Cse352

ARTIFICIAL INTELLIGENCE

Testing and Building a Classifier

(Review - Long Lecture)

Professor Anita Wasilewska
Computer Science Department
Stony Brook University

Overview

- **Introduction**
- **Basic Concept** on **training** and **testing**
- **Main Methods** of **predictive accuracy** evaluations
- **Building** a **Classifier**

Predictive Accuracy Evaluation

The **main methods** of **predictive accuracy** evaluations are:

- **Resubstitution** (N ; N)
- **Holdout** (2N/3 ; N/3)
- **k-fold cross-validation** (N- N/k ; N/k)
- **Leave-one-out** (N-1 ; 1)

where N is the number of records (instances) in the dataset

Predictive Accuracy

- **REMEMBER:** we must know the classification (class attribute values) of **all instances** (records) used in the test procedure
- **Basic Concepts**

Success: instance (record) **class** is classified **correctly**

Error: instance **class** is classified **incorrectly**

Error rate: a **percentage of errors** made over the **whole set** of instances (records) used for **testing**

Predictive Accuracy: a percentage of **well classified** data in the testing data set.

Correctly and Not Correctly Classified

- A **test** data **record is correctly classified** if and only if the following conditions hold:

(1) we **can classify** the record, i.e there is **a pattern** or **a rule** such that its LEFT side **matches** the record,

(2) **classification determined by the pattern** or the **rule** is **correct**, i.e. the RIGHT side of the rule **matches** the value of the record’s class attribute

OTHERWISE

- **the record is not correctly classified**

- **Words used:**
- not correctly = incorrectly = misclassified
- **Validation** data = **Test** data

Predictive Accuracy

- **Example:**

Testing Rules (testing record #1) = record #1.class - Succ
Testing Rules (testing record #2) not= record #2.class - Error
Testing Rules (testing record #3) = record #3.class - Succ
Testing Rules (testing record #4) = instance #4.class - Succ
Testing Rules (testing record #5) not= record #5.class - Error

Error rate:

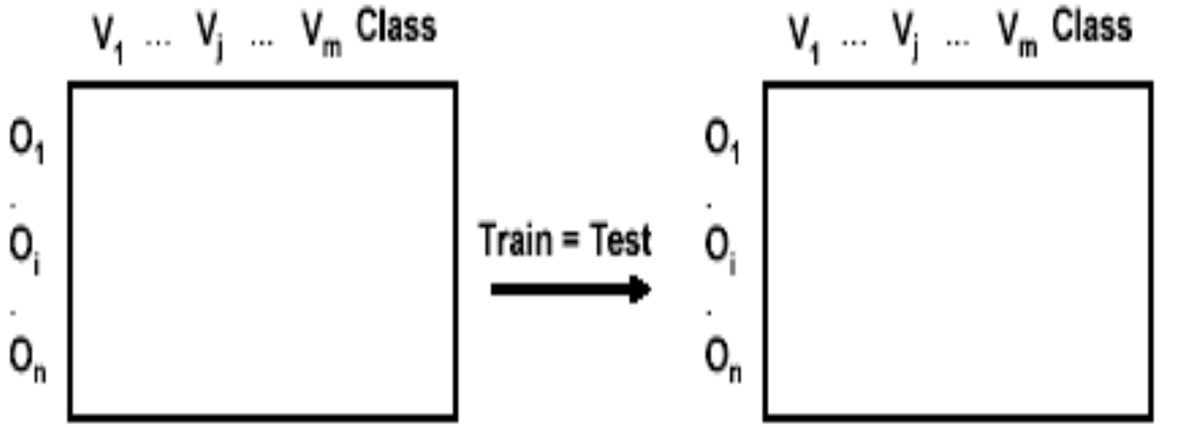
2 errors: #2 and #5

Error rate = 2/5=40%

Predictive Accuracy: 3/5 = 60%

Resubstitution (N ; N)

Testing the classification model by using the given data set (already used for „training“)



Re-substitution Error Rate

- **Re-substitution** error rate is obtained from **training data**
- **Training Data Error:** uncertainty of **the rules**
- **The error rate is not always 0%**, but usually (and hopefully) very low!
- **Re-substitution** error rate indicates only how **good (bad)** are our **results** (rules, patterns, NN) on the TRAINING data
- It expresses some **knowledge about the algorithm** used

Re-substitution Error Rate

- **Re-substitution** error rate is usually used as the **performance measure**:

The **training error rate** reflects **imprecision** of the training results

The lower training error rate the better

In the case of **rules** it is called **rules accuracy**

Predictive Accuracy

Predictive accuracy reflects how **good** are the training results with respect to the test data

The higher predictive accuracy **the better**

(N:N) re-substitution does not compute predictive accuracy

- Re-substitution error rate = **training data error rate**

Why not always 0%?

