Applied Machine Learning

Naive Bayes

Siamak Ravanbakhsh

COMP 551 (winter 2020)

Learning objectives

generative vs. discriminative classifier

Naive Bayes classifier

- assumption
- different design choices

Discreminative vs generative classification

discriminative — so far we modeled the ***conditional*** *distribution*: $p(y \mid x)$

generative — learn the ***joint*** *distribution* $p(y, x) = p(y)p(x \mid y)$

prior class probability: frequency of observing this label

likelihood of input features given the class label
(input features for each label come from a different distribution)

how to classify new input x?

Bayes rule

$$p(y = c \mid x) = \frac{p(c)p(x|c)}{p(x)}$$

posterior probability
of a given class

marginal probability of the input (evidence)
$\sum_{c'=1}^{C} p(x, c')$

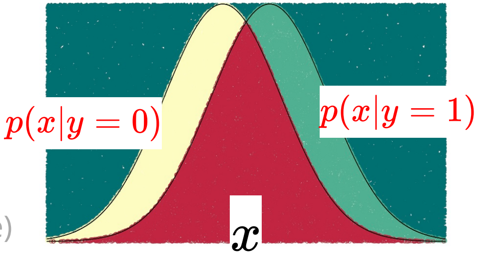


image: https://rpsychologist.com

Example: Bayes rule for classification

$y \in \{\text{yes}, \text{no}\}$ patient having cancer?

$x \in \{-, +\}$ test results, a single binary feature

prior: 1% of population has cancer $p(\text{yes}) = .01$

likelihood: $p(+|\text{yes}) = .9$ TP rate of the test (90%)

$$p(c \mid x) = \frac{p(c)p(x|c)}{p(x)}$$

posterior: $p(\text{yes}|+) = .08$

FP rate of the test (5%)

evidence: $p(+) = p(\text{yes})p(+|\text{yes}) + p(\text{no})p(+|\text{no}) = .01 \times .9 + .99 \times .05 = .189$

in a generative classifier likelihood & prior class probabilities are **learned from data**

Generative classification

prior class probability: frequency of observing this label

likelihood of input features given the class label
(input features for each label come from a different distribution)

$$p(y = c \mid x) = \frac{p(c)p(x|c)}{p(x)}$$

posterior probability
of a given class

marginal probability of the input (evidence)
$\sum_{c'=1}^{C} p(x, c')$

Some generative classifiers:

- Gaussian Discriminant Analysis: the likelihood is multivariate Gaussian
- Naive Bayes: decomposed likelihood

image: https://rpsychologist.com

Naive Bayes: model

number of input features

assumption about the likelihood $p(x|y) = \prod_{d=1}^{D} p(x_d|y)$

when is this assumption correct?

when features are **conditionally independent** given the label $x_i \perp\!\!\!\perp x_j \mid y$

knowing the label, the value of one input feature gives us no information about the other input features

chain rule of probability (true for any distribution)

$$p(x|y) = p(x_1|y)p(x_2|y, x_1)p(x_3|y, x_1, x_2) \dots p(x_D|y, x_1, \dots, x_{D-1})$$

conditional independence assumption

x1, x2 give no extra information, so $p(x_3|y, x_1, x_2) = p(x_3|y)$

Naive Bayes: objective

given the training dataset $\mathcal{D} = \{(x^{(1)}, y^{(1)}), \ldots, (x^{(N)}, y^{(N)})\}$

maximize the joint likelihood (contrast with logistic regression)

$$\ell(w, u) = \sum_n \log p_{u,w}(x^{(n)}, y^{(n)})$$

$$= \sum_n \log p_u(y^{(n)}) + \log p_w(x^{(n)}|y^{(n)})$$

$$= \sum_n \log p_u(y^{(n)}) + \sum_n \log p_w(x^{(n)}|y^{(n)})$$

using Naive Bayes assumption

$$= \sum_n \log p_u(y^{(n)}) + \sum_d \sum_n \log p_{w_{[d]}}(x_d^{(n)}|y^{(n)})$$

