CS6375: Machine Learning

Gautam Kunapuli

Mid-Term Review

THE UNIVERSITY OF TEXAS AT DALLAS
Erik Jonsson School of Engineering and Computer Science

How To Study?

For classifiers and regressors:

- Linear/ridge regression
- SVMs
- Decision trees
- k-Nearest Neighbors
- Naïve Bayes
- Logistic Regression

- What is the **hypothesis space** of the classifier/regressor?
  - *Can you draw the* ***decision boundary*** *of these classifiers given a simple example?*
  - *Can you learn/construct a classifier given a simple example?*
- What is the **general learning/optimization** procedure?
- What **hyper-parameters** does the model depend on? *Hyper-parameters trade-off between or control the influence of*
  - *the loss function*
  - *the regularization function (or mechanisms to control model complexity)*
- How do we **select** these models?
- How does the **bias-variance** behavior change with these hyper-parameters?
  - *When will these models* ***overfit****?*
  - *How can we avoid* ***overfitting****?*

For model selection:

- Training set
- Validation set
- Test set
- Cross validation

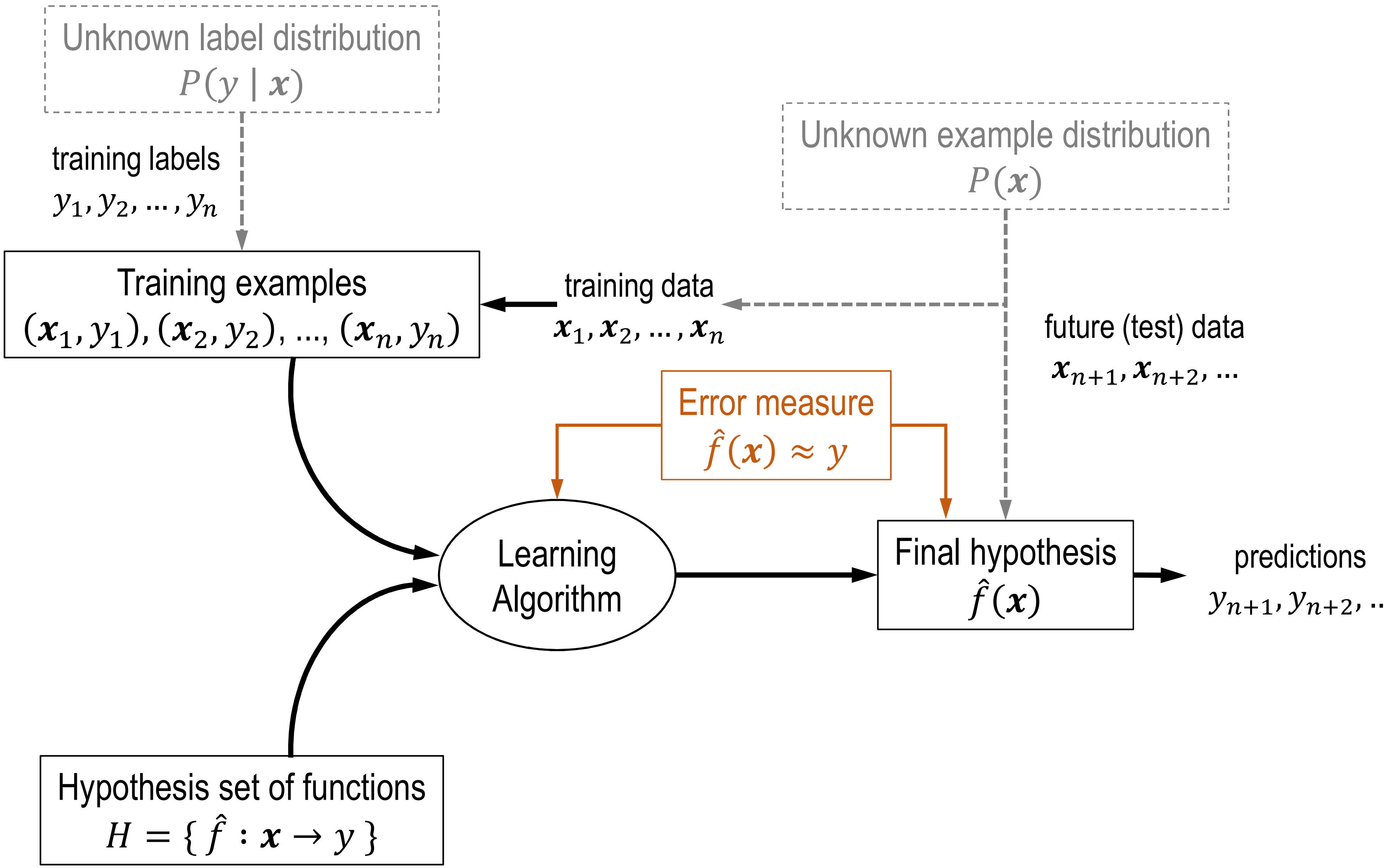
- What are they, and what are they for?
  - Can you perform cross validation given a simple example?