- The **error rate** on the training data is **not always 0%** because **algorithms** involve different (often statistical) parameters and measures that lead to uncertainties
- It is used for "**parameters tuning**"
- The error on the training data is NOT a good **indicator of performance** on **future data** since it does not measure any **not yet seen data**
- **Solution:**

  Split data into training and test set

Training and test set

- **Training** and **Test** data may differ in nature, but **must have** the same format

Example:

Given customer data from two different towns A and B.

We train the classifier with the data

from town A and we test it on data from town B, and vice-versa

Classification Learning Process

- It is important that the **test data** **is not used** in any way to create the training **rules ot other patterns**

- In fact, **classification process** operate in three stages:

  Stage 1: build the **basic patterns** structure
  -training

  Stage 2: optimize **parameter settings**; can use (N:N) re-substitution
  - **parameter tuning**

  Stage 3: use **test data** to compute
  predictive accuracy/error rate

Validation Data

- Proper **classification** process uses three sets of data:
- training data, validation data and test data
- validation data is **used** for parameter tuning
- validation data is not a test data
- validation data can be the training data, or a subset of training data
- The test data can not be used for parameter tuning!

Training and testing

- Generally, the **larger is** the training set, the **better is** the classifier

- **Larger** test data assures more **accurate** predictive accuracy, or error estimation

- **Remember:**
- the error rate of re-substitution(N;N)
  can tell us **ONLY** whether the algorithm used in
  training is good or not good
  or how good it is

Training and testing

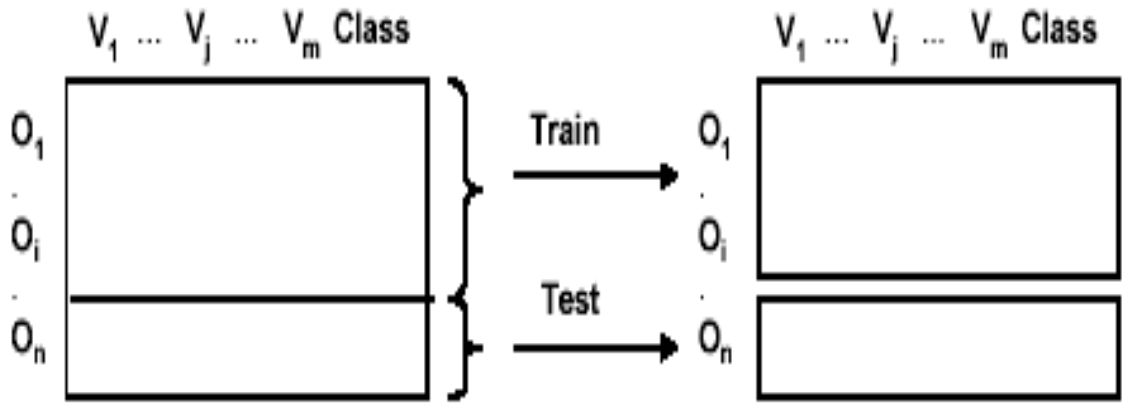
- **Holdout procedure**
  is a **method** of **splitting** original data into training and test data sets

- **Dilemma:**
- ideally **both training** and **test data** should be large!
- What to do if the **amount of data** is limited?
- How to **split** the data into training and test subsets?
- **Disjoint sets** - in the best way

Holdout

Train-and-Test (for large sample sizes) (> 1000))
dividing the given data set in

- a **training sample** for generating the classification model
- a **test sample** to test the model on independent objects with given classifications (randomly selected, 20-30% of the complete data set)


V_1 ... V_j ... V_m Class
O_1
O_i
O_n
Train
Test
V_1 ... V_j ... V_m Class
O_1
O_i
O_n

Holdout (N- N/3 ; N/3)

- The **holdout method** **reserves** a certain amount of data for testing and uses the **remainder** for training – so they are **disjoint!**

- **Usually**, one third **(N/3)** of data is used for **testing**, and **the rest** **(N -N/3) = (2N/3**) for **training**