separate MLE estimates for each part

Naive Bayes: train-test

given the training dataset $\mathcal{D} = \{(x^{(1)}, y^{(1)}), \ldots, (x^{(N)}, y^{(N)})\}$

training time

learn the prior class probabilities $p_u(y)$

learn the likelihood components $p_{w_{[d]}}(x_d|y) \quad \forall d$

test time find posterior class probabilities

$$\arg\max_c p(c|x) = \arg\max_c \frac{p_u(c) \prod_{d=1}^{D} p_{w_{[d]}}(x_d|c)}{\sum_{c'=1}^{C} p_u(c') \prod_{d=1}^{D} p_{w_{[d]}}(x_d|c')}$$

Class prior

$$p(c|x) = \frac{p_u(c) \prod_{d=1}^{D} p_{w_{[d]}}(x_d|c)}{\sum_{c'=1}^{C} p_u(c) \prod_{d=1}^{D} p_{w_{[d]}}(x_d|c')}$$

binary classification

Bernoulli distribution $p_u(y) = u^y(1-u)^{1-y}$

maximizing the log-likelihood

$$\ell(u) = \sum_{n=1}^{N} y^{(n)} \log(u) + (1 - y^{(n)}) \log(1-u)$$

$$= N_1 \log(u) + (N - N_1) \log(1-u)$$

frequency of class 1 in the dataset

frequency of class 0 in the dataset

setting its derivative to zero

$$\frac{\mathrm{d}}{\mathrm{d}u}\ell(u) = \frac{N_1}{u} - \frac{N-N_1}{1-u} = 0 \Rightarrow u^* = \frac{N_1}{N}$$

max-likelihood estimate (MLE) is the frequency of class labels

Class prior

$$p(c|x) = \frac{p_u(c) \prod_{d=1}^{D} p_{w_{[d]}}(x_d|c)}{\sum_{c'=1}^{C} p_u(c) \prod_{d=1}^{D} p_{w_{[d]}}(x_d|c')}$$

multiclass classification

categorical distribution $p_u(y) = \prod_{c=1}^{C} u_c^{y_c}$

assuming one-hot coding for labels

$u = [u_1, \ldots, u_C]$ is now a parameter vector

maximizing the log likelihood $\ell(u) = \sum_n \sum_c y_c^{(n)} \log(u_c)$

subject to: $\sum_c u_c = 1$

number of instances in class 1

closed form for the optimal parameter $u^* = [\frac{N_1}{N}, \ldots, \frac{N_C}{N}]$

all instances in the dataset

Likelihood terms

(class-conditionals)

$$p(c|x) = \frac{p_u(c) \prod_{d=1}^{D} p_{w_{[d]}}(x_d|c)}{\sum_{c'=1}^{C} p_u(c) \prod_{d=1}^{D} p_{w_{[d]}}(x_d|c')}$$

choice of likelihood distribution depends on the type of features

(likelihood encodes our assumption about "generative process")

- Bernoulli: binary features
- Categorical: categorical features
- Gaussian: continuous distribution
- ...

note that these are different from the choice of distribution for class prior

each feature x_d may use a different likelihood

separate max-likelihood estimates for each feature

$$w_{[d]}{}^* = \arg\max_{w_{[d]}} \sum_{n=1}^{N} \log p_{w_{[d]}}(x_d^{(n)} \mid y^{(n)})$$