For evaluation metrics:

- Accuracy
- ROC curves / AUC-ROC
- Precision
- Recall

- What are they, and how are they computed?
- When should you use them?
  - Can you compute them given a simple example?

Supervised Learning: General Setup



Discriminative vs. Generative Learning

- **Generative**: Create something that generates ex's
- Can create complete input feature vectors
  - Describes probability distributions for all features
  - Stochastically create a plausible feature vector
  - Example: Naïve Bayes
- Make a model that *generates* positives, negatives
- Classify a test example based on which is more likely to generate it

- **Discriminative**: What differentiates class A from class B?
- Don't try to model all the features, instead focus on the task of categorizing
  - Captures differences between categories
  - May not use all features in models
  - Examples: decision trees, SVMs, neural nets, logistic regression
- Typically more efficient and simpler

Assume some **functional form for P(X|Y), P(Y)**

–Estimate parameters of P(X|Y), P(Y) directly from training data

–Use Bayes rule to calculate P(Y|X= x)

–This is a **'*generative*' model**

•**Indirect** computation of P(Y|X) through Bayes rule

•As a result, **can also generate a sample of the data**, $P(X) = \sum_y P(y) P(X|y)$

Discriminative classifiers, e.g., Logistic Regression:

–Assume some **functional form for P(Y|X)**

–Estimate parameters of P(Y|X) directly from training data

–This is the **'*discriminative*' model**

•Directly learn P(Y|X)

•But **cannot obtain a sample of the data**, because P(X) is not available

Linear Regression

Problem Setup: Given data $(\boldsymbol{x}_i)$ and real-valued labels (y_i), find the best model that **fits current data <u>and</u> predicts future data**

Problem: Given n training examples $(\boldsymbol{x}_i, y_i)$, $i = 1, \dots, n$, find the best model $\boldsymbol{w}$ by solving

$$\underset{\boldsymbol{w}}{\text{minimize}} \ \frac{1}{n}(\boldsymbol{y}^T\boldsymbol{y} - 2\boldsymbol{y}^T X\boldsymbol{w} + \boldsymbol{w}^T X^T X\boldsymbol{w})$$

The solution to this problem is the **ordinary least squares estimator**

$$w = (X^T X)^{-1} X^T \boldsymbol{y}$$

*solution depends on the inverse of the **covariance matrix** $C = X^T X$, which can be **ill-conditioned***

***unique closed-form solution**, provided that number of data points (n) exceeds data dimension (d)*

$(X^T X)^{-1} X^T = X^+$ is called the **pseudo-inverse**

Ridge regression adds L_2 regularization

$$\underset{\boldsymbol{w}}{\text{minimize}} \ \frac{1}{n}(\boldsymbol{y} - X\boldsymbol{w})^T(\boldsymbol{y} - X\boldsymbol{w}) + \lambda\boldsymbol{w}^T\boldsymbol{w}$$

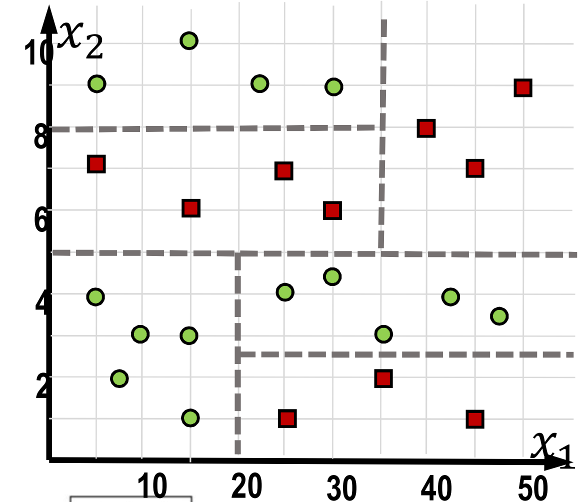
$\boldsymbol{w}^T\boldsymbol{w}$ is a **regularization term** that is used to overcome ill-conditioning, $\lambda > 0$ is the **regularization parameter**, which is **tunable**

