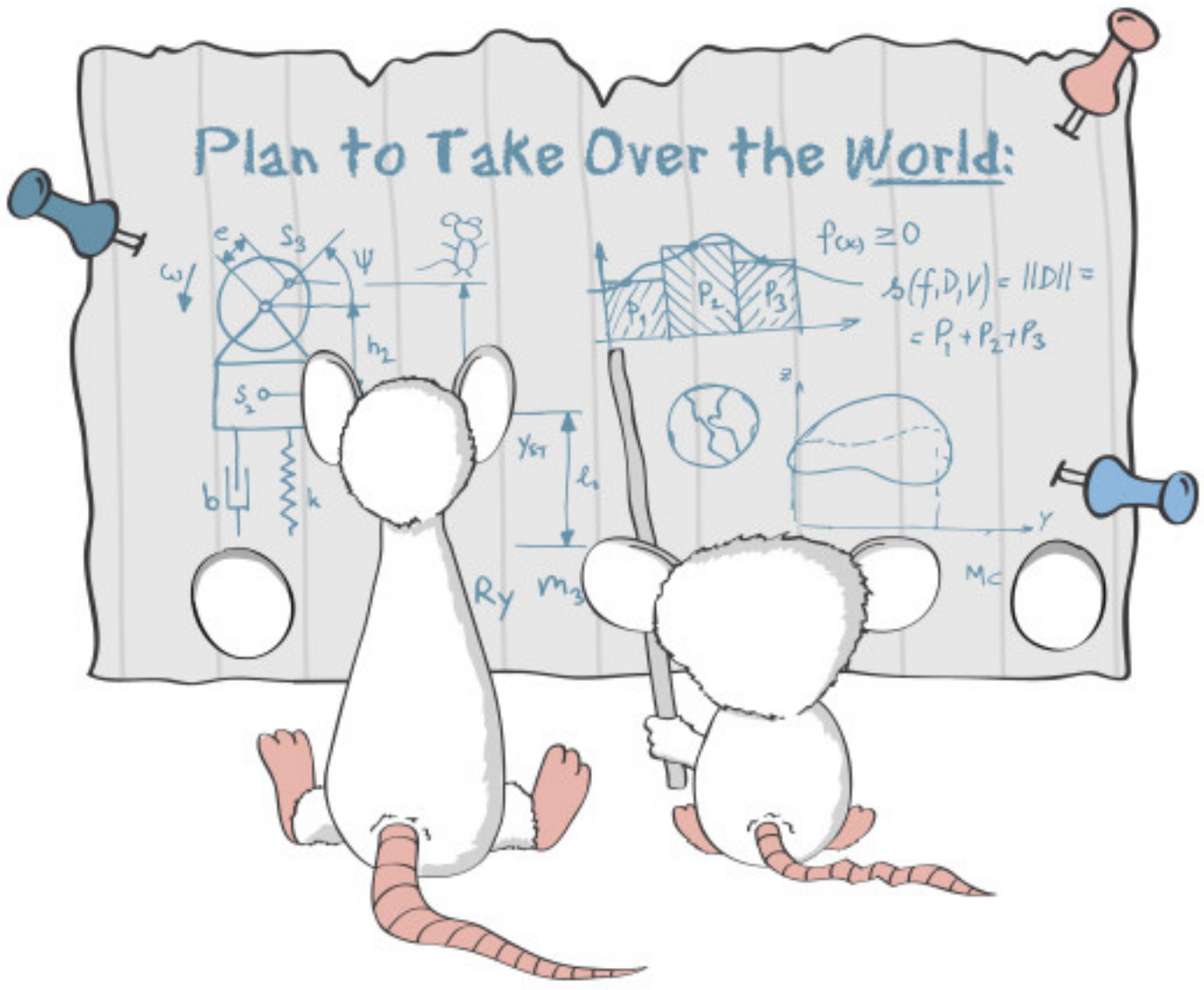


Lecture #2

F5611 Machine Learning for Astronomers
by Martin Topinka

https://github.com/toastmaker/f5611-ML4A

Guest lectures

- 24 Nov 2020 Antonio D'Isanto (MPI, Heidelberg) on redshift estimation

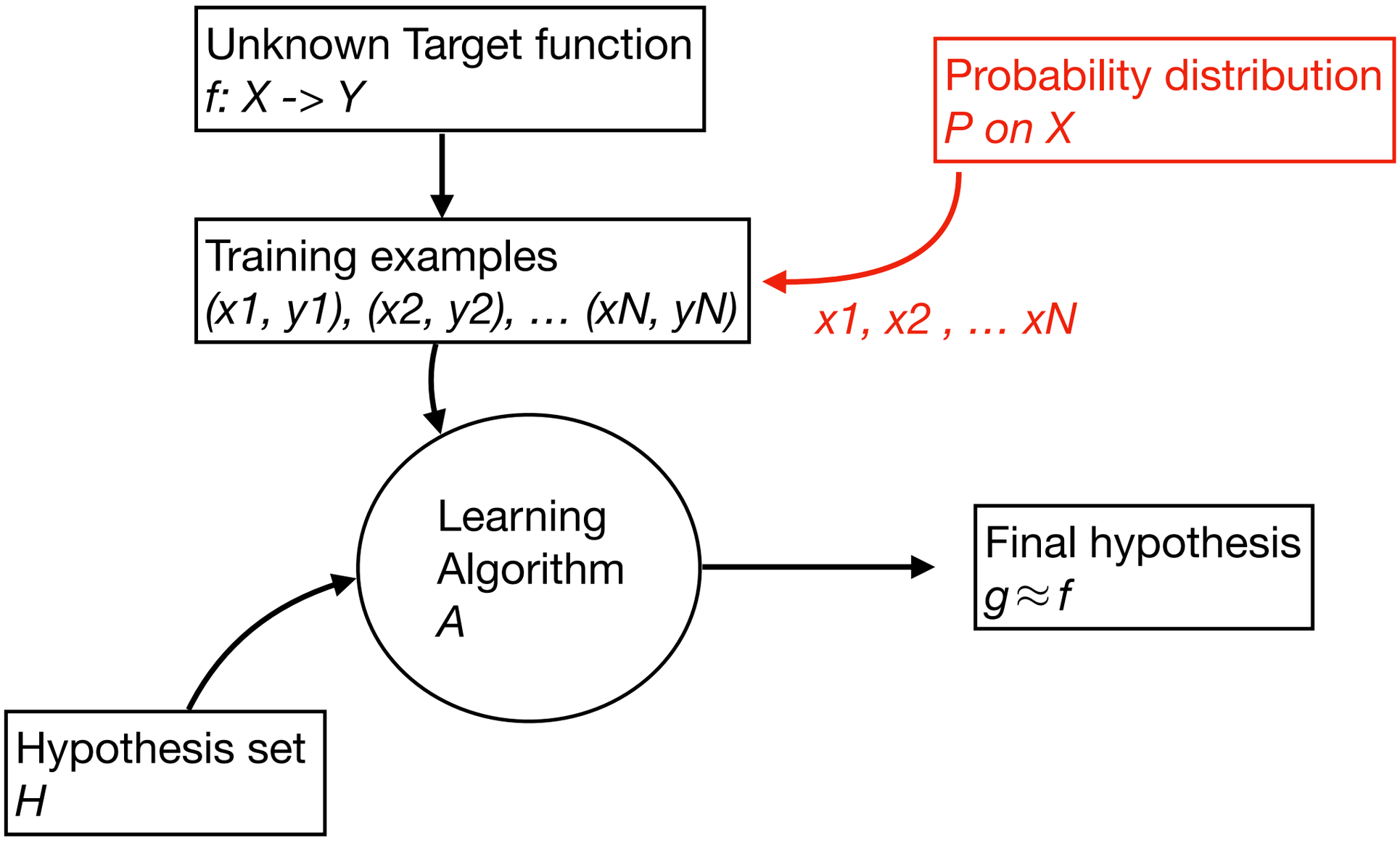
- 15 Dec 2020 Ashish Mahabal (Caltech) on ZTF transient classification

Syllabus

- Introduction to machine learning, history...
- Principles of machine learning
- Supervised, unsupervised machine learning
- Classification vs regression
- Loss function, accuracy measures
- Bias-variance tradeoff
- Model validation
- Introduction to scikit-learn and its API
- Basic machine learning algorithms (SVM, KNN, K-mean, Logistic regression, Decision Trees, Random Forest)
- Curse of dimensionality
- Feature selection, data reduction (PCA)
- Advanced algorithms (bagging, boosting, voting)

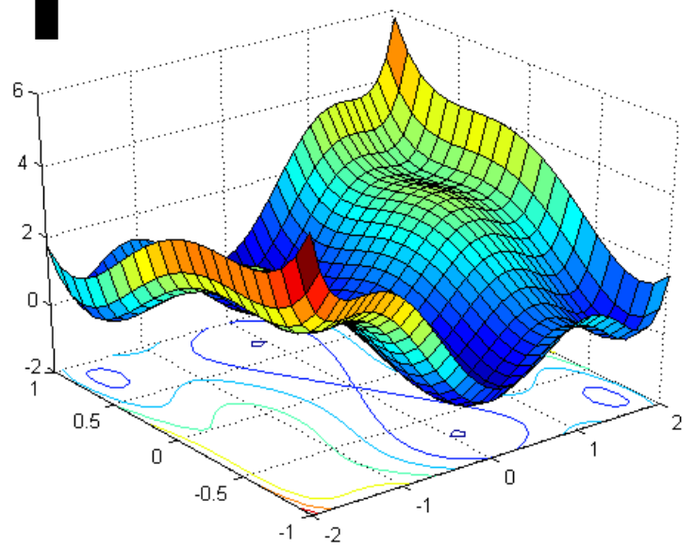
- Hands on session scikit-learn with GRB classification, QSO's vs stars…
- Hyper-parameter fine tuning
- Imbalanced classes
- Neural network, perceptron
- Deep learning neural networks
- Regularisation, dropout
- Deep learning with Convolutional Neural Networks
- Encoder-Decoder, Auto-encoder
- GAN
- Training data generators
- Introduction to Keras/TensorFlow
- Hands on session in Keras (developing a NN to classify stars/QSOs; developing a deep convNN auto-encoder for finding transients)
- Optional: Gaussian Processes

