# Data Representation
# Testing and evaluation schemes
# Labs and Tools

# Getting to Know the Data (I)

- Metadata
  - Attribute types:
    - binary, nominal (categorical), ordinal, numeric, …
  - Attribute roles:
    - Input, target, id, …
  - Attribute descriptions
- Simple visualization tools are very useful
  - Nominal attributes: histograms (Distribution consistent with background knowledge?)
  - Numeric attributes: graphs (Any obvious outliers?)
    - DO NOT UNDERESTIMATE

# Sweetviz

```
In [2]: import pandas as pd
        import sweetviz
        train = pd.read_csv(r"C:\Users\lenovo\Desktop\train.csv")
        test = pd.read_csv(r"C:\Users\lenovo\Desktop\test.csv")
```

```
In [3]: my_report = sweetviz.analyze([train, "Train"],target_feat='SalePrice')
```

```
Done! Use 'show' commands to display/save. [100%] 00:07 -> (00:00 left)
```

# Getting to Know the Data (II)

- Data quantity
  - Number of instances
    - If too few, results less reliable; use special methods (boosting, …)
  - Number of features
    - If too many, use feature reduction/selection
  - Number of targets
    - If very unbalanced, use stratified sampling
- Data relevance
  - What data is available for the task?
  - Is this data relevant?
  - Is additional relevant data available?
  - How much historical data is available?
  - Who are the data experts?

# Attribute Types

- Nominal
  - E.g., eye color={brown, blue, …}
  - No relation, ordering, or distance implied
  - Only equality tests
- Ordinal
  - E.g., grade={k, 1, …, 12}, height = {tall > med > short}
  - Order BUT no distance
- Continuous (numeric)
  - Interval quantities – integer (e.g., year)
    - Difference makes sense, not sum/product
  - Ratio quantities – real (e.g., length)
    - Measurement scheme defines 0 point, all operations allowed

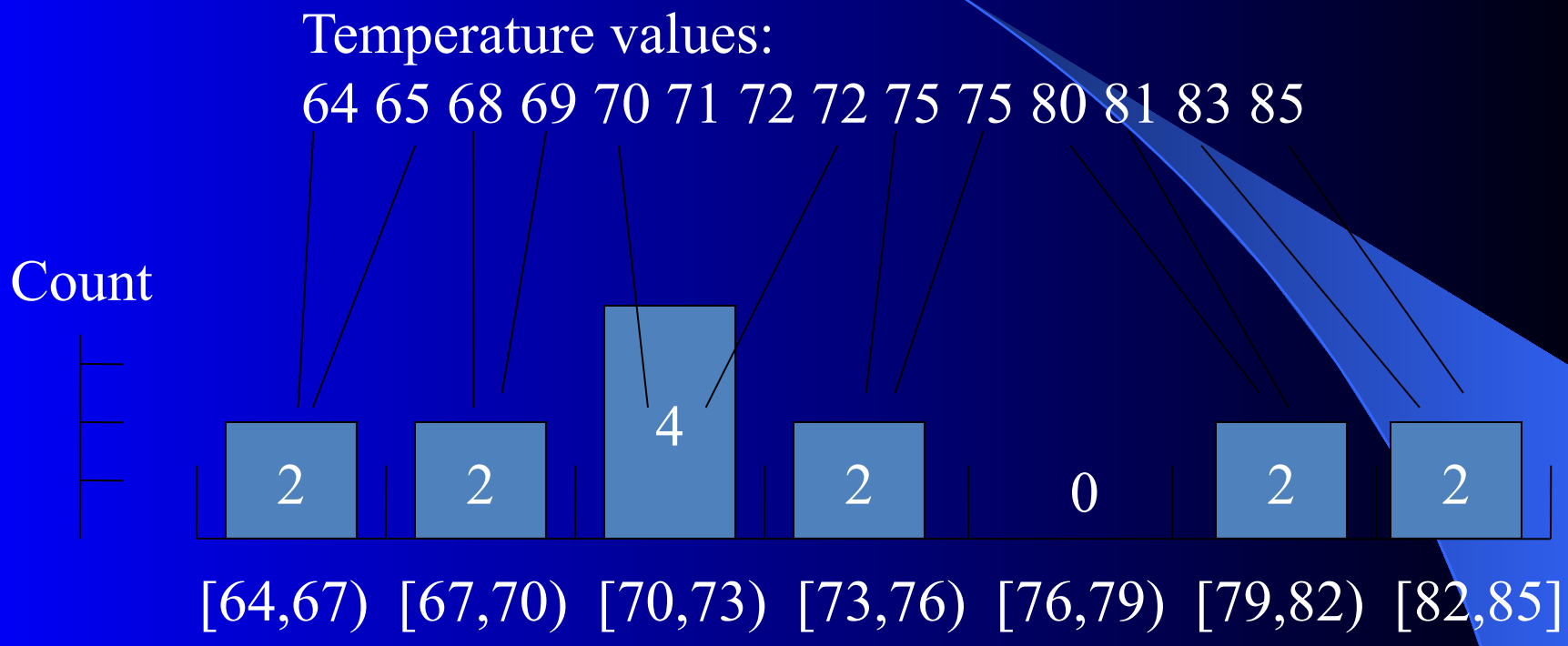
# Type Conversion

- Some algorithms require/fare better with continuous inputs (neural nets, regression, nearest neighbor)
- Some algorithms require discrete inputs (most versions of naïve Bayes, decision trees)
- Different encodings likely to produce different results
- Only show some here

# Sample Conversions

- Ord-to-Bool
  - Temp: cold, medium, hot => Temp>cold: 0/1, Temp>medium: 0/1
  - Thermometer/unary encoding (X => X 1s followed by 0)
- Bin-to-Num
  - Gender: M/F => 0/1
- Ord-to-Num (order must be preserved)
  - Grade: A, A-, … => 4.0, 3.7, …
- Nom-to-Num
  - One-hot encoding
- Num-to-Nom (e.g., discretization)

# Discretization: Equal-Width




- May produce clumping if data is skewed

# Standardization and Normalization

- Assume two input features in an astronomical task as follows:
  - Weight of the planet in grams, Diameter of the planet in light-years
- Solutions:
  - Standardization
    - Transform values into number of standard deviations from the mean
  - Normalization
    - Transform values to fall within a specified range
    - Often, range = max value – min value => [0, 1]

# Precision “Illusion”

- Example: gene expression may be reported as X83 = 193.3742, but measurement error may be +/- 20
- Actual value is in [173, 213] range, so it may be appropriate to round the data to 190
- Do not assume that every reported digit is significant!

# Missing Values (I)

- Types: unknown, unrecorded, irrelevant
  - malfunctioning equipment
  - changes in experimental design
  - collation of different datasets
  - measurement not possible

| Name | Age | Sex | Pregnant | … |
|---|---|---|---|---|
| Mary | 25 | F | N | |
| Jane | 27 | F | - | |
| Joe | 30 | M | - | |
| Anna | 2 | F | - | |
| | | | | |

In medical data, value for **Pregnant** attribute for **Jane** is missing, while for **Joe** or **Anna** should be probably be considered **Not applicable**

# Missing Values (II)

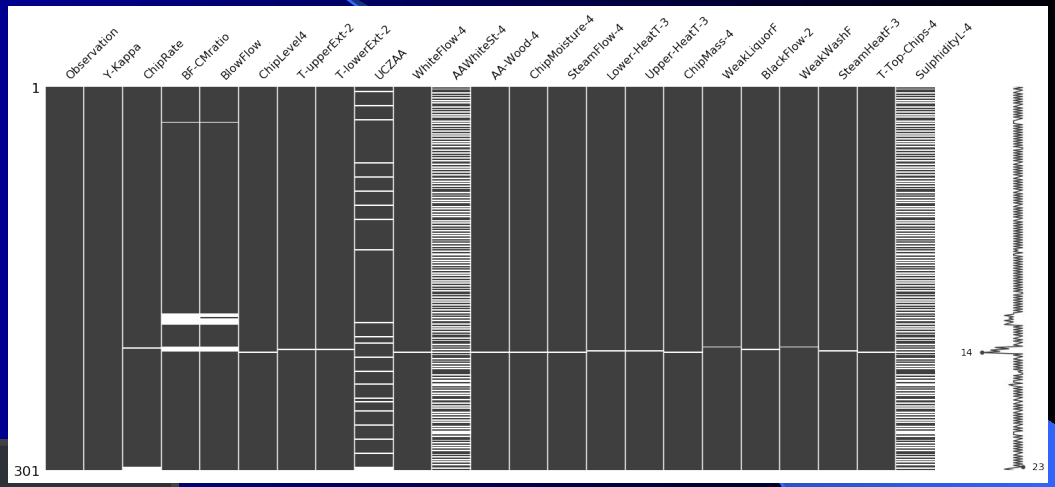
- Handling methods:
  - Remove records with missing values
  - Treat as separate value
  - Treat as don't know
  - Treat as don't care
  - Use imputation techniques
    - Mode, Median, Average
    - Regression (i.e., predict based on others)
  - Danger: BIAS
- Python missingno package

