Linear Models
Random Forests
Model Evaluation

Gerrit Gruben

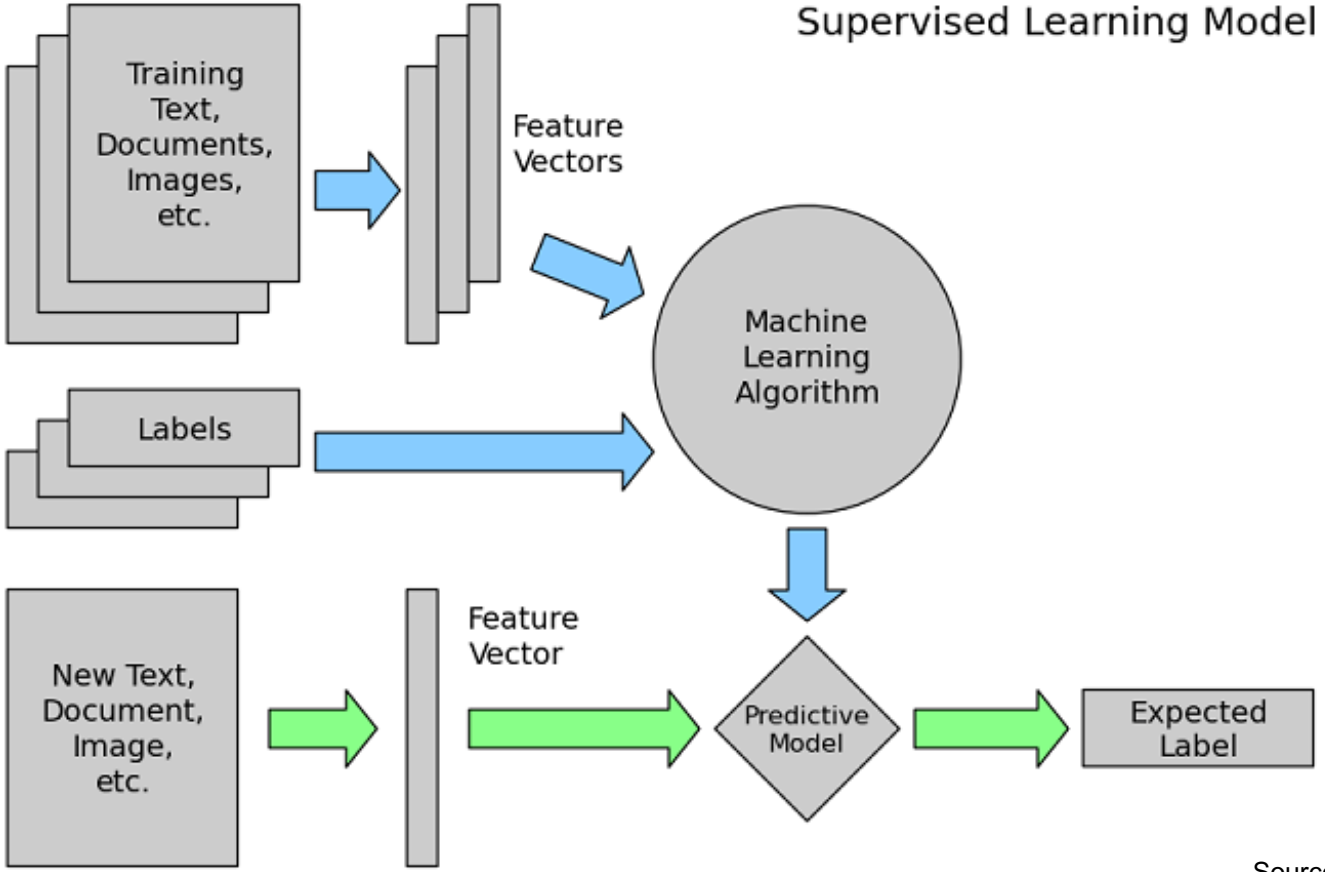
Outlook

In this module you will learn about,

- Decision tree based models, what they are and how to use them, incl.
  - Decision Trees
  - Random Forests
  - Gradient Boosted Trees
- Understand the setting of supervised learning and how to solve problems with them.
- Get the problem of overfitting and techniques to spot and deal with it.

Supervised Learning

… and the problem of overfitting.



Source: All Programming Tutorials

Supervised Learning (Mathematical)

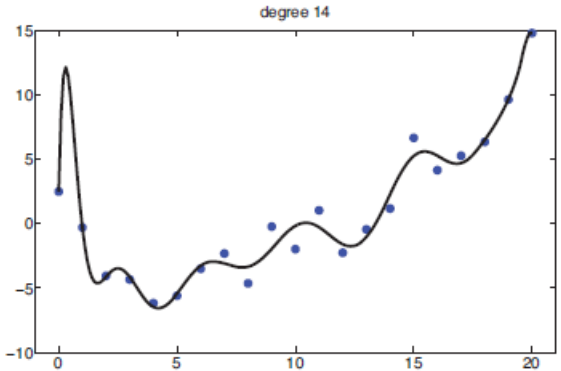
- Given from a joint distribution on X and Y:
  - training samples $x_1, \ldots, x_n$ (independent variables, features)
  - accompanying labels $y_1,\ldots,y_n$ (dependent variables)
- Find a model f: X → Y, s. t. given only a x from this distribution, f(x) is the "best" guess. Interpretations:
  - **Function Approximation:** There is a relation $y = f(x) + \varepsilon$, with ε noise, and we want to estimate f.
  - **Probabilistic:** We model p(y | x), with the relation p(x, y) = p(y | x) p(x). The "perfect" model would be $\text{argmax}_y p(y \mid x)$, i.e. the mode of the distributions p(y | x_0) for fixed x_0. This is the **MAP** (maximum a posteriori) **estimate** (we add training set to the conditional set).

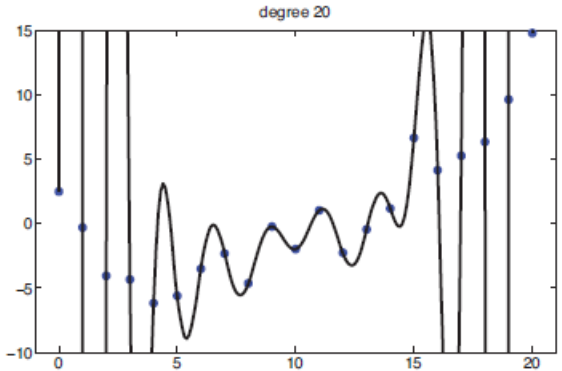
Classification / Regression

- Classification: Y discrete
  - #Y = 2: **binary classification**,
  - #Y > 2: **multiclass classification**,
  - If classes are not mutually exclusive: **multi-label classification** (best seen as **multiple output model**)
- Regression: Y real

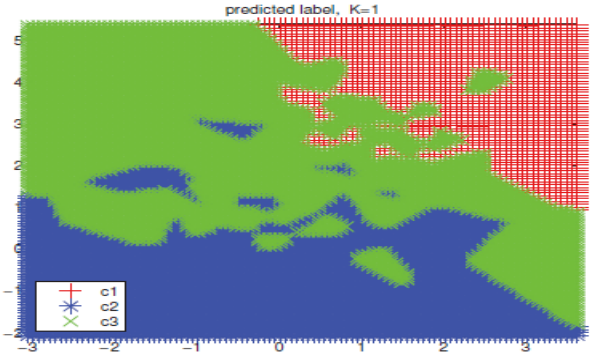
Overfitting

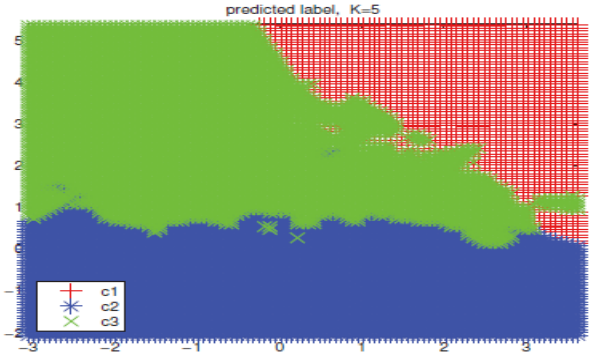
- **Overfitting** = overly complex model learns noise. (aka. *Bias-variances tradeoff)*