- **The choice** of records for **train** and **test** data is **essential**

We usually perform a **cycle:**

Train-and-test; repeat

Repeated Holdout

- **Holdout** can be made more reliable by repeating the process with **different sub-samples** (subsets of data):

  1. In each iteration, a **certain portion** is randomly selected for training, the **rest of data** is used for testing

  2. The **error rates** or **predictive accuracy** on different **iterations** are **averaged** to yield an overall error rate, or overall **predictive accuracy**
- Repeated holdout still is not **optimal**: the different test sets **overlap**

k-fold cross-validation (N - N/k ; N/k)

- This is a **cross-validation** used to **prevent** the **overlap** of the test sets

- **First step:** split data into k **disjoint subsets**
- D1, … Dk, of **equal size**, called **folds**

- **Second step:** use each subset in turn for **testing**, the remainder for **training**

- **Training** and **testing** is performed **k times**

k-fold cross-validation predictive accuracy computation

- The predictive accuracy estimate is the overall number of correct classifications from all iterations, **divided** by the total number of records in the **initial data**

Stratified cross-validation

- In the **stratified cross-validation**
- the folds are stratified; i.e.
- the **class distribution** of the tuples
- (records) in **each fold** is
- approximately **the same as** in the
- initial data

10 folds cross-validation

- In general,
- 10-fold cross-validation or
  stratified 10-fold cross-validation
- is **recommended** and
- **widely used** even if computational power allows using more folds
- **Why 10?**

Extensive experiments have shown that this is the best choice to get an accurate estimate due to its relatively low bias and variance

So interesting!

Improved Repeated Holdout

- 10-fold cross-validation is an improvement over **repeated** holdout (N- N/10 ; N/10)

repeated 10 times where we use each subset in turn for **testing**, the remainder for **training** and predictive accuracy results are averaged

In the descriptive case we can adopt the union of rules as the **new set** of rules for the final **Classifier**

A particular form of cross-validation

- k-fold cross-validation: (N -N/k ; N/k)
- If k = N, what happens?
- We get   (N-1; 1)

  It is called "leave –one –out"

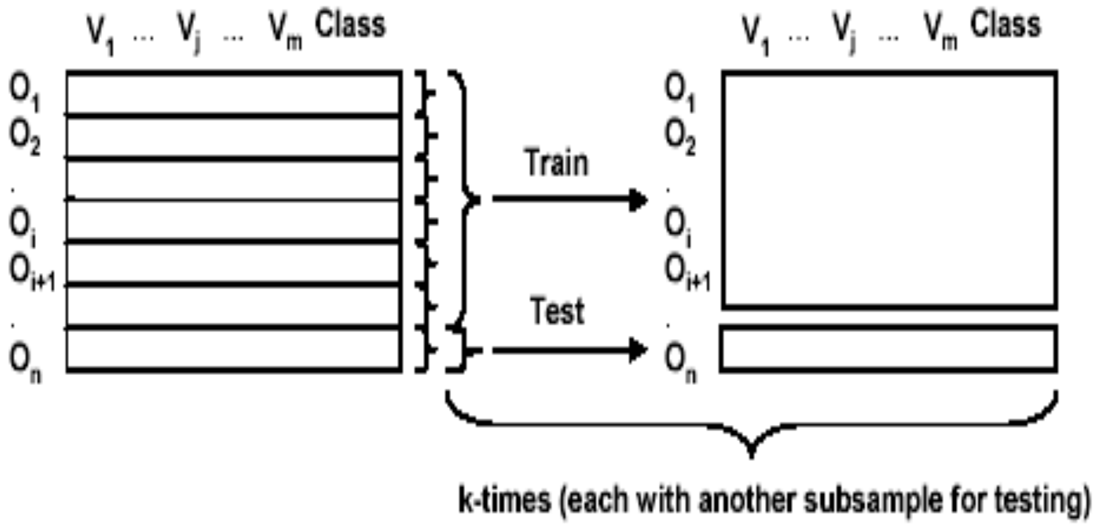
  Each sample (record) is used the same number of times for training and once for testing

Leave-one-out (N-1 ; 1)

Cross-Validation (for moderated sample sizes) → Sampling without replacemen

- Dividing the given data set into ***m*** **subsamples of equal size**
- Each subsample is tested by using a model generated from the remaining *(m-1)* subsamples

