# Evaluation and Improvement of Classification Quality

# Outline

- Model Evaluation and Selection
- Techniques to Improve Classification Accuracy: Ensemble Methods
- Multiclass Classification and Weak Supervision

# Model Evaluation and Selection

# Model Evaluation and Selection

- How to evaluate the quality of a classifier
  - A typical measure: Accuracy
  - Other metrics to consider?
- How to assess the classification quality
  - Use (independent) **validation test set** instead of training set when assessing accuracy
- Methods for estimating a classifier’s accuracy
  - Holdout method
  - Cross-validation
  - Bootstrap
- Comparing classifiers using ROC Curves

# 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)** |

- In a confusion matrix with *m* classes, $CM_{i,j}$ indicates # of tuples in class *i* that were labeled by the classifier as class *j*
  - May have extra rows/columns to provide totals
- An 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: Accuracy, Error Rate, Sensitivity, and Specificity

| 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**

- **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
- Measures handle the class imbalance problem
  - **Sensitivity** (recall): True positive recognition rate
    - **Sensitivity = TP/P**
  - **Specificity**: True negative recognition rate
    - **Specificity = TN/N**

# Classifier Evaluation Metrics: Precision and Recall, and F-measures

- **Precision** (Exactness): What percentage of tuples labeled as positive is actually positive?

$$P = \text{Precision} = \frac{TP}{TP + FP}$$

- **Recall** (Completeness): What percentage of positive tuples are labeled as positive?

$$R = \text{Recall} = \frac{TP}{TP + FN}$$

    - Range: [0, 1]
  - The "inverse" relationship between precision & recall
- ***F* measure (or *F*-score):** Harmonic mean of precision and recall
  - In general, it is the weighted measure of precision & recall

$$F_\beta = \frac{1}{\alpha \cdot \frac{1}{P} + (1 - \alpha) \cdot \frac{1}{R}} = \frac{(\beta^2 + 1)PR}{\beta^2 P + R}$$

Assigning β times as much weight to recall as to precision

  - ***F1-measure (balanced F-measure)***
    - That is, when β = 1,

$$F_1 = \frac{2PR}{P + R}$$

# Classifier Evaluation Metrics: Example

- Use the same confusion matrix, calculate the measure just introduced

| 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 | 96.50 (*accuracy*) |

  - Sensitivity = TP/P = 90/300 = 30%
  - Specificity = TN/N = 9560/9700 = 98.56%
  - Accuracy = (TP + TN)/All = (90+9560)/10000 = 96.50%
  - Error rate = (FP + FN)/All = (140 + 210)/10000 = 3.50%
  - Precision = TP/(TP + FP) = 90/(90 + 140) = 90/230 = 39.13%
  - Recall = TP/ (TP + FN) = 90/(90 + 210) = 90/300 = 30.00%
  - F1 = 2 P × R /(P + R) = 2 × 39.13% × 30.00%/(39.13% + 30%) = 33.96%

# Classifier Evaluation: Holdout & Cross-Validation

- **Holdout method**
  - The given data set 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
  - Repeated random sub-sampling validation: A variation of holdout
    - Repeat holdout k times, accuracy = average of the accuracies obtained
- **Cross-validation** (*k*-fold, where 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