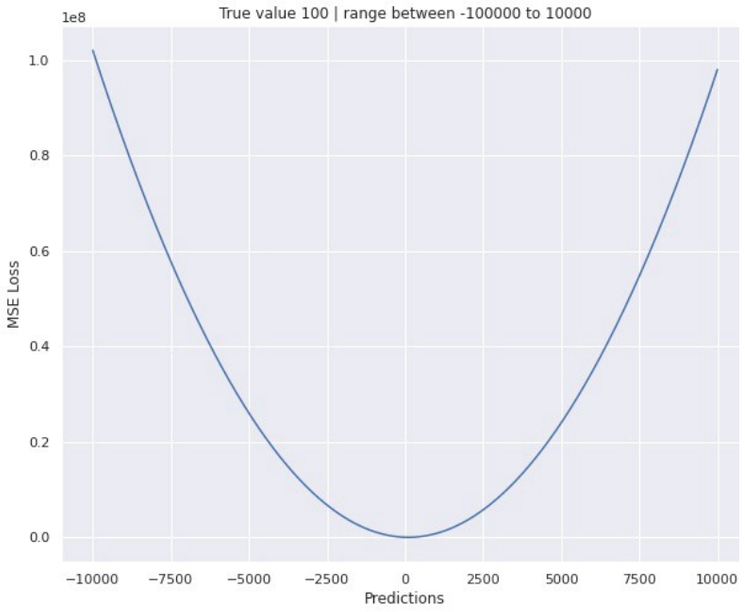
Learning Diagram



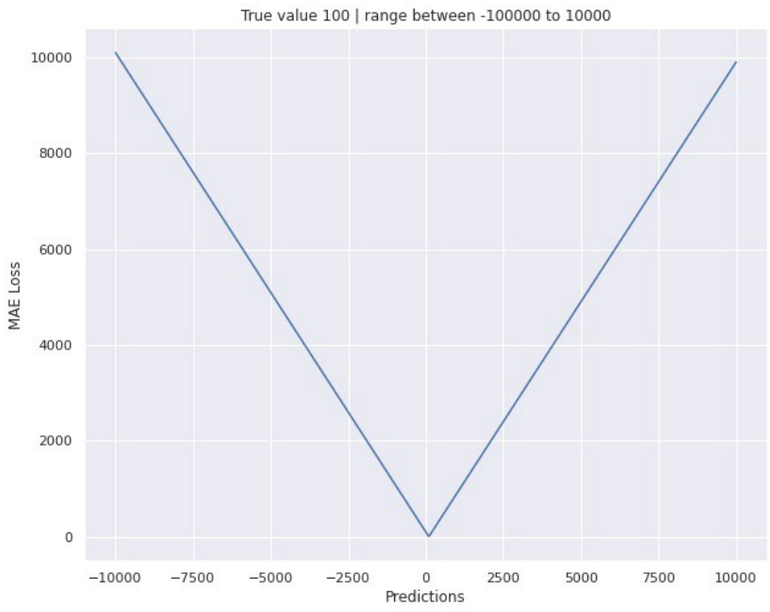
Loss function

- penalty for being wrong (L2, L1, …)
- function of model parameters

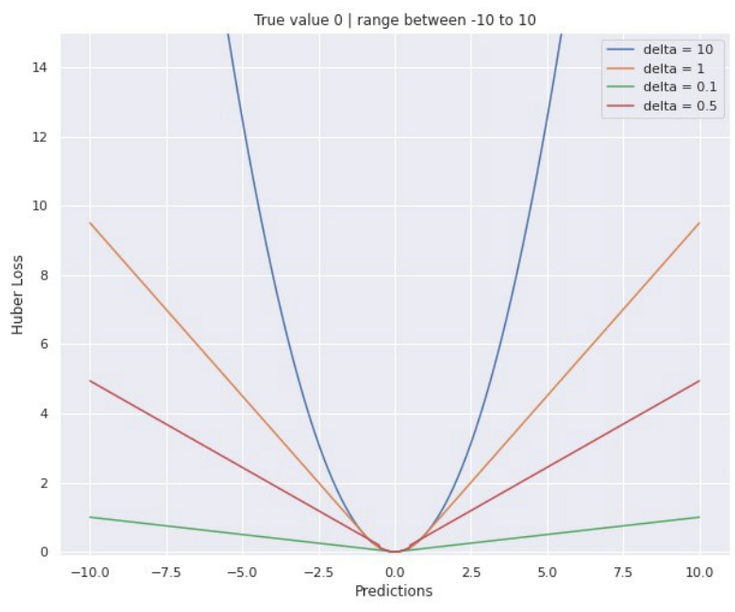




y_{true}-y_{pred}



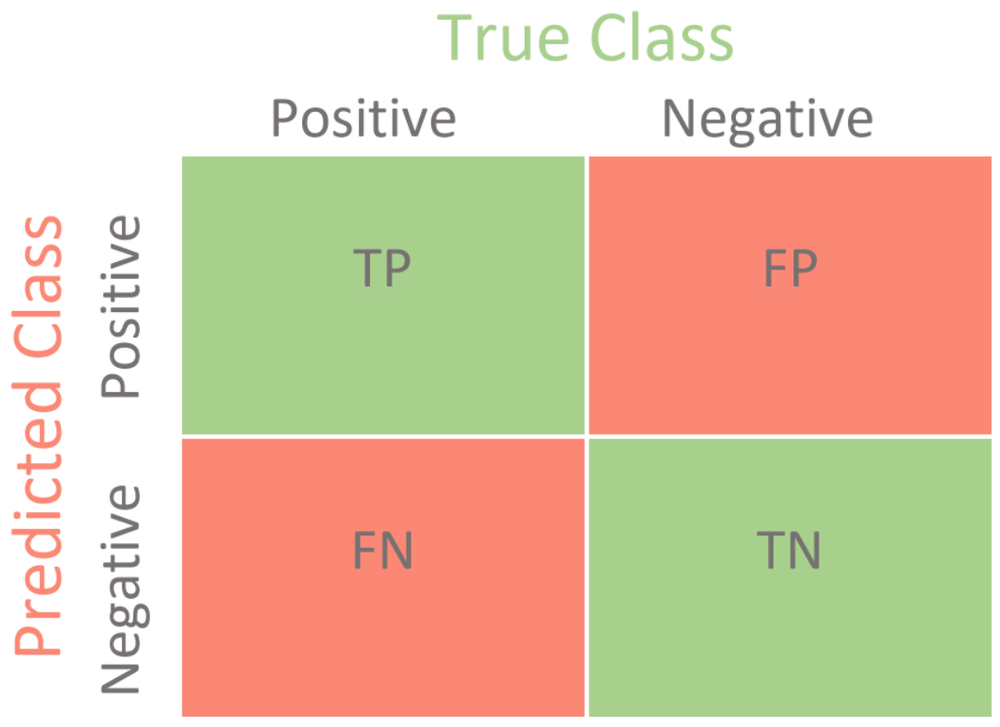
y_{true}-y_{pred}



y_{true}-y_{pred}

$$L_\delta(y, f(x)) = \begin{cases} \frac{1}{2}(y - f(x))^2 & \text{for} |y - f(x)| \le \delta, \\ \delta\,|y - f(x)| - \frac{1}{2}\delta^2 & \text{otherwise.} \end{cases}$$

Confusion matrix



$$\text{Precision} = \frac{True\ Positive}{True\ Positive + False\ Positive}$$

$$\text{Recall} = \frac{True\ Positive}{True\ Positive + False\ Negative}$$

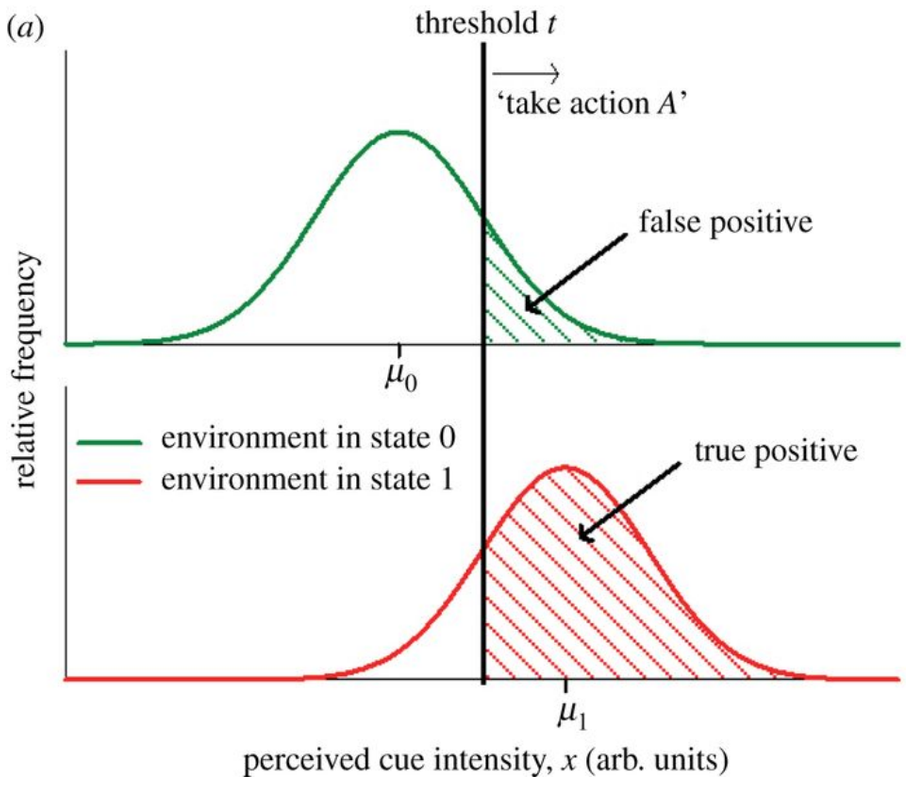
$$\text{F1} = 2 \times \frac{Precision * Recall}{Precision + Recall}$$

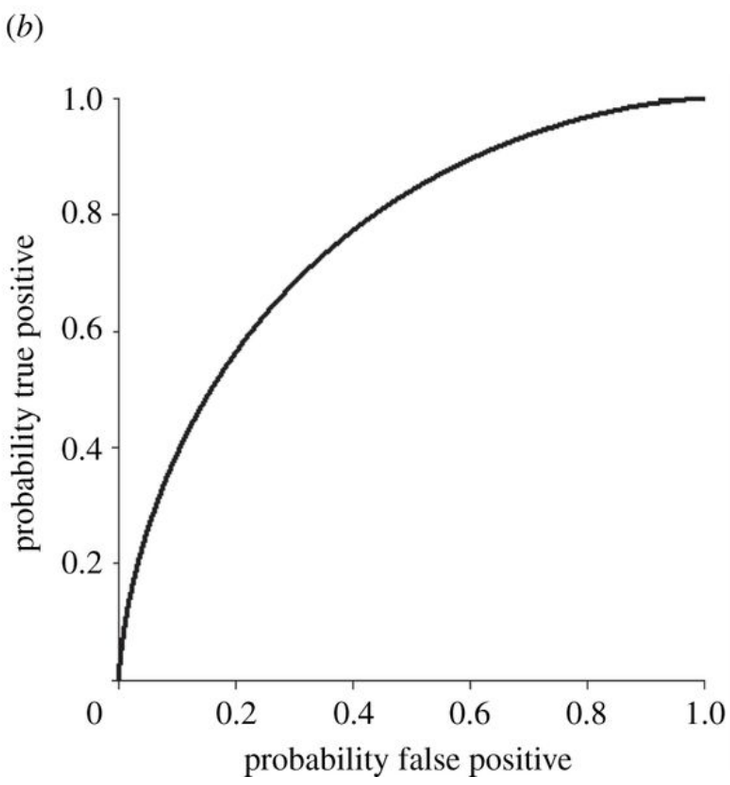
- tuneable for imbalanced classes and for weighted data

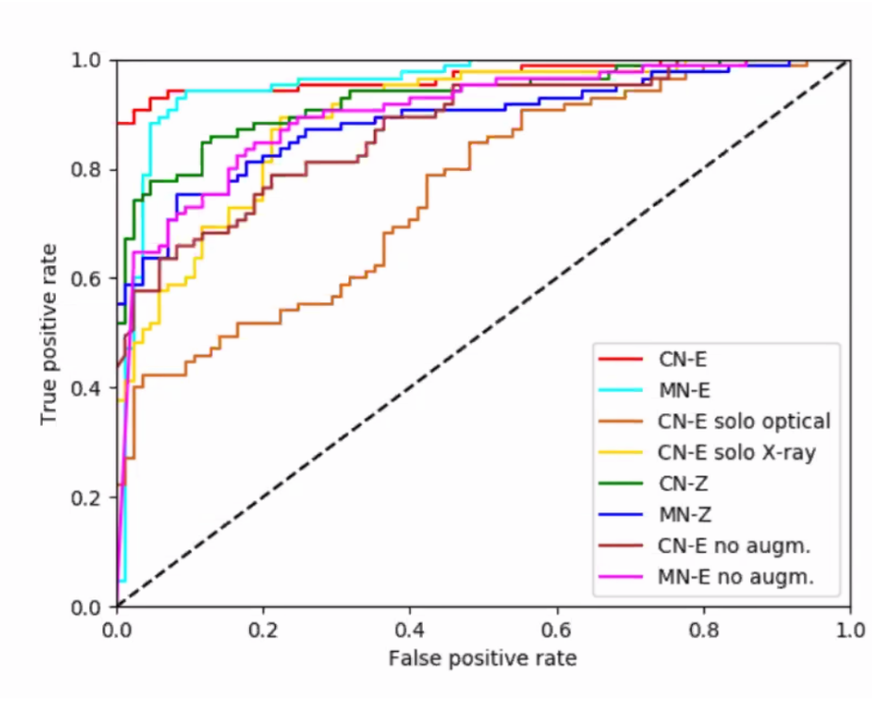
Transaction ID	True Label	Fraudulent Prob
0	0	0.01
1	0	0.1
2	1	0.2
3	0	0.3
4	1	0.4
5	0	0.5
6	0	0.6
7	1	0.7
8	1	0.8
9	1	0.9

50% dog, 50% cat?

ROC

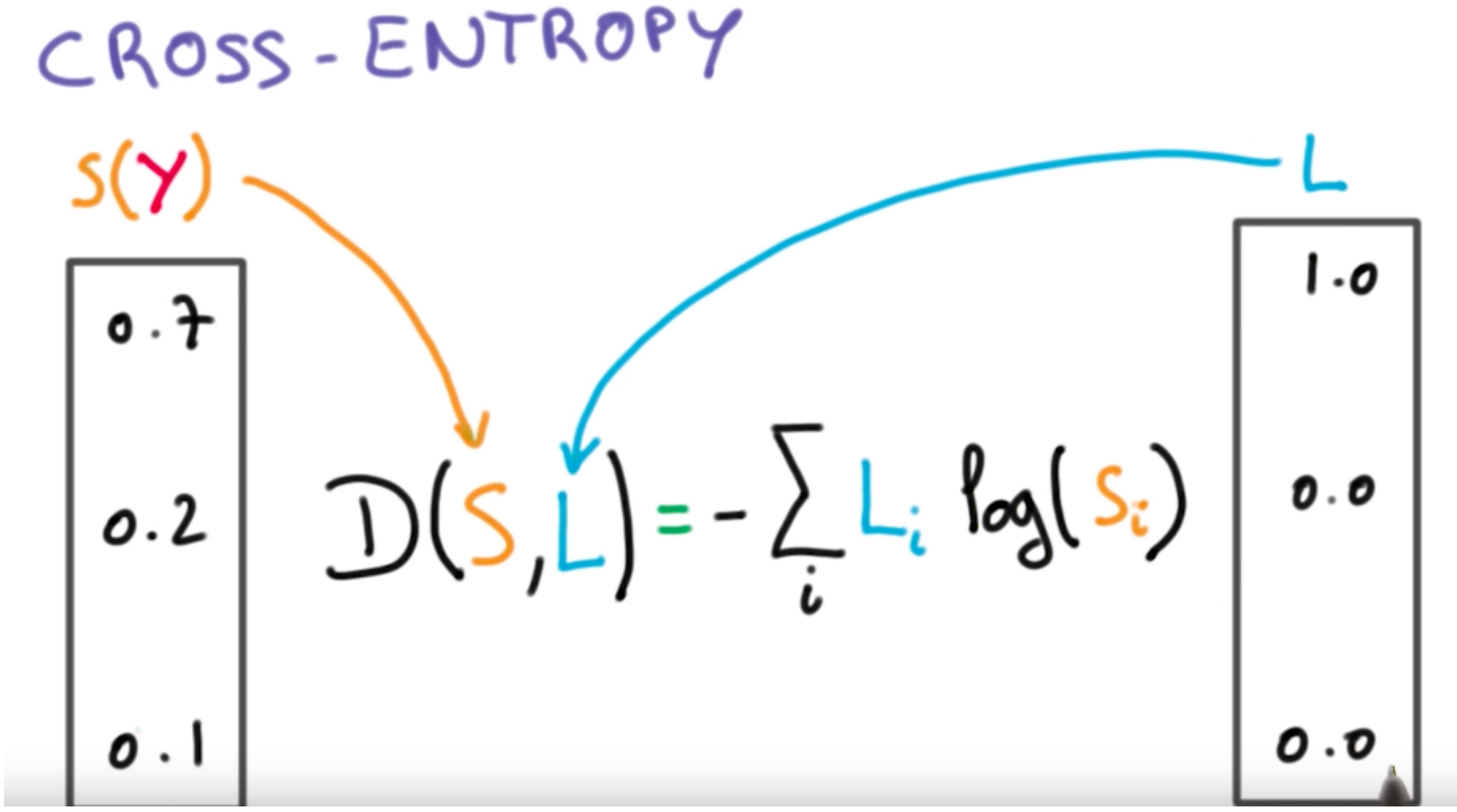






$$J = -\frac{1}{N}\left(\sum_{i=1}^{N} \mathbf{y_i} \cdot \log(\hat{\mathbf{y}}_\mathbf{i})\right)$$