→ **Leave-One-Out**: m = Number of objects


V_1 ... V_j ... V_m Class
O_1
O_2
O_i
O_{i+1}
O_n
Train
Test
V_1 ... V_j ... V_m Class
O_1
O_2
O_i
O_{i+1}
O_n
k-times (each with another subsample for testing)

Leave-one-out (N-1 ; 1)

- **Leave-one-out** is a particular form of cross-validation

  We set number of folds to number of training instances, i.e. k= N

For N instances we build classifier (repeat the training - testing) n times

Leave-one-out Procedure

- Let C(i) be the classifier (rules, patterns) built on all data **except** record x_i
- Evaluate C(i) on x_i
- Determine if it is **correct** or in **error**
- Repeat for all i=1,2,…,n
- The **total error** is the **proportion** of all the incorrectly classified x_i

- The final CLASSIFIER set of rules (patterns) can be a union of all rules obtained in the process

Leave-one-out (N-1 ; 1)

- Makes the **best** use of the data
- Involves no random sub-sampling
- Stratification is not possible
- **Computationally** expensive
- MOST commonly used

Building a Classifier

- Book Edition 2, chapter 6, sections 6.12- 6,16
- Book Edition 3, chapter 8, sections 8.5 -8.6

Building a Classifier

- **Stage 1:** build the classification **patterns** structure-**training**
- We call them a **learned classifier**
- **Stage 2:** optimize parameter settings; can use (N:N) re-substitution- **parameter tuning**
- **Stage 3:** use **test data** to compute – predictive accuracy/error rate – **testing**
- **Stage 4:** consolidate Stages 1-3 to build a **Classifier** as a **final product**

Model Evaluation and Selection (book slide)

- Evaluation metrics:
- How can we measure (predictive) accuracy?
- Other metrics to consider?
- Use **validation test set** of class-labeled tuples instead of training set when assessing accuracy
- Methods for estimating a classifier's accuracy:
  - Holdout method, random subsampling
  - Cross-validation
  - Bootstrap

Classifier, Model Terminology

- The book uses the words "classifier" and "model" interchangeably
- Sometimes "classifier" means Stage 1 basic classifier model (rules, patterns) ready for **testing**
- Sometimes "classifiers" means classifiers models (rules, patterns) obtained **by training - testing** methods (like k-fold cross validation, repeated holdout, etc..). i.e. are the results of Stages 1- 3

Classifier, Model Terminology

- When the book talks about **comparison of classifiers**, "classifier" means comparison of classifiers models (rules, patterns) obtained by **train- test methods** i.e. means comparison results of Stages 1- 3
- These **comparison methods** or other methods are called "model selection"
- Their goal is to **choose** the best one to be
- **THE CLASSIFIER**-
- the final product that would the best classify unknown records

Classifier, Model Terminilogy

- In some cases the term “learned models”
- or “base classifiers” are used for results of
- Stages 1-3

- It happens when the method is presented how to **combine** them in a way that would the best to return a class prediction for unknown records, i.e. to **build the final**
- **CLASSIFIER**

Metrics for Evaluating Classifier Performance

- The predictive accuracy is one of basic performance measures of a classifier (model) learned in Stages 1-3 when applied to predict the class label of unknown records

- Before we discuss other measures (metrics)
- We introduce some new notions

Positive, Negative

- Given classification data with n >= 2 classes

- **Positive tuples** - tuples (record) belonging to the MAIN class of interest
- **Negative tuples** - all other tuples

- This is called **Contrast Classification**
- We **contrast** one MAIN class of interest with all other classes

Classifier Evaluation Metrics

- Consider a case of n=2 classes
- Assume that the **test data** has N records
- We use the following terms that are “building blocks” used in the learned classifier (Stage 1) evaluation metrics

- **True Positives** (TP):

These are positive **test** tuples that were **correctly** labeled by the learned classifier

- We denote by TP the **number** of true positives

Classifier Evaluation Metrics

- **True Negatives** (TN):

These are negative **test** tuples that were **correctly** labeled by the learned classifier

- We denote by TN the **number** of true negatives

- **False Positives** (FP):

These are negative **test** tuples that were **incorrectly** labeled as positive by the learned classifier

- We denote by FP the **number** of false positives

Classifier Evaluation Metrics

- **False Negatives** (FN):

These are positive **test** tuples that were **incorrectly** labeled as positive by the learned classifier