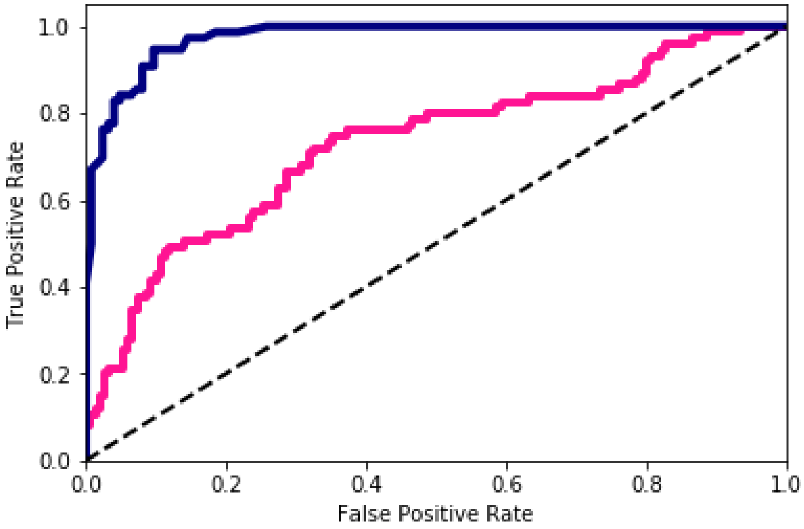
# Classifier Evaluation: Bootstrap

- **Bootstrap**
  - Works well with small data sets
  - Samples the given training tuples uniformly *with replacement*
    - Each time a tuple is selected, it is equally likely to be selected again and re-added to the training set
- Several bootstrap methods, and a common one is **.632 bootstrap**
  - 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 (since $(1 – 1/d)^d \approx e^{-1} = 0.368$)
  - Repeating the sampling procedure *k* times, the overall accuracy of the model:

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

# Model Selection: ROC Curves

- **ROC** (Receiver Operating Characteristics) curve: for visual comparison of classification models
- Originated from signal detection theory
- Shows the trade-off between the true positive rate and the false positive rate
- The area under the ROC curve (**AUC**: Area Under Curve) is a measure of the accuracy of the model
- Rank the test tuples in decreasing order: The one that is most likely to belong to the positive class appears at the top of the list
- The closer to the diagonal line (i.e., the closer the area is to 0.5), the less accurate is the model




- The vertical axis represents the true positive rate
- The horizontal axis represents the false positive rate
- The plot also shows a diagonal line
- A model with perfect accuracy will have an area of 1.0

# Issues Affecting Model Selection

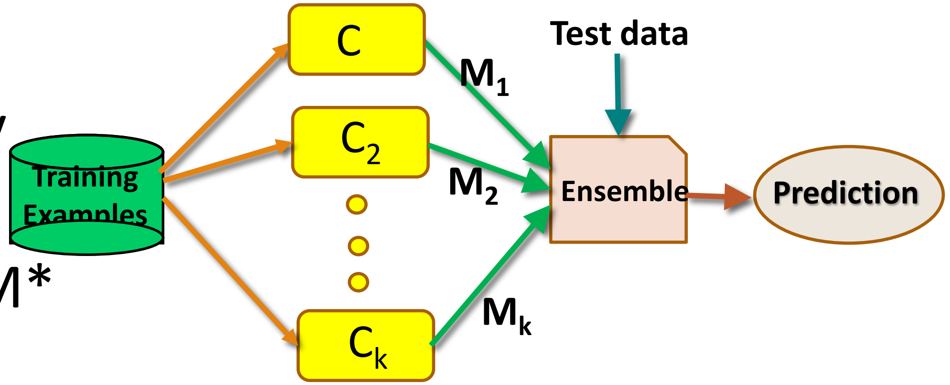
- **Accuracy**
  - Classifier accuracy: Predicting class label
- **Speed**
  - Time to construct the model (training time)
  - Time to use the model (classification/prediction time)
- **Robustness**: Handling noise and missing values
- **Scalability**: Efficiency in disk-resident databases
- **Interpretability**
  - Understanding and insight provided by the model
- Other measures
  - E.g., goodness of rules, such as decision tree size or compactness of classification rules

# Techniques to Improve Classification Accuracy: Ensemble Methods

# Outline

- Techniques to Improve Classification Accuracy: Ensemble Methods
- Bagging: Bootstrap Aggregation
- Random Forest: Basic Concepts and Methods
- Boosting and AdaBoost
- Classification of Class-Imbalanced Data Sets
- Classifying Data Streams with Skewed Distribution

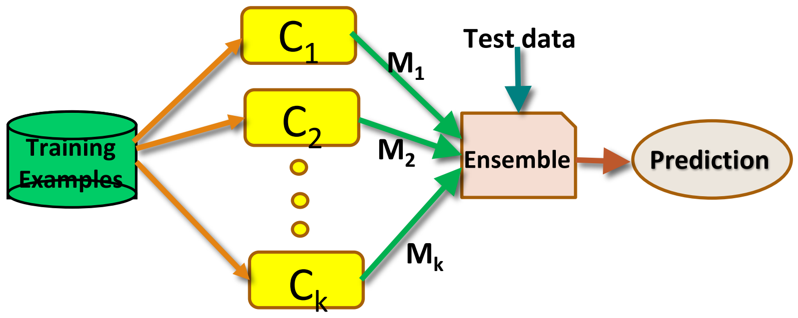
# Ensemble Methods: Increasing the Accuracy