The solution to this problem is the **regularized least squares estimator**

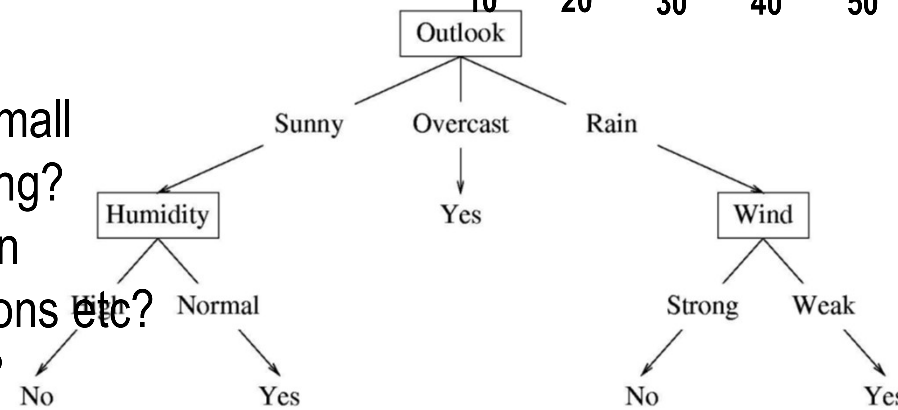
$$w = (X^T X + \lambda I_d)^{-1} X\boldsymbol{y}$$

for $\lambda > 0$, *inverse is can always be computed, algorithm more **robust***

Decision Trees



- Recursively split the features based on some statistical measure – **information gain**, **Mutual Information**
- Splits are binary in general – can you make multi-way split? What will information gain favor? Binary or multi-way?
- What is a decision stump?
- How does the **decision boundary** look like?
- **Pruning** will allow decision trees to have a reduced depth
- When will decision trees **overfit**? What will you prefer – small depths or a very large depth? How can you avoid overfitting?
- Expressiveness – Can they represent an arbitrary Boolean function? How about a disjunction of conjunctions, negations etc?
- When do they have bias? When do they exhibit variance?



Mutual information/information gain is used to select next attribute

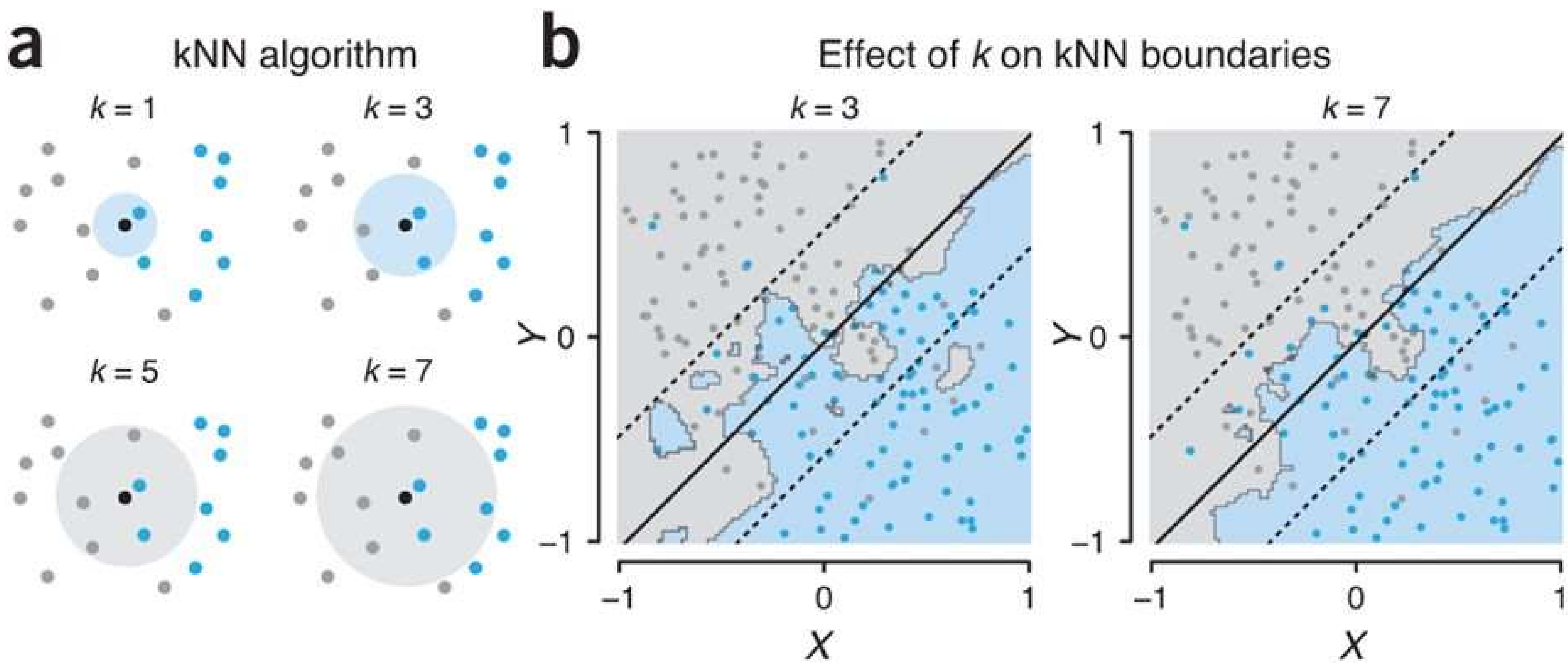
$$H(Y) = -\sum_{y} P(Y = y)\mathbf{log_2}\, \mathrm{P}(Y = y)$$

$$H(Y|X) = -\sum_{x} P(X = x) \sum_{y} P(Y = y \mid X = x)\ \log_2 (Y = y \mid X = x)$$