- We denote by FN the **number** of false negatives

- These terms are summarized in the following

  Confusion Matrix

Classifier Evaluation Metrics: Confusion Matrix

Confusion Matrix:

Actual class\Predicted class	C_1	$\neg C_1$
C_1	**True Positives (TP)**	**False Negatives (FN)**
$\neg C_1$	**False Positives (FP)**	**True Negatives (TN)**

Example of Confusion Matrix:

Actual class\Predicted class	buy_computer = yes	buy_computer = no	Total
buy_computer = yes	**6954**	**46**	7000
buy_computer = no	**412**	**2588**	3000
Total	7366	2634	10000

Classifier Evaluation Metrics: Confusion Matrix

- Given *m* classes
- An entry, ***$CM_{i,j}$*** in a **confusion matrix**
- indicates # of tuples in class *i* that were
- labeled by the classifier as class *j*

- May have extra rows/columns to provide totals

Classifier Evaluation Metrics: Accuracy, Error Rate

A\P	C	¬C	
C	**TP**	**FN**	**P**
¬C	**FP**	**TN**	**N**
	P'	**N'**	**All**

- **Classifier Accuracy**, or recognition rate: percentage of test set tuples that are correctly classified

  Accuracy = (TP + TN)/All

- **Error rate:** *1 – accuracy*, or

  Error rate = (FP + FN)/All

Classifier Evaluation Metrics: Sensitivity and Specificity

A\P	C	¬C	
C	**TP**	**FN**	**P**
¬C	**FP**	**TN**	**N**
	P'	**N'**	**All**

- **Class Imbalance Problem**:
  - One class may be *rare*, e.g. fraud, or HIV-positive
  - Significant *majority of the negative class* and minority of the positive class
  - **Sensitivity**: True Positive recognition rate
    - **Sensitivity = TP/P**
  - **Specificity**: True Negative recognition rate
    - **Specificity = TN/N**

Classifier Evaluation Metrics: Precision and Recall

- **Precision:** exactness – what % of tuples that the classifier labeled as positive are actually positive

$$precision = \frac{TP}{TP + FP}$$

$$recall = \frac{TP}{TP + FN}$$

- **Recall:** completeness –
- what % of positive tuples did the classifier label as positive?

- **Perfect score is 1.0**
- **Inverse** relationship between precision and recall

Classifier Evaluation Metrics: F-measures

- ***F* measure** (***F*$_1$** or ***F*-score**): harmonic mean of precision and recall,

$$precision = \frac{TP}{TP + FP}$$

$$recall = \frac{TP}{TP + FN}$$

- ***F*$_ß$**: weighted measure of precision and recall
  - assigns ß times
  - as much
  - weight to recall as to precision

$$F = \frac{2 \times precision \times recall}{precision + recall}$$

$$F_\beta = \frac{(1 + \beta^2) \times precision \times recall}{\beta^2 \times precision + recall}$$

Classifier Evaluation Metrics: Example

Actual Class\Predicted class	cancer = yes	cancer = no	Total	Recognition(%)
cancer = yes	**90**	**210**	300	30.00 (*sensitivity*
cancer = no	**140**	**9560**	9700	98.56 (*specificity*)
Total	230	9770	10000	96.40 (*accuracy*)

– ***Precision*** = 90/230 = 39.13% *Recall* = 90/300 = 30.00%

Evaluating Classifier Accuracy (Predictive Accuracy)

- **Holdout method**

  Given data is randomly partitioned into two independent sets

  - **Training** set (e.g., 2/3) for model construction
  - **Test** set (e.g., 1/3) for accuracy estimation

  Random sampling: a variation of holdout

  - Repeat holdout k times, accuracy = avg. of the accuracies obtained

Evaluating Classifier Accuracy (Predictive Accuracy)

- ***k*-fold Cross-validation** (k = 10 is most popular)
  - Randomly partition the data into *k mutually exclusive* subsets, each approximately equal size
  - At *i*-th iteration, use D_i as **test set** and others as **training set**

  - **Leave-one-out**: *k* folds where *k* = # of tuples, for small sized data

  - ***Stratified cross-validation***: folds are stratified so that **class distribution** in each fold is approximately the same as that in the **initial data**

Evaluating Classifier Accuracy: Bootstrap

.632 boostrap

A data set with *d* **tuples** is **sampled** *d* times, with **replacement,** resulting in a **training set** of *d* samples

The data tuples that **did not** make it into the **training set** end up forming the **test set**