- 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 $M^*$
- Popular ensemble methods
  - Bagging: Trains each model using a subset of the training set
    - Models learned independently, in parallel
    - Simple voting: Outcome is determined by the majority of the parallel models
  - Boosting: Trains each new model instance to emphasize the training instances that previous models misclassified (correcting the "errors" of previous model)
    - Models learned in order (a sequential ensemble)
    - Weighted voting: Outcome is determined by the majority - but the sequential models were built by assigning greater weights to misclassified instances of the previous models

# Bagging: Bootstrap Aggregation

- Analogy: Diagnosis based on multiple doctors' majority vote
- Training
  - Given a set D of 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 $D_i$

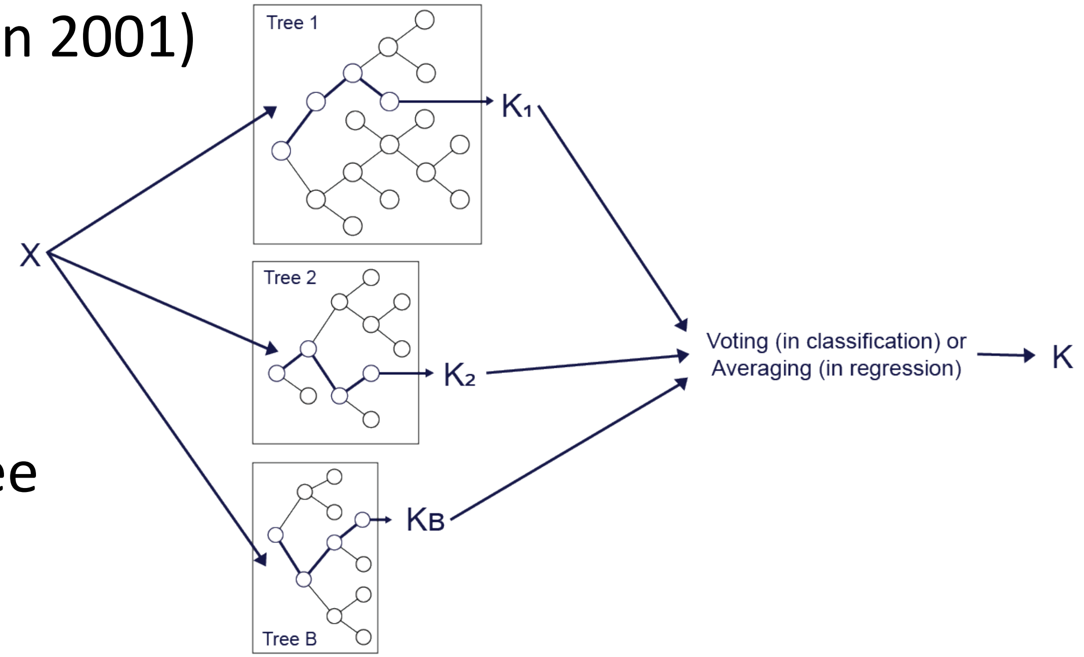



- Classification: Classify an unknown sample **X**
  - Each classifier $M_i$ returns its class prediction
  - The bagged classifier M* counts the votes and assigns the class with the most votes to **X**
- Prediction: It can be applied to the prediction of continuous values by taking the average value of each prediction for a given test tuple
- Accuracy: Improved accuracy in prediction
  - Often significantly better than a single classifier derived from D
  - For noise data: Not considerably worse, more robust

# Random Forest: Basic Concepts

- Random Forest (first proposed by L. Breiman in 2001)
  - A variation of bagging for *decision trees*
  - *Data bagging*
    - Use a subset of training data

      by sampling with replacement for each tree
  - *Feature bagging*
    - At each node use a random selection of attributes as candidates and split by the best attribute among them
  - Comparing with original bagging, the *random forests* method increases the diversity among generated trees
  - During classification, each tree votes and the most popular class is returned