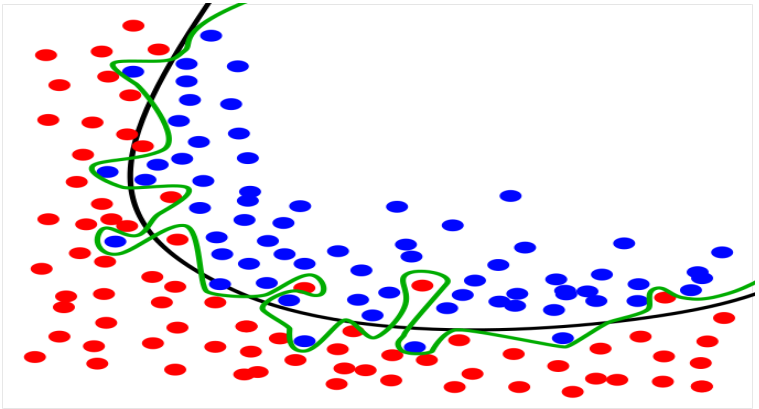




Overfitting Surfaces







Linear Models

Linear regression, logistic regression, L1 & L2 regularization

Linear Models

- Linear models are a power horse, esp. with non-linear features.
- Robust, well-understood, works on big data (if you do not overdo feature extraction).
- Requires one-hot encoding of categorical features, numerical ones should be Z-normalized to make regularization work well.
- Other trick: to make linear models affine (i.e. add a constant **bias**), extend each data row by a one.

```
import sklearn.preprocessing as pp
# One-hot encoding
oh_enc = pp.OneHotEncoder(
n_values="auto", # num. of. cats
# which columns are categorical?
categorical_features="all",
sparse=True # hold as sparse?
)
oh_enc.fit(some_data)
oh_enc.transform(some_data)
# optionally: to_array() to go back from sparse representation
# it's always fit/transform
```

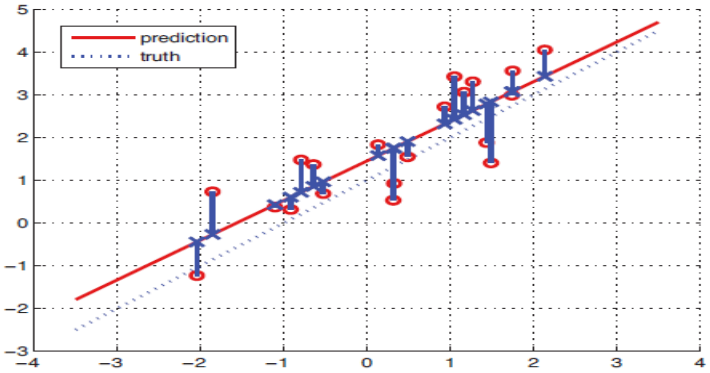
```
# Label encoding
le = pp.LabelEncoder()
le.fit([1, 2, 2, 6])
# also fit_transform
# le.classes_ == array([1, 2, 6])
le.transform([1, 2, 2, 6])
# == array([0, 0, 1, 2])
# Trick with Pandas
from collections import defaultdict
d = defaultdict(pp.LabelEncoder)
fit = df.apply(lambda x: d[x.name].fit_transform(x))
df_new.apply(lambda x: d[x.name].transform(x))
```

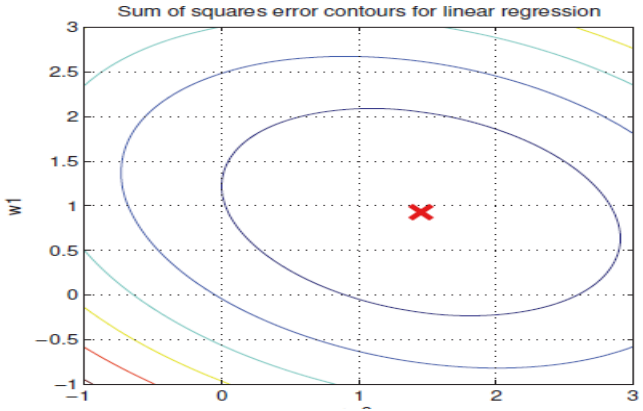
Empirical loss

- Given a data set D of d-dimensional feature vectors $x_1, \ldots, x_n$ and labels $y_1, \ldots, y_n$.
  - Define loss on a sample l(y', y, x), y' prediction. Often independent of x.
  - i-th. loss: $l_i(\theta) = l(f_\theta(x_i), y_i)$.
  - **empirical loss**: $l(\theta) = \sum_{i=1}^{n} l_i(\theta)$.

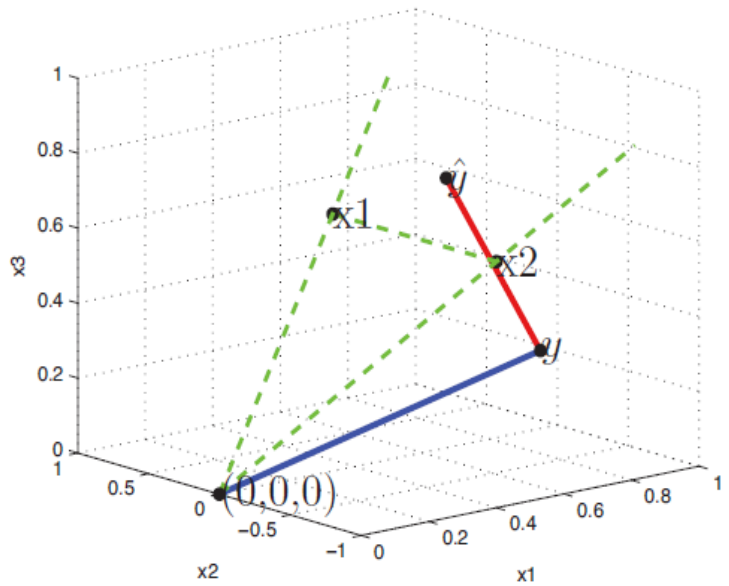
Least Squares

- Model parameter Θ is a d-dim. **weight vector** w.
- $l(w) = RSS(w) = \sum_{i=1}^{n}(y_i - w^t x_i)^2$ **(residual sum of squares)**





Geometric Proof (N >= D)



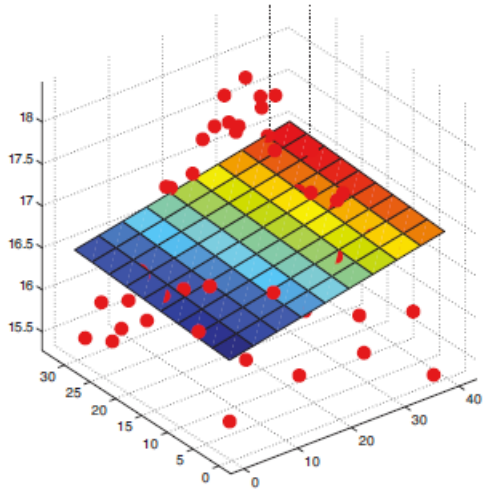
Idea: Project y on the **column space** of X (data matrix with instances as rows). Coordinate vector of y is then w.

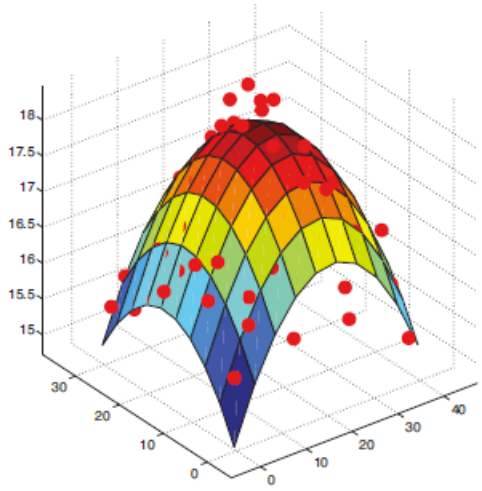
$$P = X(X^T X)^{-1} X^T$$
$$\Rightarrow Pw = y' = Xw'$$
$$\Rightarrow w' = (X^T X)^{-1} X^T y.$$

P is called **hat matrix**. This formula can also be calculated by calculus: Derive loss by w and equate to 0, solve by w like in school.

Carl-Friedrich Gauß (1777-1855)

Polynomial feats
on remote sensing data





Problems with Least Squares

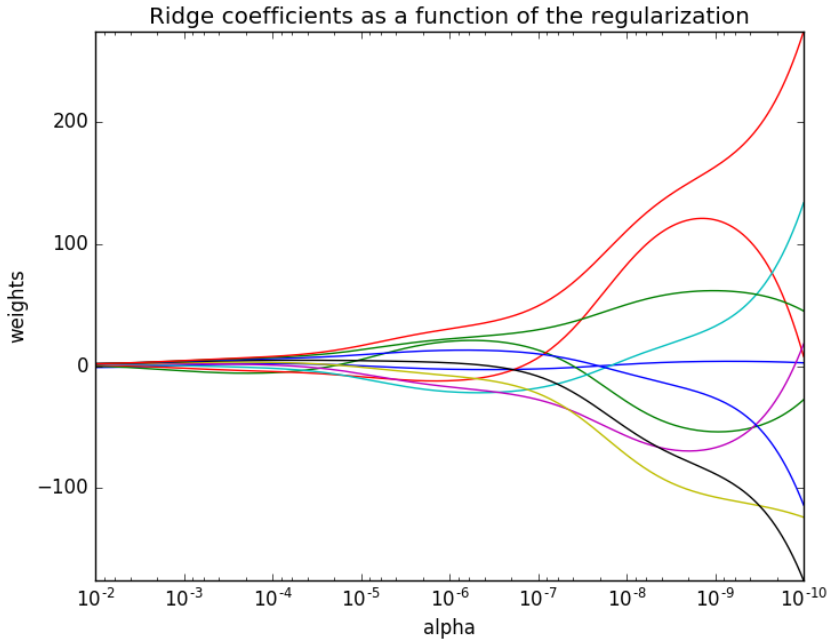
- Weights become arbitrarily large → overfitting.
- $(X^T X)$ sometimes not invertible: $(X^T X + \lambda I)$ for a non-negative lambda is more robust.
- Equivalent to modifying loss to:

$$l(w, \lambda) = RSS(w) + \lambda L_2(w)$$

$$= \sum_{i=1}^{n} (y_i - w^t x_i)^2 + \lambda({w_1}^2 + \cdots + {w_d}^2)$$

- This is **ridge regression** or linear regression with L2 regularization.