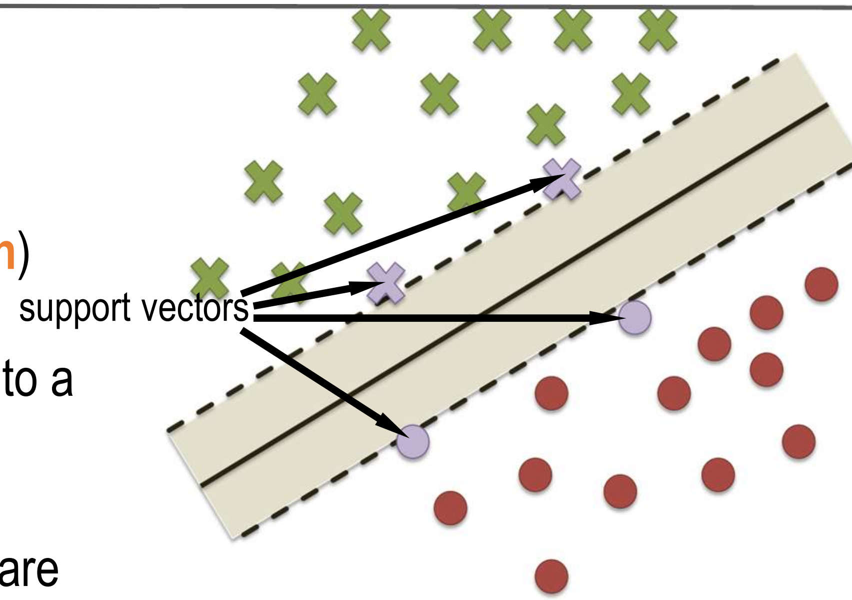
$$I(X, Y) = H(Y) - H(Y|X)$$

K-Nearest Neighbor

- Lazy algorithm: does not build a classifier from all data. Instead builds as every example comes in
- Decision boundaries are drawn between examples of **opposite** classes. What are distance measures?
- Complex decision boundaries – Voronoi diagram. Small k leads to more complex decision boundaries
- **Choosing k**: increasing k reduces variance, increases bias
- How does noise affect NN? How can their effect be reduced?
- For high-dimensional space, problem that the nearest neighbor may not be very close at all!



Support Vector Machines



- In the simplest case, SVMs search for the hyperplane that maximizes the separation between the two classes (**margin**)
- The minimization problem is a quadratic optimization with linear inequality constraints. The key idea is to convert this to a dual problem with smaller number of constraints
- Hypothesis is a linear combination of training examples
  - only some training examples have non-zero weights, are called **support vectors**
- Can replace the inner product in the dual formulation with a **kernel**; called the **kernel trick**
  - Understand kernels, how to get kernels from explicit transformations
- What can be represented? When can SVMs overfit? How can you prevent that? (Hint: Think linear SVMs vs non-linear SVMs.)
- When do they have bias? When do they exhibit variance?

soft-margin support vector machine

$$\min \quad \frac{1}{2}\|\mathbf{w}\|^2 + C\sum_{i=1}^{n}\xi_i$$

$$\text{s.t.} \quad y_i(\mathbf{w}'\mathbf{x}_i - b) \geq 1 - \xi_i \quad \forall i = 1\ldots n$$

$$\xi_i \geq 0$$

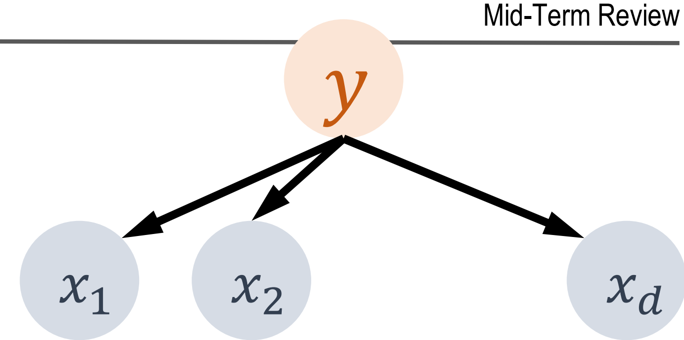
soft-margin svm dual

$$\max \quad -\frac{1}{2}\sum_{i=1}^{n}\sum_{j=1}^{n}\alpha_i\alpha_j y_i y_j \mathbf{x}_i'\mathbf{x}_j + \sum_{i=1}^{n}\alpha_i$$

$$\text{s.t.} \quad \sum_{i=1}^{n}\alpha_i y_i = 0$$

$$0 \leq \alpha_i \leq C, \quad \forall i = 1\ldots n$$

Naïve Bayes



- **Generative model** – Learns the joint distribution of the labels and features P(y,**x**)
- **Naïve Bayes assumption**: **features are conditionally independent given** y that is,

$$P(x_1, x_2, \dots, x_d | y) = \prod_{j=1}^{d} P(x_j | y)$$

- When is that a good one? When is it a bad assumption?
- Learning is very simple with **maximum likelihood estimation** (MLE).