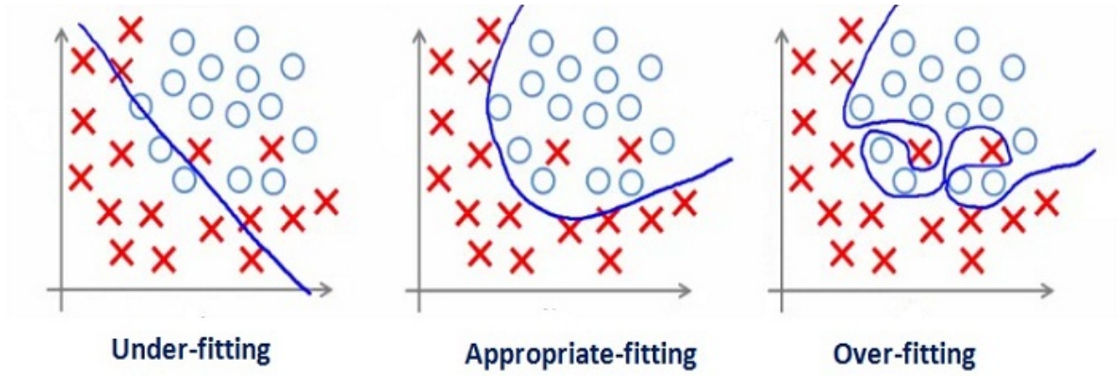
A way how to compare the predicted distribution of class probabilities to the true class probabilities

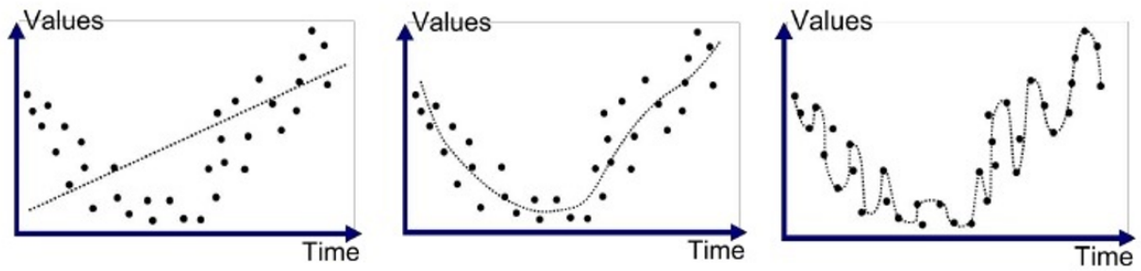


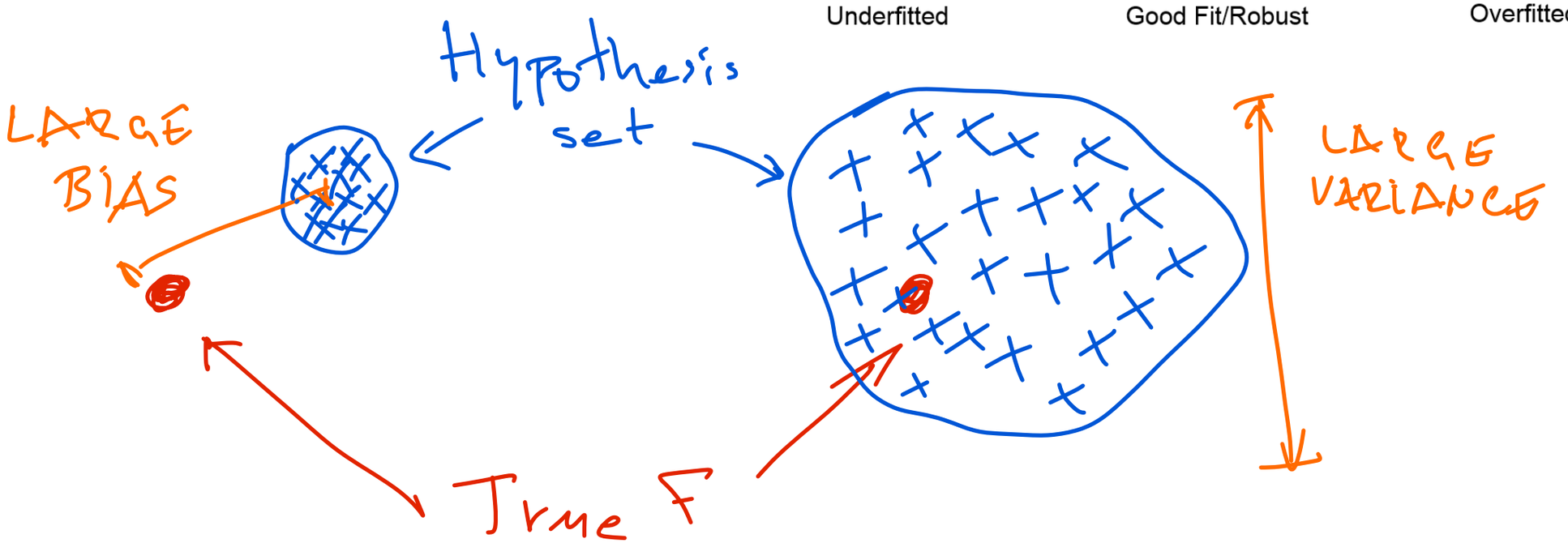
$$J(\mathbf{w}) \;=\; \frac{1}{N}\sum_{n=1}^{N} H(p_n, q_n) \;=\; -\frac{1}{N}\sum_{n=1}^{N} \Big[y_n \log \hat{y}_n + (1-y_n) \log (1-\hat{y}_n) \Big]$$

Bias - Variance Trade-off

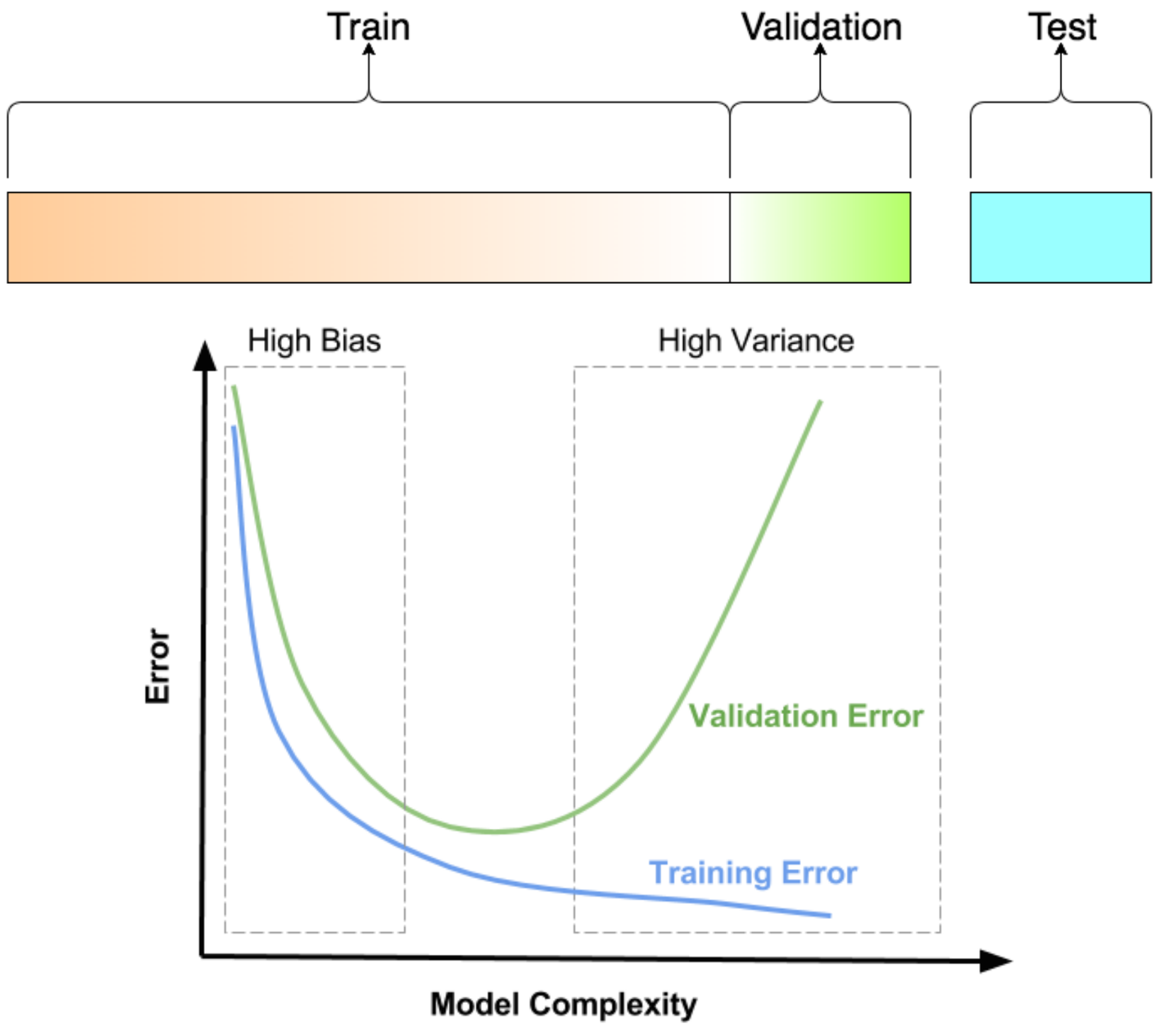
- Under-fitting/over-fitting
- in sample error vs out of sample error
- VC dimension





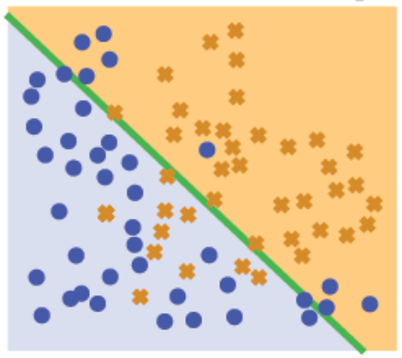


(cross)-Validation



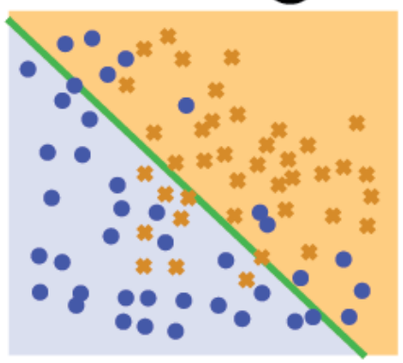
Model 1...

...on Training data. 1



30 10 error: 22.5%
32 8 acc. : 77.5%

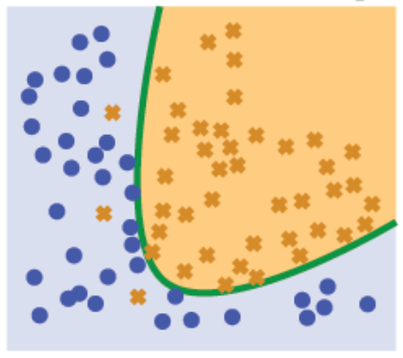
...on Test data. 4



32 8 error: 23.8%
29 11 acc. : 76.2%

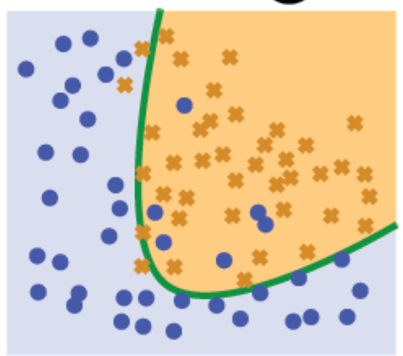
Model 2...