Lambda / Weights



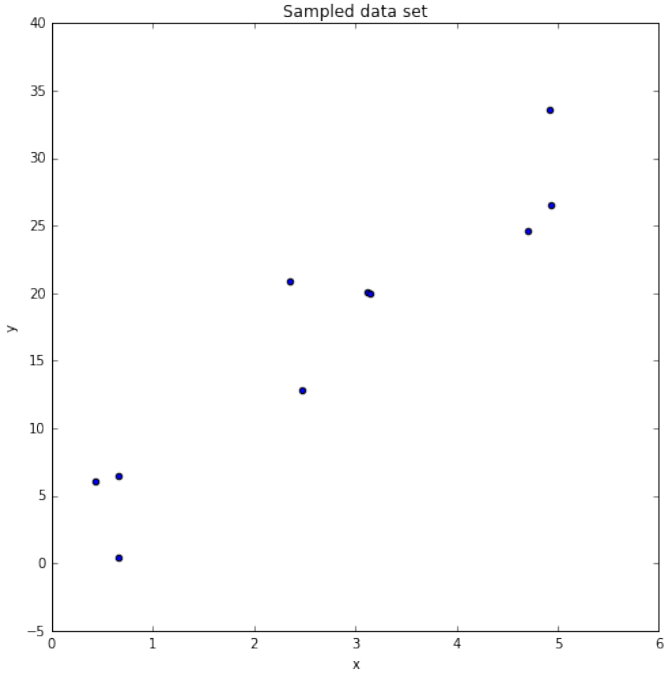
Lambda

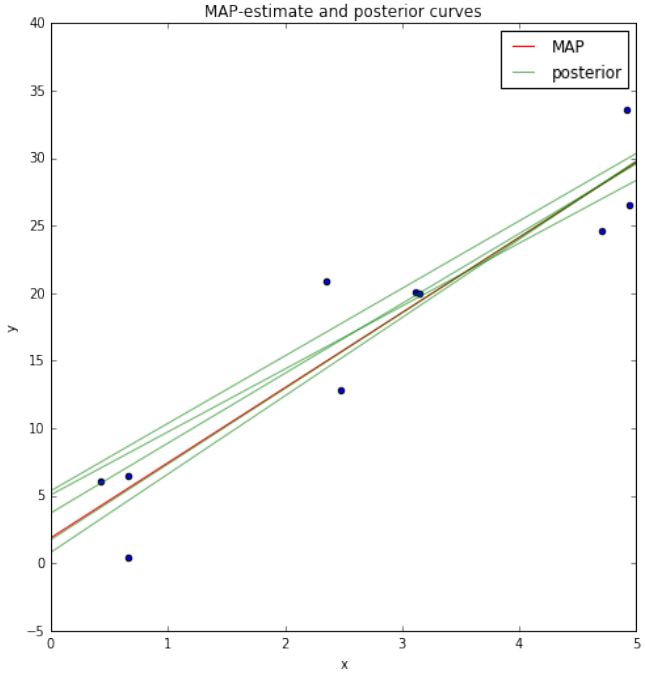
- Lambda is a **hyper parameter**.
- Chosen on a log-scale (try …0.05, 0.1, 0.5, 1, 5, …).
- The smaller the lambda, the larger the weights can get and the model can become more complex.
- More data → smaller lambda.
- If you add more features → larger lambda. **But** that depends on the precise feature added and how it correlates to the target variable.