# Quality Investigation – Missing Values

- missingno package

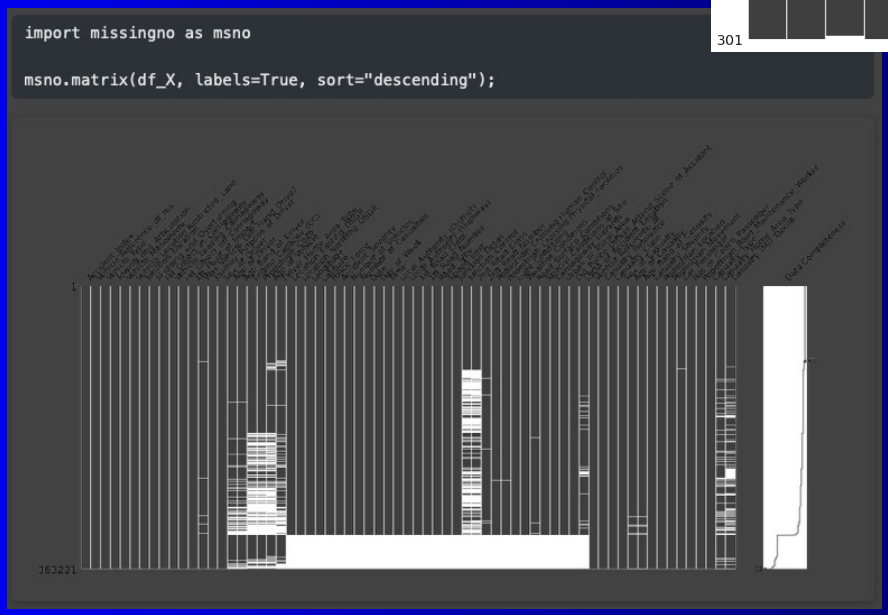
import missingno as msno

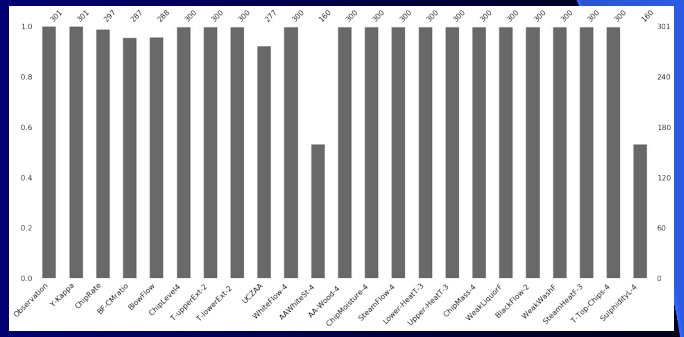
msno.matrix(df)
msno.bar(df)




```
import missingno as msno

msno.matrix(df_X, labels=True, sort="descending");
```





# Missing Data

- Structurally missing
  - Missing because it does not exist (# pregnant men)
- Missing completely at random
  - Whether or not a value is missing is unrelated to its value or the value of any other variable
  - Missing values can be excluded or imputed with mean or median
- Missing at random
  - Whether or not a value is missing does not depend on its value after controlling for another variable
  - Missing values can be predicted from other variables
- Missing not at random
  - Missing values that are not MCAR or MAR
  - Hardest type of missingness

For more information: https://www.uvm.edu/~statdhtx/StatPages/Missing_Data/Missing.html

# Outliers and Errors

- Outliers are values thought to be out of range
- Approaches:
  - Do nothing
  - Enforce upper and lower bounds
  - Let binning/discretization handle the problem

# Class Imbalance (I)

- Sometimes, class distribution is skewed
  - Monthly attrition: 97% stay, 3% defect
  - Medical diagnosis: 90% healthy, 10% disease
  - eCommerce: 99% don't buy, 1% buy
  - Security: >99.99% of Americans are not terrorists
- Similar situation with multiple classes
- Majority class classifier can be 97% correct, yet completely useless

# Class Imbalance (II)

- Two classes
  - Undersample (select desired number of minority class instances, add equal number of randomly selected majority class)
  - Oversample (select desired number of majority class, sample minority class with replacement)
  - Use boosting, cost-sensitive learning, etc.
- Generalize to multiple classes
  - Approximately equal proportions of each class in training and test sets (stratification)

- Python imbalanced-learn package
  - Synthetic Minority Oversampling Technique (SMOTE)

# Feature Issues

- Typically do not use features where
  - Almost all instance have the same value (no information)
    - If there is a significant, though small, percentage of other values, then might still be useful
  - Almost all instances have unique values (e.g., SSN, phone-numbers)
    - Might be able to use a variation of the feature (such as area code)
  - The feature is highly correlated with another feature
    - In this case the feature may be redundant and only one is needed
  - Careful if feature is too highly correlated with the target (leaky)
    - Check this case as the feature may just be a synonym with the target and will thus lead to overfitting (e.g., the output target was bundled with another product so they always occurred together)

# Improving Performance

- When trying to improve performance, you may need
  - More data
  - Better features
  - Different ML models or hyperparameters
  - Etc.
- One way to decide if you need more/better data
  - Compare your accuracy on training and test set
    - bad training set accuracy, probably need better data/ features. (though might need a different learning model/hyperparameters)
    - test set accuracy much worse than training set accuracy, gathering more data usually a good direction. (though regularization or learning model/hyperparameters could still be significant issue)

# Data Files

- Come in many varieties
  - Most commonly csv, arff
- Places to find data
  - UC Irvine data repository - https://archive.ics.uci.edu/
  - Kaggle datasets - https://www.kaggle.com/datasets
  - Data.gov - https://data.gov/
  - Census.gov - https://www.census.gov/
  - HealthData.gov - https://healthdata.gov/
  - Sports-Statistics.com - https://sports-statistics.com/sports-data/sports-data-sets-for-data-modeling-visualization-predictions-machine-learning/

# CSV Files

- Comma Separated Value
- Can come with a header or without a header line
- One line per row of the dataframe
- No data types
  - Type must be inferred or set by the programmer

```
ID,Last Name,First Name,City,State,Gender,Student Status,Major,Country,Age,SAT,Average score (grade),Height (in),Newspaper readership (times/wk),,,,,,,,,
1,DOE01,JANE01,Los Angeles,California,Female,Graduate,Politics,US,30,2263,67,61,5,,,,,,,,,
2,DOE02,JANE02,Sedona,Arizona,Female,Undergraduate,Math,US,19,2006,63,64,7,,,,,,,,,
3,DOE01,JOE01,Elmira,New York,Male,Graduate,Math,US,26,2221,78,73,6,,,,,,,,,
4,DOE02,JOE02,Lackawana,New York,Male,Graduate,Econ,US,33,1716,78,68,3,,,,,,,,,
5,DOE03,JOE03,Defiance,Ohio,Male,Graduate,Econ,US,37,1701,65,71,6,,,,,,,,,
6,DOE04,JOE04,Tel Aviv,Israel,Male,Graduate,Econ,Israel,25,1786,69,67,5,,,,,,,,,
7,DOE05,JOE05,Cimax,North Carolina,Male,Graduate,Politics,US,39,1577,96,70,5,,,,,,,,,
8,DOE03,JANE03,Liberal,Kansas,Female,Undergraduate,Politics,US,21,1842,87,62,5,,,,,,,,,
9,DOE04,JANE04,Montreal,Canada,Female,Undergraduate,Math,Canada,18,1813,91,62,6,,,,,,,,,
10,DOE05,JANE05,New York,New York,Female,Graduate,Math,US,33,2041,71,66,5,,,,,,,,,
11,DOE06,JOE06,Hot Coffe,Mississippi,Male,Undergraduate,Econ,US,18,1787,82,67,3,,,,,,,,,
12,DOE06,JANE06,Java,Virginia,Female,Graduate,Math,US,38,1513,79,59,5,,,,,,,,,
13,DOE07,JOE07,Varna,Bulgaria,Male,Graduate,Politics,Bulgaria,30,1637,79,63,4,,,,,,,,,
14,DOE08,JOE08,Moscow,Russia,Male,Graduate,Politics,Russia,30,1512,70,75,6,,,,,,,,,
```

# ARFF Files

- An ARFF (Attribute-Relation File Format) file is an ASCII text file that describes a Machine Learning dataset (or relation).
  - Developed at the University of Waikato (NZ) for use with the Weka machine learning software (http://www.cs.waikato.ac.nz/~ml/weka).
  - We will commonly use the ARFF format for CS 270
- ARFF files have two distinct sections:
  - Metadata information
    - Name of relation (Data Set)
    - List of attributes and domains
  - Data information
    - Actual instances or rows of the relation
- Optional comments may also be included which give information about the Data Set (lines prefixed with %)

# Sample ARFF File

```
% 1. Title: Pizza Database
% 2. Sources:
%      (a) Creator: BYU CS 270 Class…
%      (b) Statistics about the features, etc.

@RELATION Pizza

@ATTRIBUTE Weight        real
@ATTRIBUTE Crust         {Pan, Thin, Stuffed}
@ATTRIBUTE Cheesiness    real
@ATTRIBUTE Meat          {True, False}
@ATTRIBUTE Quality       {Good, Great}

@DATA
.9,  Stuffed,    99,     True,     Great
.1,  Thin,       2,      False,    Good
?,   Thin,       60,     True,     Good
.6,  Pan,        60,     True,     Great
```

- Any column could be the output, but we will assume that the last column(s) is the output
- Assume cheesiness is linear (an integer between 0 and 100)
- What would you do to this data before using it with a perceptron and what would the perceptron look like? – Show an updated ARFF row

# Performance Measures

- There are a number of ways to measure the performance of a learning algorithm:
  - Predictive accuracy of the induced model (or error)
  - Size of the induced model
  - Time to compute the induced model
  - etc.
- We will focus mostly on accuracy/error
- Fundamental Assumption:

  *Future novel instances are drawn from the same/similar distribution as the training instances*

# Evaluation of Classification

| predicted outcome \ actual outcome | 1 | 0 |
|---|---|---|
| 1 | *a* | *b* |
| 0 | *c* | *d* |

Accuracy = (a+d) / (a+b+c+d)

- Not always the best choice
  - Assume 1% fraud,
  - model predicts no fraud
  - What is the accuracy?

| Predicted Class \ Actual Class | Fraud | No Fraud |
|---|---|---|
| **Fraud** | **0** | **0** |
| **No Fraud** | **10** | **990** |

# Evaluation of Classification

Other options:

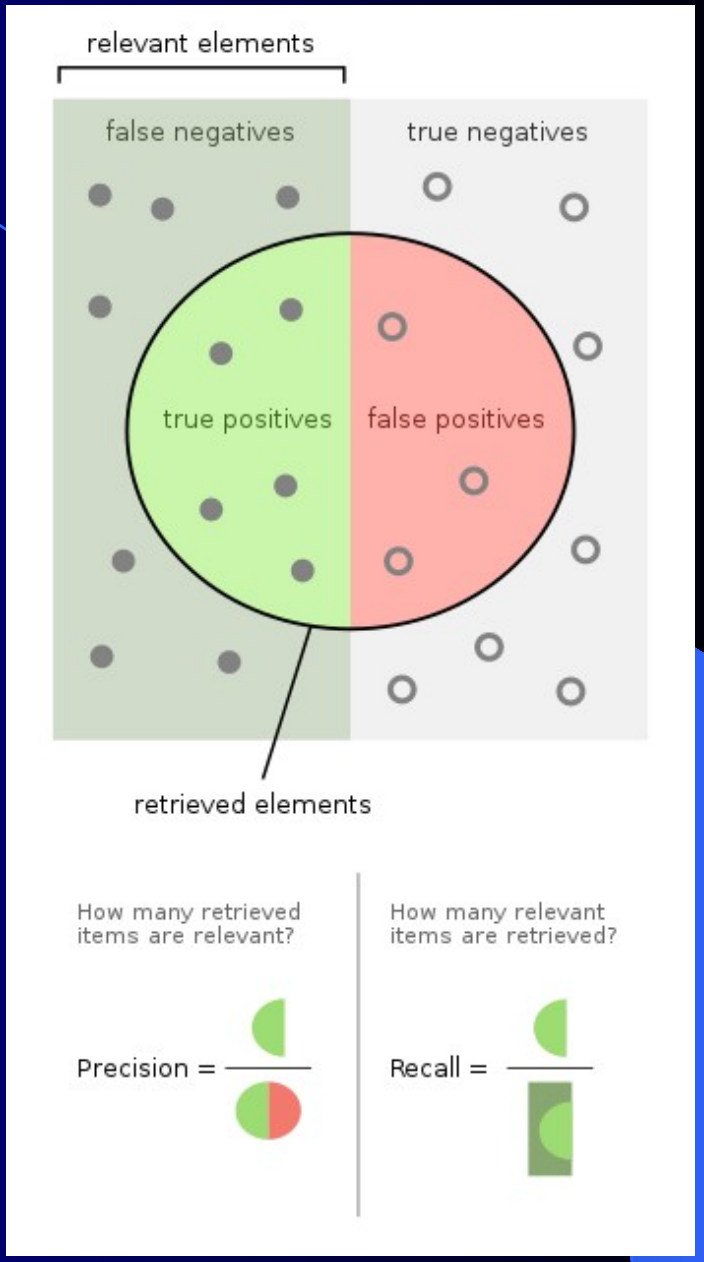
- **recall** or sensitivity (how many of those that are really positive did you predict?):
  - a/(a+c)
- **precision** (how many of those predicted positive really are?)
  - a/(a+b)
- **Specificity** (how many of the negatives did you get right?)
  - d/(b+d)

| | | actual outcome 1 | actual outcome 0 |
|---|---|---|---|
| predicted outcome | 1 | *a* | *b* |
| | 0 | *c* | *d* |

| | | Predicted | |
|---|---|---|---|
| | | **Negative** | **Positive** |
| **Actual** | **Negative** | True Negative | False Positive |
| | **Positive** | False Negative | True Positive |

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

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



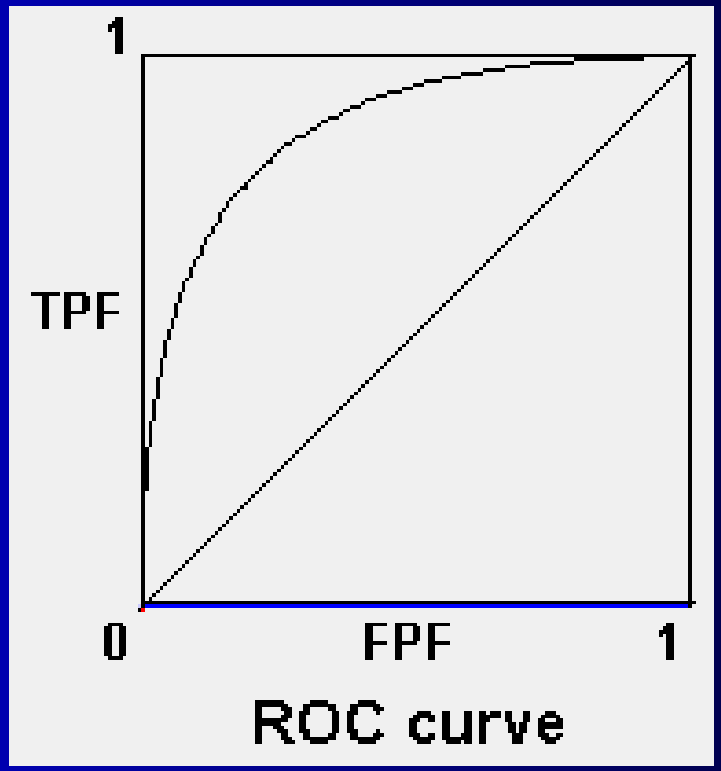
# F1

- Together, precision and recall give more information than just accuracy.
- Precision and recall are always in tension
  - Increasing one tends to decrease another
- Can combine for a single measure – F1
  - F1 = 2 * precision * recall / (precision + recall)
- Harmonic mean of precision and recall
  - Use harmonic mean when units are different
  - Eg. - Average MPH over different distances

# ROC CURVES

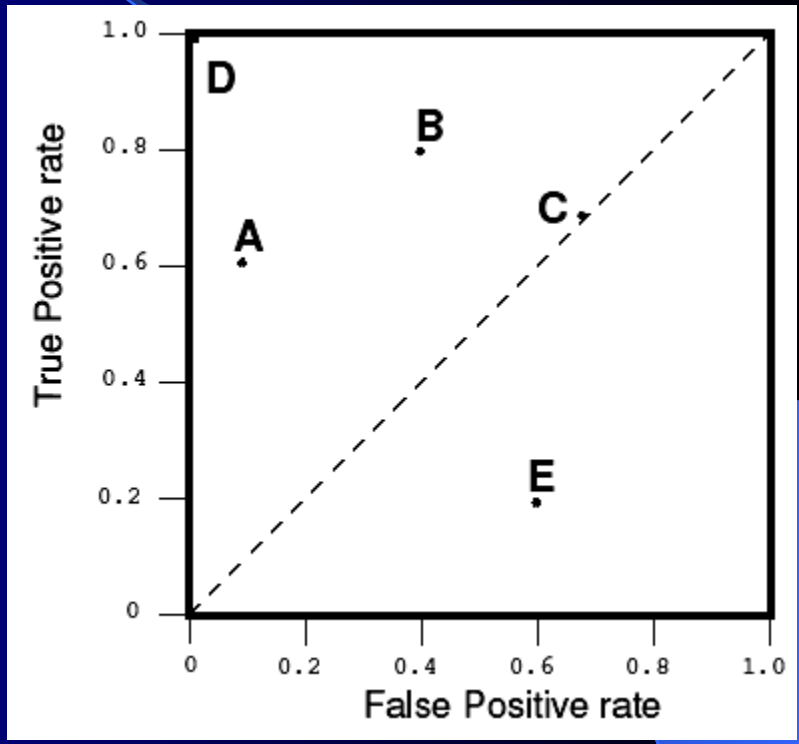
# Receiver Operating Characteristic curve

- ROC curves were developed in the 1950's as a by-product of research into making sense of radio signals contaminated by noise. More recently it's become clear that they are remarkably useful in decision-making.
- True positive and False positive fractions are plotted as we move the dividing threshold. They look like:



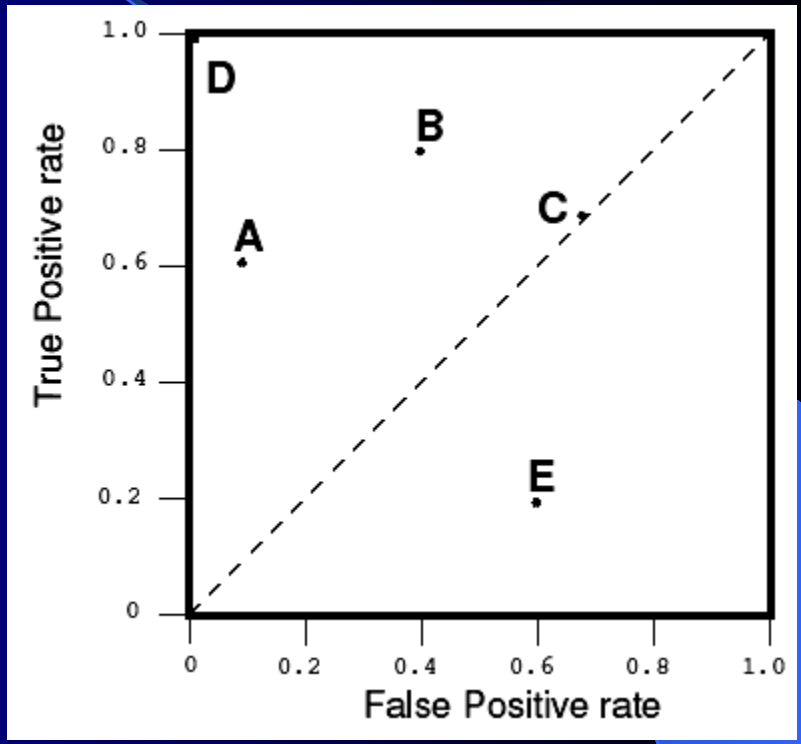
# ROC Space

- ROC graphs are two-dimensional graphs in which TP rate is plotted on the Y axis and FP rate is plotted on the X axis.
- An ROC graph depicts relative trade-offs between benefits (true positives) and costs (false positives).
- Figure shows an ROC graph with five classifiers labeled A through E.
- A discrete classier is one that outputs only a class label.
- Each discrete classier produces an (fp rate, tp rate) pair corresponding to a single point in ROC space.
- Classifiers in figure are all discrete classifiers.