# Methods for Constructing Random Forest

- Two Methods to construct Random Forest:
  - Forest-RI (*random input selection*)
    - Randomly select, at each node, *F* attributes as candidates for the split at the node
      - The CART methodology is used to grow the trees to maximum size
  - Forest-RC (*random linear combinations*)
    - Creates new attributes (or features) that are a linear combination of the existing attributes (reduces the correlation between individual classifiers)
- Comparable in accuracy to Adaboost, but more robust to errors and outliers
- Insensitive to the number of attributes selected for consideration at each split, and faster than typical bagging or boosting

# 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 $M_i$
  - The final **M* combines the votes** of each individual classifier, where the weight of each classifier's vote is a function of its accuracy
- 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 (Freund and Schapire, 1997)

- Given a set 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 (i.e., 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**
- Error rate: $err(\mathbf{X_j})$ is the misclassification error of tuple $\mathbf{X_j}$
- Classifier $M_i$ error rate is the sum of the weights of the misclassified tuples:

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

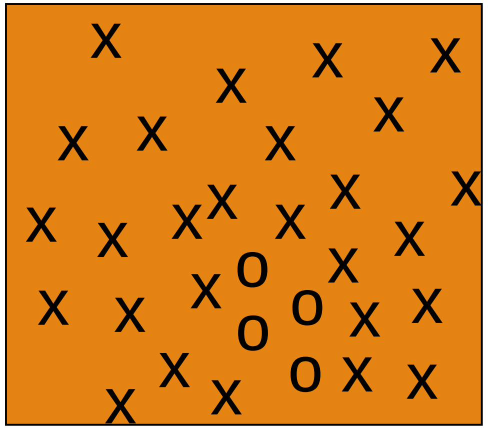
- The weight of classifier $M_i$'s vote is $\log \frac{1-error(M_i)}{error(M_i)}$



8

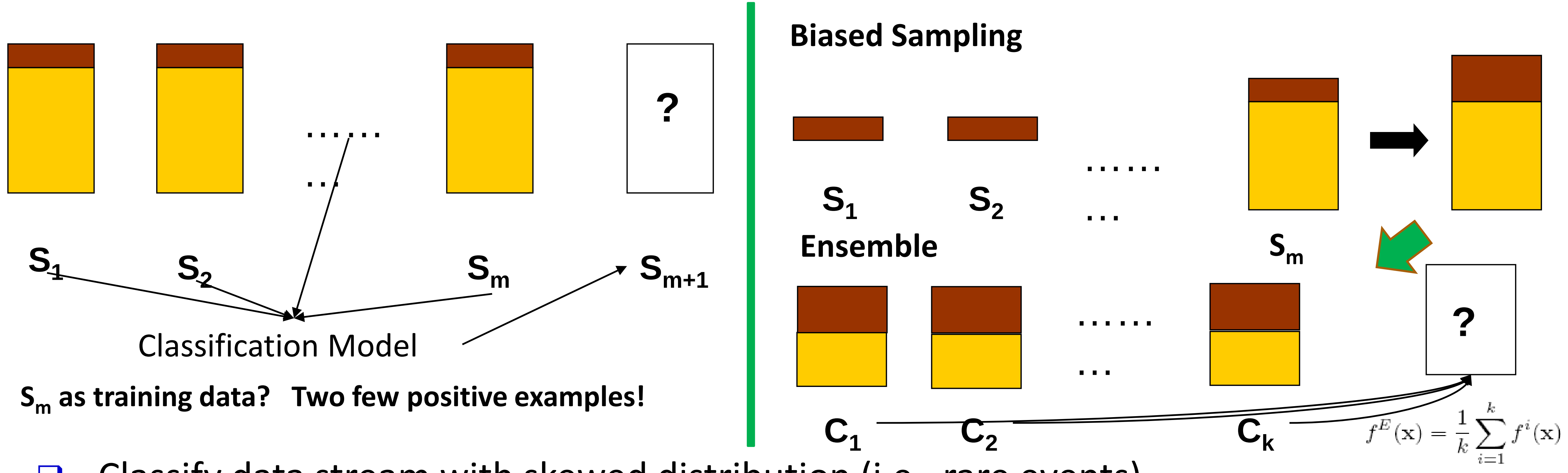
# Classification of Class-Imbalanced Data Sets