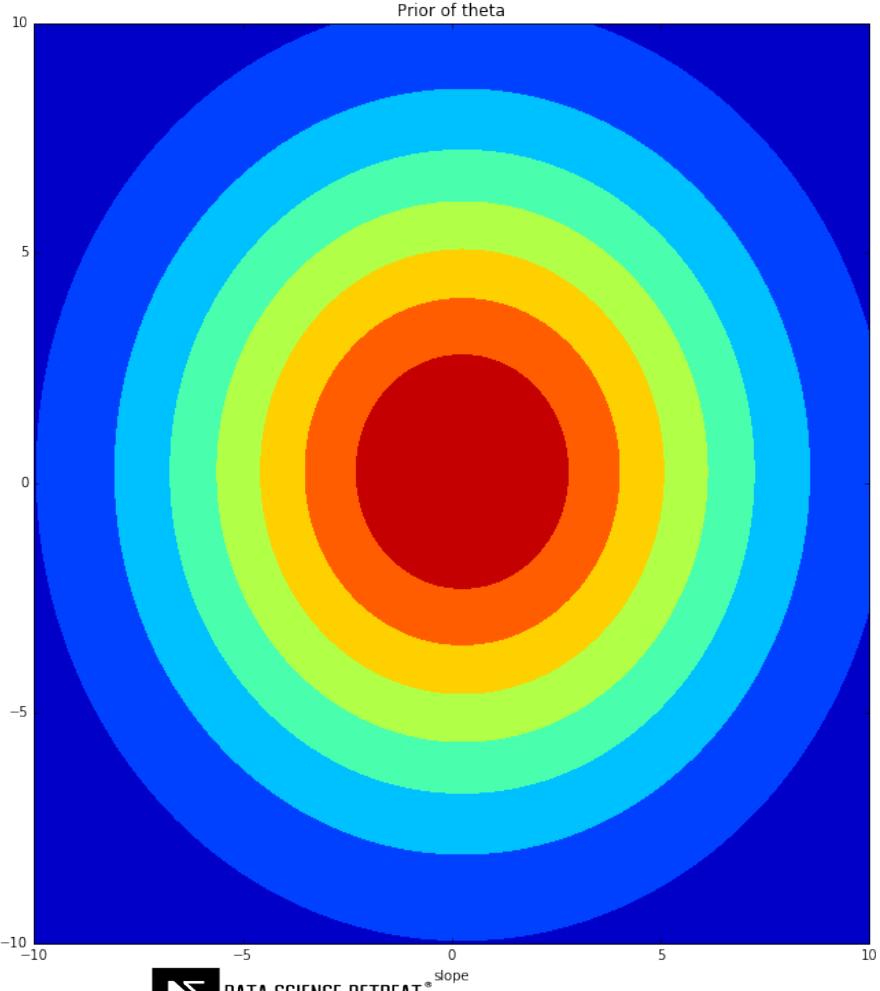
Other variants

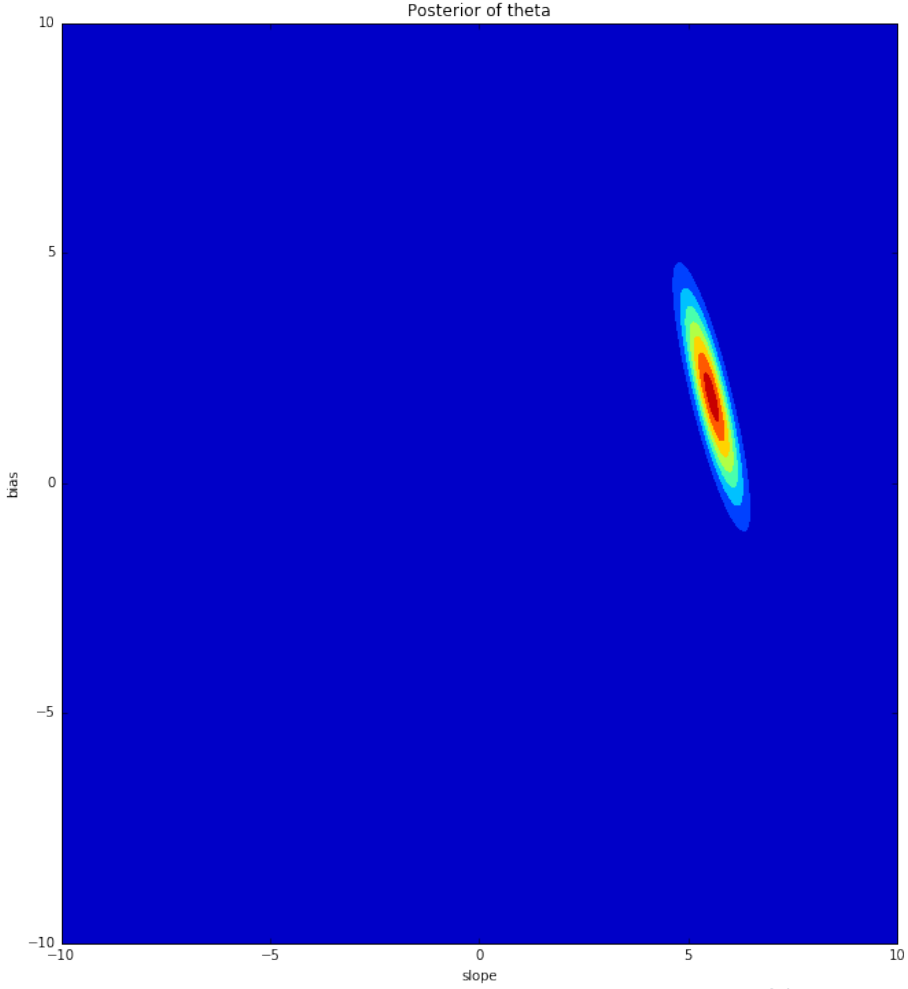
- **Lasso**: $\lambda L_1(w) = \lambda(|w_1| + \cdots + |w_d|)$
  - Leads to **sparsity** of features (presses weights to 0).
- **Elastic net**: Combine both L1 and L2 on all weights.
  - better deals with correlated features,
  - harder to tune as a new hyper parameter is added.
- **Robust regression**: More stable to outliers, rarely used as one has to solve a quadratic program (QP).
- **Bayesian Linear Regression**: Do not compute only one weight, but keep a distribution the weights!

Bayesian Linear Regression example





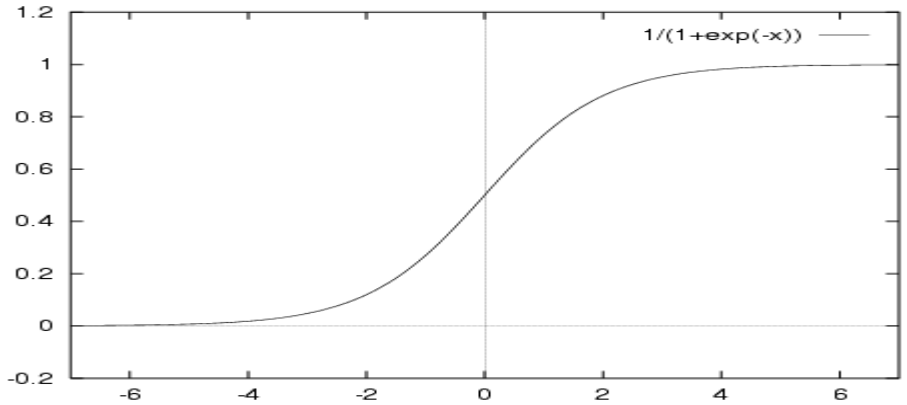

Prior of theta
10
5
0
−5
−10
−10
−5
0
5
10
slope


Posterior of theta
10
5
0
−5
−10
bias
−10
−5
0
5
10
slope

Logistic Regression

- Logistic Regression is simply linear regression + **squashing**, with the **sigmoid** function:

- $\sigma(h) = \frac{1}{1+ e^{-h}}, \; h = w^t x$



Logistic Regression II

- Outputs a probability in [0, 1] that outputs the likelihood of the label being 1: P(y = 1 | x, D)
  → binary classification problem.

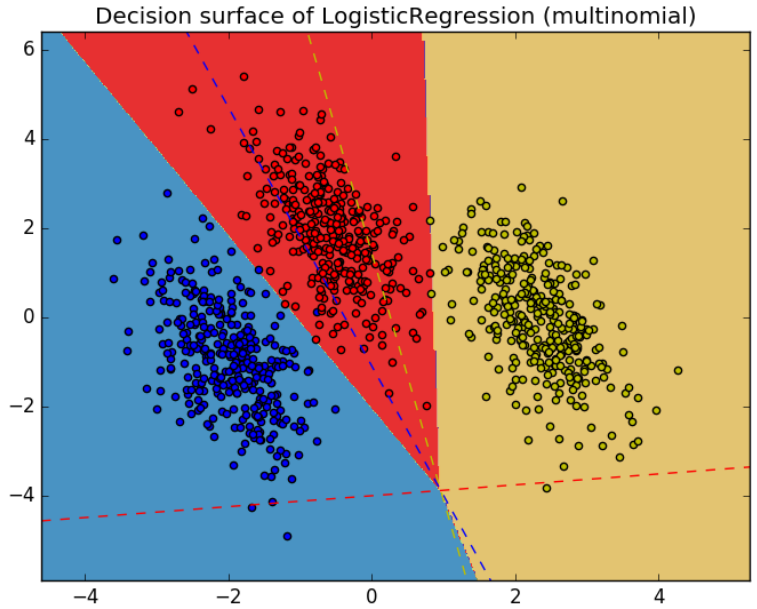
$$logloss(w) = \sum_{i=1}^{n} \ln(1 + e^{-y_i(w^t x_i)})$$

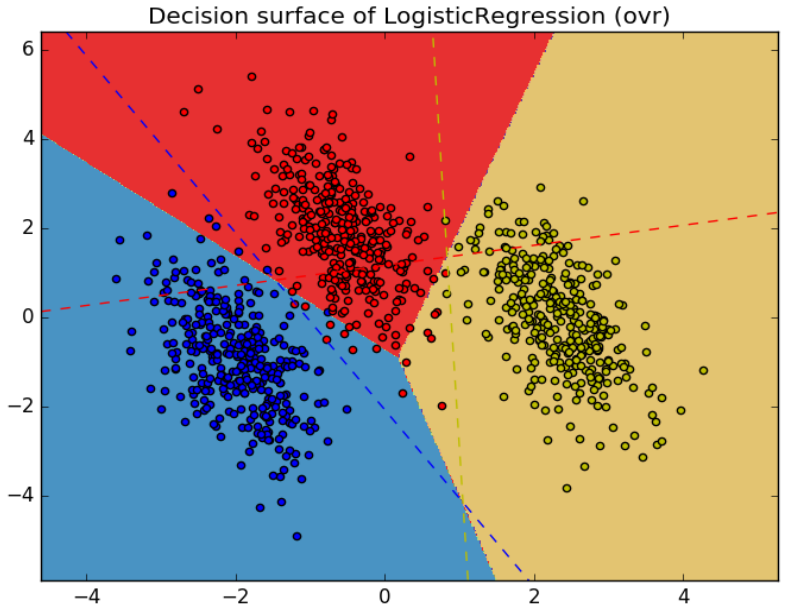
- **multinomial logistic regression**: ($\mathbf{w}_i$ are model parameters)

$$P(y = c \mid x, \boldsymbol{w}_1, \ldots, \boldsymbol{w}_C) = \frac{\exp(w_c^T x)}{\sum_{c'=1}^{C} \exp(w_{c'}^T x)}$$

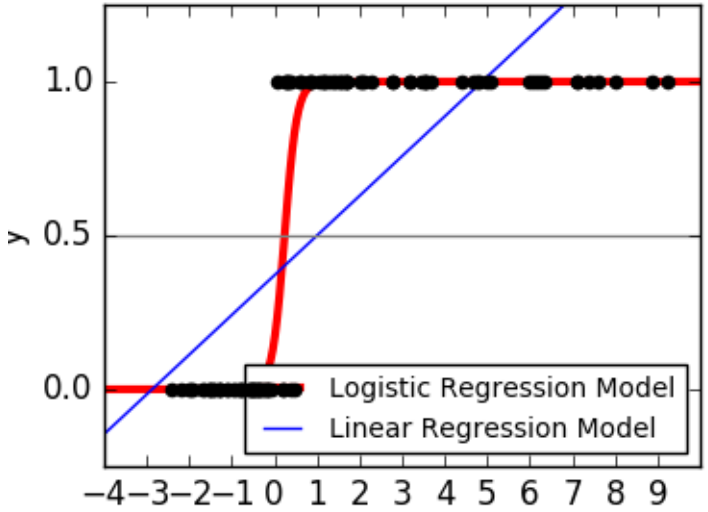
  Alternative: one-vs-all approach.