$$y = \underset{y}{\operatorname{argmax}}\, P(y|\boldsymbol{x}) = \underset{y}{\operatorname{argmax}}\, P(y) \cdot \prod_{j=1}^{d} P(x_j | y)$$

remember these expressions!

What is the issue with a simple MLE? How can you fix this?

- Can handle a variety of data types. Why?
- First thing to try in most problems – simple yet very efficient
- When do they have bias? When do they exhibit variance?

Logistic Regression

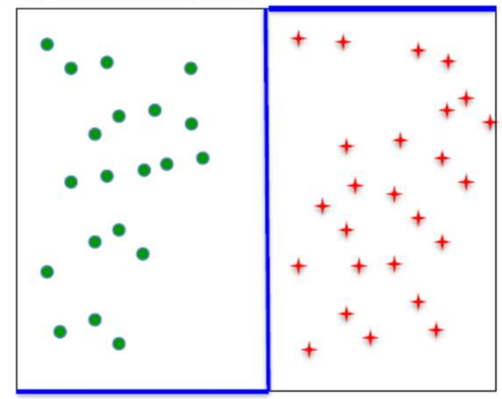
Logistic Loss Function: (probabilities from hard classification)
Learn or $p(y|\boldsymbol{x})$ directly from the data

- Assume a functional form, (e.g., a linear classifier $f(\boldsymbol{x}) = \boldsymbol{w}^T\boldsymbol{x} + b$) such that
- $p(y = -1|\,\boldsymbol{x}) = \frac{1}{1+\exp\,(\boldsymbol{w}^T\boldsymbol{x}+b)}$ on one side and
- $p(y = 1|\,\boldsymbol{x}) = \frac{\exp(\boldsymbol{w}^T\boldsymbol{x}+b)}{1+\exp\,(\boldsymbol{w}^T\boldsymbol{x}+b)}$ on the other side

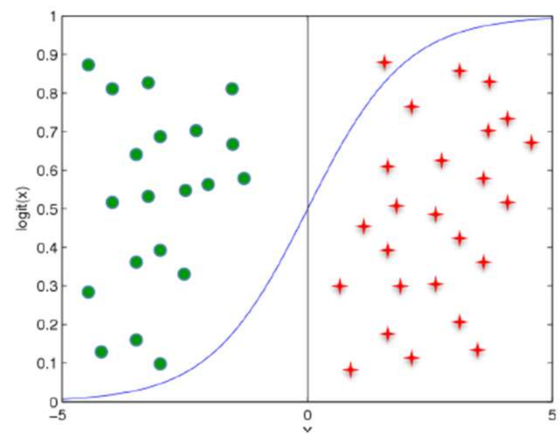
  that is $p(y = -1|\,\boldsymbol{x}) = 1 - p(y = 1|\,\boldsymbol{x})$

- Differentiable, easy to learn, handles noisy labels naturally

$p(Y = 1|x) = 0$



$p(Y = 1|x) = 1$



Logistic regression **implements a linear classifier** as it maximizes the **log-odds of a training example** belonging to class $y = 1$ are:

$$\log \frac{p(y = 1|\,\boldsymbol{x})}{p(y = -1|\,\boldsymbol{x})} = \boldsymbol{w}^T\boldsymbol{x} + b$$

Consider a **prior distribution** on the weights to prevent overfitting

- assume weights from a normal distribution with zero mean, identity covariance:

$$\mathrm{P}(\boldsymbol{w}) = \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left(-\frac{\|\boldsymbol{w}\|^2}{2\sigma^2}\right)$$