...on Training data. 2



37 3 error: 7.5%
37 3 acc. : 92.5%

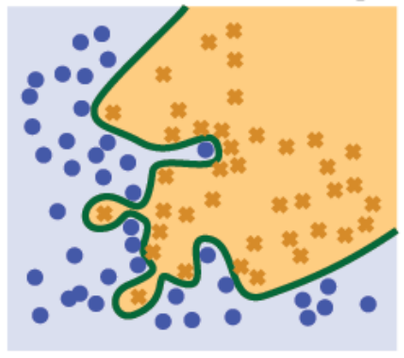
...on Test data. 5



37 3 error: 11.3%
34 6 acc. : 88.7%

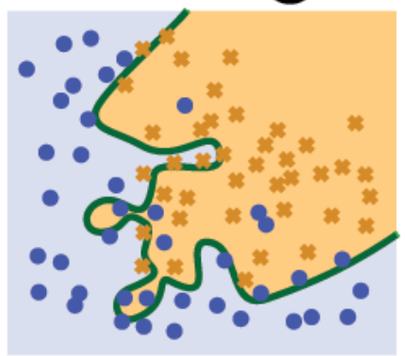
Model 3...

...on Training data. 3

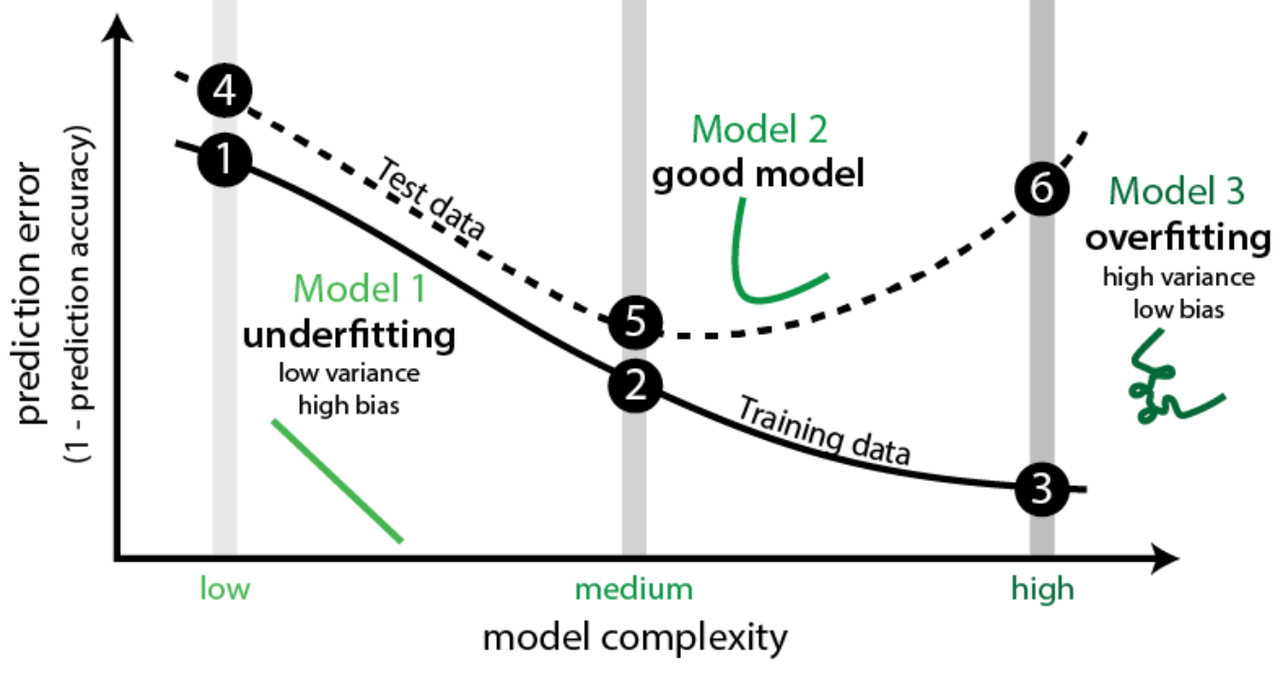


37 0 error: 0%
37 0 acc. : 100%

...on Test data. 6



34 6 error: 21.3%
29 11 acc. : 78.7%



	Remedies

High Bias	• Train longer
	• Increase model complexity
	  • more features
	  • more parameters,
	  • richer architecture

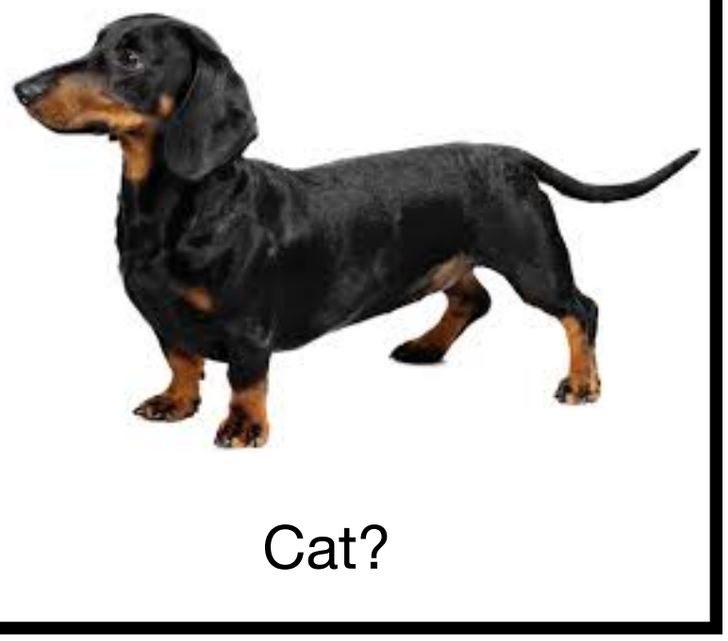
High Variance	• Get more data
	• Decrease model complexity
	  • less features
	  • less parameters,
	  • simpler architecture
	• Regularization
	• Early stopping
	• Drop-out

Data Augmentation:

- When more data are needed, make up new ones! (The way of the god.)
- Translate, rotate, flip, crop, lighten/darken, add noise, de-phase, etc.

Cats

Dogs



Cat?

50% dog, 50% cat?

Machines are lazy and love shortcuts

- A 1-2 years old needs about 2-4 cats to generalise how a cat looks like and to recognise new cats without over-fitting

- Neural Network needs 50k objects to learn with much less neurons and still risks over-fitting

- *Garbage in, garbage out* — non-representative or poor quality data (noise, errors, outliers), irrelevant features

Linear Regression

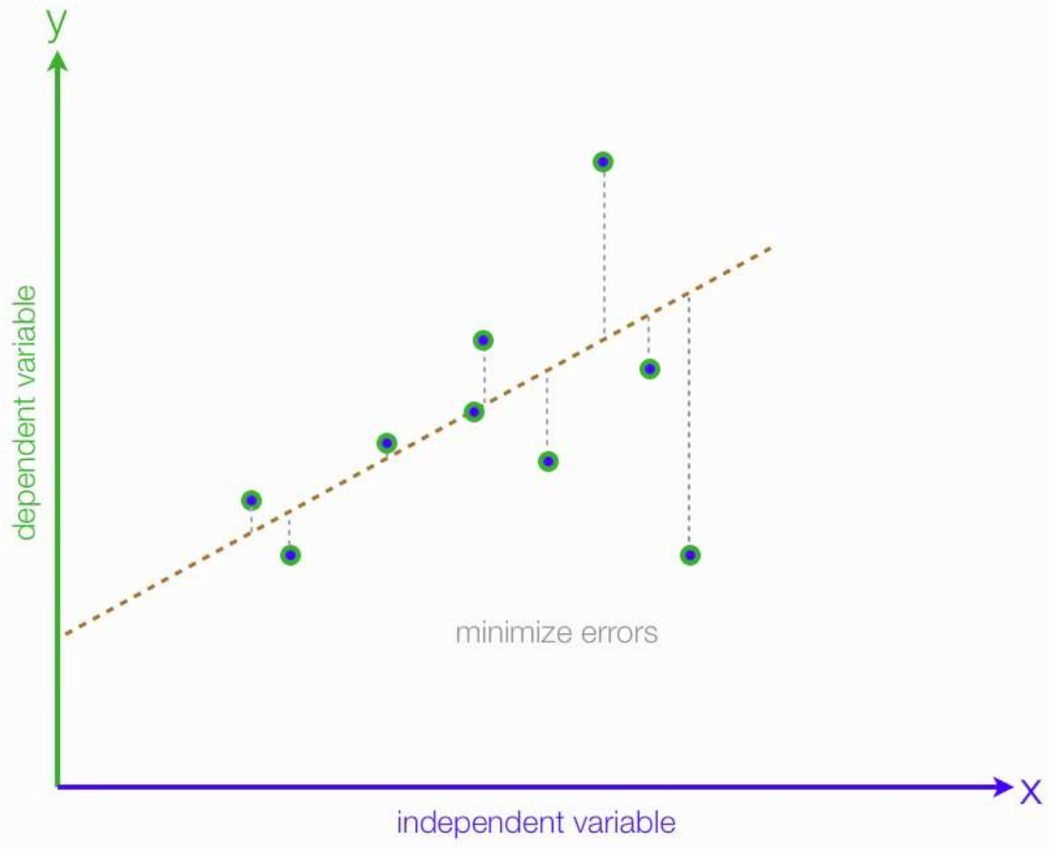
- Linear fitting in N-dim space
  $y = a\,x + b$, or more general $y = \sum a_i\, x^i$

- we have training pairs (x_i, y_i)

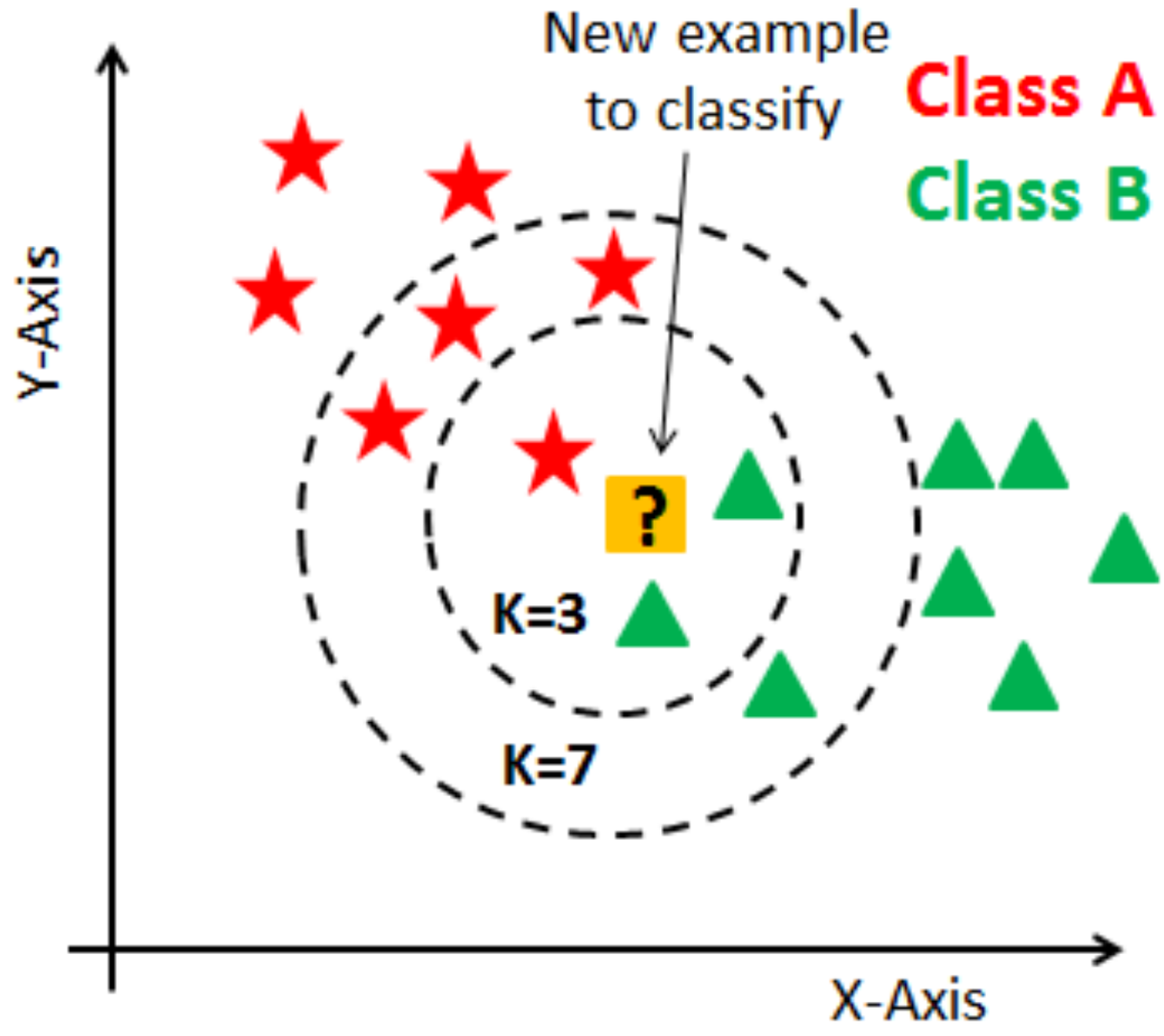
- **Linear in weights!** => works well with polynomials
  $a_0\, x^0 + a_1\, x^1 + a_2\, x^2 + a_3\, x^3 \ldots$

- We can transform our data points in a non-linear way and still use linear regression to find the fit