About **63.2%** of the **original data** end up in the **bootstrap**, and the remaining **36.8%** form the **test set**

Repeat the sampling procedure *k* times, overall accuracy of the model:

$$Acc(M) = \frac{1}{k}\sum_{i=1}^{k}(0.632 \times Acc(M_i)_{test\_set} + 0.368 \times Acc(M_i)_{train\_set})$$

Evaluating Classifier Accuracy: Bootstrap

- **Bootstrap**

  samples the given **training tuples** uniformly *with replacement*

  i.e., each time a **tuple** is selected, it is equally likely to be selected again and re-added to the training **set**

- There are several bootstrap methods, and a common one is **.632 boostrap**

Evaluating Classifier Accuracy: Bootstrap

- **.632 boostrap**

  A data set with *d* **tuples** is **sampled** *d* times,

  with **replacement**

  Resulting is a **training set** of *d* samples

  The data tuples that **did not** make it into the

  training set end up forming the **test set**

  About **63.2%** of the **original data** end up in the **bootstrap**, and the remaining **36.8%** form the **test set** (since $(1 - 1/d)^d \approx e^{-1} = 0.368$)

Compare Learned Models M_1 vs. M_2

- **Suppose** we have **learned** 2 classifiers, M_1 and M_2
- **Which one is better?**
- Use 10-fold cross-validation to obtain $\overline{err}(M_1)$
- and $\overline{err}(M_2)$
- These mean error rates are just ***estimates*** of error on the true population of *future* data cases
- Want to **choose** one for the final **Classifier**

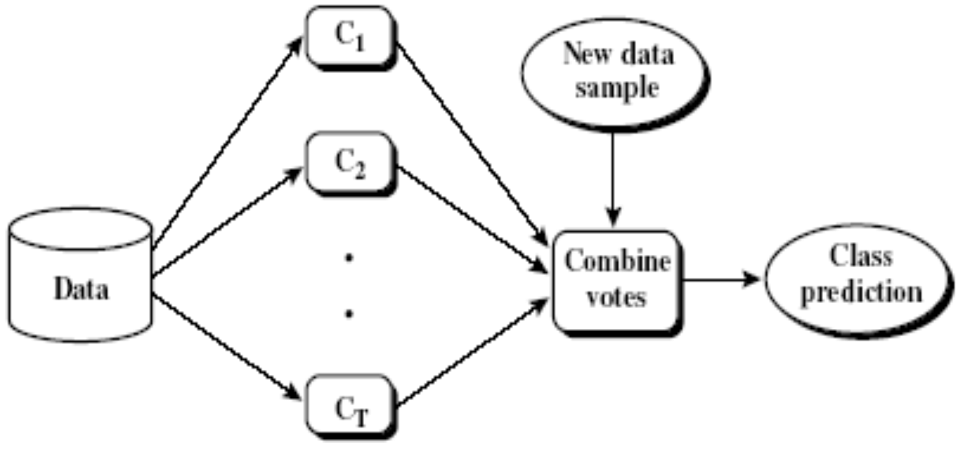
Choosing Models M_1 vs. M_2

- What if the **difference** between the 2 error rates is just **attributed** to *chance*?
- **We use t-test** (or **Student's t-test**)
- **Null Hypothesis**: M_1 & M_2 mean error rates **are the same**
- If we can **reject** null hypothesis, then
  - we conclude that the difference between M_1 & M_2 is **statistically significant**