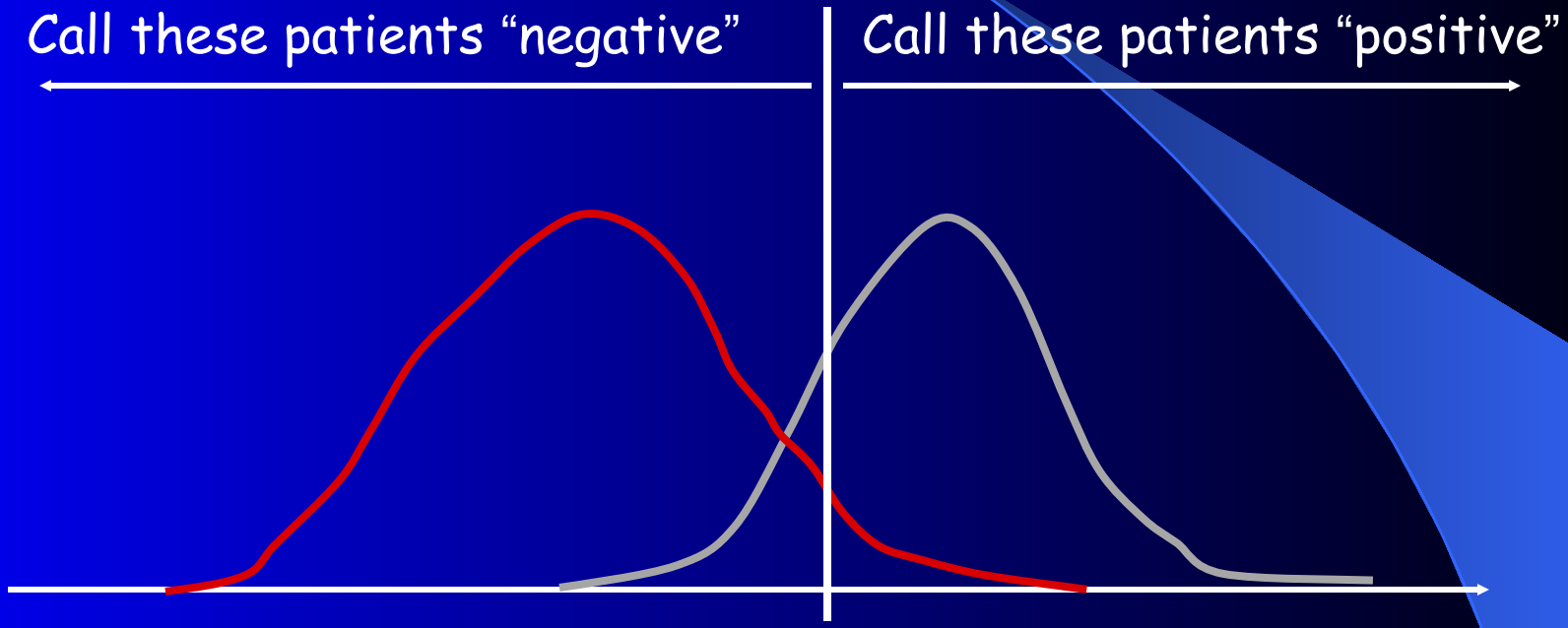
# Several Points in ROC Space

- **Lower left point (0, 0)** represents the strategy of never issuing a positive classification;
  - such a classier commits no false positive errors but also gains no true positives.
- **Upper right corner (1, 1)** represents the opposite strategy, of unconditionally issuing positive classifications.
- **Point (0, 1)** represents perfect classification.
  - D's performance is perfect as shown.
- Informally, one point in ROC space is better than another if it is to the northwest of the first
  - tp rate is higher, fp rate is lower, or both.