Bernoulli Naive Bayes

binary **features**: *likelihood is Bernoulli*

$$\begin{cases} p_{w_{[d]}}(x_d \mid y=0) = \text{Bernoulli}(x_d; w_{[d],0}) \\ p_{w_{[d]}}(x_d \mid y=1) = \text{Bernoulli}(x_d; w_{[d],1}) \end{cases}$$

one parameter per label

short form: $p_{w_{[d]}}(x_d \mid y) = \text{Bernoulli}(x_d; w_{[d],y})$

max-likelihood estimation is similar to what we saw for the prior

closed form solution of MLE →

$$w^*_{[d],c} = \frac{N(y=c, x_d=1)}{N(y=c)}$$

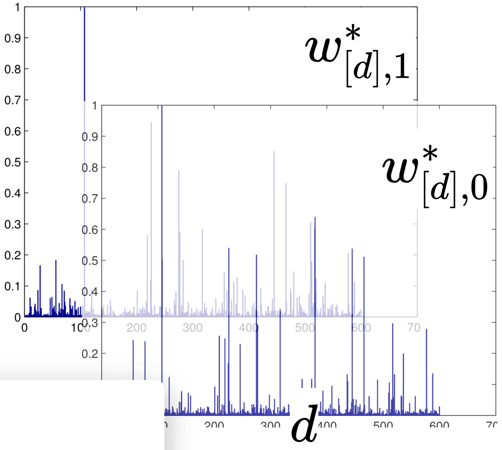
number of training instances satisfying this condition

Example: Bernoulli Naive Bayes

using naive Bayes for **document classification**:

- 2 classes (documents types)
- 600 binary features
  - $x_d^{(n)} = 1$ word d is present in the document n (vocabulary of 600)

likelihood of words in two document types →




```python
def BernoulliNaiveBayes(prior,# vector of size 2 for class prior
                        likelihood, #600 x 2: likelihood of each word under each class
                        x, #vector of size 600: binary features for a new document
                       ):
  logp = np.log(prior) + np.sum(np.log(likelihood * x[:,None]), 0) + \
                np.sum(np.log((1-likelihood) * (1 - x[:,None])), 0)
  log_p -= np.max(log_p) #numerical stability
  posterior = np.exp(log_p) # vector of size 2
  posterior /= np.sum(posterior) # normalize
  return posterior # posterior class probability
```

Multinomial Naive Bayes

what if we wanted to use *word frequencies* in document classification

$x_d^{(n)}$ is the number of times word d appears in document n

Multinomial likelihood: $p_w(x|c) = \frac{(\sum_d x_d)!}{\prod_{d=1}^{D} x_d!} \prod_{d=1}^{D} w_{d,c}^{x_d}$

we have a vector of size D for each class $C \times D$ (parameters)

MLE estimates: $w_{d,c}^{*} = \frac{\sum x_d^{(n)} y_c^{(n)}}{\sum_n \sum_{d'} x_{d'}^{(n)} y_c^{(n)}}$

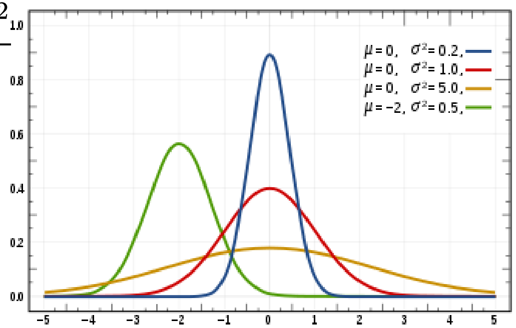
count of word d in all documents labelled **y**

total word count in all documents labelled **y**

Gaussian Naive Bayes

Gaussian likelihood terms

$$p_{w_{[d]}}(x_d \mid y) = \mathcal{N}(x_d; \mu_{d,y}, \sigma^2_{d,y}) = \frac{1}{\sqrt{2\pi\sigma_{d,y}{}^2}} e^{-\frac{(x_d - \mu_{d,y})^2}{2\sigma_{d,y}{}^2}}$$



