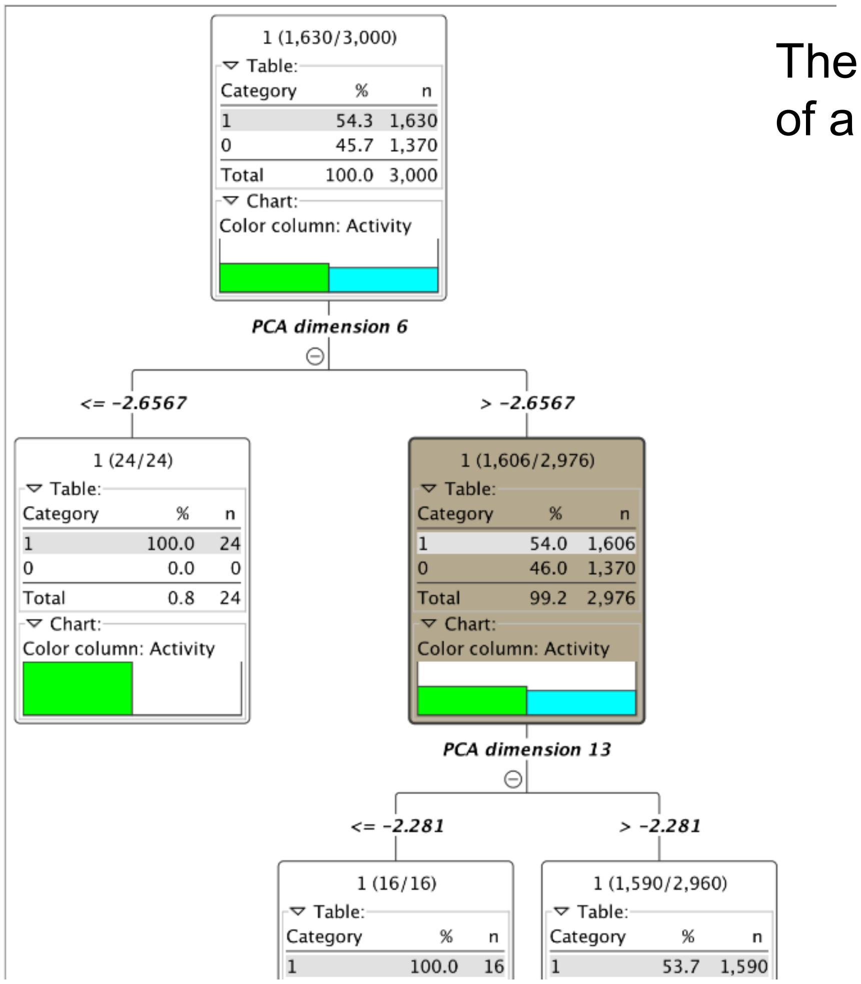
- Can be read as a series of if-then-else rules



“If descriptor 1 = 1 and
descriptor 2 = 0 then predict 1.”

Let’s start with decision trees

- The final model is a tree
- Can be read as a series of if-then-else rules



1 (1,630/3,000)

Table:

Category	%	n
1	54.3	1,630
0	45.7	1,370
Total	100.0	3,000

Chart:
Color column: Activity

1 (24/24)

Table:

Category	%	n
1	100.0	24
0	0.0	0
Total	0.8	24

Chart:
Color column: Activity

1 (1,606/2,976)

Table:

Category	%	n
1	54.0	1,606
0	46.0	1,370
Total	99.2	2,976

Chart:
Color column: Activity

1 (16/16)

Table:

Category	%	n
1	100.0	16

1 (1,590/2,960)

Table:

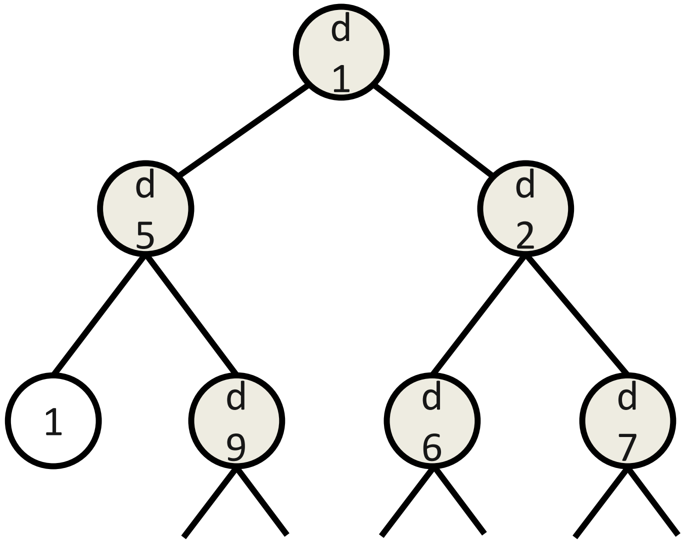
Category	%	n
1	53.7	1,590

The first three levels of a decision tree

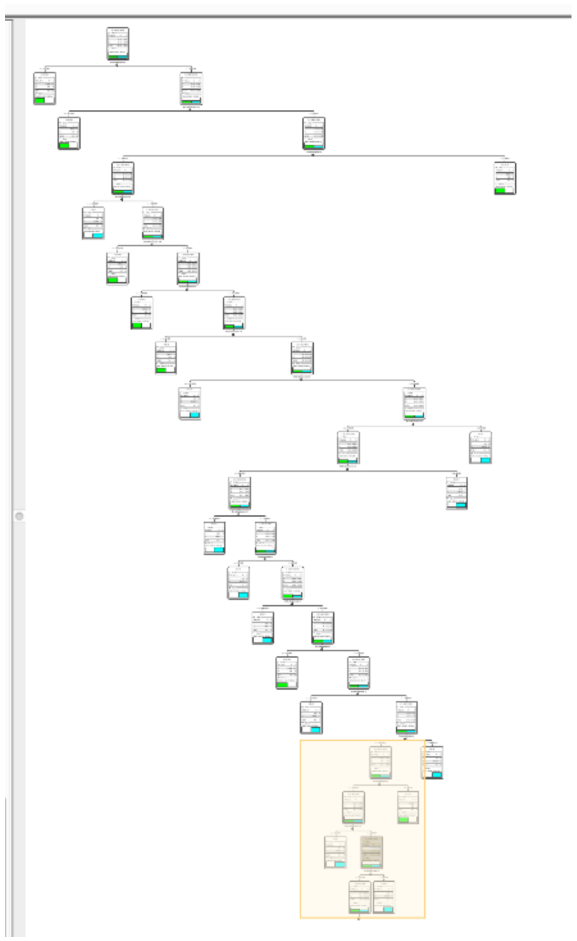
Let’s start with decision trees

Building a decision tree:

- While not done:
  - Pick best, legal attribute to split on
  - Split data set

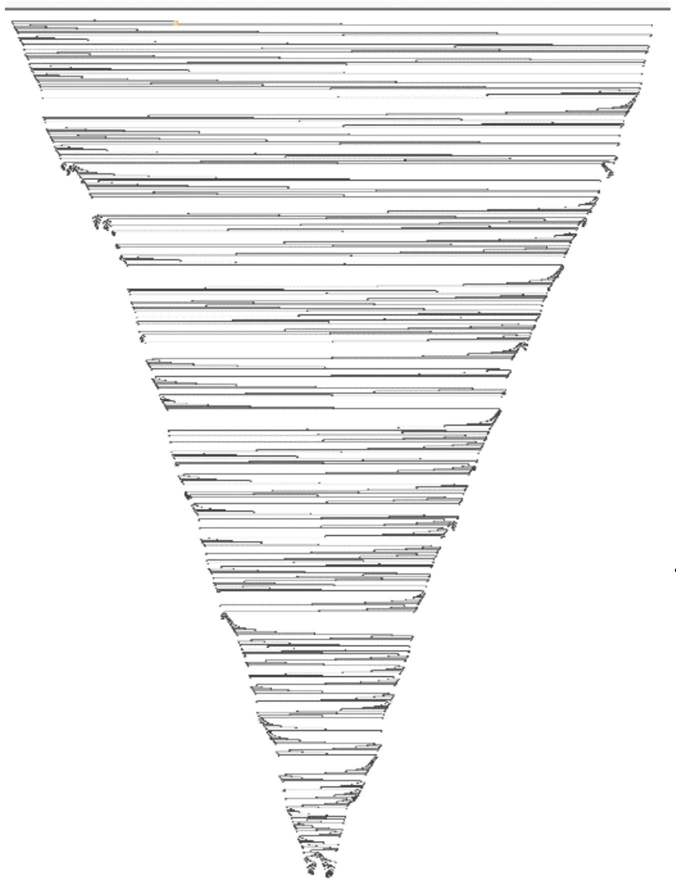


What comes out?



Part of the decision tree for our dataset

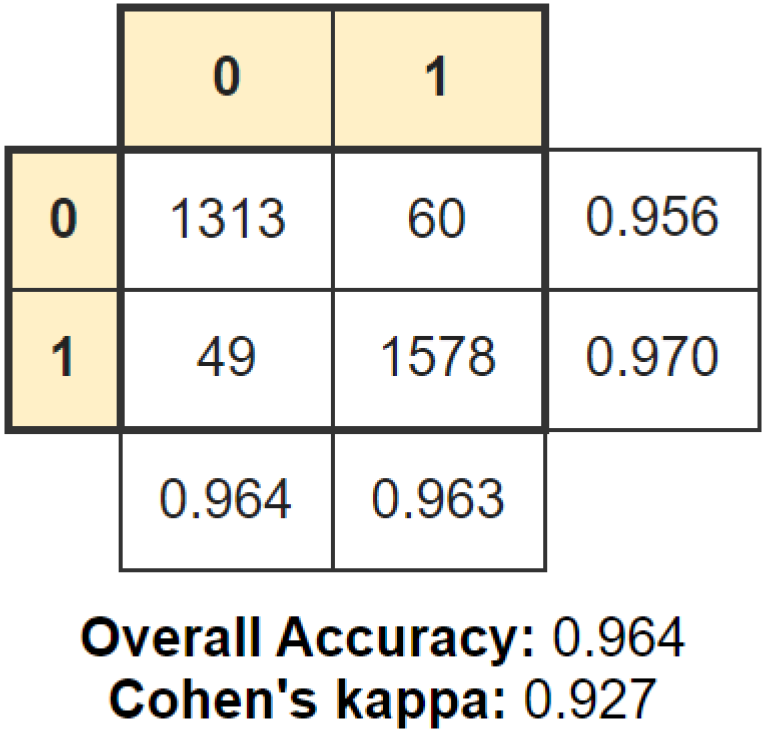
The whole tree



Think that *might* be overfit?