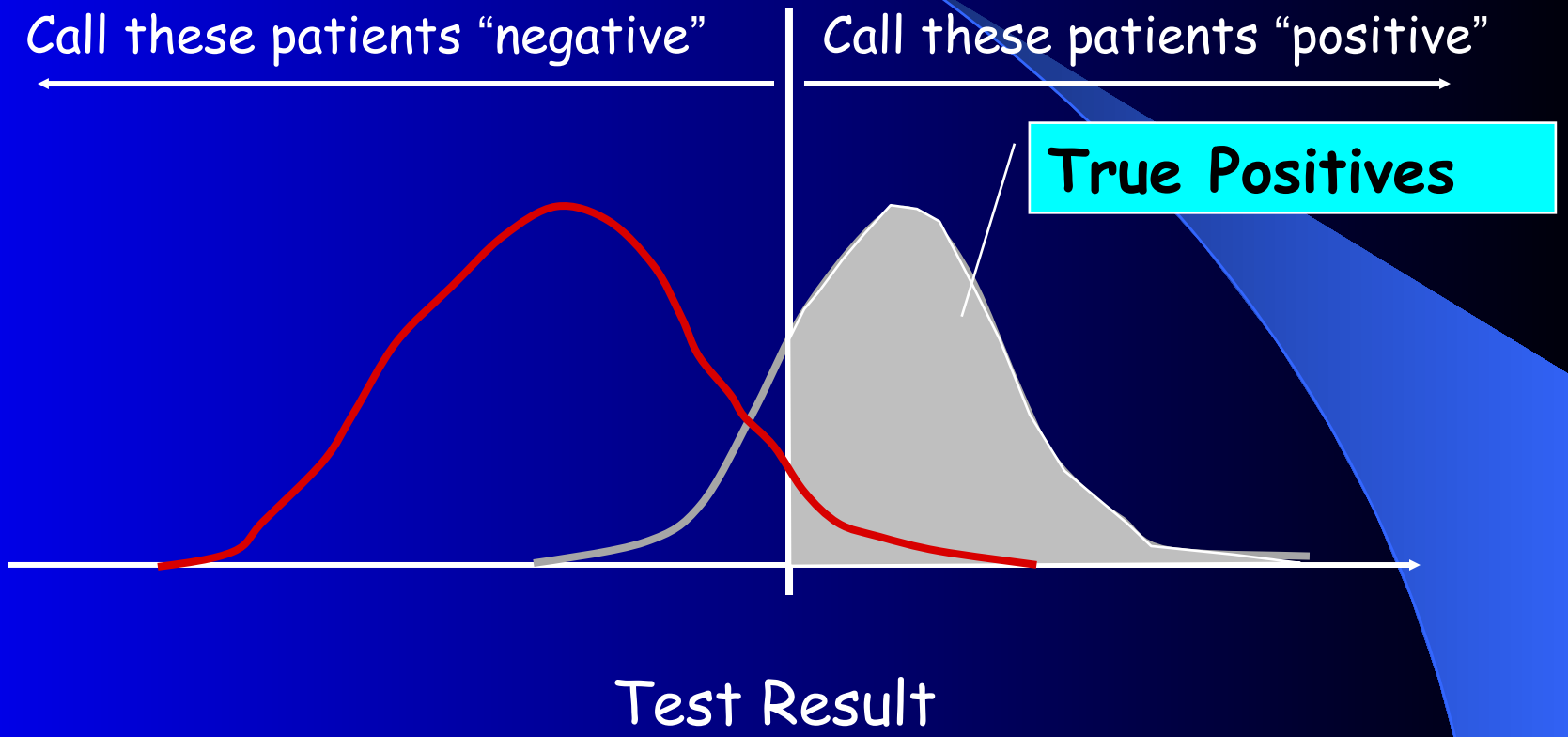


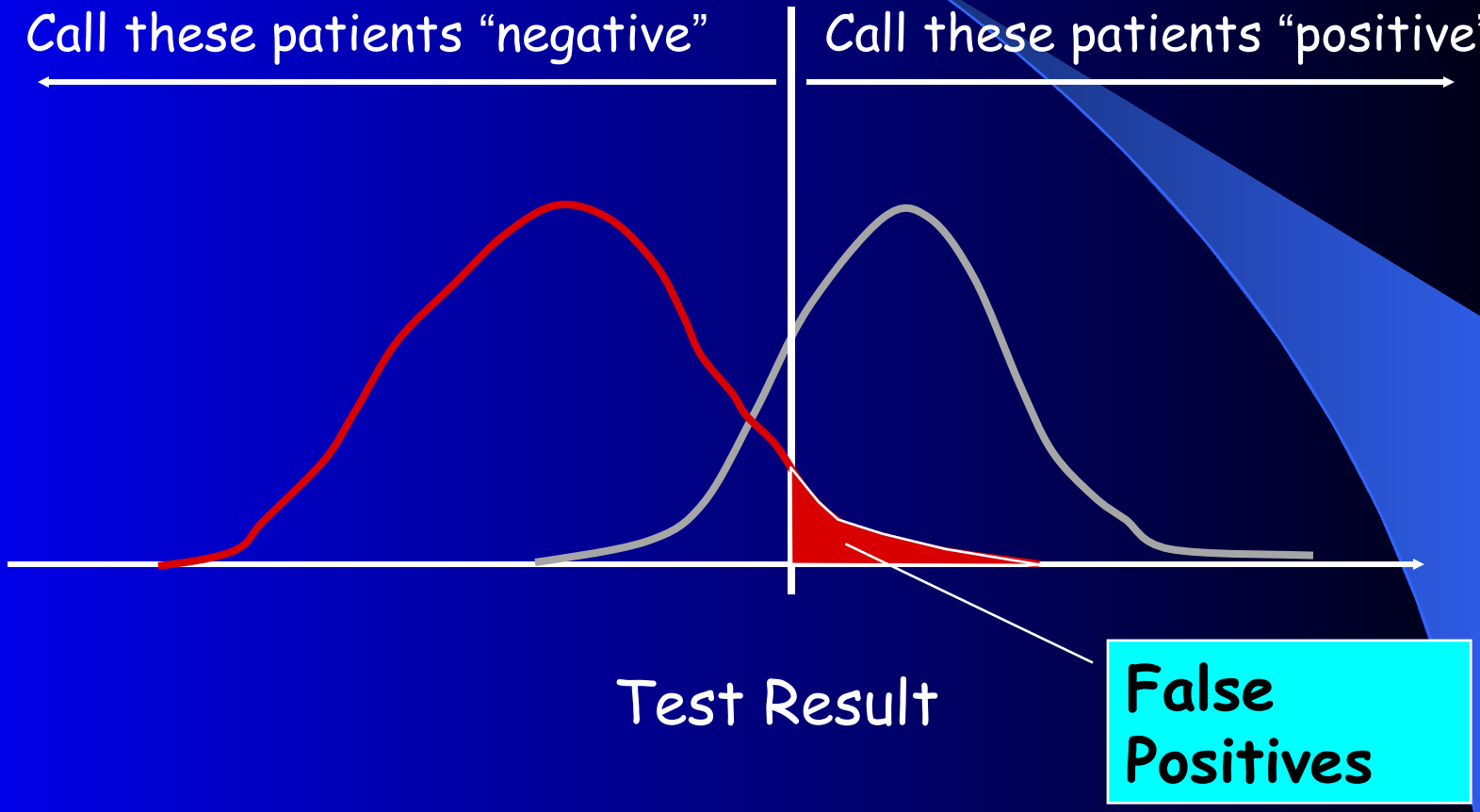
# Specific Example

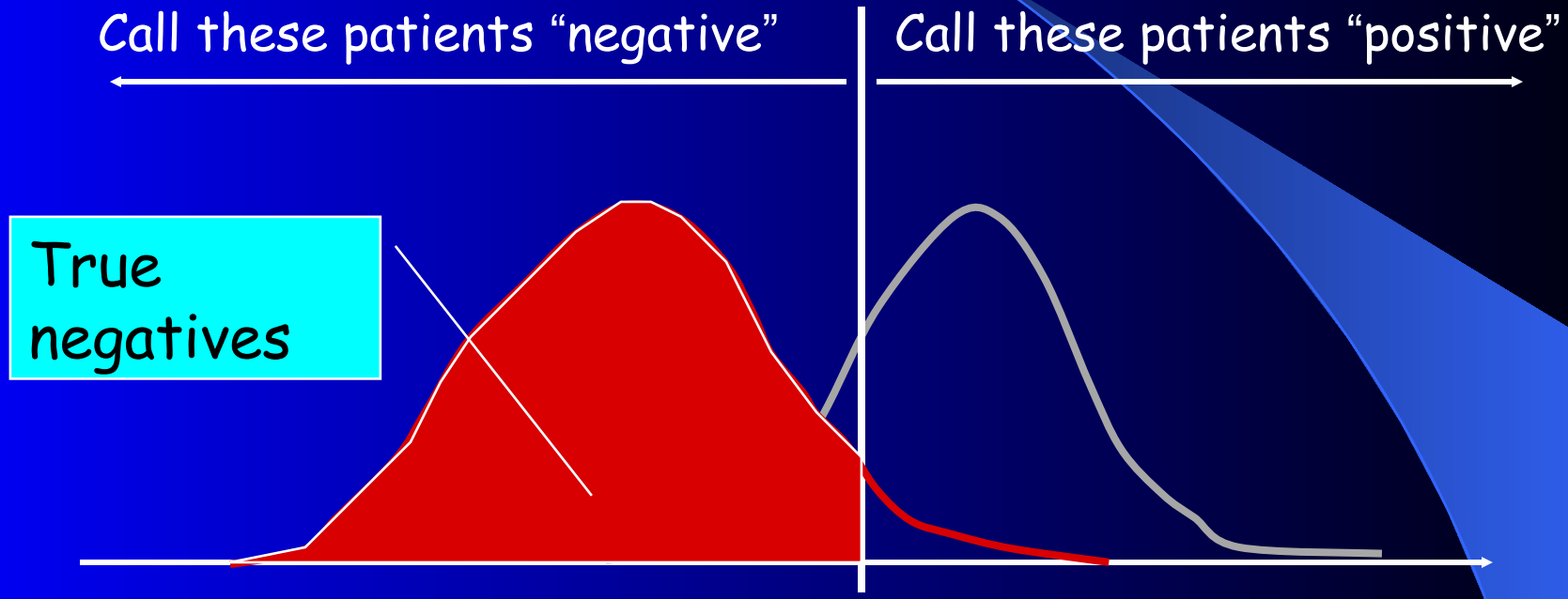
# Threshold


Call these patients “negative”
Call these patients “positive”
Test Result

# Some definitions ...

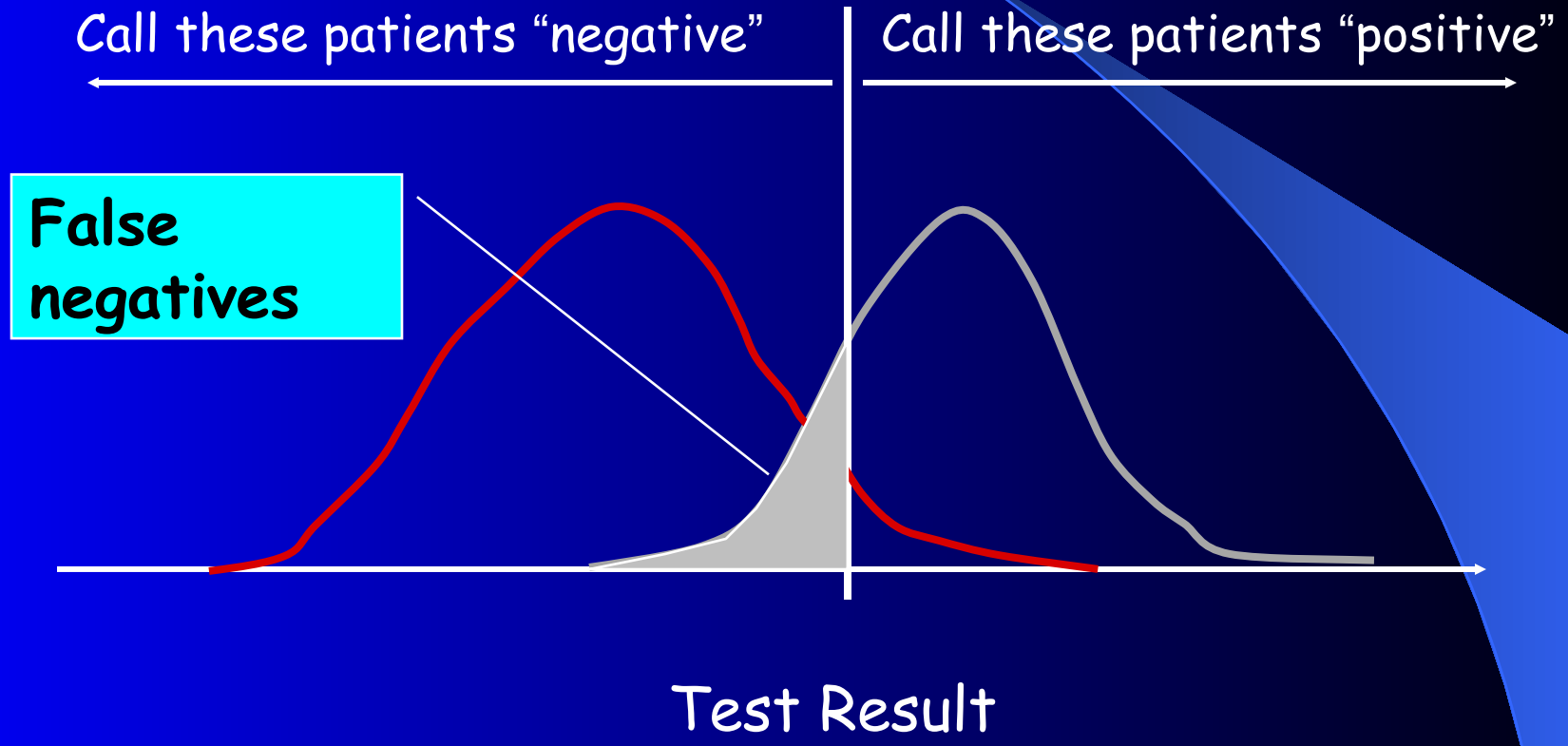



Call these patients “negative”
Call these patients “positive”
Test Result
False Positives
without the disease
with the disease




without the disease

with the disease


Call these patients "negative"
Call these patients "positive"
False negatives
Test Result
without the disease
with the disease