- Class-imbalance problem: Rare positive examples but numerous negative ones
  - E.g., medical diagnosis, fraud transaction, accident (oil-spill), and product fault
- Traditional methods assume a balanced distribution of classes and equal error costs: Not suitable for class-imbalanced data



- Typical methods on imbalanced data in two-class classification
  - **Oversampling**: Re-sampling of data from positive class
  - **Under-sampling**: Randomly eliminate tuples from negative class
  - **Threshold-moving**: Move the decision threshold, t, so that the rare class tuples are easier to classify, and hence, less chance of costly false negative errors
  - **Ensemble techniques**: Ensemble multiple classifiers introduced above
- Still difficult for class imbalance problem on multiclass tasks

# Classifying Data Streams with Skewed Distribution

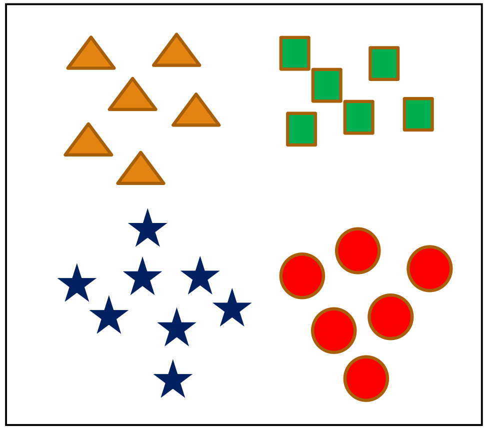



- Classify data stream with skewed distribution (i.e., rare events)
  - **Biased sampling:** Save only the positive examples in the streams
  - **Ensemble:** Partition negative examples of $S_m$ into k portions to build k classifiers
  - Effectively reduce classification errors on the minority class

(Gao, Fan, & Han, 2007)