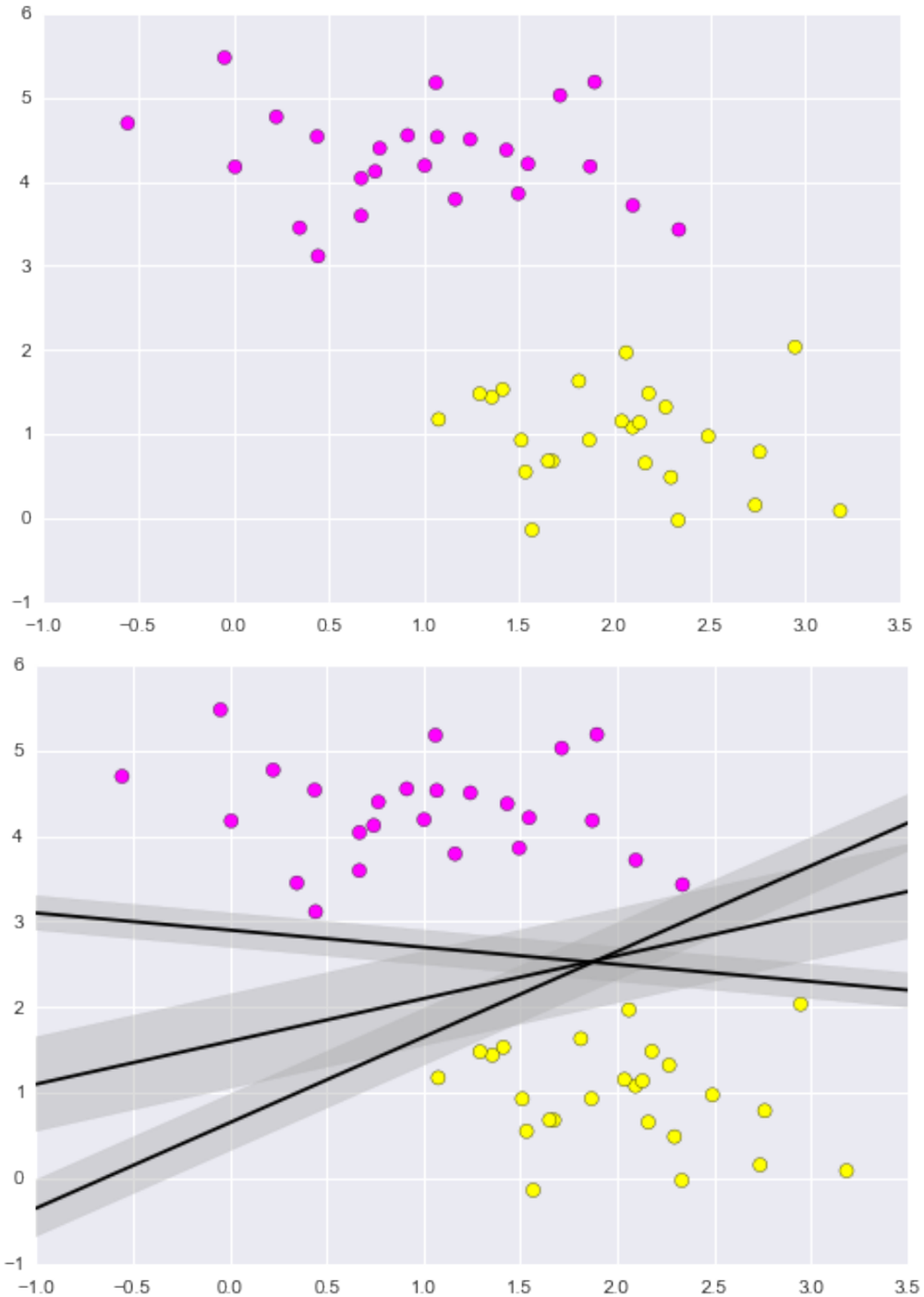
- We control the loss function *L1, L2*, …