Performance of the decision tree

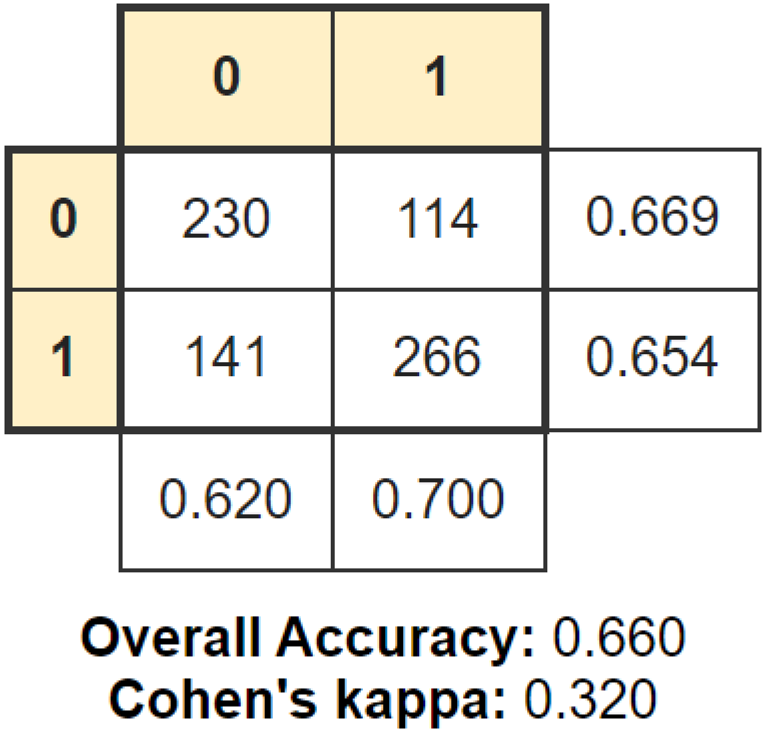
Training data



	0	1	
0	1313	60	0.956
1	49	1578	0.970
	0.964	0.963	

Overall Accuracy: 0.964
Cohen's kappa: 0.927

Test data



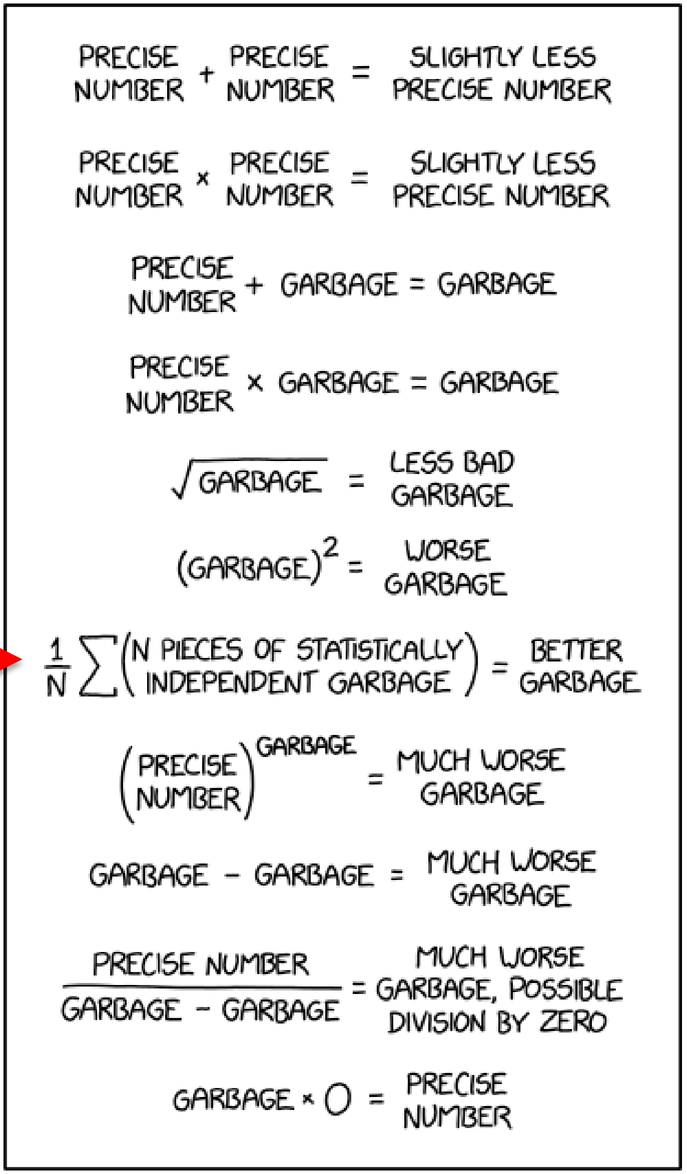
	0	1	
0	230	114	0.669
1	141	266	0.654
	0.620	0.700	

Overall Accuracy: 0.660
Cohen's kappa: 0.320

Oh yeah, it’s overfit!

Bagging and random forests

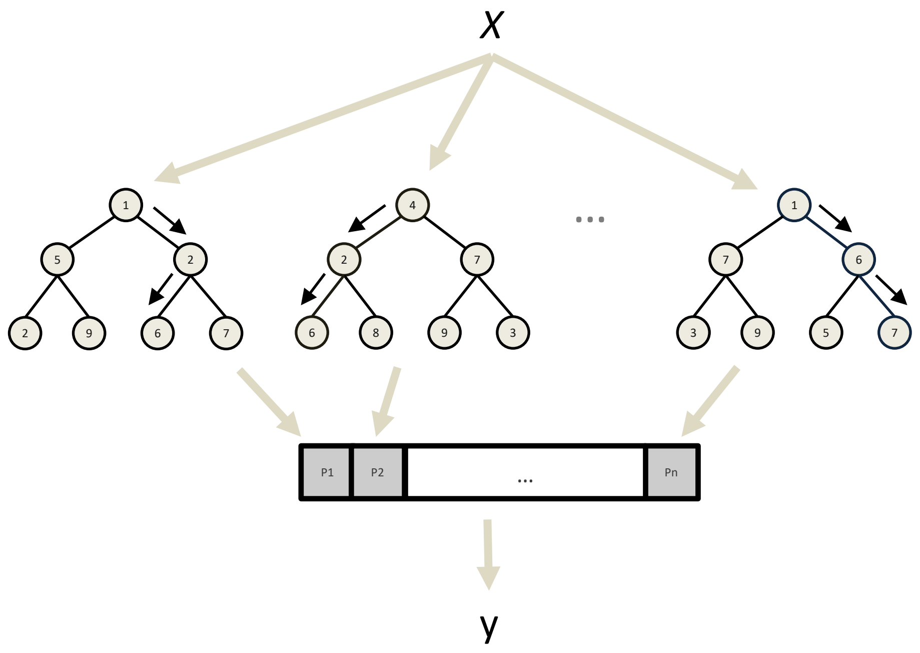
- The general idea is to take advantage of the “wisdom of the crowd”: combining predictions from a large number of weak predictors leads to a more accurate predictor.