# Multiclass Classification and Weak Supervision

# Multiclass Classification

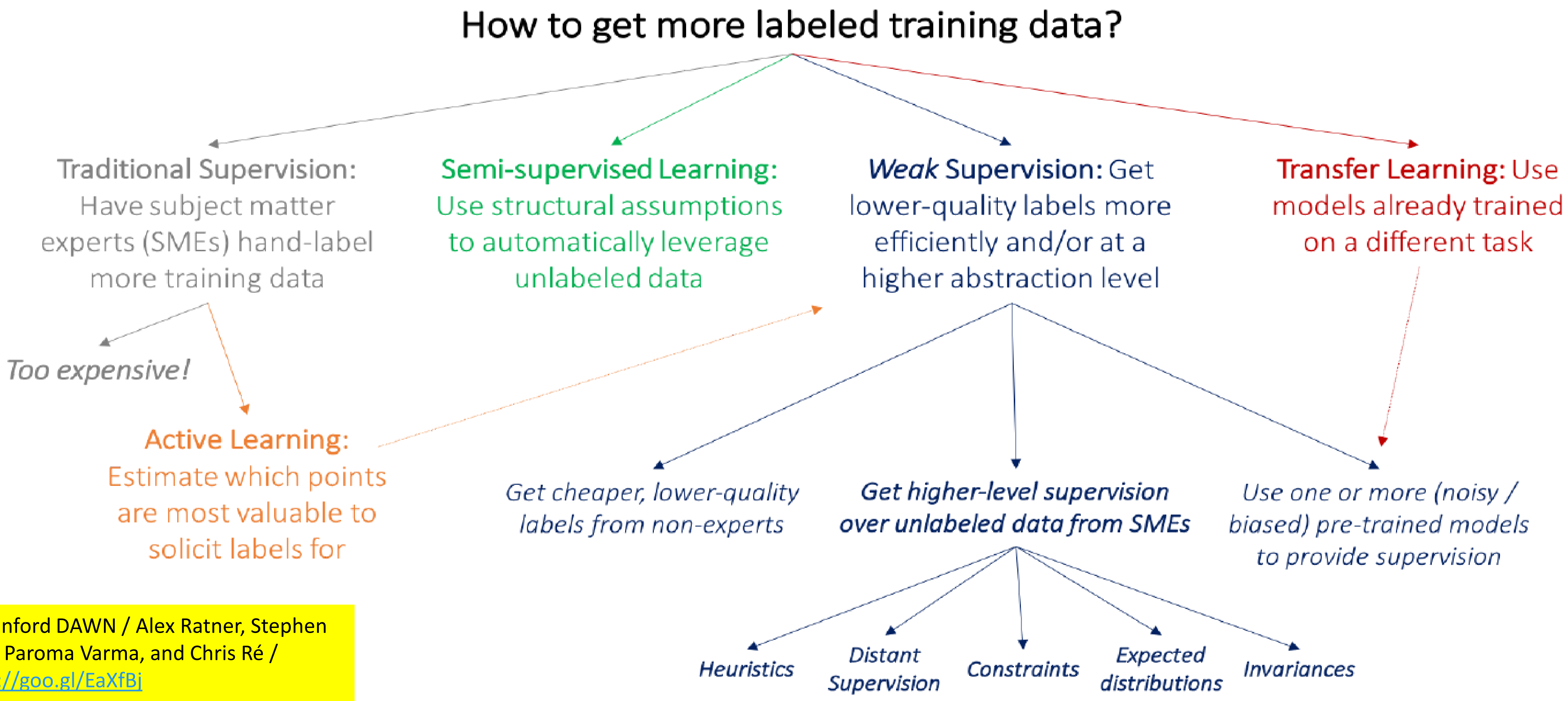


- Classification involving more than two classes (i.e., > 2 Classes)
- Methodology
  - Reducing the multi-class problem into multiple binary problems
- Method 1. **One-vs.-rest** (or **one-vs.-all**)
  - Given *m* classes, train *m* classifiers: One for each class
  - Classifier j: Treat tuples in class j as *positive* & **all the rest** as *negative*
  - To classify a tuple **X**, the set of classifiers vote as an ensemble
- Method 2. **One-vs.-one** (or **all-vs.-all**): Learn a classifier for each pair of classes
  - Given *m* classes, construct $m(m-1)/2$ binary classifiers
  - A classifier is trained using tuples of the two classes
  - To classify **X**, each classifier votes: **X** is assigned to the class with maximal vote
- Comparison: One-vs.-one tends to perform better than one-vs.-rest
- Many new algorithms have been developed to go beyond binary classifier method

# Weak Supervision: A New Programming Paradigm for Machine Learning

- Overcome the training data bottleneck
  - Leverage higher-level and/or noisier input from experts
- Exploring *weak label distributions* provided more cheaply and efficiently by
  - Higher-level, less precise supervision (e.g., heuristic rules, expected label distributions)
  - Cheaper, lower-quality supervision (e.g., crowdsourcing)
  - Existing resources (e.g., knowledge bases, pre-trained models)
- These weak label distributions could take many forms
  - Weak labels from crowd workers, output of heuristic rules, or the result of distant supervision (from KBs), or the output of other classifiers, etc.
  - Constraints and invariances (e.g., from physics, logic, or other experts)
  - Probability distributions (e.g., from weak or biased classifiers, user-provided labels, feature expectations, or measurements)