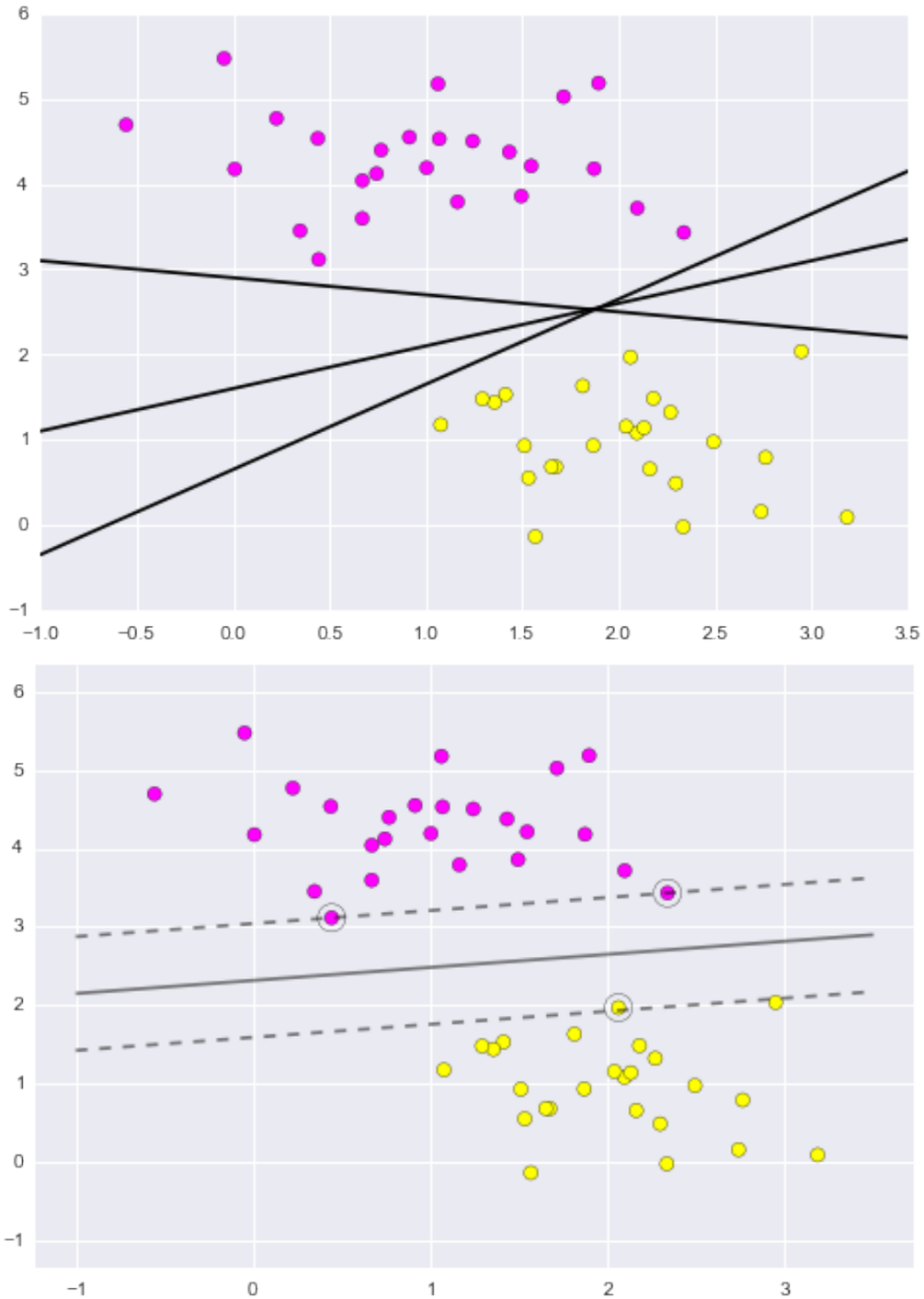


K Nearest Neighbours

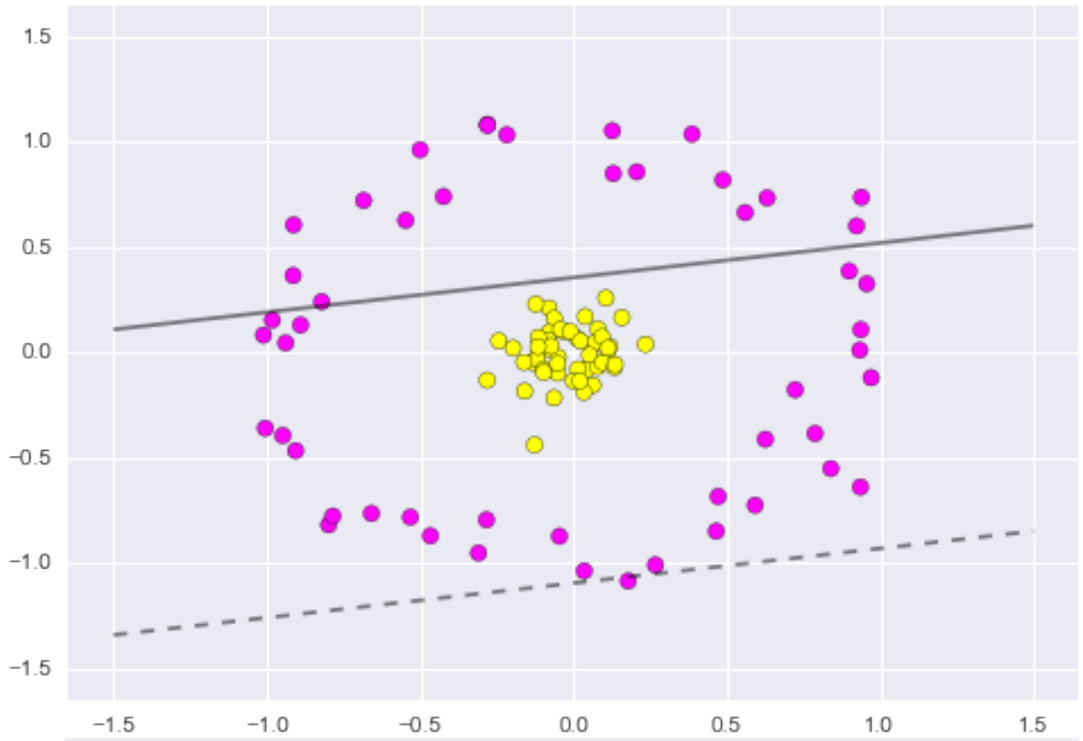


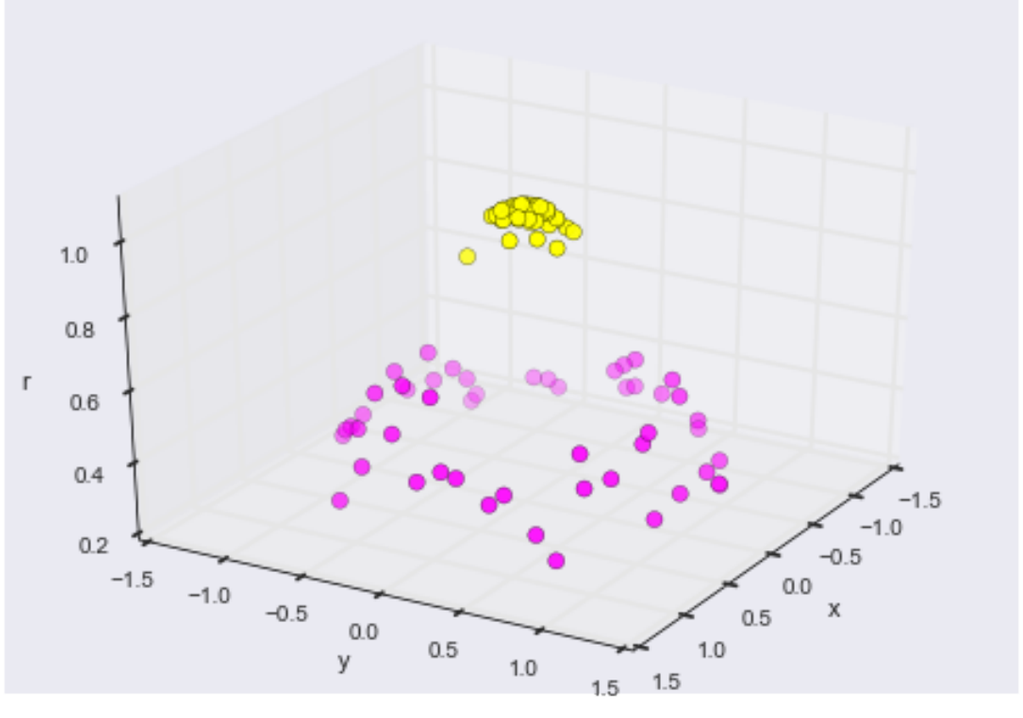
Support Vector Machine

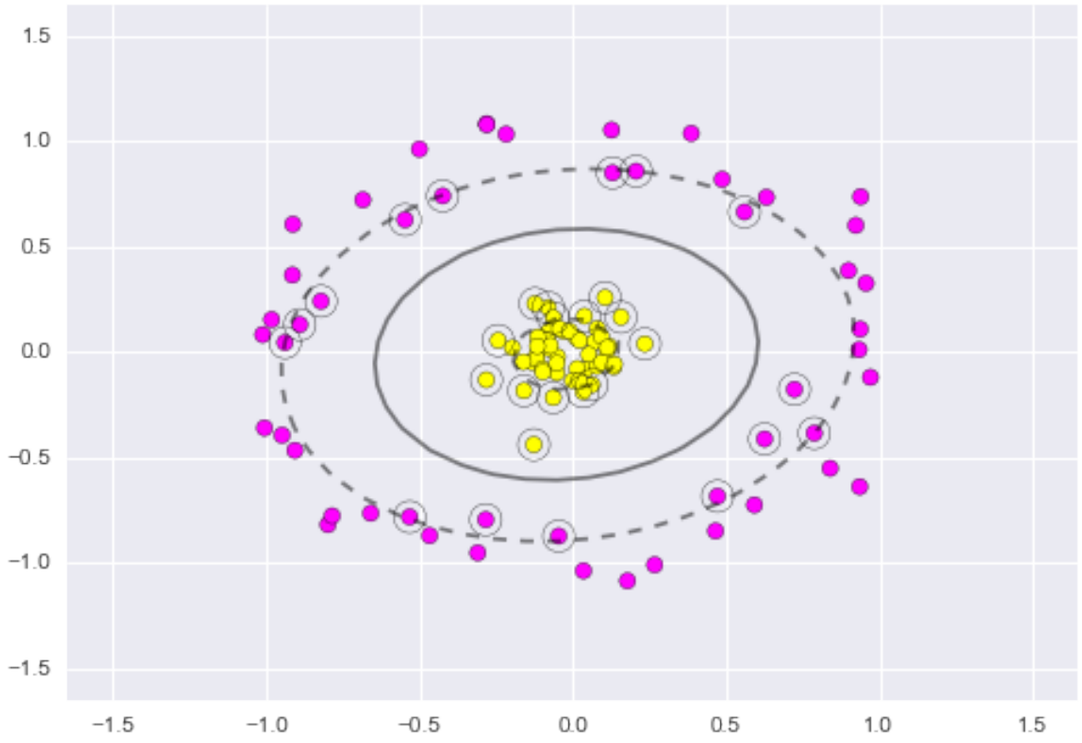




SVM with non-linear (kernel) transformation

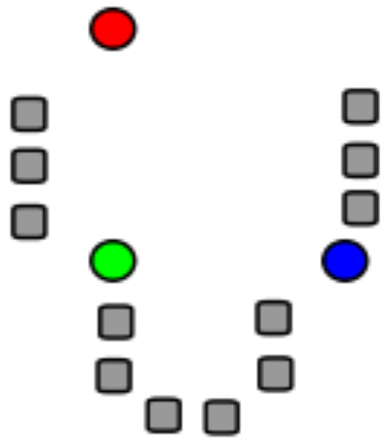




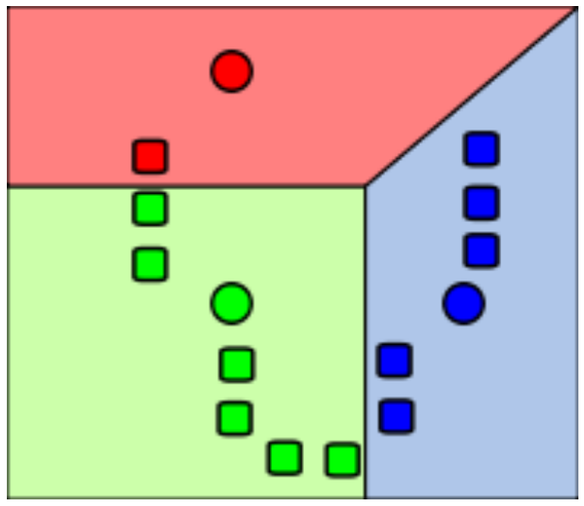


K-Mean Clustering (unsupervised)

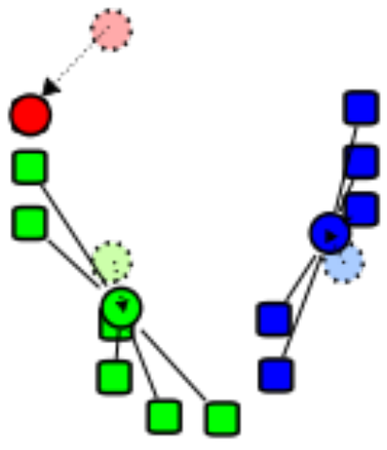
1)



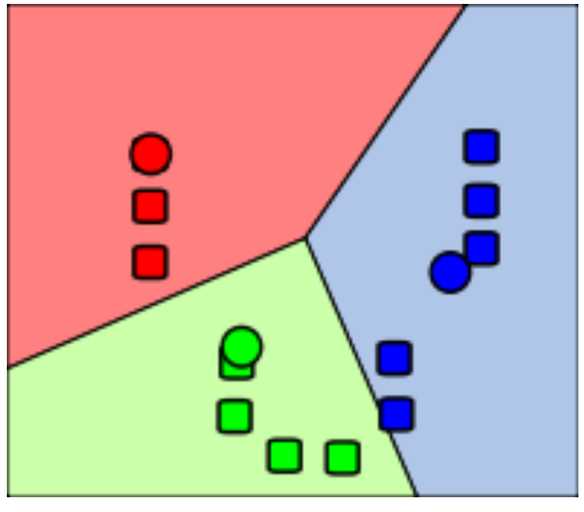
2)



3)



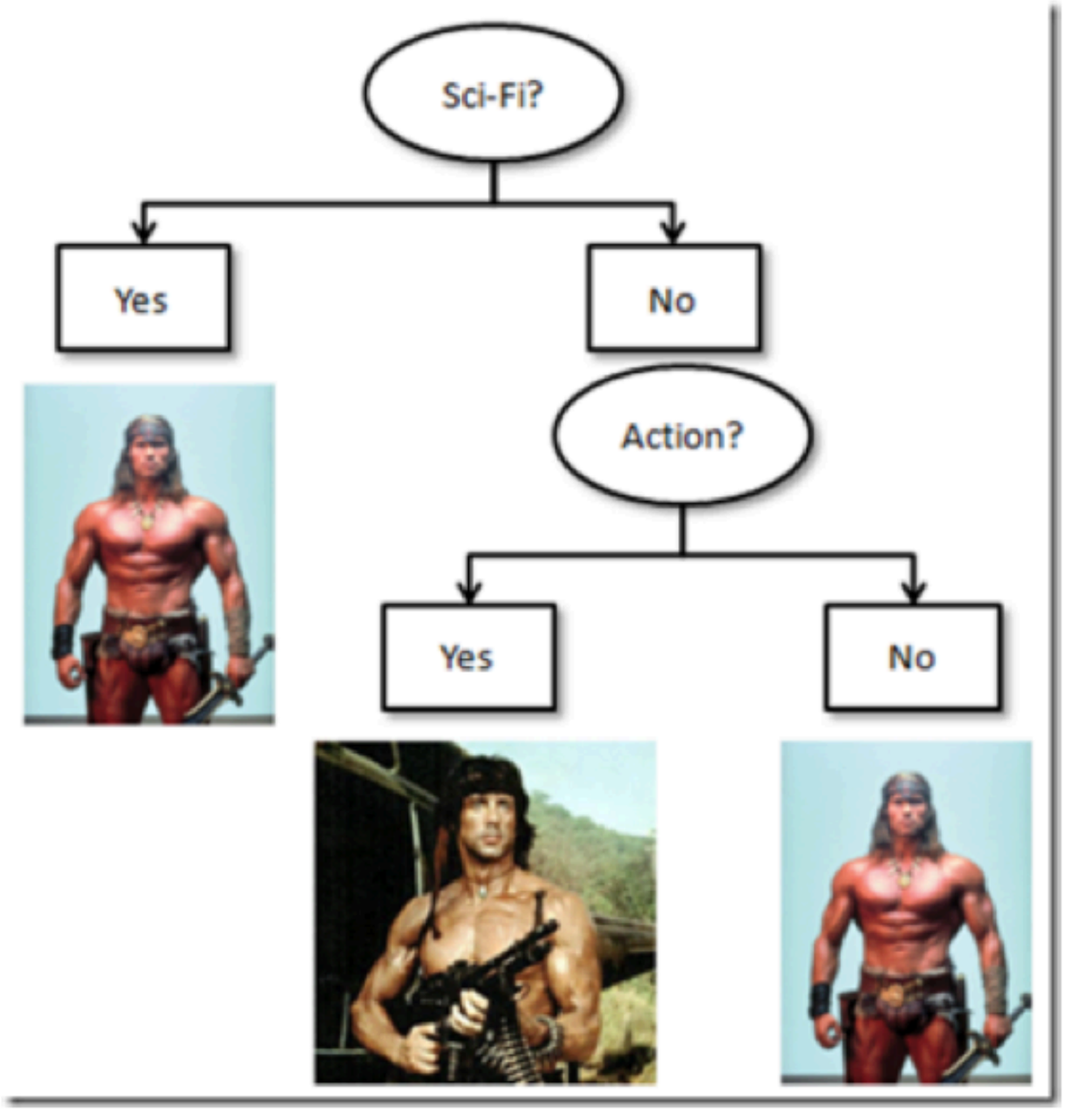
4)