Multinomial vs One-vs-rest





Logistic Regression III



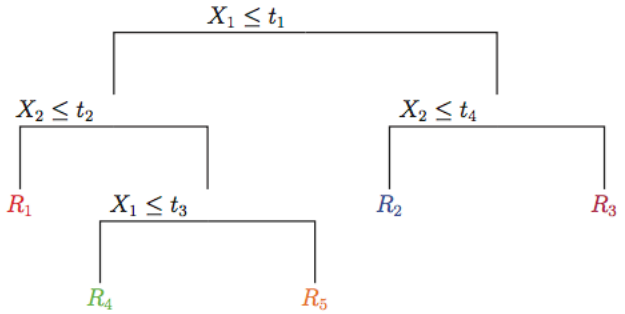
Linear Regression sklearn

```
from sklearn.linear_model
import Ridge

l2_reg = Ridge(alpha=0.5) # lambda
l2.fit(X, y)
y_pred = l2.predict(X_train)

l2._coef # returns w
l2._intercept # returns constant
```

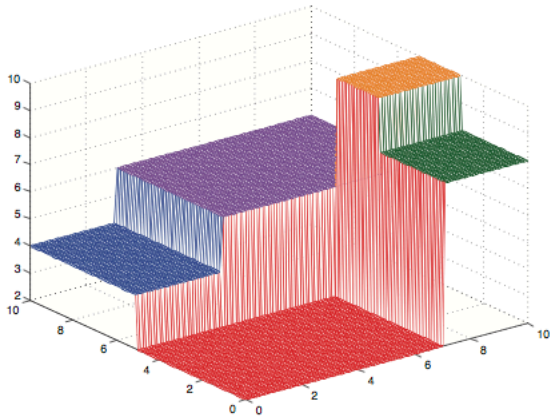
Logistic Regression sklearn

```
from sklearn.linear_model import
LogisticRegression

lr_clf = LogisticRegression(
penalty="l2", # or "l1"
C=1./lambda,  # capacity
fit_intercept=true, # add bias?
n_jobs=-1) # -1 = all cores

lr_clf.fit(X_train, y_train)
y_pred = lr_clf.predict_proba(X_test)
```

Decision Trees



(a)

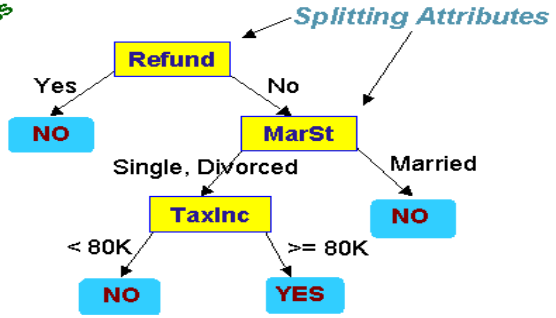


(b)

Basic Idea

- Use a tree in which inner nodes split the data into two branches and terminal leafs contain a result.
- Split is taken to be the one that carries “maximal” information gain.
- Measured in *entropy* or *gini*.

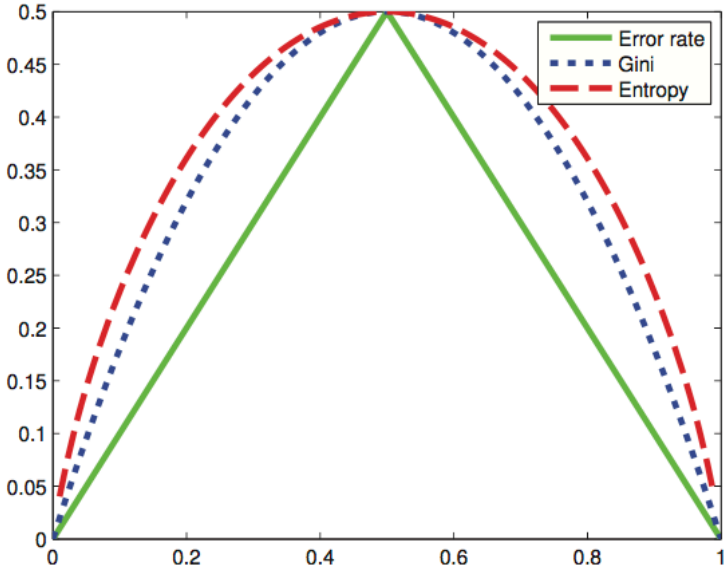
		categorical	categorical	continuous	class
Tid	Refund	Marital Status	Taxable Income	Cheat
1	Yes	Single	125K	No
2	No	Married	100K	No
3	No	Single	70K	No
4	Yes	Married	120K	No
5	No	Divorced	95K	Yes
6	No	Married	60K	No
7	Yes	Divorced	220K	No
8	No	Single	85K	Yes
9	No	Married	75K	No
10	No	Single	90K	Yes



The splitting attribute at a node is determined based on the Gini index.

Split Criterias

- For a proposed split, given K classes, a leaf would have a sample distribution by class $\pi_1, \ldots, \pi_K$.
- Entropy: $H(\pi_1, \ldots, \pi_K) = -\sum_{k=1}^{K} \pi_k \log(\pi_k)$. Minimizing this is maximizing the information gain.
- Gini: $\sum_{k=1}^{K} \pi_k (1 - \pi_k)$. This is the expected error rate.

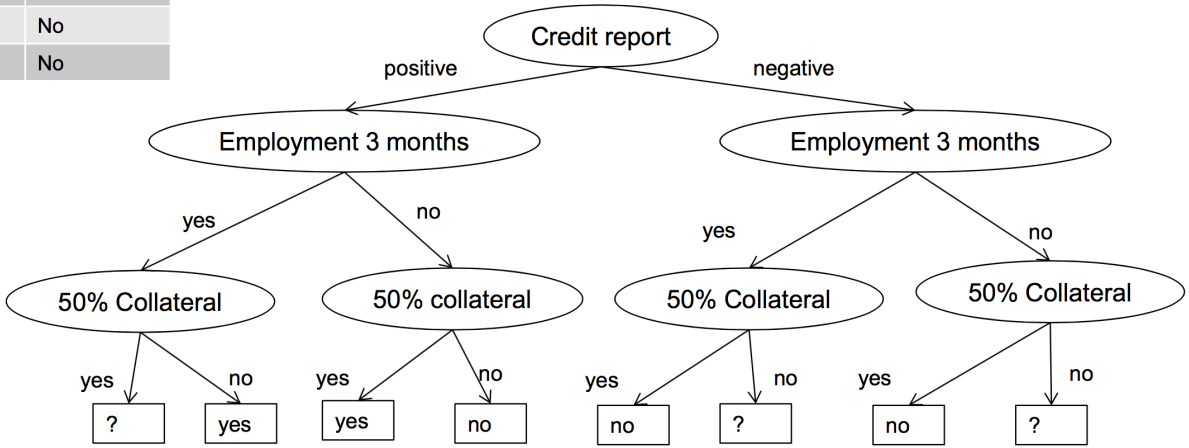

Error rate
Gini
Entropy
0.5
0.45
0.4
0.35
0.3
0.25
0.2
0.15
0.1
0.05
0
0
0.2
0.4
0.6
0.8
1

Example

Loan	Credit report	Employment last 3 months	Collateral > 50% loan	Payed back in full
1	Positive	Yes	No	Yes
2	Positive	No	Yes	Yes
3	Positive	No	No	No
4	Negative	No	Yes	No
5	Negative	Yes	No	No

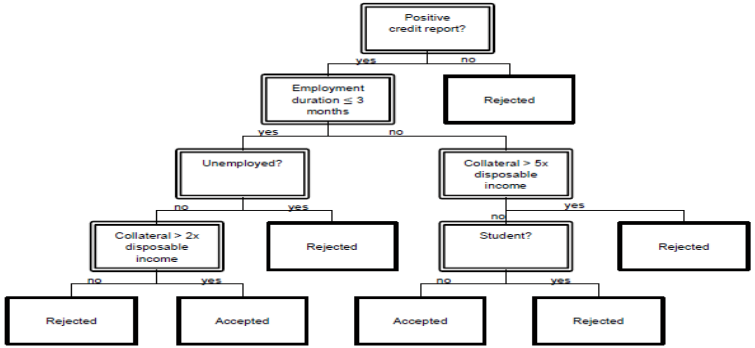
Example

Loan	Credit report	Employment last 3 months	Collateral > 50% loan	Payed back in full
1	Positive	Yes	No	Yes
2	Positive	No	Yes	Yes
3	Positive	No	No	No
4	Negative	No	Yes	No
5	Negative	Yes	No	No



Decision Trees - Algorithms

- Usually trained by CART (Breiman, 1984) or C4.5/ID3 (Quinlan, 1993/1986).
- Optimal Split problem is NP-complete.



Decision Trees II

- Every leaf has data points that **supports it.**
- sklearn.tree.DecisionTreeClassifier
  - criterion: "gini" / "entropy" (kind of information)
  - max_features: how many features to consider at split
    - can be integer or ratio (float), other: "auto", "sqrt", "log2", None = all
    - lower values to speed up, possibly avoids overfitting (unlikely)
  - max_depth: limits depth of tree (deep = complex = overfitting), None → unlimited expansion.
  - min_samples_split / leaf: minimum support for split / at leaf.
    - low number could lead to overfitting.
  - min_impurity_split: min. impurity to split.
  - class_weight: weight classes for splitting, "balanced": balances imbalanced classes out (usually a good thing).