PRECISE NUMBER + PRECISE NUMBER = SLIGHTLY LESS PRECISE NUMBER
PRECISE NUMBER × PRECISE NUMBER = SLIGHTLY LESS PRECISE NUMBER
PRECISE NUMBER + GARBAGE = GARBAGE
PRECISE NUMBER × GARBAGE = GARBAGE
√GARBAGE = LESS BAD GARBAGE
(GARBAGE)² = WORSE GARBAGE
1/N Σ(N PIECES OF STATISTICALLY INDEPENDENT GARBAGE) = BETTER GARBAGE
(PRECISE NUMBER)^GARBAGE = MUCH WORSE GARBAGE
GARBAGE − GARBAGE = MUCH WORSE GARBAGE
PRECISE NUMBER / (GARBAGE − GARBAGE) = MUCH WORSE GARBAGE, POSSIBLE DIVISION BY ZERO
GARBAGE × 0 = PRECISE NUMBER

https://xkcd.com/2295/

Bagging and random forests

- The general idea is to take advantage of the “wisdom of the crowd”: combining predictions from a large number of weak predictors leads to a more accurate predictor.

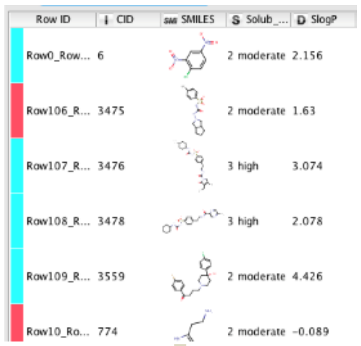


Typically: for classification the individual models vote and the majority wins

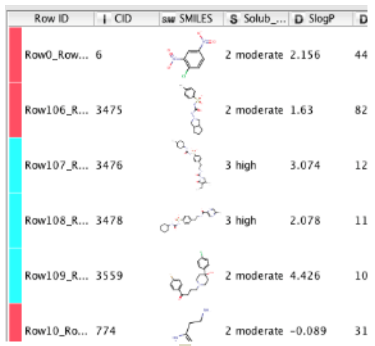
Breiman, L. Bagging predictors. *Mach. Learn.* **24,** 123–140 (1996).

How do we get all those different predictors?

Pick a different random subset of the training data for each model in the ensemble (bag).

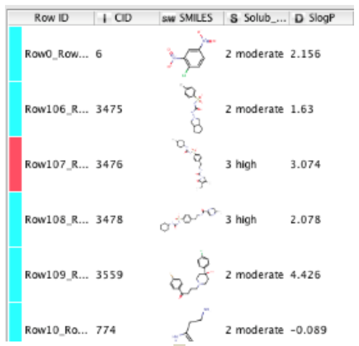


	Row ID	CID	SMILES	Solub_...	SlogP
🟦	Row0_Row...	6		2 moderate	2.156
🟥	Row106_R...	3475		2 moderate	1.63
🟦	Row107_R...	3476		3 high	3.074
🟦	Row108_R...	3478		3 high	2.078
🟦	Row109_R...	3559		2 moderate	4.426
🟥	Row10_Ro...	774		2 moderate	-0.089

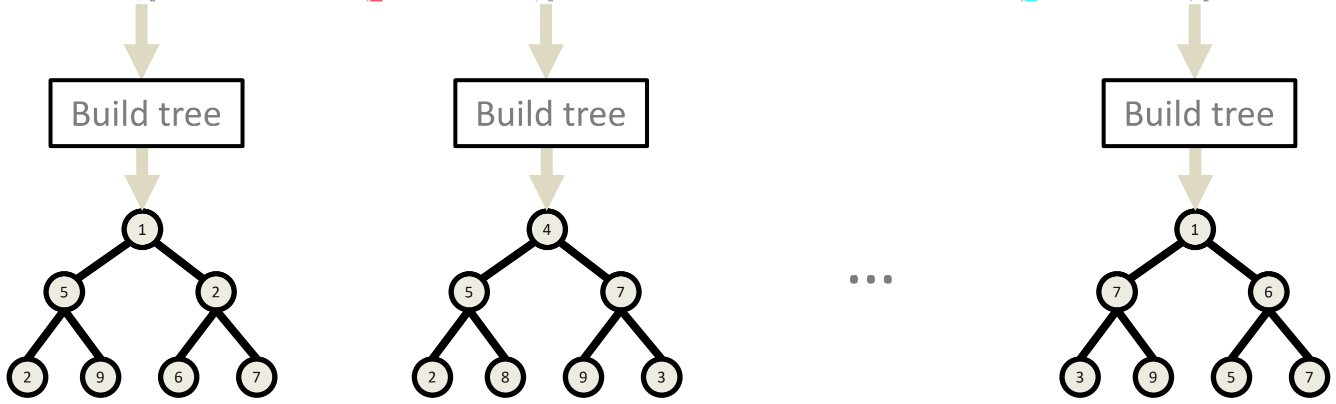


	Row ID	CID	SMILES	Solub_...	SlogP	D
🟥	Row0_Row...	6		2 moderate	2.156	44
🟥	Row106_R...	3475		2 moderate	1.63	82
🟦	Row107_R...	3476		3 high	3.074	12
🟦	Row108_R...	3478		3 high	2.078	11
🟦	Row109_R...	3559		2 moderate	4.426	10
🟥	Row10_Ro...	774		2 moderate	-0.089	31

...