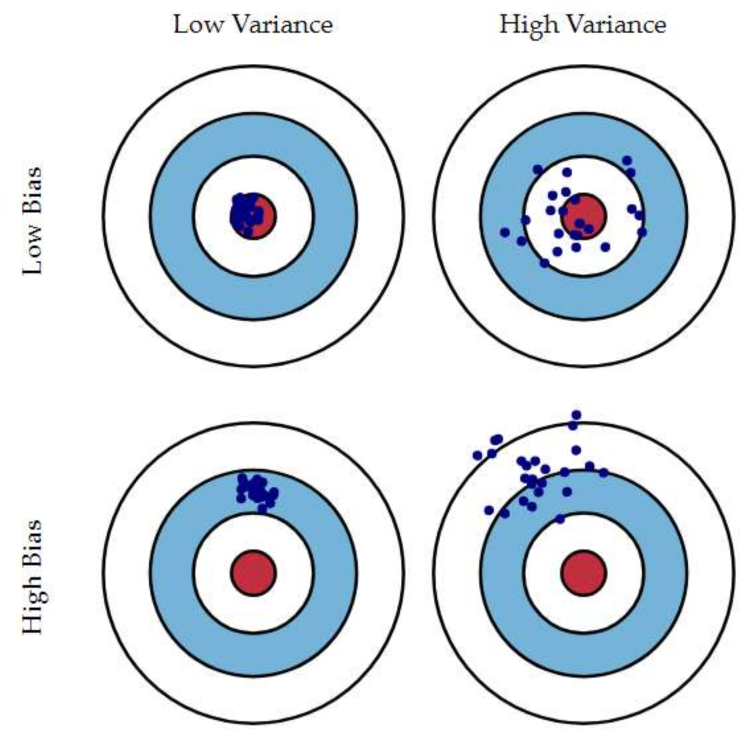
- **maximizing** $\mathrm{P}(\boldsymbol{w})$ pushes weights to zero, which **minimizes the complexity** of classifier; also helps avoid large weights and overfitting

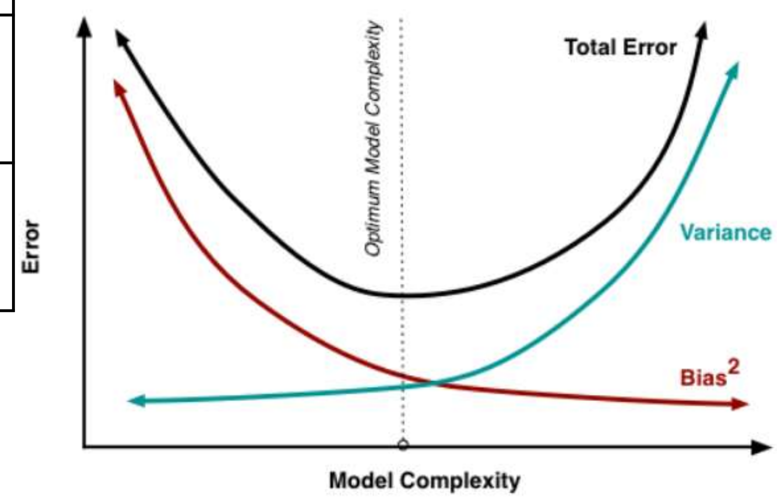
taking the logarithm gives us $\log \mathrm{P}(\boldsymbol{w}) = -\|\boldsymbol{w}\|^2 + \text{const}$

$$\max_{\boldsymbol{w},b} \sum_{i=1}^{n} y_i(\boldsymbol{w}^T\boldsymbol{x}_i + b) - \log(1 + \exp(\boldsymbol{w}^T\boldsymbol{x}_i + b)) \; - \frac{\lambda}{2}\|\boldsymbol{w}\|^2$$

The Bias-Variance Tradeoff

Logistic regression, linear regression, SVM, neural networks with an $\lambda \, \|w\|_2^2$ (L2) penalty in the objective	Higher λ means... more/less **variance** more/less **bias**
Logistic regression, linear regression, SVM, neural networks with an $\lambda \, \|w\|_1$ (L2) penalty in the objective	Higher λ means... more/less **variance** more/less **bias**
Decision tree: n, an upper limit on the number of nodes in the tree	Higher n means... more/less **variance** more/less **bias**
Feature selection with mutual information scoring: include a feature in the model only if its MI(feat, class) is higher than a threshold t	Higher t means... more/less **variance** more/less **bias**
Increasing k in k-nearest neighbor models	Higher k means... more/less **variance** more/less **bias**
Removing all the non-support vectors in an SVM	This means... more/less **variance** more/less **bias**
Dimension reduction as preprocessing: instead of using all features, reduce the training data down to k dimensions	Higher k means... more/less **variance** more/less **bias**