# Moving the Threshold: right


without the disease

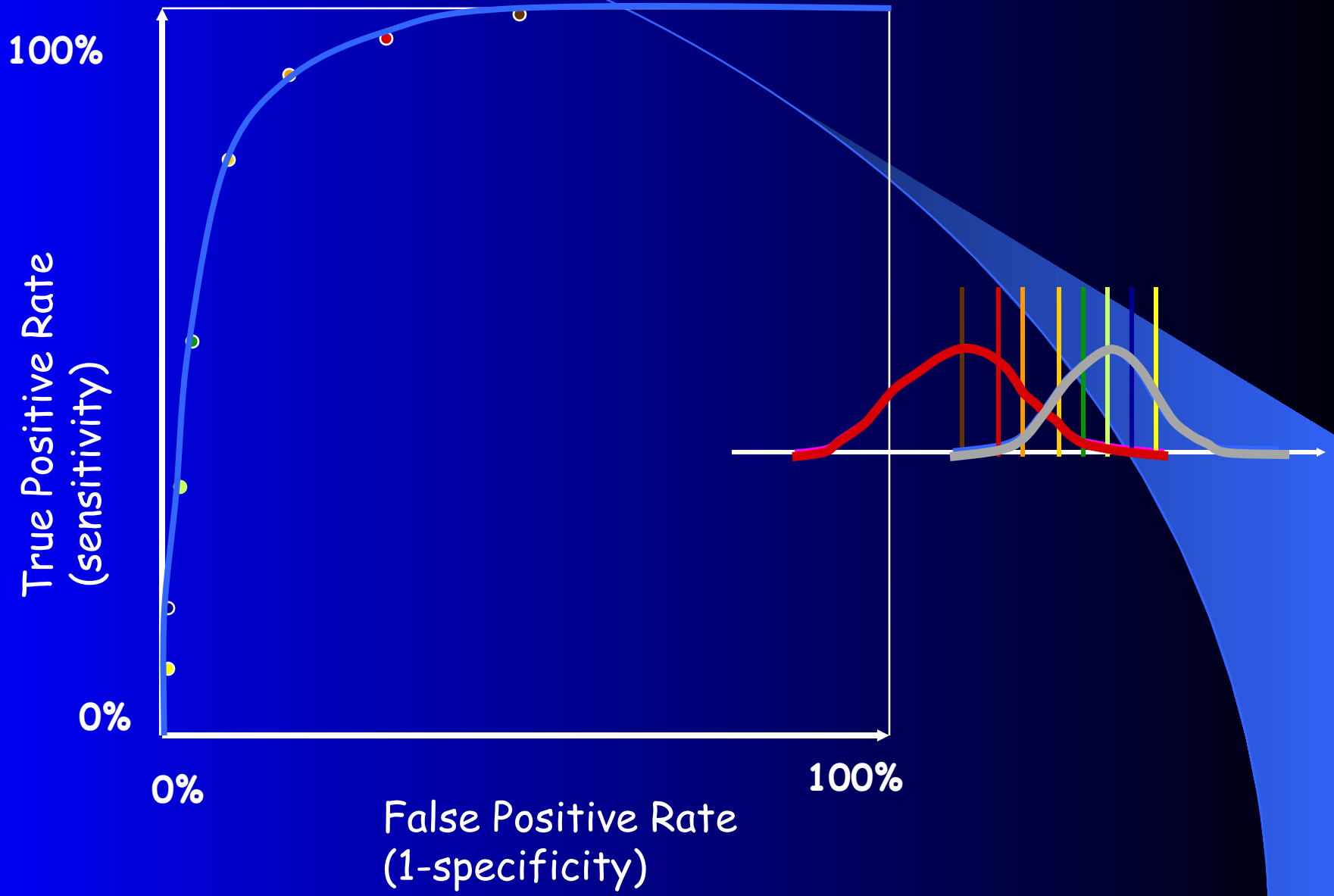
with the disease

# Moving the Threshold: left


without the disease

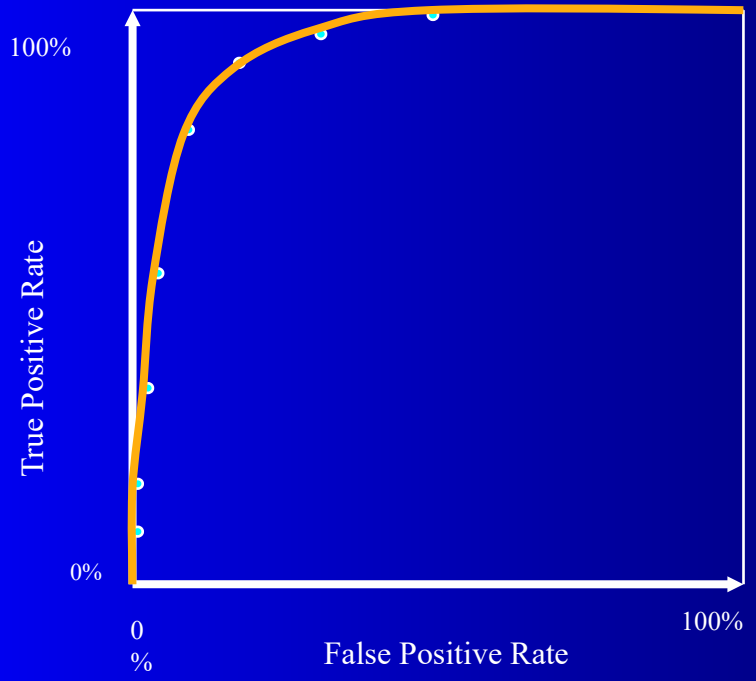
with the disease

# ROC curve

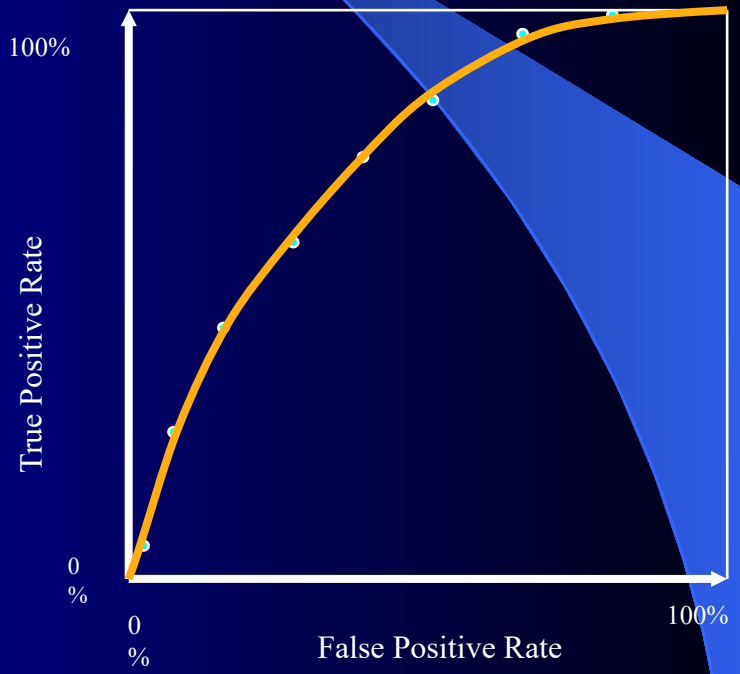

100%
True Positive Rate
(sensitivity)
0%
0%
100%
False Positive Rate
(1-specificity)

# ROC curve comparison

A good test:

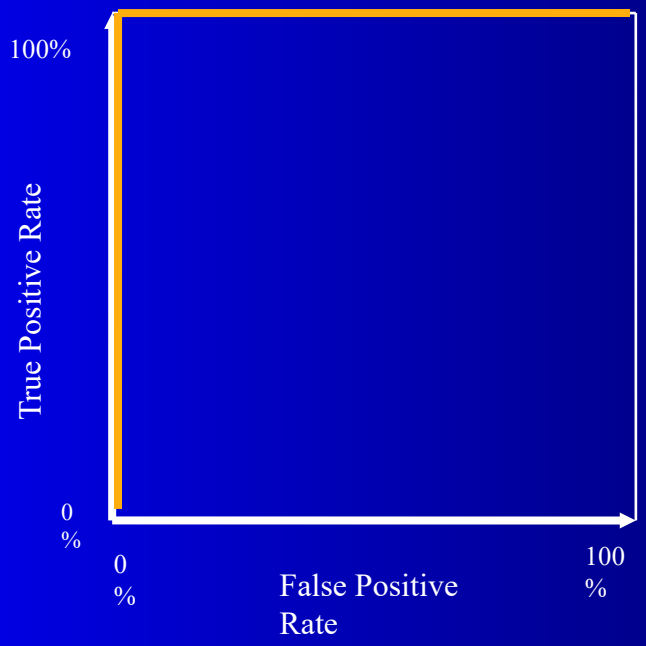



A poor test:



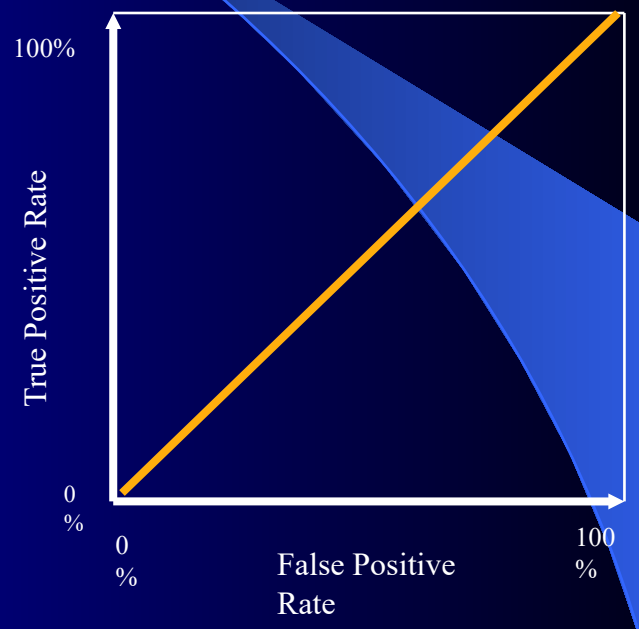
# ROC curve extremes

Best Test:




The distributions don't overlap at all

Worst test:




The distributions overlap completely

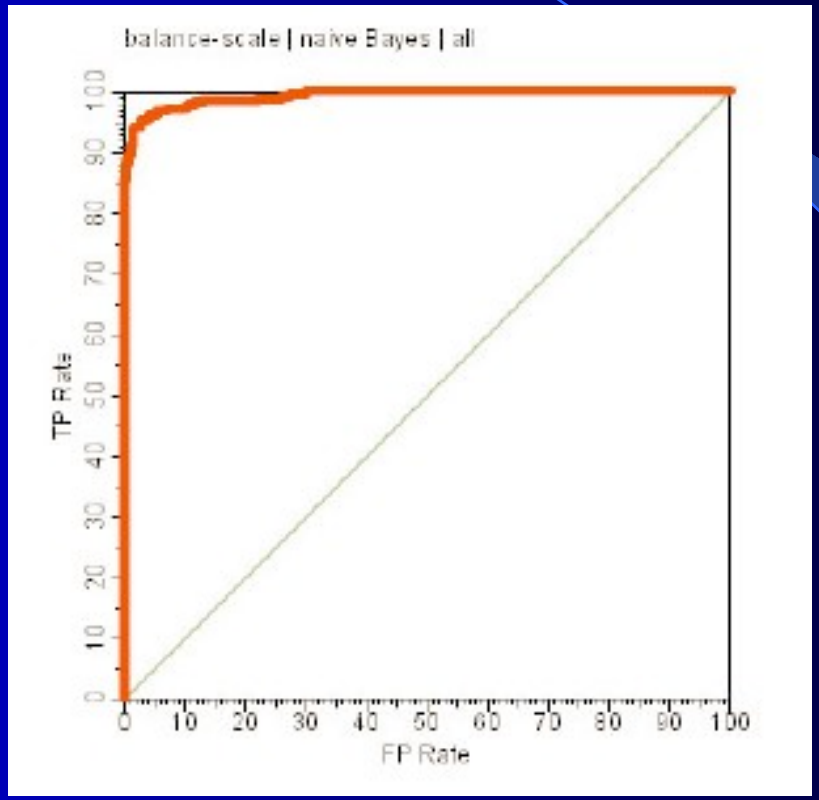
# How to Construct ROC Curve for one Classifier

- Sort the instances according to their $P_{pos}$.
- Move a threshold on the sorted instances.
- For each threshold define a classifier with confusion matrix.
- Plot the TPr and FPr rates of the classfiers.

| $P_{pos}$ | True Class |
|---|---|
| 0.99 | pos |
| 0.98 | pos |
| 0.7 | neg |
| 0.6 | pos ← |
| 0.43 | neg |