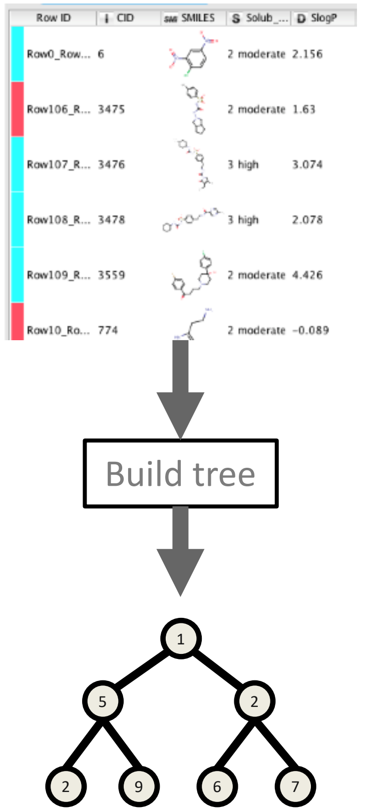


	Row ID	CID	SMILES	Solub_...	SlogP
🟦	Row0_Row...	6		2 moderate	2.156
🟦	Row106_R...	3475		2 moderate	1.63
🟥	Row107_R...	3476		3 high	3.074
🟦	Row108_R...	3478		3 high	2.078
🟦	Row109_R...	3559		2 moderate	4.426
🟦	Row10_Ro...	774		2 moderate	-0.089



Random Forests

- Bags of decision trees, but an extra element of randomization is applied when building the trees: each node in the decision tree only “sees” a subset of the input descriptors, typically $\sqrt{N}$.
- Random forests tend to be very robust with respect to overfitting (though the individual trees are almost certainly overfit)
- Extra benefit: training tends to be much faster since there are less attributes to consider



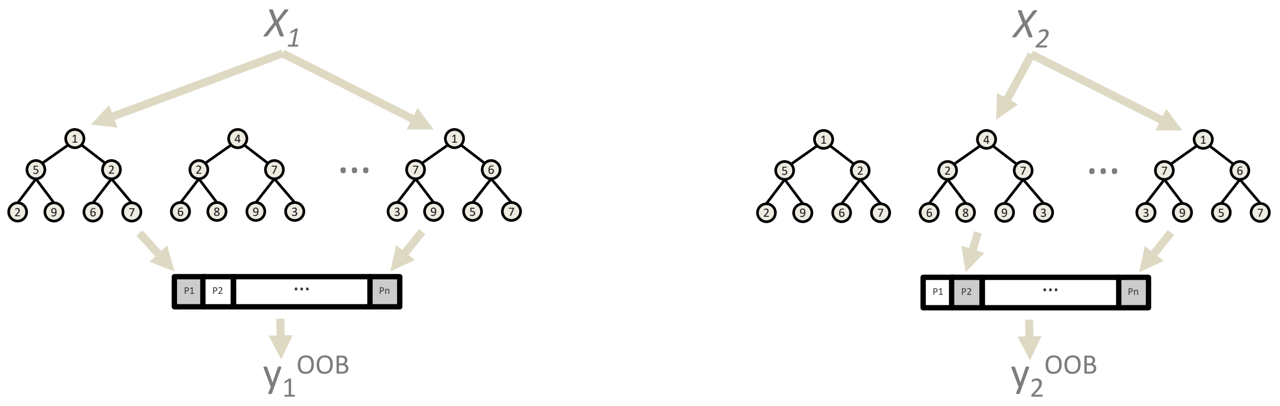
Row ID	CID	SMILES	Solub_...	SlogP
Row0_Row...	6		2 moderate	2.156
Row106_R...	3475		2 moderate	1.63
Row107_R...	3476		3 high	3.074
Row108_R...	3478		3 high	2.078
Row109_R...	3559		2 moderate	4.426
Row10_Ro...	774		2 moderate	−0.089

Nice benefits of bagging 1

- You get prediction probabilities:
  - Simple: Fraction of predictors voting for an outcome
  - More complex: average of predicted probabilities from individual predictors

Nice benefits of bagging 2

- Out of bag estimation: Allows testing the model using the training data: when validating, each model should only vote on data points that were not used to train it



- This seems like either cheating or magic, but it's statistically valid and tends to provide an accurate measure of model performance

Algorithm for building a bag (ensemble) of trees

- For each tree you want in the ensemble:
  - Generate a subset of the data to use as a local training set
  - Train a tree using that subset using a standard decision-tree algorithm
  - Add the tree to the bag

- Same idea for the random forest, but each node can only choose from a random subset of the descriptors