Cross Validation

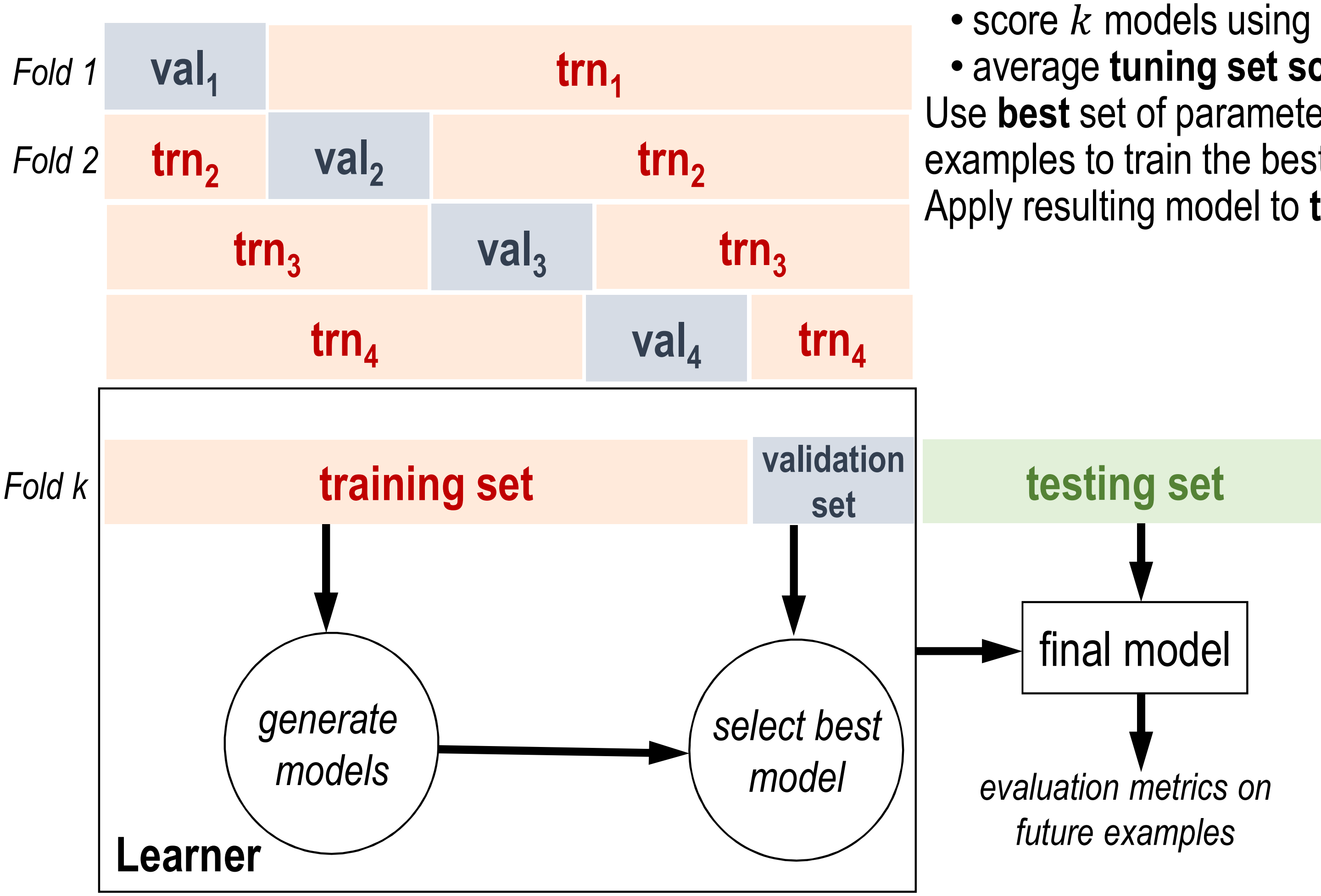
Using a **single** tuning set can be **unreliable** predictor, plus some data "**wasted**"; cross validation can help with **model selection**:

For each possible set of parameters, θ_p

- Divide <u>training</u> data into k **folds**
- **train** k models using trn_k with θ_p
- score k models using val_k
- average **tuning set score** over the k models

Use **best** set of parameters θ_* and **<u>all</u> (train + tune)** examples to train the best model

Apply resulting model to **test set**



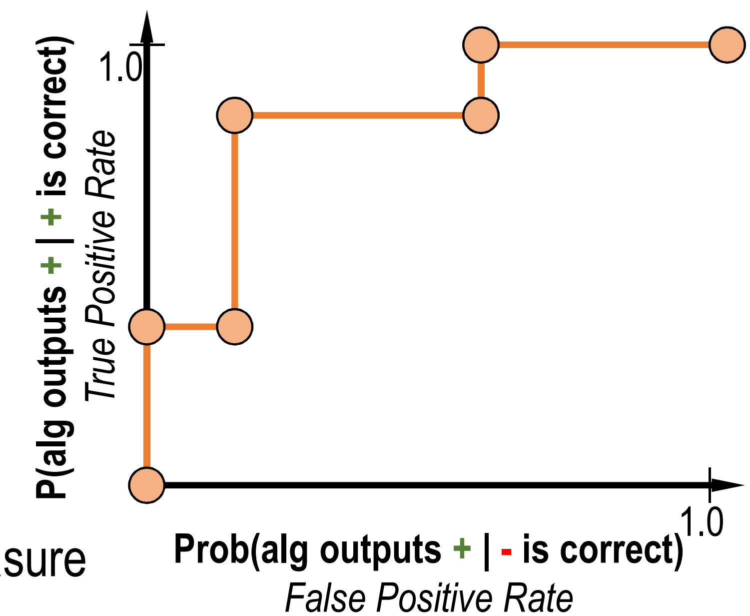
Receiver-Operator Characteristic Curves

An **ROC curve** (receiver operating characteristic curve) is a graph showing the performance of a classification model at **all classification thresholds**.