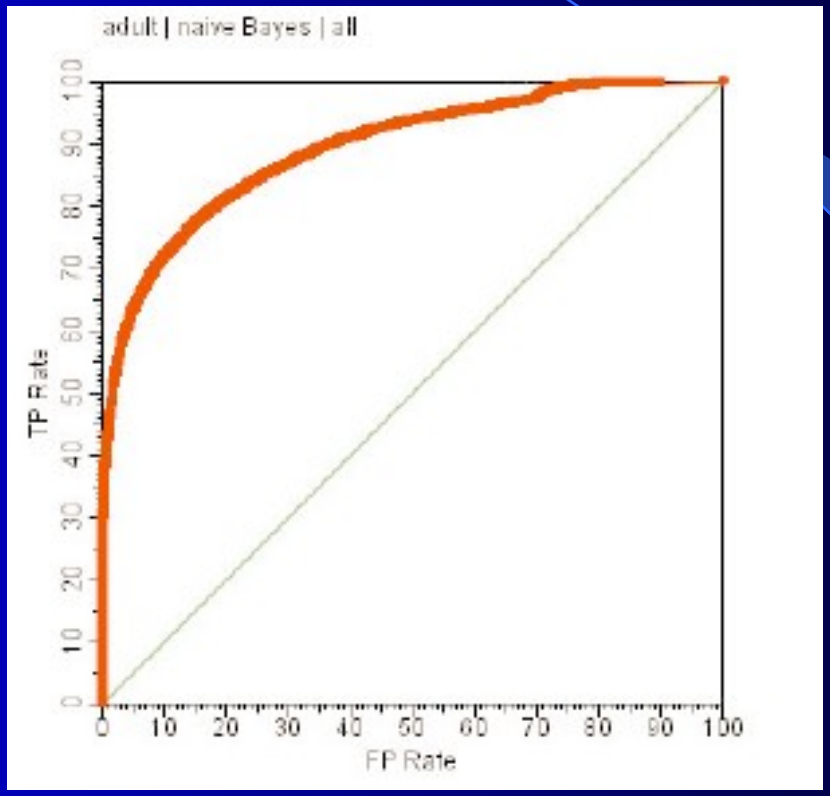
| True | Predicted | |
|---|---|---|
| | pos | neg |
| pos | 2 | 1 |
| neg | 1 | 1 |

# ROC for one Classifier



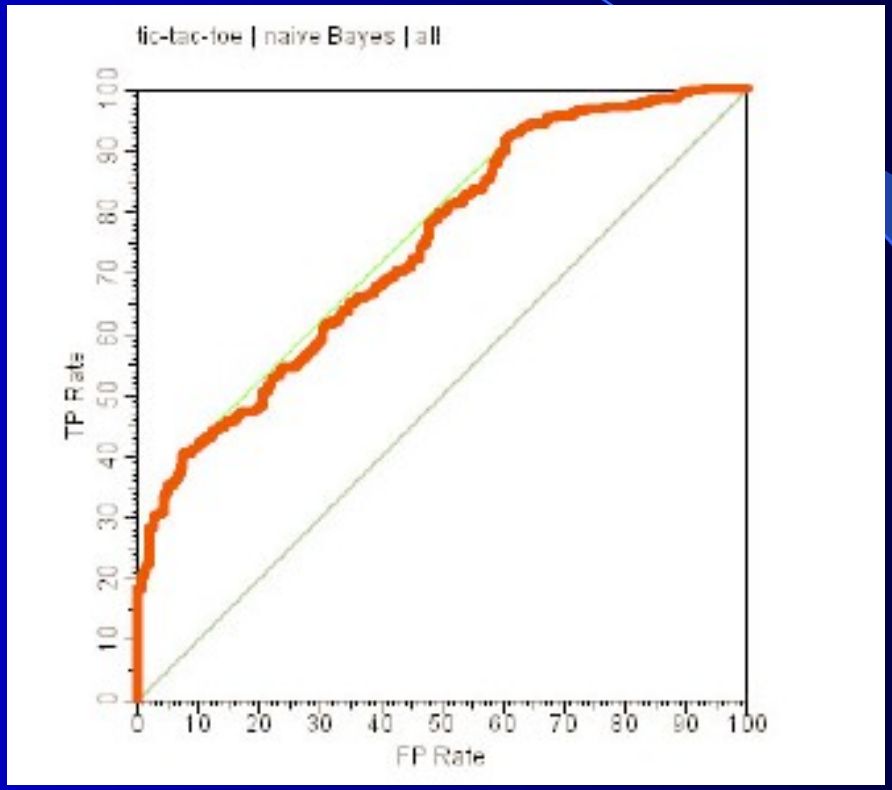

Good separation between the classes, convex curve.

# ROC for one Classifier



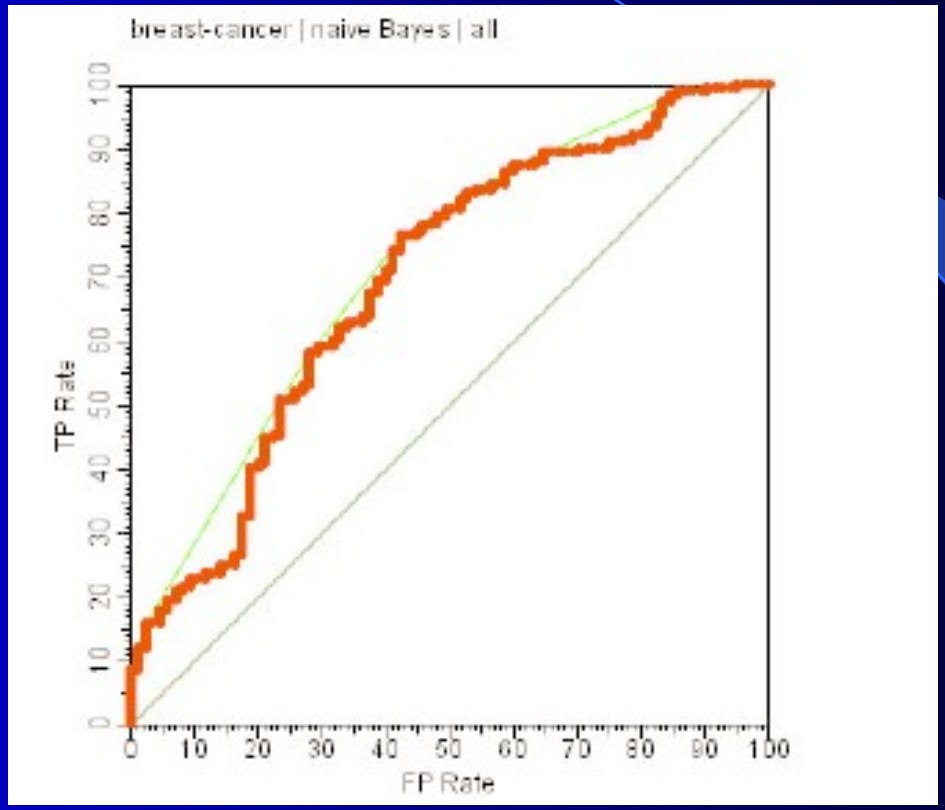

Reasonable separation between the classes, mostly convex.

# ROC for one Classifier



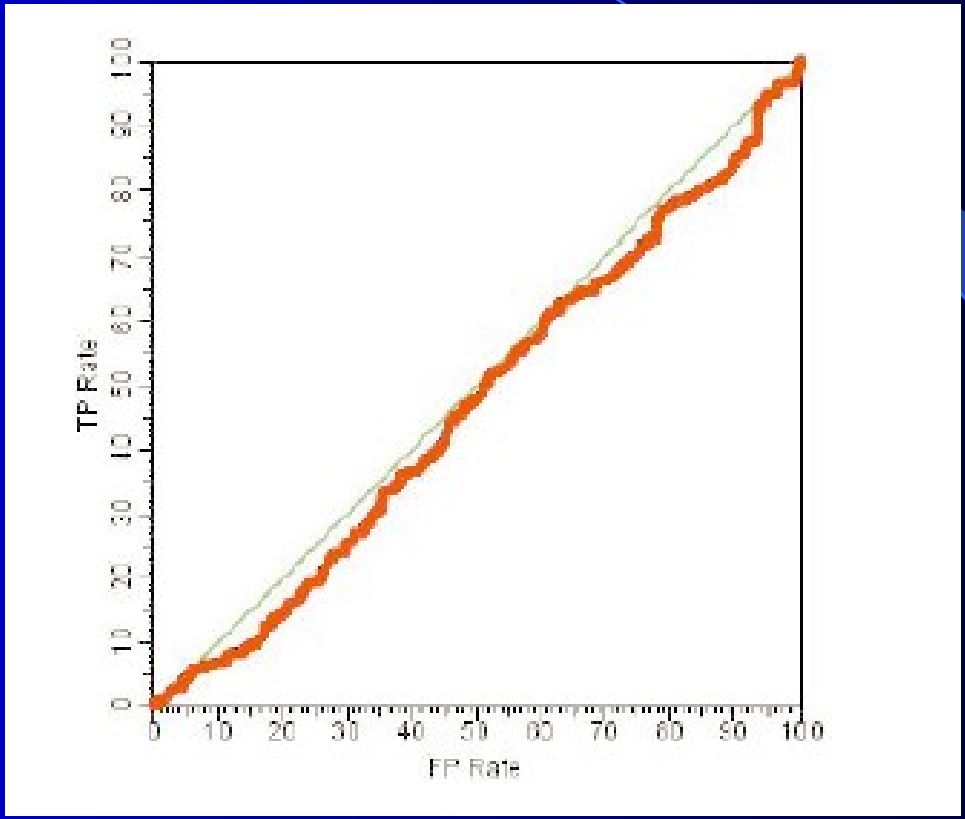

Fairly poor separation between the classes, mostly convex.

# ROC for one Classifier




Poor separation between the classes, large and small concavities.