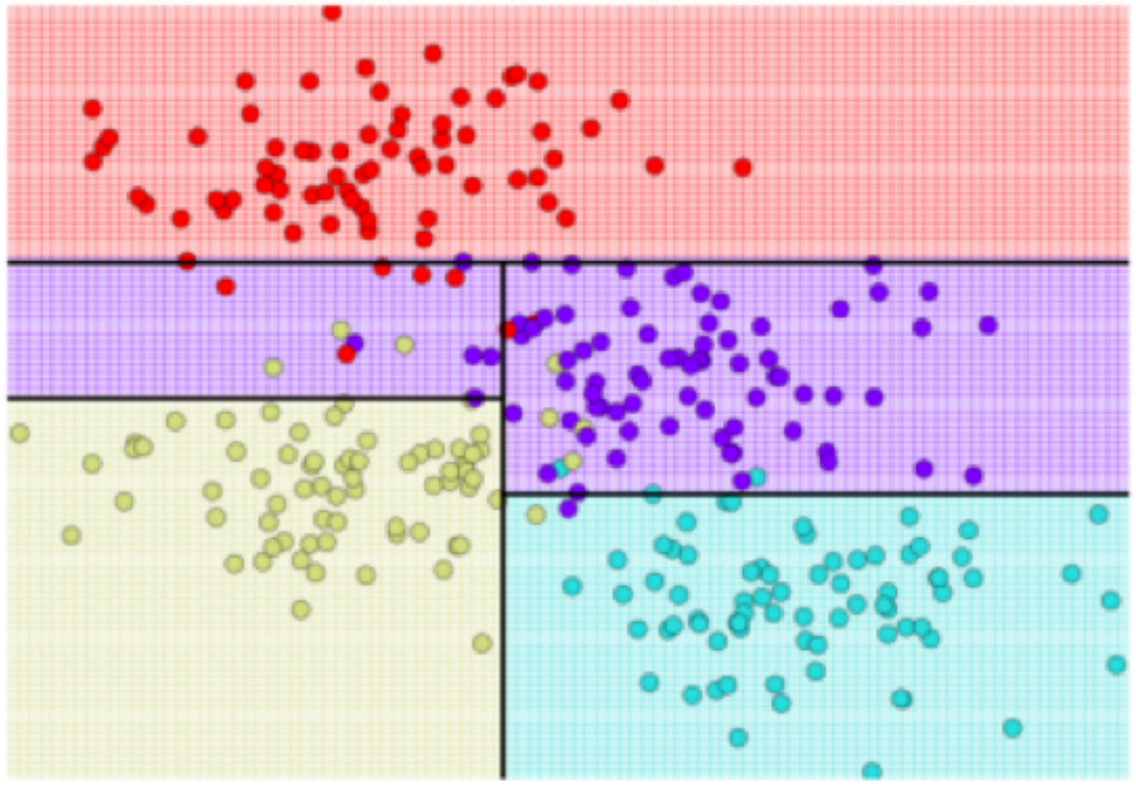


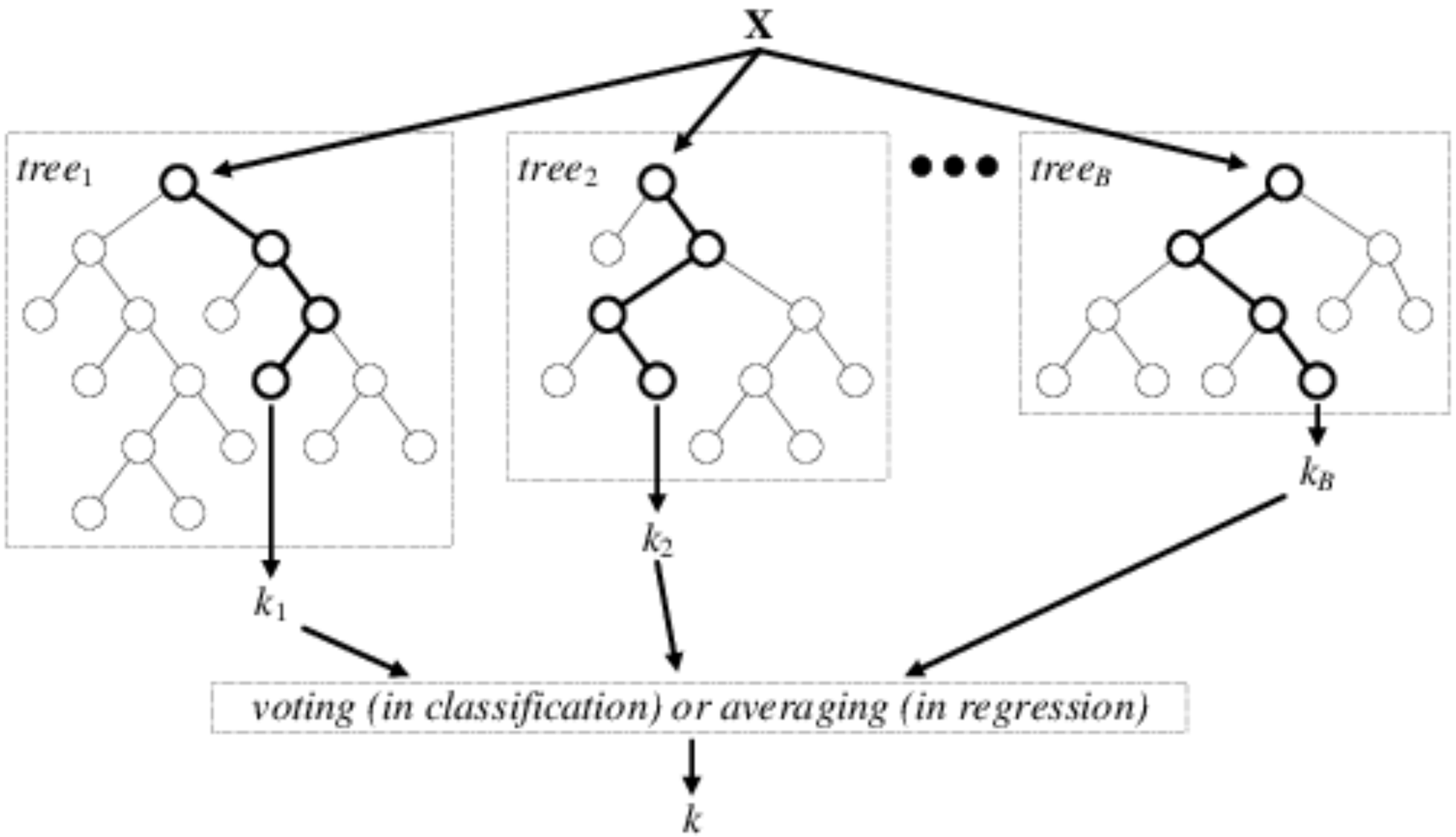
More advanced algorithms

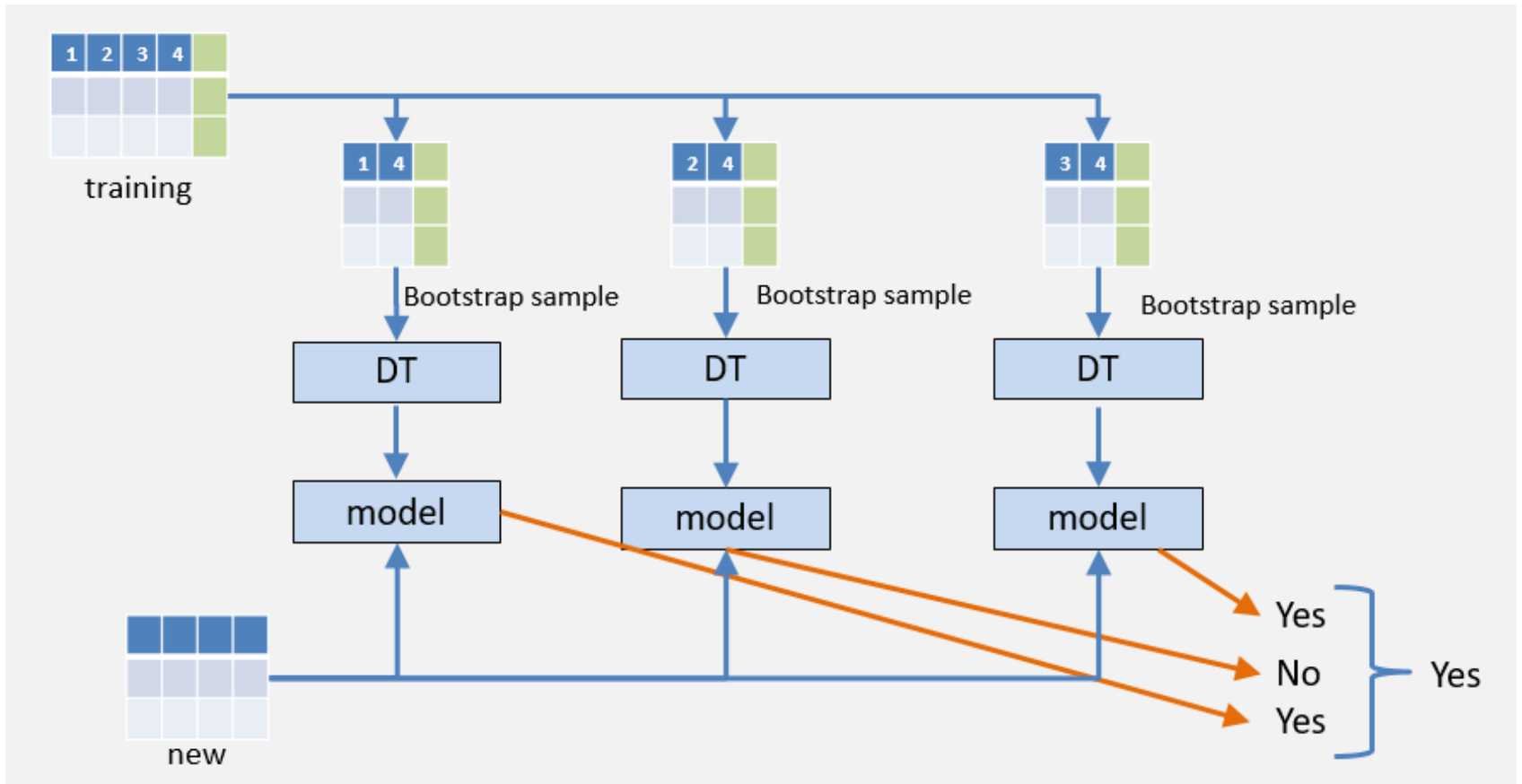
- Ensemble methods:
    - Bagging (averaging lowers variance) / bootstrapping
    - Voting
    - Boosting
- Neural networks

Random Forest (Ensemble of Decision Trees)

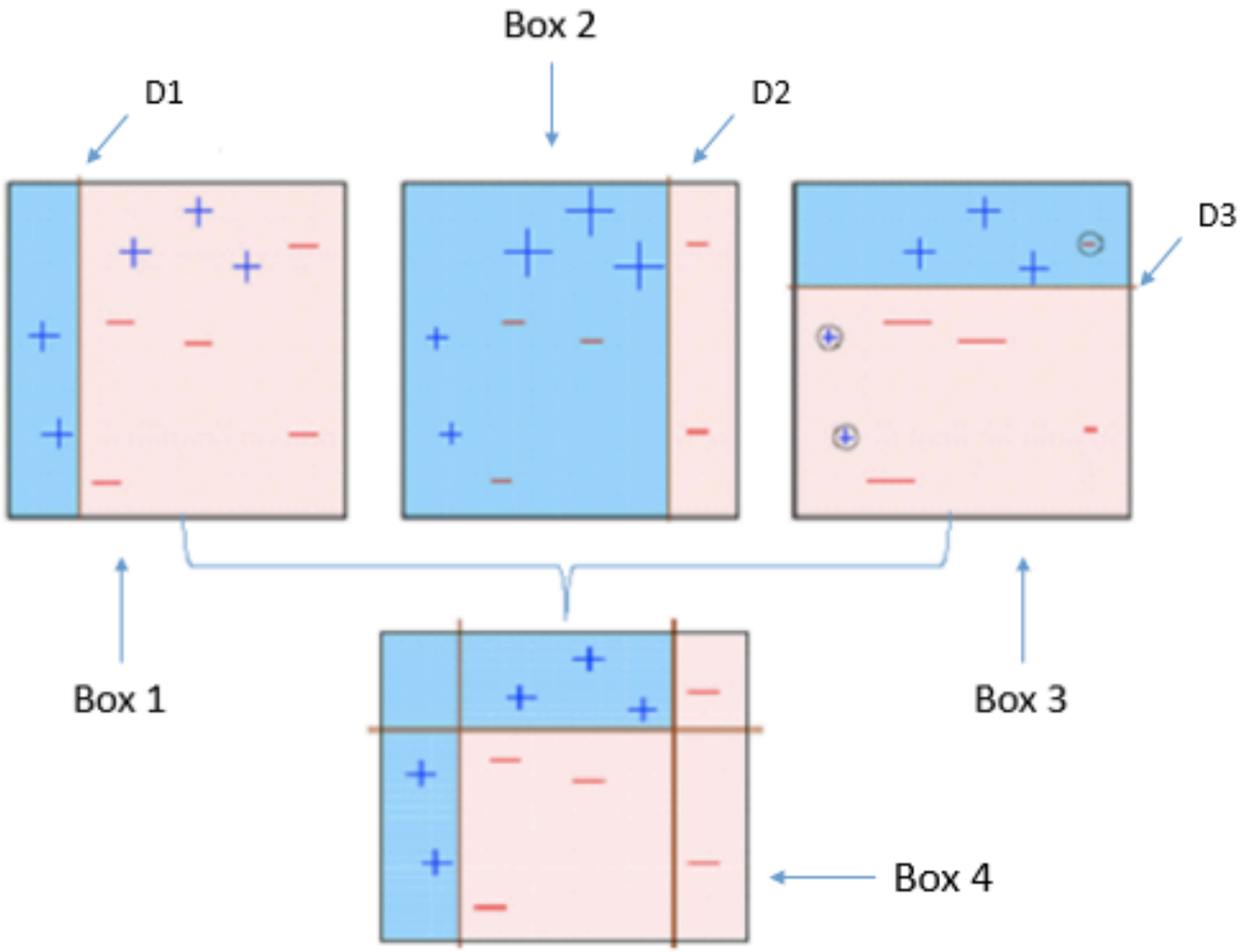




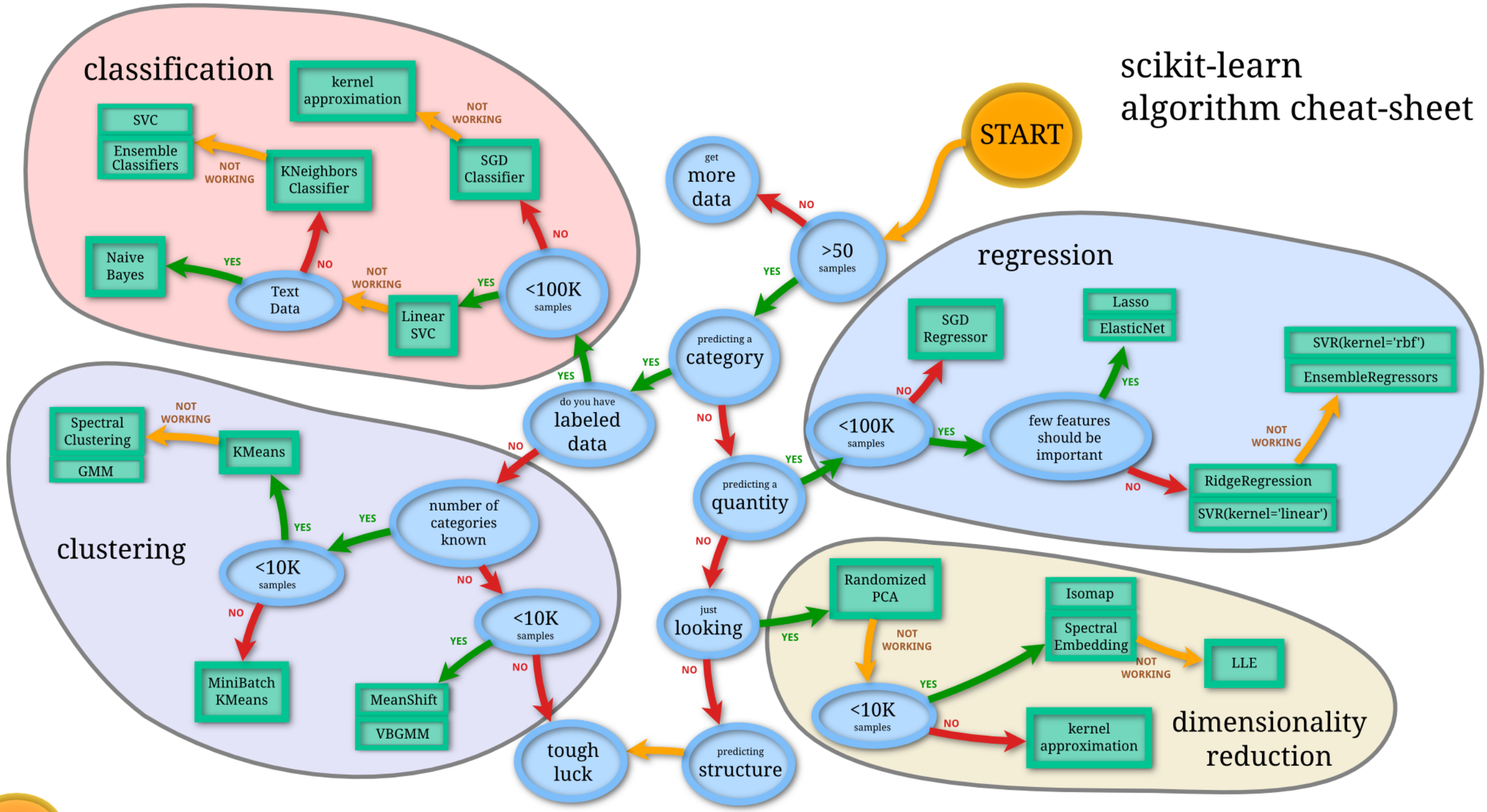

X
tree1
tree2
treeB
k1
k2
kB
voting (in classification) or averaging (in regression)
k


training
Bootstrap sample
Bootstrap sample
Bootstrap sample
DT
DT
DT
model
model
model
new
Yes
No
Yes
Yes