- judging algorithms on accuracy alone may not be good enough when getting a positive example wrong **costs more** than getting a negative example wrong (**or vice versa**)
- **lowering** the classification **threshold** classifies **more** items as **positive**, thus increasing both False Positives and True Positives

Procedure to construct an ROC curve:

- **sort** predictions on test set
- locate a **threshold** between examples with opposite categories
- **compute** TPR & FPR for each threshold
- **connect** the dots



Area under the ROC Curve (AUC) provides an aggregate measure of performance across all possible classification thresholds

- One way of interpreting AUC is as the **probability** that the model ranks a random **positive example** more **highly** than a random **negative example**
- can compare performance of different algorithms using AUC
- can use AUC/ROC to select a good threshold for classification in order to weight false positives and false negatives differently

Evaluation: Precision and Recall

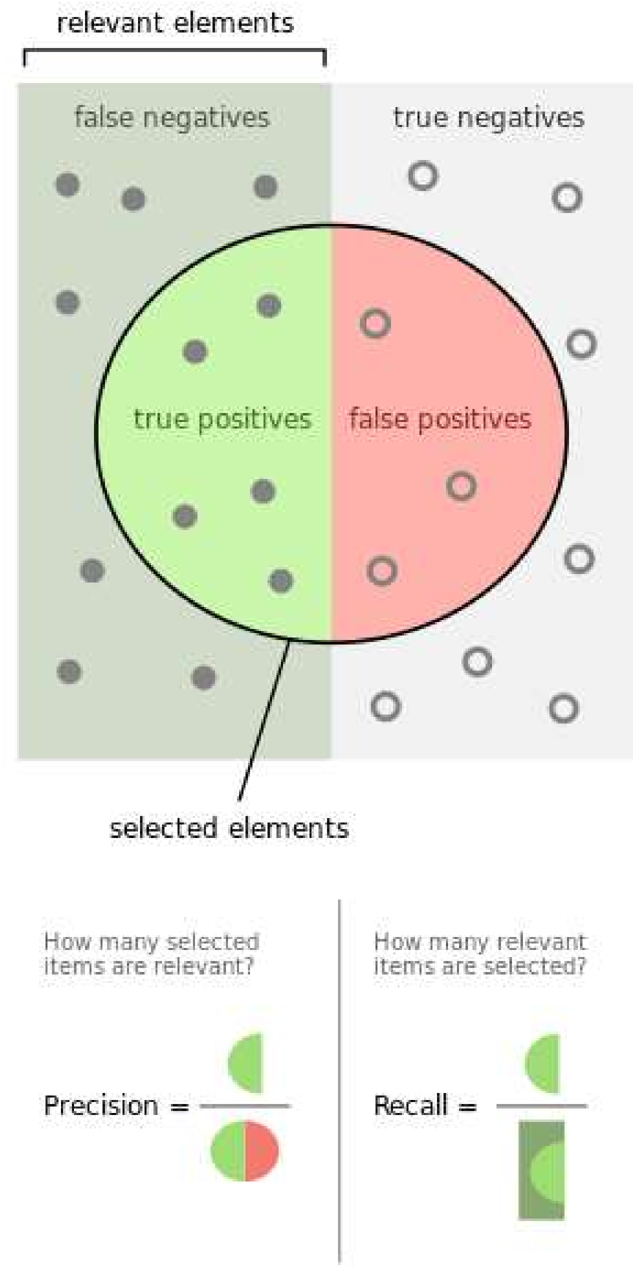
precision $= \frac{\#\, of\ relevant\ items\ retrieved}{total\ \#\, of\ items\ retrieved} = \frac{TP}{TP+}$

interpretation: Prob(is positive | called positive)

recall $= \frac{\#\, of\ relevant\ items\ retrieved}{total\ \#\, of\ items\ that\ exist} = \frac{TP}{TP+FN}$

interpretation: Prob(called positive | is positive)

Notice that the count of true negatives (TN) is not used in either formula; therefore you get **no credit for filtering out irrelevant items**



Case Study 1: For applications such as **medical diagnosis,** require **high recall to reduce false negatives**

Case Study 2: For applications such as **spam-filtering** and **recommendations systems**, require **high precision to reduce false positives**