> $w_{[d]} = (\mu_{d,1}, \sigma_{d,1}, \ldots, \mu_{d,C}, \sigma_{d,C})$
> one mean and std. parameter for each class-feature pair

writing log-likelihood and setting derivative to zero we get maximum likelihood estimate:

$$\mu_{d,y} = \frac{1}{N_c} \sum_{n=1}^{N} x_d^{(n)} y_c^{(n)}$$

$$\sigma^2_{d,y} = \frac{1}{N_c} \sum_{n=1}^{N} y_c^{(n)} (x_d^{(n)} - \mu_{d,y})^2$$

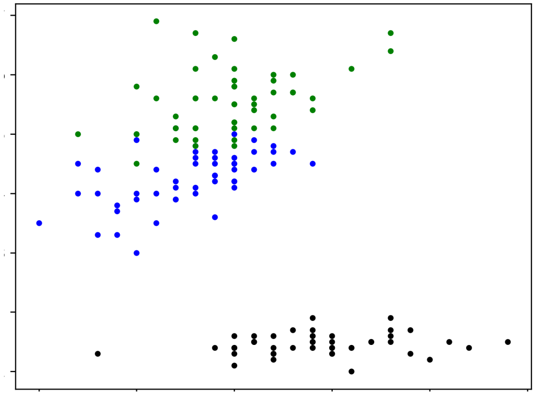
empirical mean & std of feature x_d
across instances with label y

Example: Gaussian Naive Bayes

classification on **Iris flowers dataset**:

(a classic dataset originally used by Fisher)

$N_c = 50$ samples with D=4 features, for each of C=3 species of Iris flower

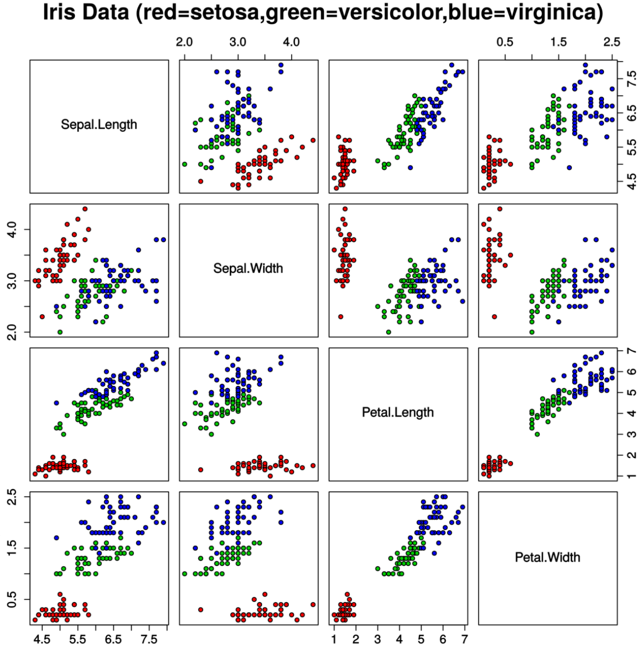


our setting

3 classes

2 features

(septal width, petal length)