Decision Tree III

- Decision trees are famous for overfitting.
- Long lost trick: Set min_samples_leaf "high" and train models for each leaf, in other words: the decision tree pre-partitions the data set roughly, then more computation intensive / better models do the rest.
- High explainability.
- Got popular because of ensembling / boosting.

Pruning

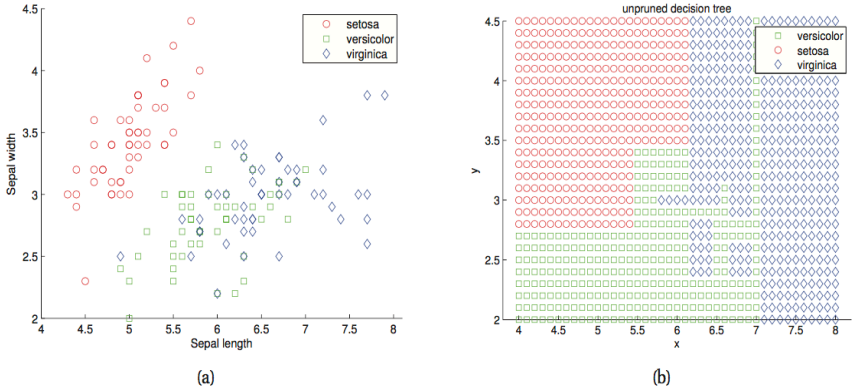


Figure 16.4 (a) Iris data. We only show the first two features, sepal length and sepal width, and ignore petal length and petal width. (b) Decision boundaries induced by the decision tree in Figure 16.5(a).

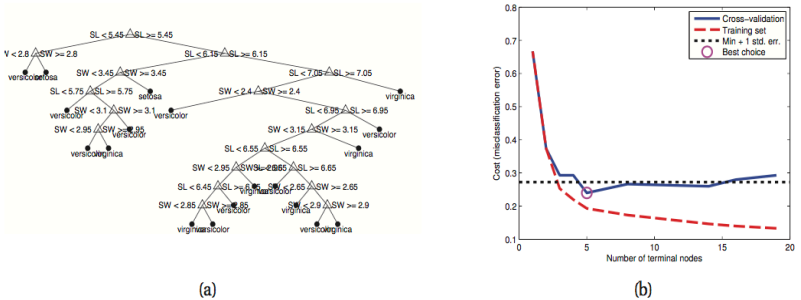


Figure 16.5 (a) Unpruned decision tree for Iris data. (b) Plot of misclassification error rate vs depth of tree. Figure generated by `dtreeDemoIris`.

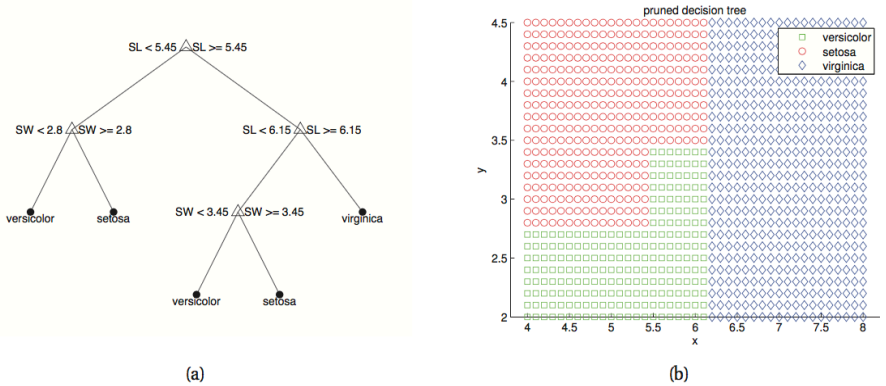


Figure 16.6 Pruned decision tree for Iris data. Figure generated by `dtreeDemoIris`.

CART models

Pros

- Easy to interpret
- Handle mixed discrete & continuous input, robust to monotone transformations
- Automatic variable selection
- Robust to outliers
- Scales well to large data

Cons

- Overfitting deluxe (high variance estimators)
- Not useful for online learning (unstable to data changes)
- Often weaker compared to other models

Random Forests

Ensembles, Bagging, Independence…

Random Forests

- **Ensemble** of decision trees.
- **Bagging** used to fit the trees.
- **Random Subspace** training on a feature subset.
- **Independence** of the training process of each tree.

Ensemble

- Having K classifiers $C_1, \ldots, C_K$ the ensemble classifier C predicts the class that is the majority vote of the C_i,

$$\mathrm{C(x)} = argmax_c(\#k \,|C_k(x) = c)$$

- Accordingly for regressors the average is taken,

$$\mathrm{C(x)} = \frac{1}{K}\sum_{k=1}^{K} C_k(\,x)$$

Bagging (Bootstrap Aggregation)

- The idea of bagging is to fit a model on only a subset B of the training data D, the **bag**.
- This is combined with ensembling in RFs.
- Other advantage: Can compute error metric on D – B, the *out-of-bag error*.
- Disadvantage: Highly correlated base models.

Random Subset / Independence

- For further decorrelation only a random subset of the features can be used, this also speeds up the training process.
- In an ensemble the base models can – and should – be trained independently. => Parallelization possible.

Variants

- ExtraTrees (in *sklearn.tree* and *sklearn.ensemble*): Only a subset of features is used at *each* node split.

Boosting

Boosting

- Boosting: Using a weak base classifier by building a strong one by ensembling successively.
- Example: AdaBoost with Decision Stumps (DTs of depth 1).
- On step m we have model F_m and ideally want to have for each (x,y):

$$F_{m+1}(x) = F_m(x) + h(x) = y$$

Thus we optimize $h(x) = y - F_m(x)$. This is *Gradient Boosting*.

Evaluation

ERM, Experimental Design, Bias/Variance, CV

Empirical Risk Minimization

- Two assumptions:
  - Data follows a distribution $p(x, y)$ on $X \times Y$.
  - Learner sees data set $D = \{(x_1, y_1), \ldots, (x_N, y_N)\}$ of **independent** and **identically sampled** (from p) data (*IID assumption*).

- **The IID assumption is almost always broken in practice.**

Empirical Loss

- Given a loss function l the *empirical loss/risk is,*

$$R_D[f] = \sum_{n=1}^{N} l(f(x_n), y_n, x_n).$$

- This is a statistical estimator if you vary the selection of D.

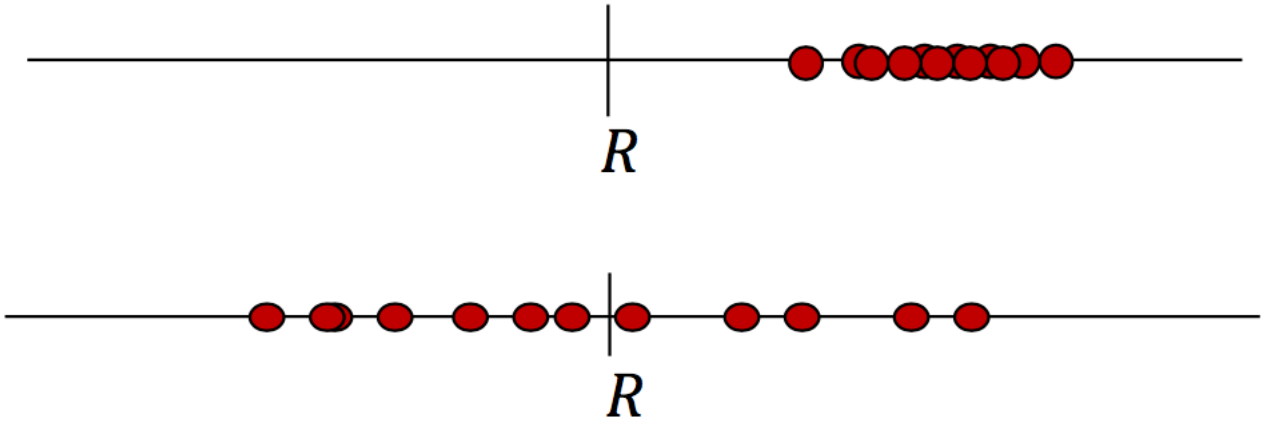
Empirical Risk Minimization II

- $R_D[f]$ is supposed to estimate the *generalization error/risk,*

$$R[f] = \int l(f(x), y, x) dp(x, y) \,.$$

- The larger $|D|$ the more accurate.
- $R_D[f]$ has a bias (distance to R$[f]$) and a variance.