# Relationships Among Different Kinds of Supervisions

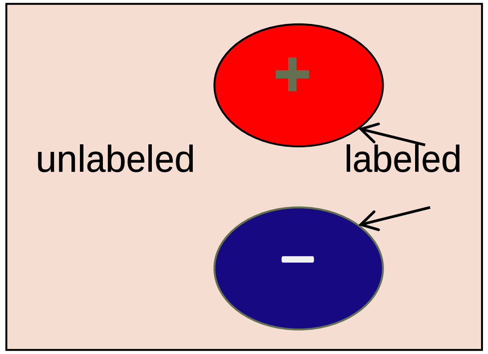




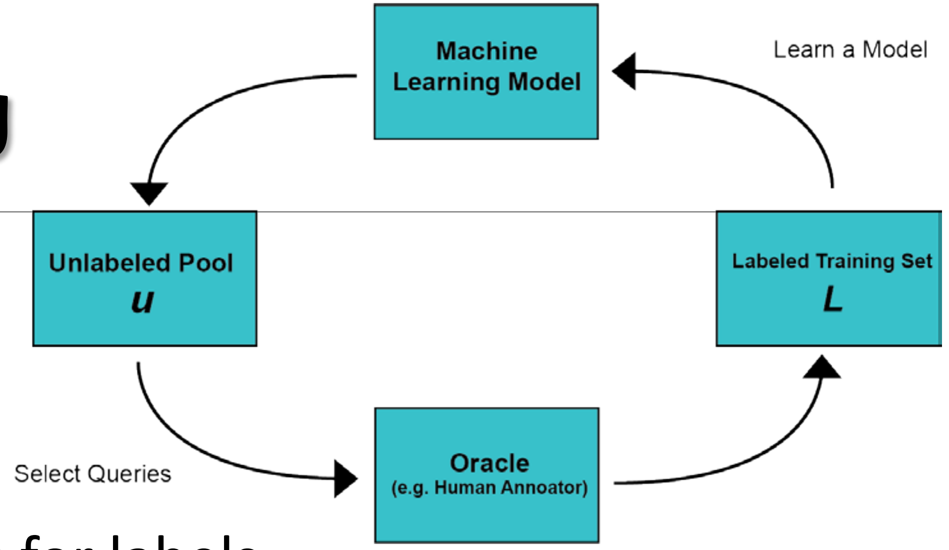

*Many areas of machine learning are motivated by the bottleneck of labeled training data, but are divided at a high-level by what information they leverage instead.*

# Semi-Supervised Classification

- Semi-supervised: Uses labeled and unlabeled data to build a classifier
- Self-training
  - Build a classifier using the labeled data
  - Use it to label the unlabeled data, and those with the most confident label prediction are added to the set of labeled data
  - Repeat the above process
  - Adv.: Easy to understand; Disadv.: May reinforce errors
- Co-training: Use two or more classifiers to teach each other
  - Each learner uses a mutually independent set of features of each tuple to train a good classifier, say $f_1$ and $f_2$
  - Then $f_1$ and $f_2$ are used to predict the class label for unlabeled data X
  - Teach each other: The tuple having the most confident prediction from $f_1$ is added to the set of labeled data for $f_2$ & vice versa
- Other methods include joint probability distribution of features and labels