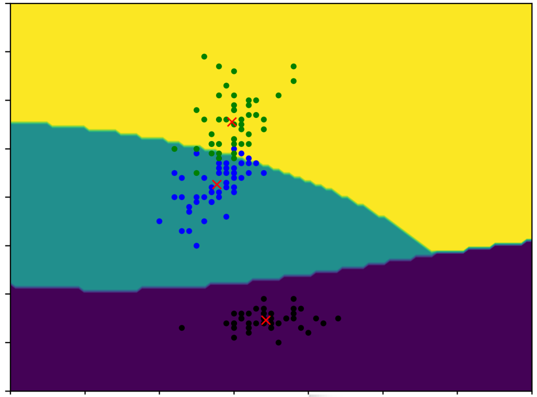


Example: Gaussian Naive Bayes

categorical class prior & Gaussian likelihood

```python
def GaussianNaiveBayes(
                X, # N x D
                y, # N x C
                Xtest,# N_test x D
               ):
    N,C = y.shape
    D = X.shape[1]
    mu, s = np.zeros((C,D)), np.zeros((C,D))
    for c in range(C): #calculate mean and std
        inds = np.nonzero(y[:,c])[0]
        mu[c,:] = np.mean(X[inds,:], 0)
        s[c,:] = np.std(X[inds,:], 0)
    log_prior = np.log(np.mean(y, 0))[:,None]
    log_likelihood = - np.sum( np.log(s[:,None,:]) +.5*(((Xt[None,:,:] - mu[:,None,:])/s[:,None,:])**2), 2)
    return log_prior + log_likelihood #N_text x C
```

decision boundaries are not linear!

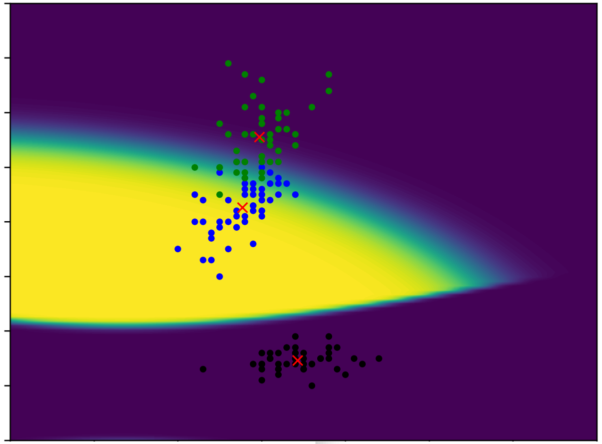


Example: Gaussian Naive Bayes

categorical class prior & Gaussian likelihood

```
def GaussianNaiveBayes(
                X, # N x D
                y, # N x C
                Xtest,# N_test x D
               ):
    N,C = y.shape
    D = X.shape[1]
    mu, s = np.zeros((C,D)), np.zeros((C,D))
    for c in range(C): #calculate mean and std
        inds = np.nonzero(y[:,c])[0]
        mu[c,:] = np.mean(X[inds,:], 0)
        s[c,:] = np.std(X[inds,:], 0)
    log_prior = np.log(np.mean(y, 0))[:,None]
    log_likelihood = - np.sum( np.log(s[:,None,:]) +.
- mu[:,None,:])/s[:,None,:])**2), 2)
    return log_prior + log_likelihood #N_text x C
```

posterior class probability for c=1



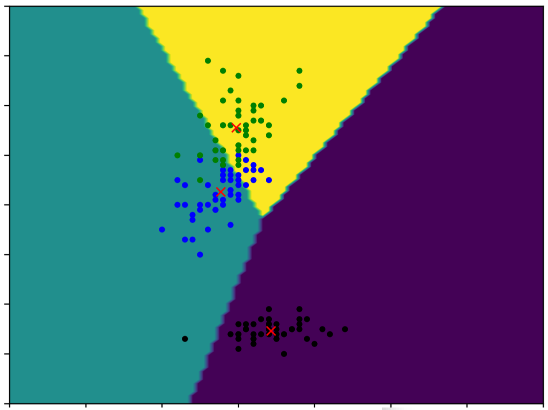
Example: Gaussian Naive Bayes

using the **same variance** for all classes

its value does not make a difference

```
def GaussianNaiveBayes(
                X, # N x D
                y, # N x C
                Xtest,# N_test x D
               ):
    N,C = y.shape
    D = X.shape[1]
    mu, s = np.zeros((C,D)), np.zeros((C,D))
    for c in range(C): #calculate mean and std
        inds = np.nonzero(y[:,c])[0]
        mu[c,:] = np.mean(X[inds,:], 0)
    log_prior = np.log(np.mean(y, 0))[:,None]
    log_likelihood = - np.sum(.5*(((Xt[None,:,:] - mu[:,None,:]))**2), 2)
    return log_prior + log_likelihood #N_text x C
```