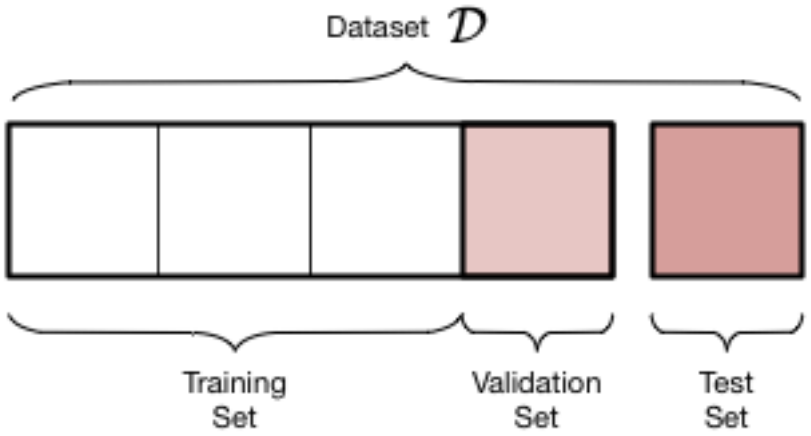
Bias / Variance



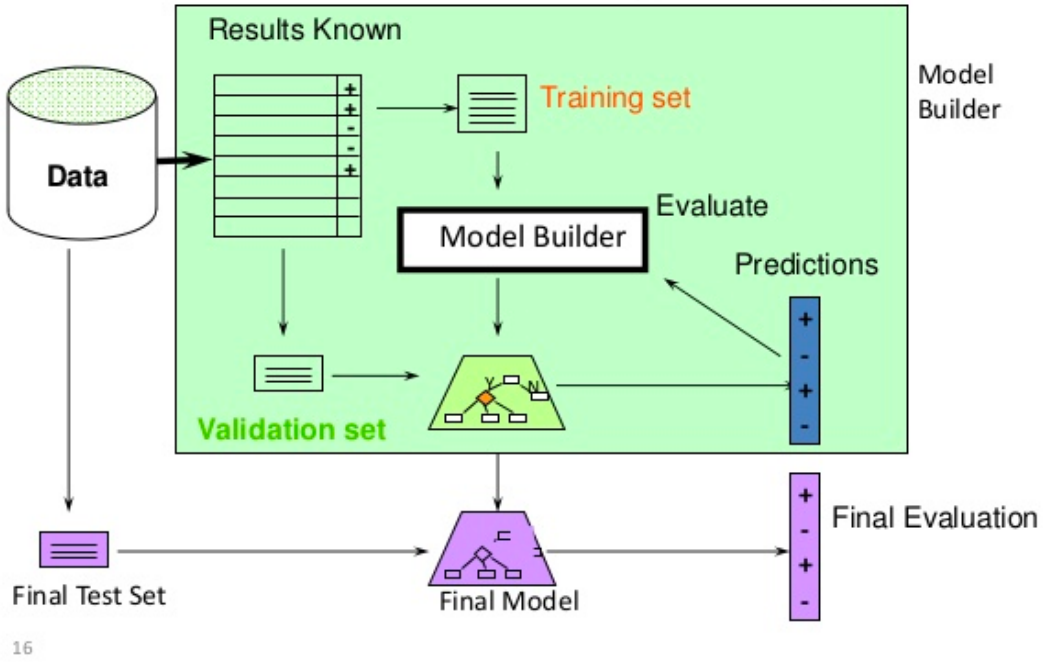
- **Model fitting:** finding the best model parameter:

$$\theta = argmin_{\theta} R_D[f_\theta]$$

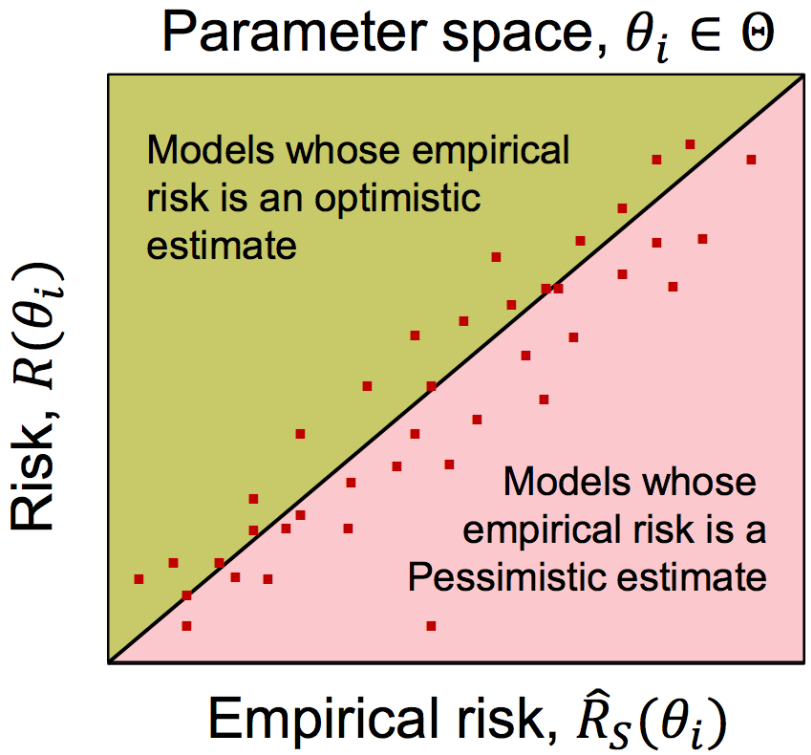
- **Model selection:** find the best fitting model family / hyperparams.
- **Model evaluation:** estimate the generalization risk.



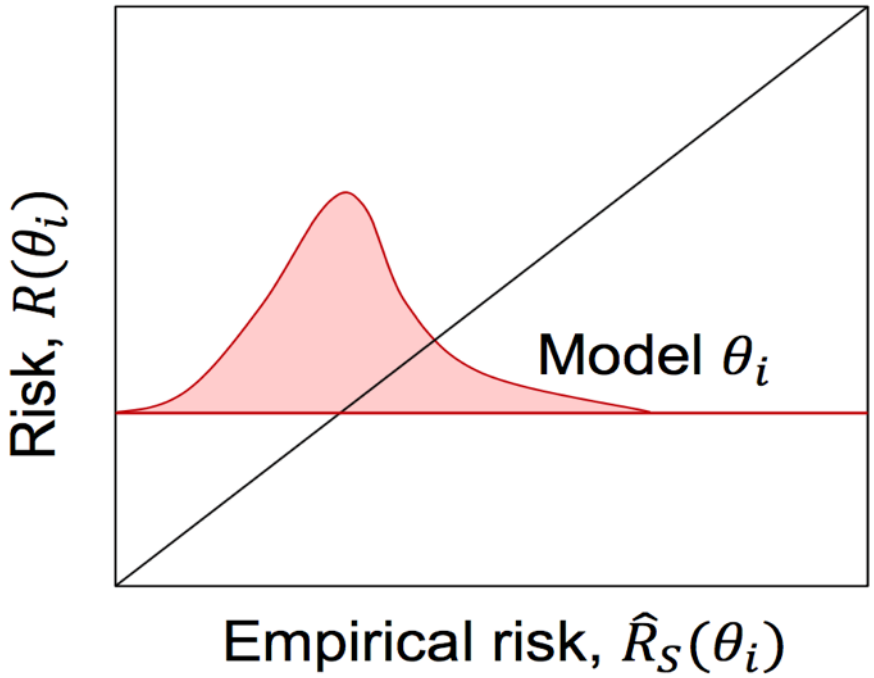
Classification:
Train, Validation, Test split