Results 1: Impact of model size

1 Tree from Bag

	0	1	
0	203	144	0.585
1	123	281	0.696
	0.623	0.661	

Overall Accuracy: 0.644
Cohen's kappa: 0.282

10 Tree Bag

	0	1	
0	257	90	0.741
1	136	268	0.663
	0.654	0.749	

Overall Accuracy: 0.699
Cohen's kappa: 0.400

200 Tree Bag

	0	1	
0	261	86	0.752
1	105	299	0.740
	0.713	0.777	

Overall Accuracy: 0.746
Cohen's kappa: 0.490

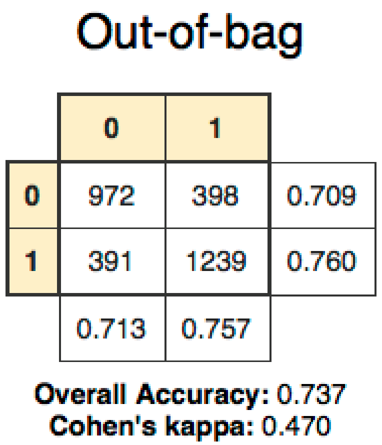
500 Tree Bag

	0	1	
0	250	97	0.720
1	102	302	0.748
	0.710	0.757	

Overall Accuracy: 0.735
Cohen's kappa: 0.468

Results 2: Out-of-bag vs Test set

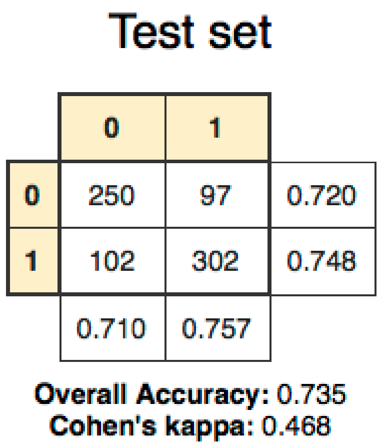
500 tree bag classifier



Out-of-bag

	0	1	
0	972	398	0.709
1	391	1239	0.760
	0.713	0.757	

Overall Accuracy: 0.737
Cohen's kappa: 0.470



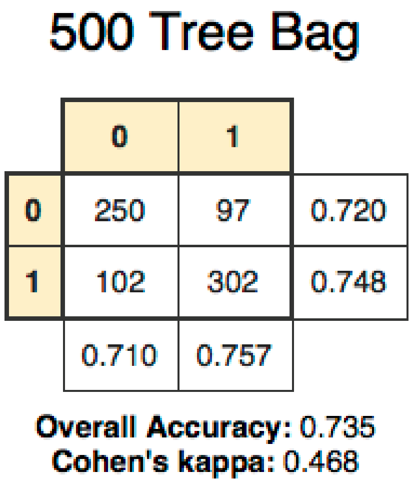
Test set

	0	1	
0	250	97	0.720
1	102	302	0.748
	0.710	0.757	

Overall Accuracy: 0.735
Cohen's kappa: 0.468

Out-of-bag prediction really does work!

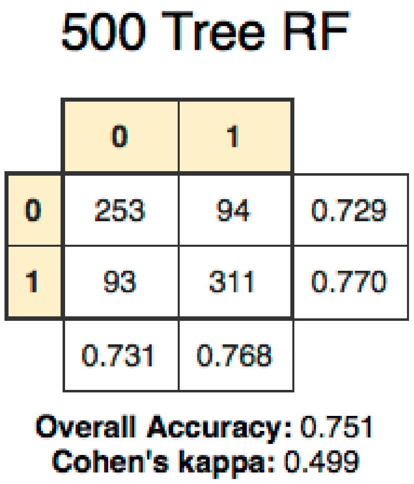
Results 3: Random forest vs Tree ensemble



500 Tree Bag

	0	1	
0	250	97	0.720
1	102	302	0.748
	0.710	0.757	

Overall Accuracy: 0.735
Cohen's kappa: 0.468



500 Tree RF

	0	1	
0	253	94	0.729
1	93	311	0.770
	0.731	0.768	

Overall Accuracy: 0.751
Cohen's kappa: 0.499

On this dataset the random forest is a bit better

Bags and random forests: Some general comments/observations

- The good:
  - Algorithm and parameters are easy to understand
  - Quite robust w.r.t. overfitting
  - On any given dataset, if any method is going to work, a random forest will likely do ok
  - There aren’t a huge number of hyper-parameters to optimize
  - Normally pretty quick to build a model, particularly with random forests. (Embarassingly parallel)

- The bad:
  - Models can be *huge*
  - You can forget about direct interpretation

Boosting

- Another algorithm for creating ensembles of decision trees
- Starts with a single tree built from the data
- Builds additional trees to fit the residual errors and combines them with a weight based on their accuracies
- Typically uses fairly shallow trees
- Can introduce randomness in choice of data subsets (“stochastic gradient boosting”) and in variable choice.

Boosting (conceptually)

- Start with a model (tree) built from the data
- While not done
  - Increase the weight of misclassified examples
  - Train a new tree
  - Combine the new tree with the existing ones using a weight proportional to its accuracy

- There's math too, if you want it:
  - Freund, Y. & Schapire, R. E. A Short Introduction to Boosting. *J. Japanese Soc. Artif. Intell.* **15,** 771–780 (1999).
  - Friedman, J. H. Greedy Function Approximation: A Gradient Boosting Machine. *Ann. Stat.* **29,** 1189–1232 (2001).