decision boundaries **are linear**



Decision boundary in generative classifiers

decision boundaries: two classes have the same probability $p(y|x) = p(y'|x)$

which means $\log \frac{p(y=c|x)}{p(y=c'|x)} = \log \frac{p(c)p(x|c)}{p(c')p(x|c')} = \log \frac{p(c)}{p(c')} + \log \frac{p(x|c)}{p(x|c')} = 0$

not a function of x (ignore)

this ratio is linear (in some bases) for a large family of probabilities

(called linear exponential family)

$$p(x|c) = \frac{e^{w_{y,c}^T \phi(x)}}{Z(w_{y,c})} \rightarrow \log \frac{p(x|c)}{p(x|c')} = (w_{y,c} - w_{y,c'})^T \phi(x) + g(w_{y,c}, w_{y,c'})$$

linear using some bases

not a function of x

e.g., Bernoulli is a member of this family with $\phi(x) = x$

→ *Bernoulli Naive Bayes has a linear decision boundary linear.*

Discreminative vs generative classification

$p(y, x) = p(y)p(x \mid y)$ **generative**

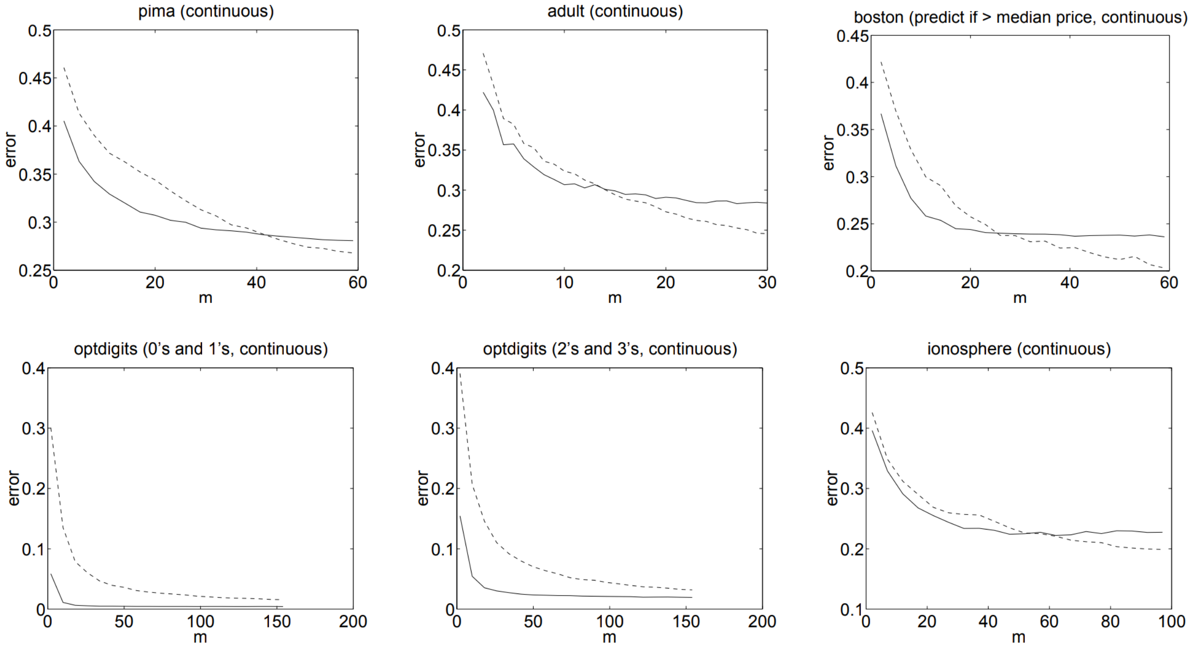
- maximize joint likelihood
- it makes assumptions about $p(x)$
- can deal with missing values
- can learn from unlabelled data
- often works better on smaller datasets

discriminative $p(y \mid x)$

- maximize ***conditional*** *likelihood*
- makes no assumption about $p(x)$
- often works better on larger datasets