Source: http://bit.ly/2vDfoLp

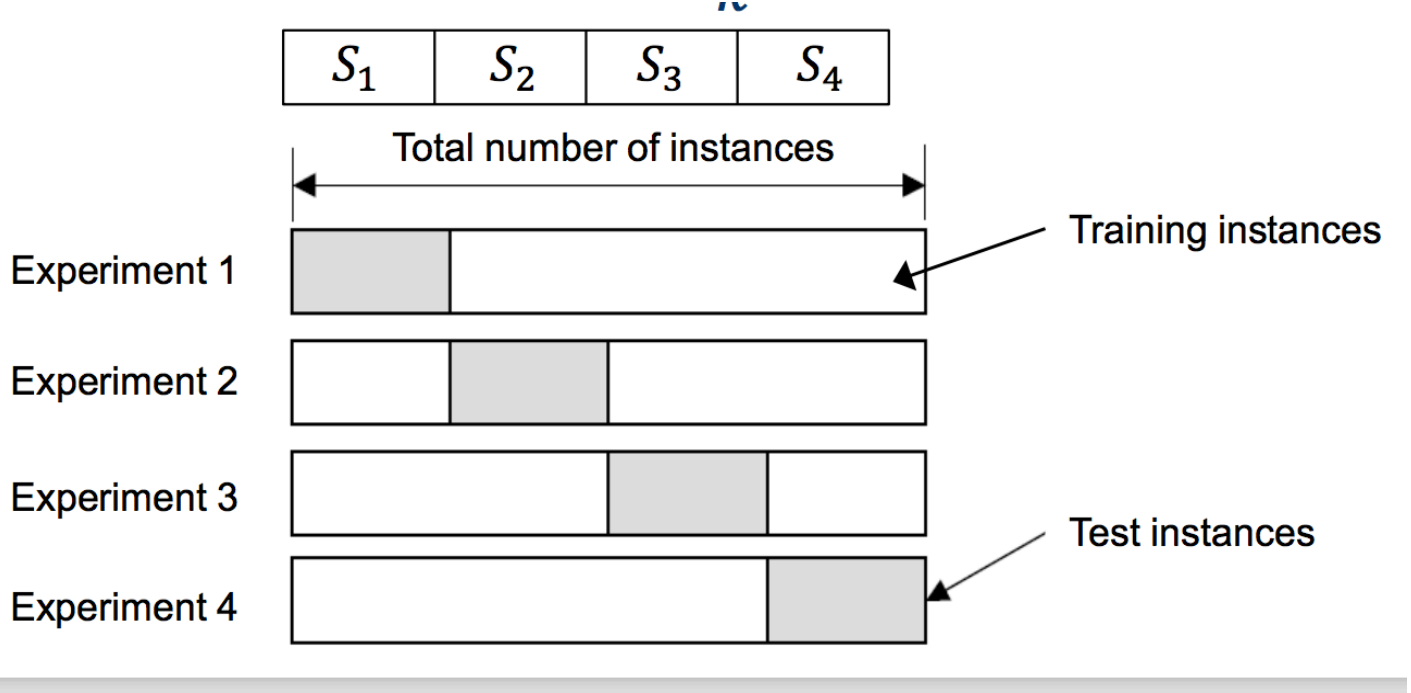


Source: University of Potsdam

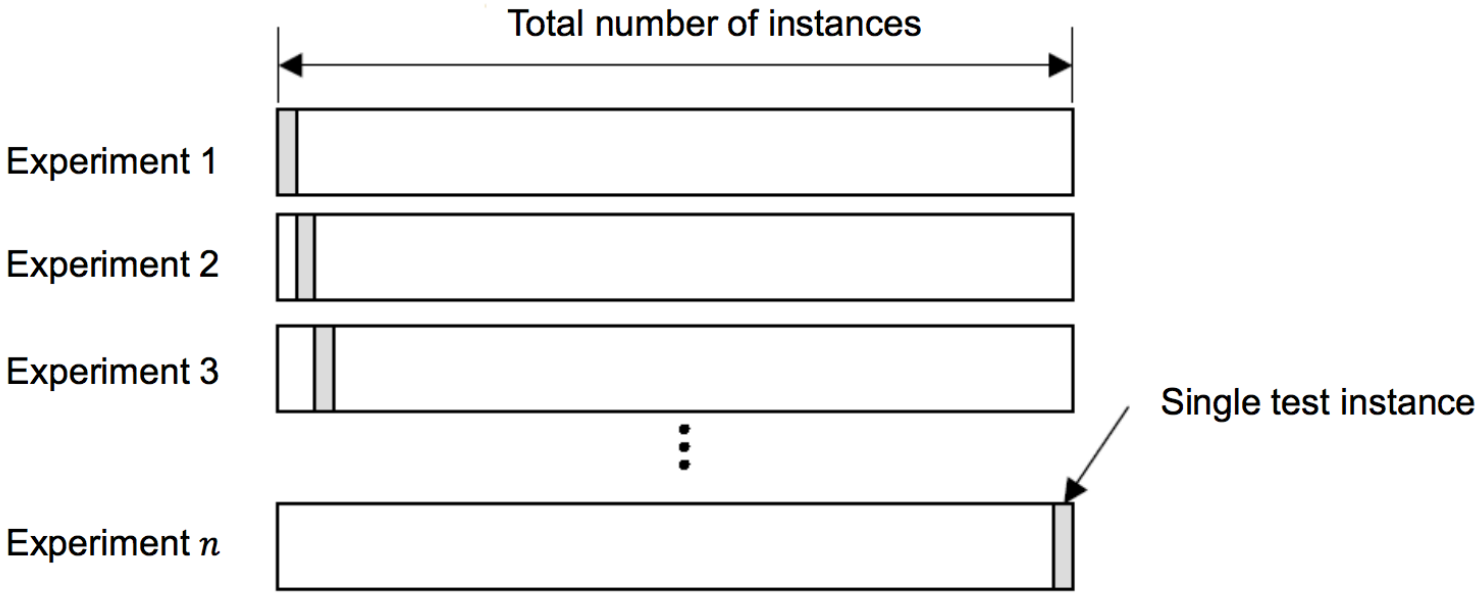


Source: University of Potsdam

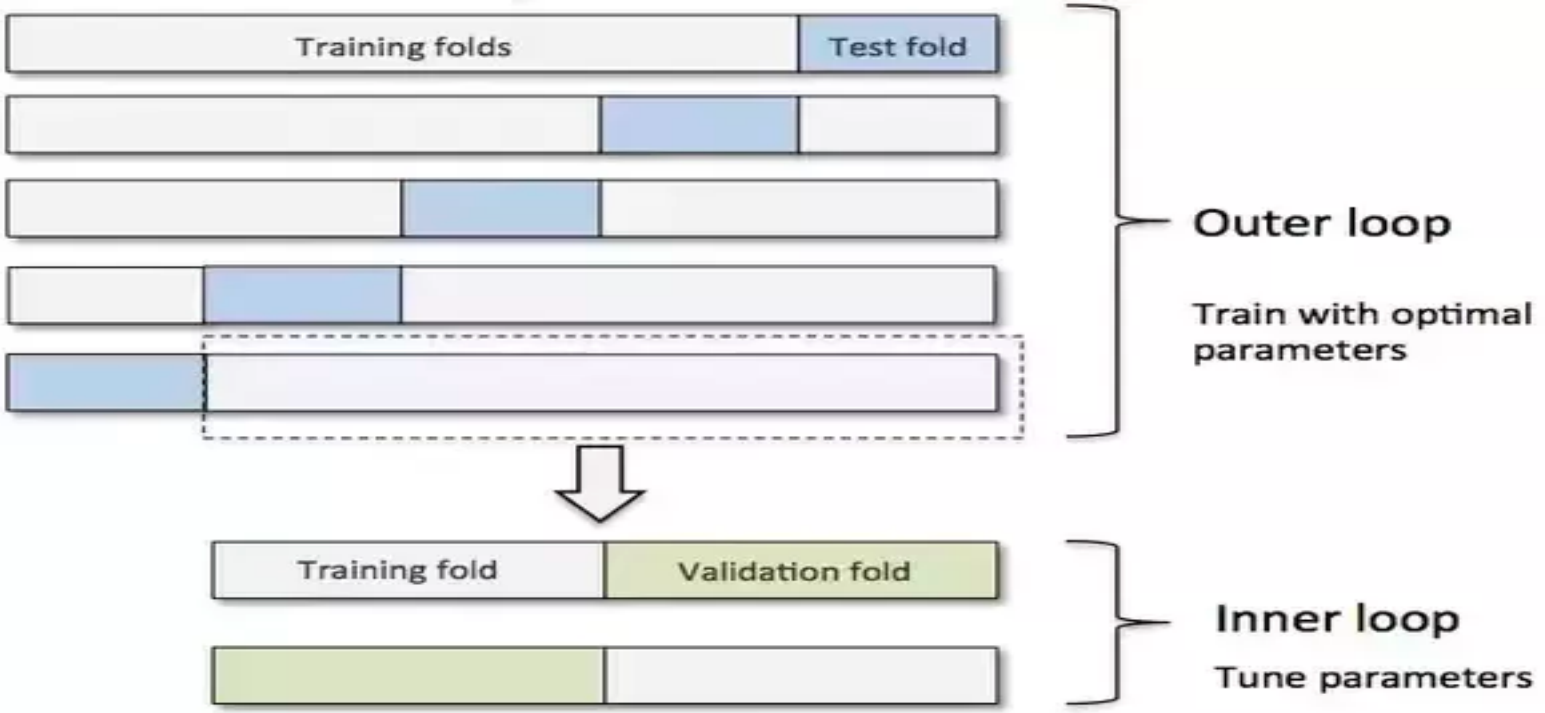
CV


S_1
S_2
S_3
S_4
Total number of instances
Training instances
Experiment 1
Experiment 2
Experiment 3
Test instances
Experiment 4

LOOCV


Total number of instances
Experiment 1
Experiment 2
Experiment 3
Experiment n
Single test instance

Nested CV



From Quora: http://bit.ly/2wvz2aZ

Messing up your experiments

- Data split strategy is part of experiment.
- Mainly care for:
  - Class distribution
  - Problem domain relevant issues such as time

> *”Validation and Test sets should model nature and nature is not accommodating."* --- **Data Scientist’s Proverbs**