We chose model with **lower** error rate

- **Otherwise**, **conclude** that any difference is **chance**

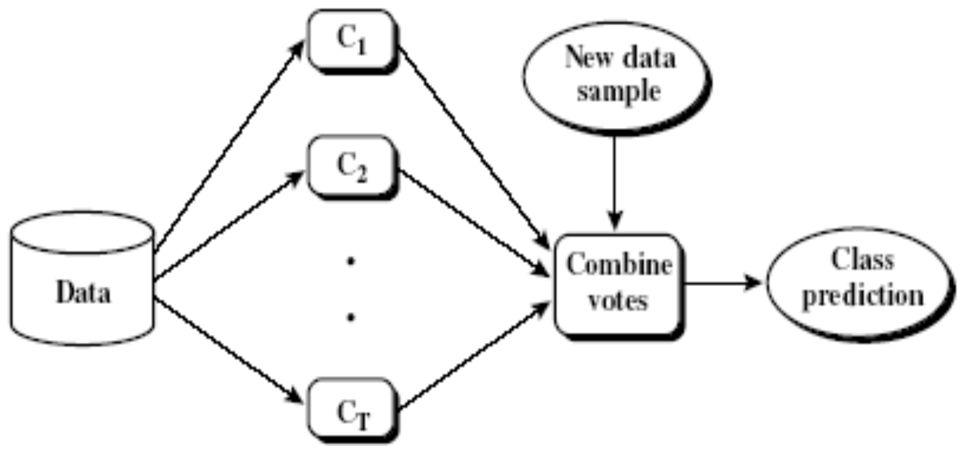
Ensemble Methods



- Ensemble methods
  - Use a combination of models to increase accuracy
  - Combine a series of k learned models

  M_1, M_2, …, M_k, with the aim of creating an improved model M* as a CLASSIFIER

Building the CLASSIFIER



- Popular ensemble methods of **building the CLASSIFIER**
  - Bagging: averaging the prediction over a collection of classifiers
  - Boosting: weighted vote with a collection of classifiers
  - Random Forest: *decision tree* classifier

Bagging: Boostrap Aggregation

- Analogy: Diagnosis based on multiple doctors' majority vote
- **Training**

  Given a set D of *d* tuples, at each iteration *i* a **training** set D_i of *d* tuples is **sampled** with **replacement** from D (i.e., bootstrap)

  – A classifier model M_i is learned for each **training** set D_i

Bagged Classifier

- Classifier: we build to classify an **unknown** sample **X**
- We proceed as follows
  - Each classifier model M_i returns its class prediction
  - The **bagged classifier** M^* counts the votes and **assigns** the **class** with the most votes to **X**
- Accuracy
  - Often significantly better than a **single classifier** derived from D

Boosting

- Analogy:
  Consult several doctors, based on a combination of weighted diagnoses—weight assigned based on the previous diagnosis accuracy

How boosting works?

Weights are assigned to each training tuple

A series of k classifiers is iteratively **learned**

After a classifier M_i is **learned**,
the **weights** are updated to allow the
subsequent classifier M_{i+1}
to pay **more** attention to the **training tuples**
that were **misclassified** by M_i

Boosting

- Boosting algorithm can be extended for numeric prediction

- Comparing with bagging:

- Boosting tends to have greater accuracy, but it also risks overfitting the model to misclassified data

Adaboost

Given a set D of *d* class-labeled tuples

$(\mathbf{X}_1, y_1), \ldots, (\mathbf{X}_d, y_d)$

Initially, all the **weights** of tuples are set the same (1/d)

- **Generate** k classifiers in k rounds
- At round i
  - Tuples from D are **sampled** (with replacement) to form a **training** set D_i of the same size

    Each tuple's **chance** of being selected is based on its **weight**
  - A classification model M_i is derived from D_i

    Its error rate is calculated using D_i as a **test** set
  - If a tuple is **misclassified**, its **weight** is increased, otherwise it is decreased

Adaboost

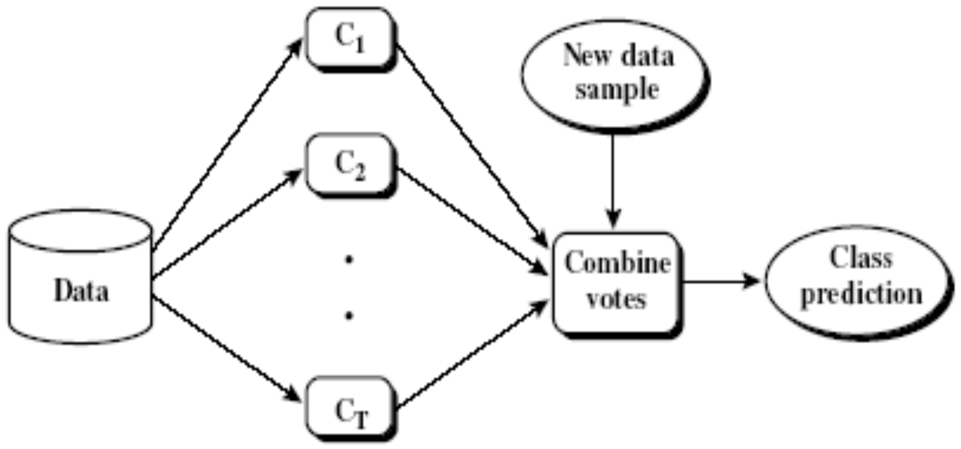
- **Error rate:**
- err($\mathbf{X}_j$) is the **misclassification** error of tuple $\mathbf{X}_j$
- Classifier model M_i **error rate** is the sum of the **weights** of the misclassified tuples:

$$error(M_i) = \sum_{j}^{d} w_j \times err(\mathbf{X_j})$$

The **weight** of classifier M_i' s **vote** is $\log \frac{1 - error(M_i)}{error(M_i)}$