Discriminative vs generative classification

Example naive Bayes vs logistic regression on UCI datasets



——— naive Bayes

- - - - - - logistic regression

m is #instances

from: Ng & Jordan 2001

Summary

- generative classification
  - learn the class prior and likelihood
  - Bayes rule for conditional class probability
- Naive Bayes
  - assumes conditional independence
    - *e.g., word appearances indep. of each other given document type*
  - class prior: Bernoulli or Categorical
  - likelihood: Bernoulli, Gaussian, Categorical...
  - MLE has closed form (in contrast to logistic regression)
  - estimated separately for each feature and each label
- evaluation measures for classification accuracy

Measuring performance

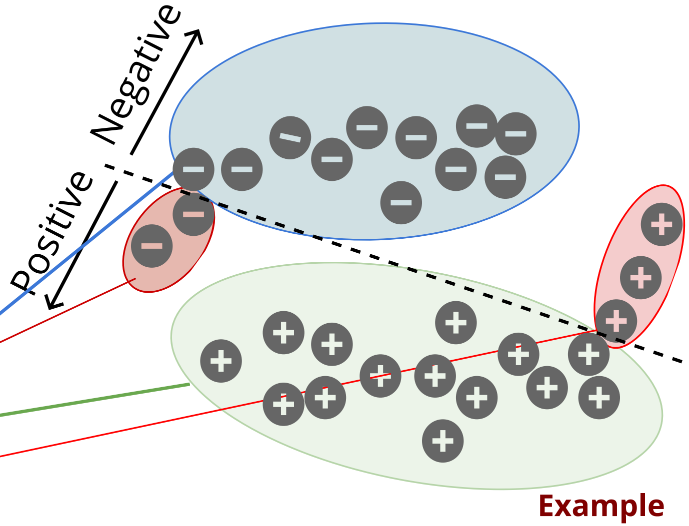
A side note on measuring performance of classifiers

binary classification

	Truth		Σ
Result	TP	FP	RP
	FN	TN	RN
Σ	P	N	

We use the confusion matrix
count the combinations of y and $\hat{y}$

	Truth		Σ
Result	14	2	16
	3	11	14
Σ	17	13	



Measuring performance

binary classification

use the confusion matrix to quantify difference metrics

	Truth		Σ
Result	TP	FP	RP
	FN	TN	RN
Σ	P	N	

marginals:

$$RP = TP + FP$$
$$RN = FN + TN$$
$$P = TP + FN$$
$$N = FP + TN$$

$$Accuracy = \frac{TP+TN}{P+N}$$

$$Error\ rate = \frac{FP+FN}{P+N}$$

$$Precision = \frac{TP}{RP}$$

$$Recall = \frac{TP}{P}$$

$$F_1 score = 2\frac{Precision \times Recall}{Precision + Recall}$$

{Harmonic mean}

Measuring performance

binary classification

	Truth		Σ
Result	TP	FP	RP
	FN	TN	RN
Σ	P	N	

$$Accuracy = \frac{TP+TN}{P+N}$$

$$Precision = \frac{TP}{RP}$$

$$Recall = \frac{TP}{P}$$

$$F_1 score = 2\frac{Precision \times Recall}{Precision + Recall}$$

{Harmonic mean}

Less common

$$Miss\ rate = \frac{FN}{P}$$

$$Fallout = \frac{FP}{N}$$

$$False\ discovery\ rate = \frac{FP}{RP}$$

$$Selectivity = \frac{TN}{N}$$

$$False\ omission\ rate = \frac{FN}{RN}$$

$$Negative\ predictive\ value = \frac{TN}{RN}$$

Threshold invariant: ROC & AUC

ROC as a function of threshold

TPR = TP/P (**recall**, sensitivity)

FPR = FP/N (**fallout**, false alarm)