Gradient Boosting: Results

200 Tree Bag

	0	1	
0	261	86	0.752
1	105	299	0.740
	0.713	0.777	

Overall Accuracy: 0.746
Cohen's kappa: 0.490

500 Tree RF

	0	1	
0	253	94	0.729
1	93	311	0.770
	0.731	0.768	

Overall Accuracy: 0.751
Cohen's kappa: 0.499

100 tree GBM

	0	1	
0	243	104	0.700
1	99	305	0.755
	0.711	0.746	

Overall Accuracy: 0.730
Cohen's kappa: 0.456

100 tree Stochastic GBM

	0	1	
0	252	95	0.726
1	101	303	0.750
	0.714	0.761	

Overall Accuracy: 0.739
Cohen's kappa: 0.476

Gradient Boosting: Results

100 tree Stochastic GBM

	0	1	
0	252	95	0.726
1	101	303	0.750
	0.714	0.761	

Overall Accuracy: 0.739
Cohen's kappa: 0.476

200 tree Stochastic GBM

	0	1	
0	249	98	0.718
1	99	305	0.755
	0.716	0.757	

Overall Accuracy: 0.738
Cohen's kappa: 0.472

200 tree Stochastic RF GBM

	0	1	
0	249	98	0.718
1	102	302	0.748
	0.709	0.755	

Overall Accuracy: 0.734
Cohen's kappa: 0.465

Model sizes

	Tree Ensemble	Random Forest	Gradient boosting[1]
# trees	200	200	200
Min depth	55	33	5
Max depth	142	70	5
Avg depth	87	48	5
Min # nodes	691	799	<=32
Max # nodes	835	935	<=32
Avg # nodes	766	861	<=32

[1]I'm kind of guessing here

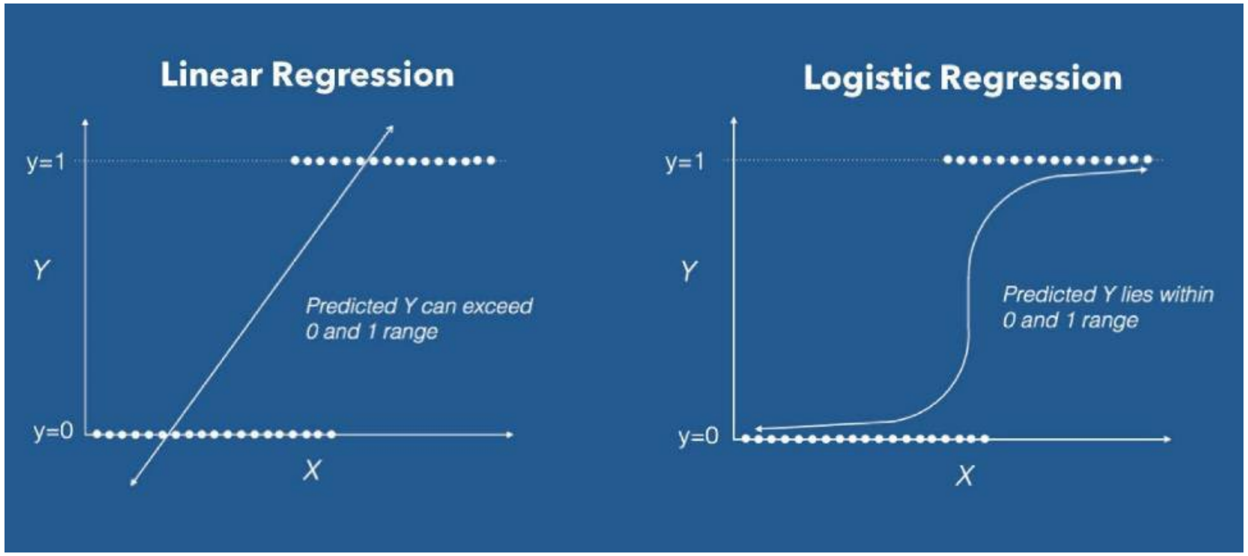
GBMs: some general comments/observations

- The good:
  - Quite robust w.r.t. overfitting
  - Comparatively small models
  - There aren’t a huge number of hyper-parameters to optimize
  - Amazingly fast when using something like XGBoost
- The bad:
  - You can forget about direct interpretation
  - Not embarassingly parallel[1]
  - More hyper-parameters than a random forest

[1] **Note:** this observation goes right out the window when using tools like XGBoost

Logistic regression (it’s not all trees!)

- Similar to linear regression (or multiple-linear regression) except we’re fitting the logistical function to the data instead of a straight line and predicting classes instead of values



Linear Regression
y=1
Y
y=0
X
Predicted Y can exceed 0 and 1 range
Logistic Regression
y=1
Y
y=0
X
Predicted Y lies within 0 and 1 range

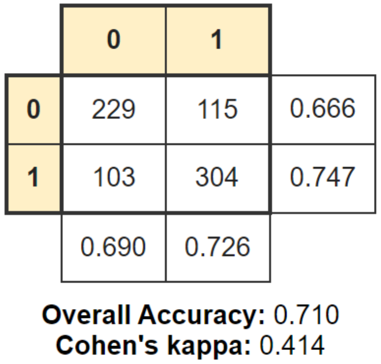
https://towardsdatascience.com/introduction-to-logistic-regression-66248243c148

- The resulting models are quite simple and can be quite effective

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

Logistic regression results

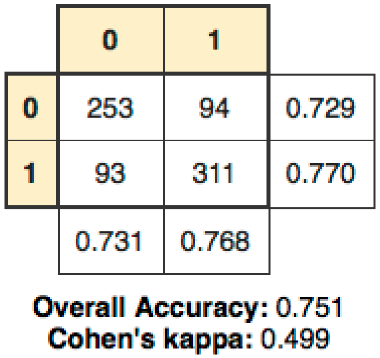
Logistic Regression



	0	1	
0	229	115	0.666
1	103	304	0.747
	0.690	0.726	

Overall Accuracy: 0.710
Cohen's kappa: 0.414

500 Tree RF



	0	1	
0	253	94	0.729
1	93	311	0.770
	0.731	0.768	

Overall Accuracy: 0.751
Cohen's kappa: 0.499

In this case the LR model is less effective than the RF. It's worth keeping in mind that the model is orders of magnitude simpler though.