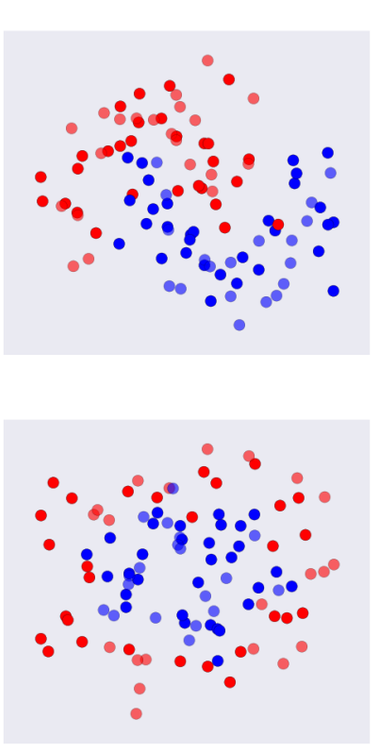
Voting (max/average) with weights (gradient descent)

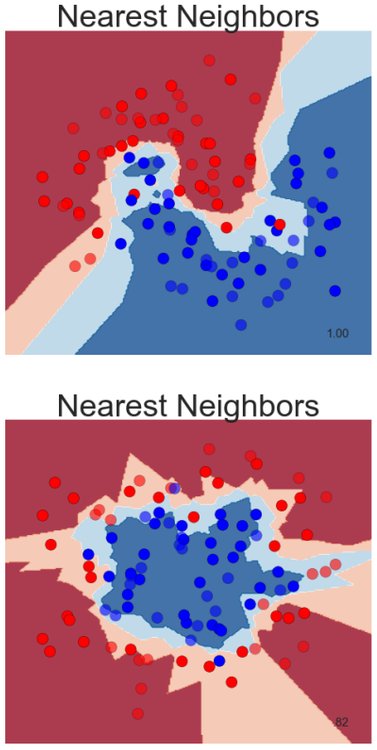


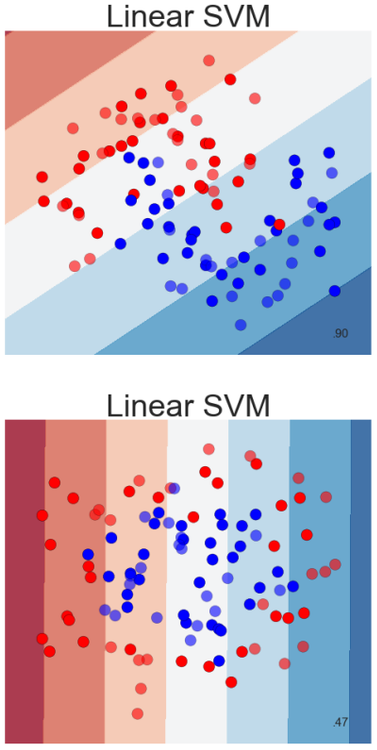
scikit-learn algorithm cheat-sheet

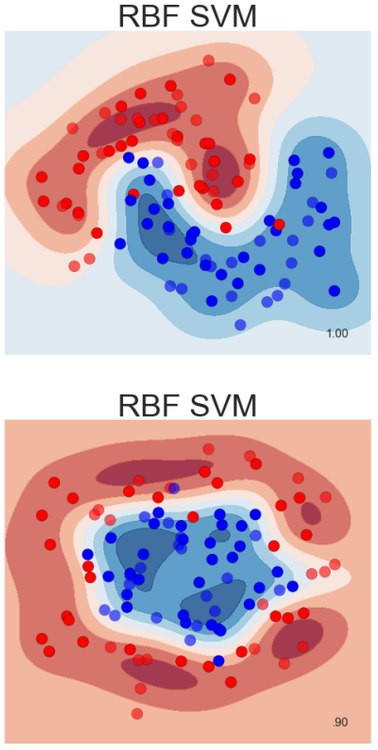


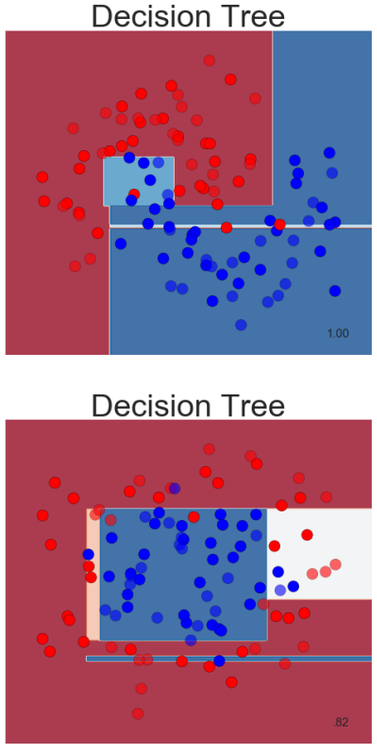
Back

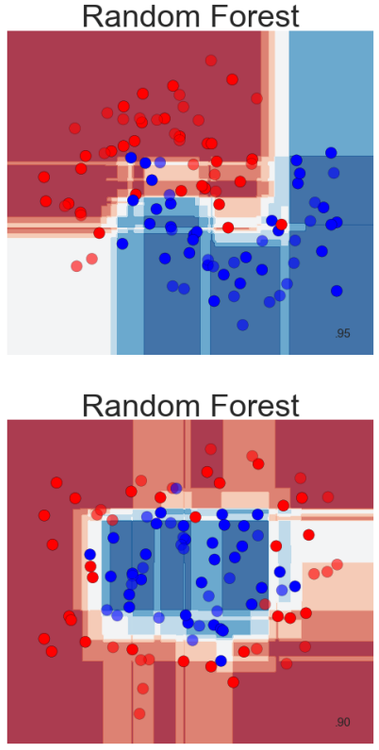



Nearest Neighbors
1.00


Linear SVM
.90


RBF SVM
1.00


Decision Tree
1.00


Random Forest
.95

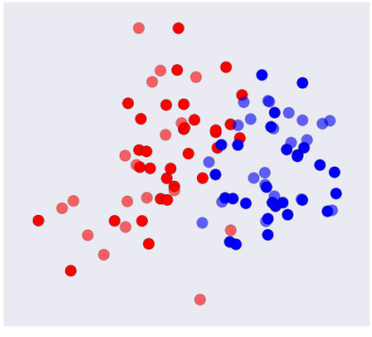
Nearest Neighbors
.82

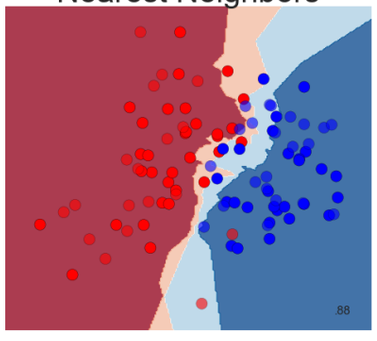
Linear SVM
.47

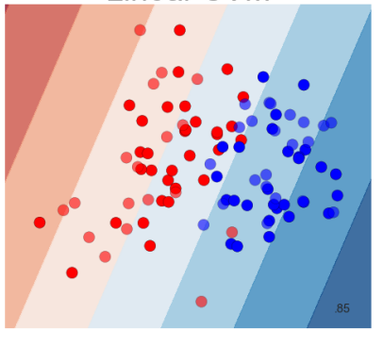
RBF SVM
.90

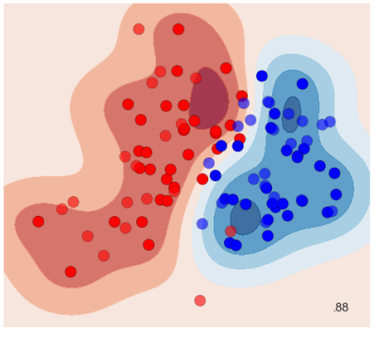
Decision Tree
.82

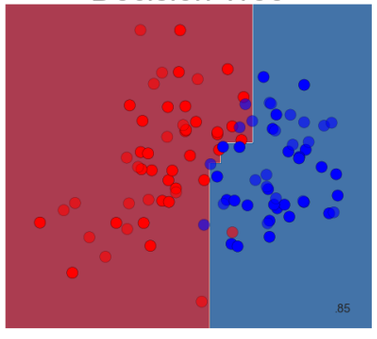
Random Forest
.90

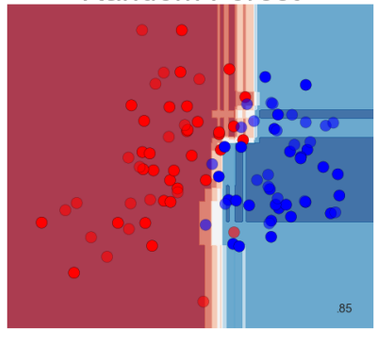



Nearest Neighbors
.88


Linear SVM
.85

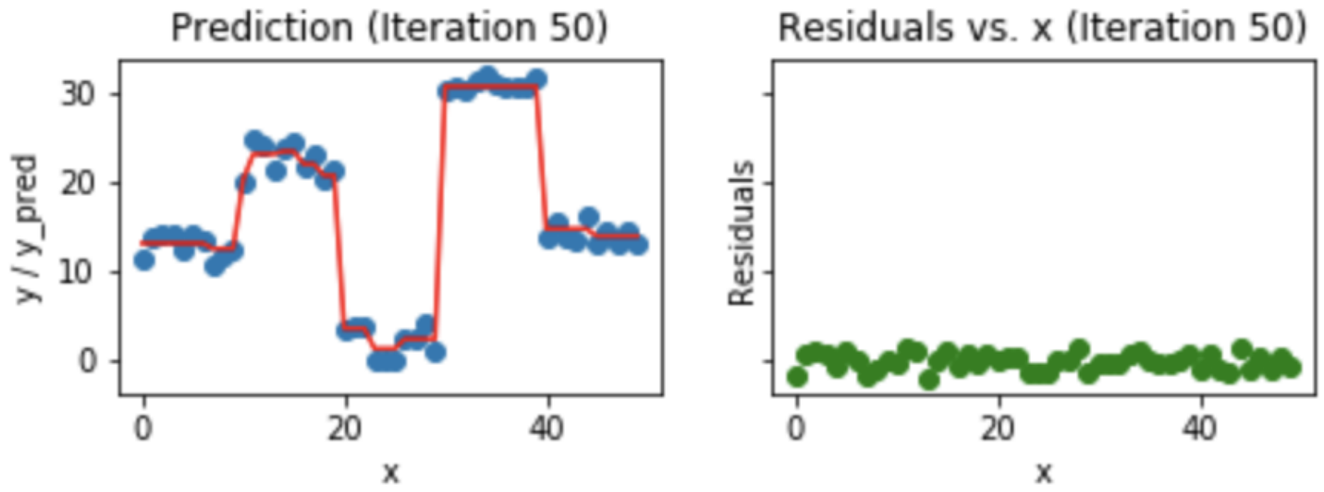

RBF SVM
.88


Decision Tree
.85


Random Forest
.85

Top 5 (supervised)

Algorithm	Comments
Neural Networks	• Take long to train - lot of CPU • Overfits • Requires lot of data
Gradient Boosted Trees	• Fast • Overfit danger
Random Forest	• Robust to overfitting
SVM w/non-linear kernel	• Pretty good
Gaussian Processes	• non-parametric fitting



Scikit-learn

http://scikit-learn.org

Pros:

- ★ Written in Python, language #1 in astronomy today
- ★ Even complicated powerful algorithms are provided
- ★ Single and relatively simple API for all tasks
- ★ Actively being developed, open source, free
- ★ Object oriented, extensible
- ★ Great and practical online documentation with tons of examples

Cons:

- ◉ Not suitable for big data (but it can be tweaked with *dask*, *partial_fit* method, moreover, many other software packages follows a similar API concept)

Really Simple API

0) Import your model class

```
from sklearn.svm import LinearSVC
```

1) Instantiate an object and set the parameters

```
svm = LinearSVC()
```

2) Fit the model

```
svm.fit(X_train, y_train)
```

3) Apply / evaluate

```
print(svm.predict(X_train))
print(y_train)
```

- hyperparameters specifying the model in the family of models
- feature vectors **X**

  N rows = number of points

  m columns = number of features
- values/labels **y**

```
from sklearn.ensemble import RandomForestClassifier
```

```
rf = RandomForestClassifier(n_estimators=100)
```

```
rf.fit(X_train, y_train)
```

Hands-on Session

- Datasets:

  - Ondrejov 2m telescope spectral lines (pca+classification)
    spectral_lines.npz
    spec=spectra, target=class

  - SDSS photometry stars + galaxies (classification)
    sdss_photo.csv
    u,g,r,i,z, u-g, g-r, r-i, i-z, target class

  - SDSS photometric redshift of quasars (regression)
    sdssphotoz.npy
    u,g,r,i,z,redshift

  - BATSE duration distribution of GRBs (clustering)
    T90 is the 5th column

to be continued…

**“Once you stop (machine) learning,
you start dying.”**

– Albert Einstein