The **final classifier M* combines the votes** of each individual classifier

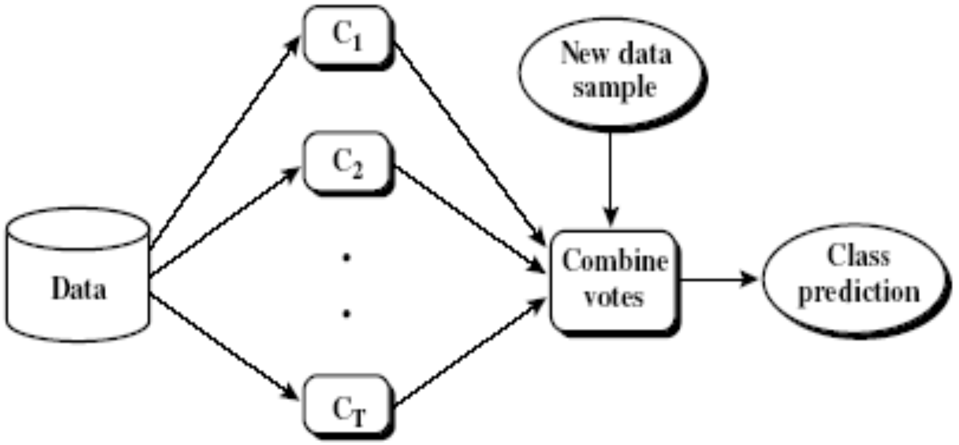
Ensemble Methods



- Ensemble methods
  - Use a combination of models to increase accuracy
  - Combine a series of k learned models

 M_1, M_2, …, M_k, with the aim of creating an improved model M* as a CLASSIFIER

Building the CLASSIFIER



- Popular ensemble methods of **building the CLASSIFIER**
  - Bagging: averaging the prediction over a collection of classifiers
  - Boosting: weighted vote with a collection of classifiers
  - Erandom Forest: *decision tree* classifier

Random Forest

- Random Forest:

  each classifier in the ensemble is a *decision tree* classifier

  It is **generated** using a random selection of attributes at each **node** of the tree to determine the **split**

  In **final classifier**, each tree **votes** and the most popular class is **returned**

Random Forest

- **Two Methods** to construct Random Forest:

Forest-RI (*random input selection*):

Randomly select, at each node, Forest attributes as candidates for the split at the **node**

The CART methodology is used to grow the trees to maximum size

Random Forest

Forest-RC (*random linear combinations*)*:*

Creates new Forest attributes (or features)

that are a linear combination of the

existing ensemble attributes

It reduces the correlation between individual

classifiers

- **Random Forest**
- **Insensitive** to the number of attributes selected for consideration at each **split**
- **Faster** than bagging or boosting

Classification of Class-Imbalanced Data Sets

- **Class-imbalance problem:**

  Rare positive example but **numerous** negative once

For example:

- medical diagnosis, fraud, oil-spill, fault, etc.

- Traditional **methods** assume a balanced distribution of **classes** and equal error **costs**
- This is **not suitable** for class-imbalanced data

Classification of Class-Imbalanced Data Sets

Typical methods for imbalance data in 2-class classification:

Oversampling:
re-sampling of data from positive class

Under-sampling:
randomly eliminate tuples from negative class

Classification of Class-Imbalanced Data Sets

Threshold-moving:
moves the decision threshold (t)
so that the **rare class** tuples are easier
to classify and there is less chance of
costly **false negative** errors

Ensemble techniques:
Ensemble multiple learned classifiers

- All are **difficult** for class imbalance problem on **multiclass** tasks

Book Summary

- Stratified k-fold cross-validation is a **recommended** method for accuracy estimation
- Bagging and boosting can be used to increase overall accuracy by learning and combining a series of individual models.
- Significance tests and ROC curves are useful for model selection - building a **final** CLASSIFIER