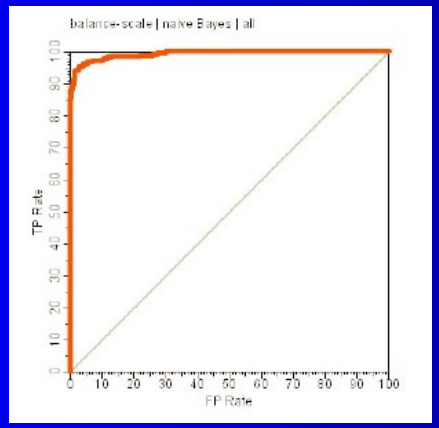
# ROC for one Classifier

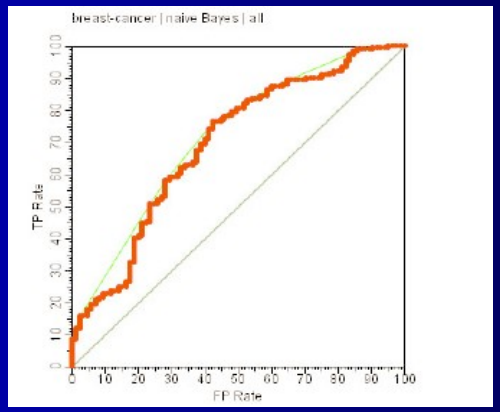


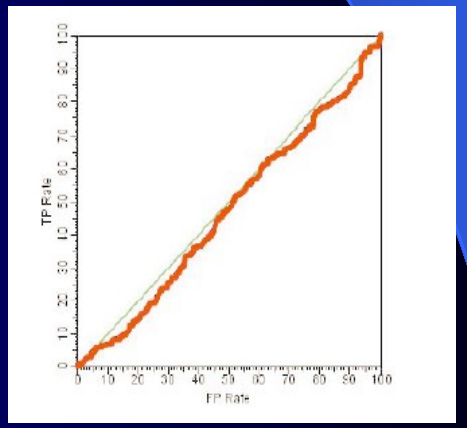

Random performance.

# The AUC Metric

- The area under ROC curve (AUC) assesses the ranking in terms of separation of the classes.
- AUC estimates that randomly chosen positive instance will be ranked before randomly chosen negative instances.

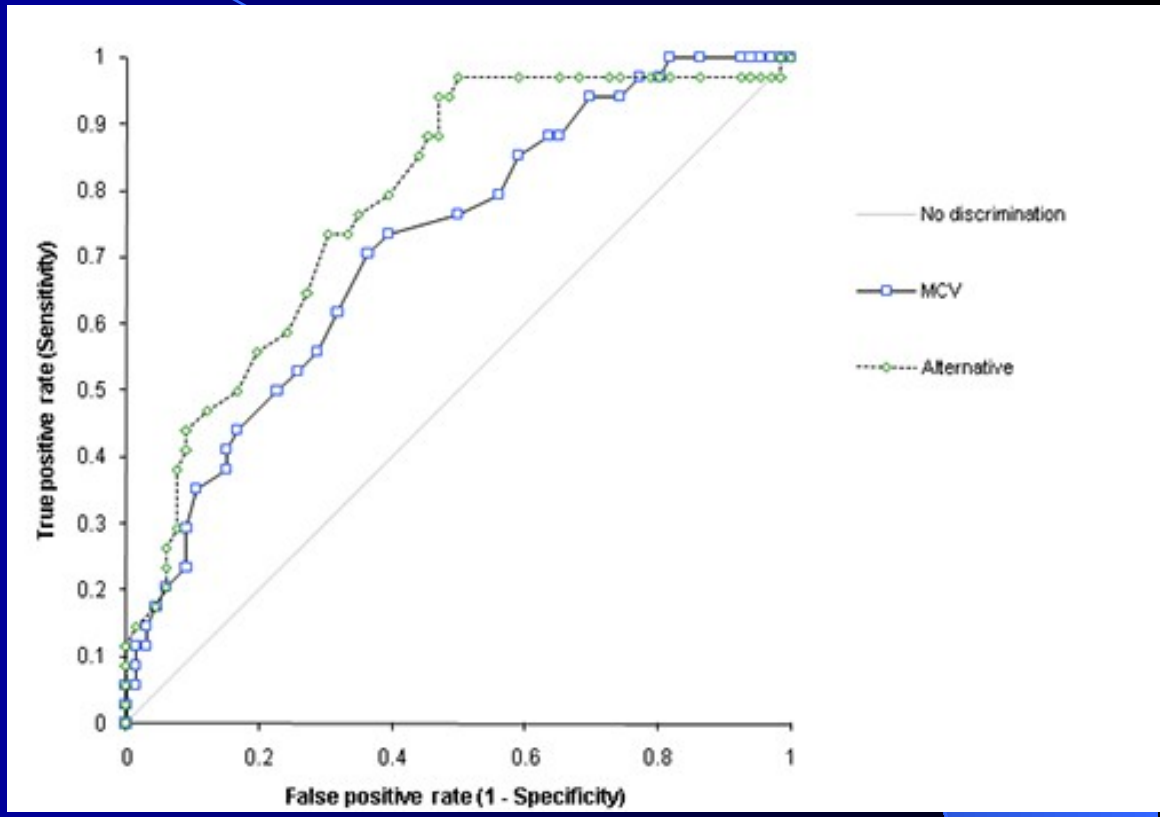


# Comparing Models

- Highest AUC wins
- But pay attention to 'Occam's Razor'
  - 'the best theory is the smallest one that describes all the facts'
  - Also known as the 'parsimony principle'
  - If two models are similar, pick the simpler one



# Training/Testing Alternatives

- Four methods that we commonly use:
  - Training set method
  - Static split test set
  - Random split test set CV
  - *N*-fold cross-validation
  - The last two are the more accurate approaches

# Training Set Method

- Procedure
  - Build model from the training set
  - Compute accuracy on the same training set
- Simple but least reliable estimate of future performance on unseen data (a rote learner could score 100%!)
- Not used as a performance metric but it is often important information in understanding how a machine learning model learns
- This is information which you will often report in your labs and then compare it with how the learner does on a better method

# Static Training/Test Set

- Static Split Approach
  - The data owner makes available to the machine learner two distinct datasets:
    - One is used for learning/training (i.e., inducing a model), and
    - One is used exclusively for testing
- Note that this gives you a way to do repeatable tests
- Can be used for challenges (e.g. to see how everyone does on one particular unseen set, method we use for helping grade your labs.)
- Be careful not to overfit the Test Set ("Gold Standard")

# Cross-Validation (CV)

- Cross-Validation (CV) – Cross-validation is a resampling method that uses different portions of the data to test and train a model on different iterations
- We then average the results of these iterations
- With CV we avoid having data just used for either training or test, and give all data a chance to be part of each, thus getting more accurate results

# Random Training/Test Set Approach

- Random Split CV Approach (aka holdout method)
  - The data owner makes available to the machine learner a single dataset
  - The machine learner splits the dataset into a training and a test set, such that:
    - Instances are randomly assigned to either set
    - The distribution of instances (with respect to the target class) is hopefully similar in both sets due to randomizing the data before the split
      - Stratification is an option to ensure proper distribution
    - Typically 60% to 90% of instances is used for training and the remainder for testing – the more data there is the more that can be used for training and still get statistically significant test predictions
  - Useful quick estimate for computationally intensive learners
  - Not statistically optimal (high variance, unless lots of data)
    - Could get a lucky or unlucky test set
  - Best to do multiple training runs with different random splits. Train and test *m* different splits and then average the accuracy over the *m* runs to get a more statistically accurate prediction of generalization accuracy.

# *N*-fold Cross-validation

- Use all the data for both training and testing
  - Statistically more reliable
  - All data can be used which is good, especially for small data sets
- Procedure
  - Partition the randomized dataset (call it $D$) into $N$ equally-sized subsets $S_1, \ldots, S_N$
  - For $k = 1$ to $N$
    - Let $M_k$ be the model induced from $D - S_k$
    - Let $a_k$ be the accuracy of $M_k$ on the instances of the test fold $S_k$
  - Return $(a_1+a_2+\ldots+a_N)/N$

# *N*-fold Cross-validation (cont.)

- The larger *N* is, the smaller the variance in the final result
- The limit case where $N = |D|$ is known as *leave-one-out CV* and provides the most reliable estimate. However, it is typically only practical for small instance sets
- Commonly, a value of $N$=10 is considered a reasonable compromise between time complexity and reliability
- Still must chose an actual model to use during execution – how?

# *N*-fold Cross-validation (cont.)

- The larger *N* is, the smaller the variance in the final result
- The limit case where $N = |D|$ is known as *leave-one-out CV* and provides the most reliable estimate. However, it is typically only practical for small instance sets
- Commonly, a value of $N$=10 is considered a reasonable compromise between time complexity and reliability
- Still must chose an actual model to use during execution - how?
  - Could select the one model that was best on its fold?
  - All data! With any of the approaches
- Note that N-fold CV is just a better way to estimate how well we will do on novel data, rather than a way to do *model selection*

# Machine Learning Tools

- Lots of new Machine Learning Tools
  - Weka was the first main site with lots of ready to run models
  - Scikit-learn now very popular
  - Languages:
    - Python with NumPy, matplotlib, pandas, other libraries
    - R (good statistical packages), but with growing Python libraries…
  - Deep Learning Neural Network frameworks – GPU capabilities
    - Tensorflow - Google
    - PyTorch – Multiple developers (Facebook, twitter, Nvidia...) - Python
    - Others: Caffe2 (Facebook), Keras, Theano, CNTK (Microsoft)
  - Data Mining Business packages – Visualization, Expensive
- Great for experimenting and applying to real problems
- But important to "get under the hood" and not just be black box ML users

# Doing Your Labs

- Will use scikit-learn in individual labs
  - Whatever you want in group project
- Program in Python in Jupyter notebooks
  - NumPy library – Great with arrays, etc.
- Recommended tools and libraries
  - Colab – Google IDE for Python and Jupyter notebooks
  - Pandas – Data Frames and tools are very convenient
  - MatplotLib
  - Scikit-Learn

# scikit-learn (SK)

- One of the most used and powerful machine learning toolkits out there
- Lots of implemented models and tools to use for machine learning applications
  - Sometimes missing some things we would like, but it is continually evolving
  - Source is available, and you can always override methods with your own, etc.
- Basically a Python Library to call from your Python code
- Familiarize yourself with the scikit-learn website as you will be using it for all labs

# Perceptron Project

- Content Section of LS (Learning Suite) for project specifications
  - Review carefully the introductory part regarding all projects
- For each project carefully read the specifications for the lab in the Jupyter notebook on GitHub
- You can just copy the Perceptron notebook from the GitHub site to your computer and then add your work in the code and text boxes