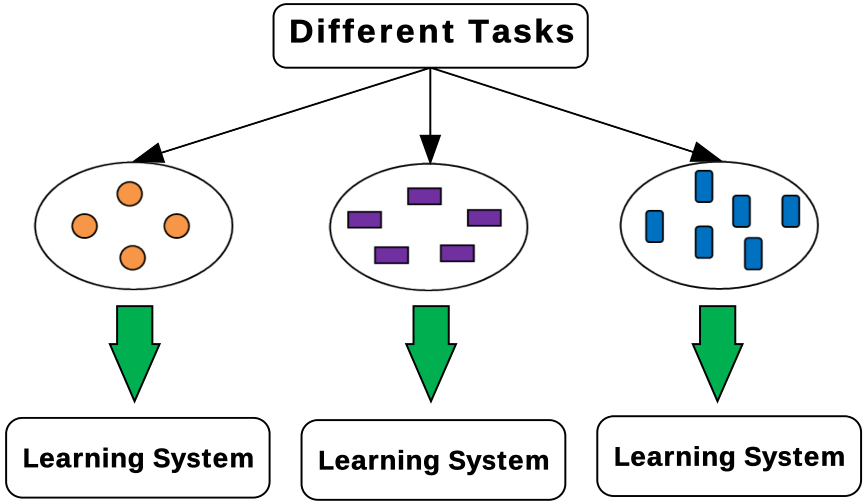
# Active Learning



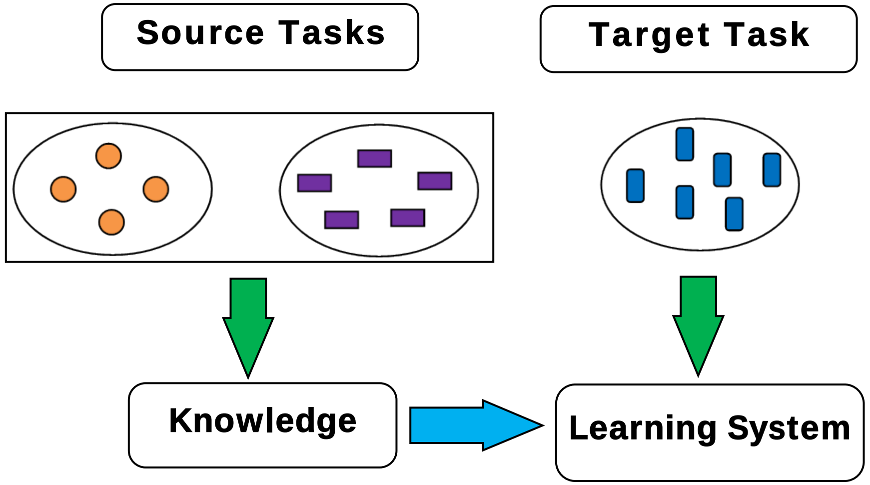

- A special case of semi-supervised learning
  - Unlabeled data: Abundant
  - Class labels are expensive to obtain
- Active learner: Interactively query teachers (oracle) for labels
- Pool-based approach: Uses a pool of unlabeled data
  - L: A small subset of D is labeled; U: A pool of unlabeled data in D
  - Use a query function to carefully select one or more tuples from U and request labels from an oracle (a human annotator)
  - The newly labeled samples are added to L, and learn a model
  - Goal: **Achieve high accuracy using as few labeled data as possible**
- Evaluated using *learning curves*: Accuracy as a function of the number of instances queried (# of tuples to be queried should be small)
- A lot of algorithms have been developed for active learning

# Transfer Learning: Conceptual Framework

- Transfer learning: Extract knowledge from one or more source tasks (e.g., recognizing cars) and apply the knowledge to a target task (e.g., recognizing trucks)
- Traditional learning: Build a new classifier for each new task
- Transfer learning: Build new classifier by applying existing knowledge learned from source tasks
- Many algorithms are developed, applied to text classification, spam filtering, etc.



**Traditional Learning Framework**



**Transfer Learning Framework**


7

# Summary

# Summary

- Model Evaluation and Selection
- Techniques to Improve Classification Accuracy: Ensemble Methods
- Multiclass Classification and Weak Supervision

# Recommended Readings


- Breiman, L. (1996). Bagging predictors. *Machine Learning, 24*(2), 123-140. Retrieved from https://link.springer.com/article/10.1023/A:1018054314350
- Efron, B. & Tibshirani, R. (1993). *An introduction to the bootstrap*. London, UK: Chapman & Hall/CRC.
- Freund, Y. & Schapire, R. E. (1997). A decision-theoretic generalization of on-line learning and an application to boosting. *JCSS, 55*(1), 119-139. Retrieved from https://www.sciencedirect.com/science/article/pii/S002200009791504X
- Gao, J., Fan, W., & Han, J. (2007). A general framework for mining concept-drifting data streams with skewed distributions. *Proc. of SDM*. DOI: 10.1137/1.9781611972771.1. Retrieved from http://epubs.siam.org/doi/abs/10.1137/1.9781611972771.1
- Grossman, R., Seni, G., Elder, J., Agarwal, N., & Liu, H. (2010). *Ensemble methods in data mining: Improving accuracy through combining predictions*. San Rafael, CA: Morgan & Claypool.
- Kohavi, R. (1995). A study of cross-validation and bootstrap for accuracy estimation and model selection. *IJCAI, 2*, 1137-1143. Retrieved from https://dl.acm.org/citation.cfm?id=1643047
- Pan, S. J. & Yang, Q. (2010). A Survey on transfer learning. *IEEE Trans. on Knowledge and Data Eng.* Retrieved from http://ieeexplore.ieee.org/document/5288526/
- Sun, Y., Wong, A. K. C., & Kamel, M. S. (2009). Classification of imbalanced data: A review. *Int. Journal of Pattern Recognition and Artificial Intelligence, 23*(4), 687. Retrieved from http://www.worldscientific.com/doi/abs/10.1142/S0218001409007326
- Zhou, Z.-H. (2012). *Ensemble methods: Foundations and algorithms*. Boca Raton, FL: CRC Press.

# References


- Stanford DAWN. (2017). *Labeled training data graphic* [Online image]. Retrieved from https://goo.gl/EaXfBj