Comparing LR to other methods

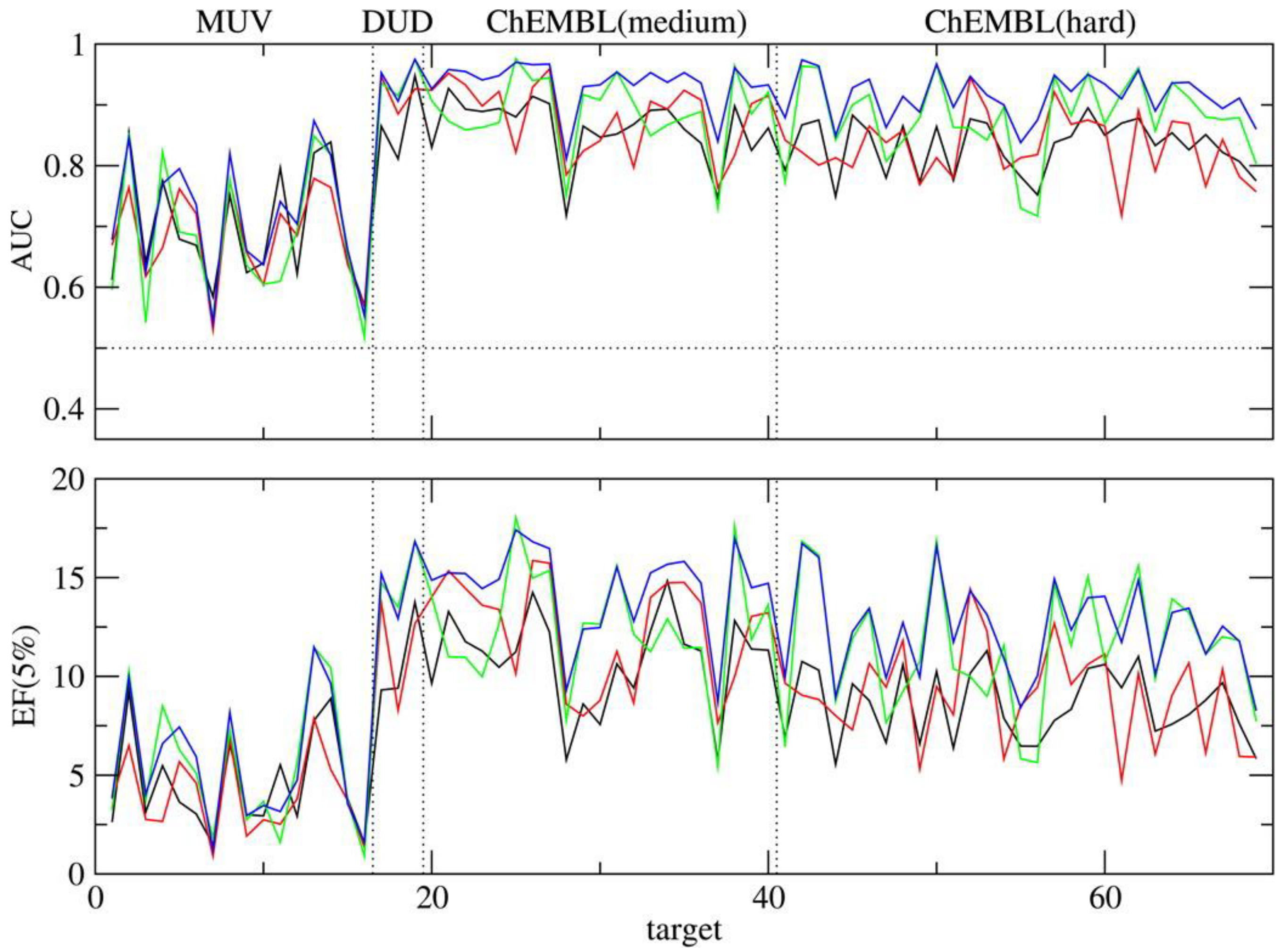


Figure 5. Average performance of RF(AP) (black), NB(Morgan2) (red), **LR(Morgan2) (green)**, and RF(AP)-NB(Morgan2)-LR(Morgan2) (blue) measured with AUC (top) and EF(5%) (bottom.

https://doi.org/10.1021/ci400466r

Some other common classification methods

- Support vector machines
- Naïve Bayes classifiers
- Gaussian process classfiers
- Regression models with binning

- many, many more

Neural networks

- Saved these for last because they are currently **the** hot thing.
- Very flexible models with a bunch of hyperparameters to optimize
- Quite data hungry (due to the number of hyperparameters)
- Quite computationally expensive to train/use.
- From the perspective of building QSAR models, they are generally not significantly better than simpler methods like random forests or XGBoost[1]

- However, they do enable something quite new and powerful: multi-task learning – building one model which predicts multiple outputs simultaneously. This opens a number of really interesting doors
- But that's a topic for another time

[1] Caveat: it is *always* possible to find an example where one method out-performs another

Regression models

- There are a huge variety of methods available here too.
- I’m not going to get into details today
- Some common methods:
  - Linear regression
  - Partial least squares
  - Support vector regression
  - Random forest regression
  - Gradient boosting regression
  - Neural networks

Acknowledgements

These slides are assembled from a set of lectures that Sereina Riniker originally created and that we've worked together on and expanded for the past couple of years.