Outline

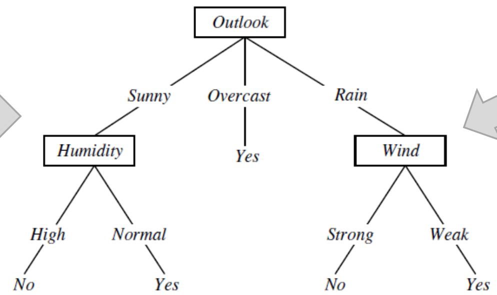
- Classification basics
- Decision tree classifiers
- Overfitting
- Lazy vs Eager Learners
- k-Nearest Neighbors (or learning from your neighbors)
- Evaluation of classifiers

True vs predicted class labels

- The quality of a classifier is evaluated over a *test set*, different from the training set
  - For each instance in the test set, we know its true class label
  - Compare the predicted class label (by the classifier) with the true class of the test instances

Training set

Day	Outlook	Temperature	Humidity	Wind	PlayTennis
D1	Sunny	Hot	High	Weak	No
D2	Sunny	Hot	High	Strong	No
D3	Overcast	Hot	High	Weak	Yes
D4	Rain	Mild	High	Weak	Yes
D5	Rain	Cool	Normal	Weak	Yes
D6	Rain	Cool	Normal	Strong	No
D7	Overcast	Cool	Normal	Strong	Yes
D8	Sunny	Mild	High	Weak	No
D9	Sunny	Cool	Normal	Weak	Yes
D10	Rain	Mild	Normal	Weak	Yes
D11	Sunny	Mild	Normal	Strong	Yes
D12	Overcast	Mild	High	Strong	Yes
D13	Overcast	Hot	Normal	Weak	Yes
D14	Rain	Mild	High	Strong	No



true class label

Test set

Day	Outlook	Temperature	Humidity	Wind	Play Teniss	Prediction
D16	Overcast	Cool	Normal	Weak	Yes	Yes
D17	Overcast	High	Normal	Weak	Yes	No
D18	Sunny	Hot	Normal	Weak	No	No
D19	Overcast	Cool	Normal	Weak	No	Yes

predicted class labels

Confusion matrix

- Terminology
  - Positive tuples: tuples of the main class of interest (e.g., “Play tennis = yes”)
  - Negative tuples: all other tuples
- A useful tool for analyzing how well a classifier performs is the *confusion matrix*
  - For an *m*-class problem, the matrix is of size *mxm*
- An example of a matrix for a 2-class problem:

Predicted class

Actual/ true class

	yes	no	totals
yes	TP (true positive)	FN (false negative)	P
no	FP(false positive)	TN (true negative)	N
Totals	P’	N’	

Classifier evaluation measures

- Accuracy
- Error rate
- Sensitivity
- Specificity
- Precision
- Recall
- F–measure
- F_β-measure
- ...

Classifier evaluation measures 1/3

Predicted class

	C_1	C_2	totals
C_1	TP (true positive)	FN (false negative)	P
C_2	FP(false positive)	TN (true negative)	N
Totals	P'	N'	

Actual class

- Accuracy/ Recognition rate:

  % of test set instances correctly classified

$$accuracy(M) = \frac{TP + TN}{P + N}$$

Predicted class

classes	buy_computer = yes	buy_computer = no	total
buy_computer = yes	6954	46	7000
buy_computer = no	412	2588	3000
total	7366	2634	10000

Actual class

→Accuracy(M)=95.42%

- Error rate/ Missclassification rate: error_rate(M)=1-accuracy(M)

$$\text{error\_rate}(M) = \frac{FP + FN}{P + N}$$

→Error_rate(M)=4.58%

Limitations of accuracy and error rate

- Consider a 2-class problem
  - Number of Class 0 examples = 9990
  - Number of Class 1 examples = 10
- If model predicts everything to be class 0, accuracy is 9990/10000 = 99.9 %
  - Accuracy is misleading because model does not detect any class 1 example

!!! Accuracy and error rate are more effective when the class distribution is relatively *balanced*

Classifier evaluation measures 2/3

If classes are *imbalanced*:

- Sensitivity/ True positive rate/ recall:

  % of positive tuples that are correctly recognized

$$sensitivity(M) = \frac{TP}{P}$$

- Specificity/ True negative rate : % of negative tuples that are correctly recognized

$$specificity(M) = \frac{TN}{N}$$

Predicted class

Actual class	C_1	C_2	totals
C_1	TP (true positive)	FN (false negative)	P
C_2	FP(false positive)	TN (true negative)	N
Totals	P'	N'	

Predicted class

Actual class: classes	buy_computer = yes	buy_computer = no	total
buy_computer = yes	6954	46	7000
buy_computer = no	412	2588	3000
total	7366	2634	10000

➔Accuracy(M)=95.42%

➔sensitivity(M)=99.34%

➔specificity(M)=86.27%

Classifier evaluation measures 3/3

- Precision: % of tuples labeled as positive which are actually positive

$$precision(M) = \frac{TP}{TP + FP}$$

- Recall: % of positive tuples labeled as positive

$$recall(M) = \frac{TP}{TP + FN} = \frac{TP}{P}$$

  - Precision biased towards TP and FP
  - Recall biased towards TP and FN
  - Higher precision → less FP
  - Higher recall → less FN

Predicted class

Actual class

	C_1	C_2	totals
C_1	TP (true positive)	FN (false negative)	P
C_2	FP(false positive)	TN (true negative)	N
Totals	P'	N'	

Recall the definition of precision/recall in IR:

- *Precision: % of selected items that are correct*
- *Recall: % of correct items that are selected*

Predicted class

Actual class

classes	buy_computer = yes	buy_computer = no	total
buy_computer = yes	6954	46	7000
buy_computer = no	412	2588	3000
total	7366	2634	10000

→precision(M)=94.41%

→recall(M)=99.34%

Classifier evaluation measures 3/3

- F-measure/ F_1 score/F-score combines both

$$F(M) = \frac{2 * precision(M) * recall(M)}{precision(M) + recall(M)}$$

It is the harmonic mean of precision and recall

Predicted class

Actual class

	C_1	C_2	totals
C_1	TP (true positive)	FN (false negative)	P
C_2	FP(false positive)	TN (true negative)	N
Totals	P'	N'	

More on harmonic mean: http://mathworld.wolfram.com/HarmonicMean.html

- F_β-measure is a weighted measure of precision and recall

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

Common values for β:
- $\beta=1 \rightarrow F_1$
- $\beta=0.5$

β is used to justify the importance of recall w.r.t. precision: recall is considered β times as important as precision !!!

- For our example, F(M)=2*94.41%*99.34%/(94.41%+99.34%)=96.81%

Evaluation setup

- How to create the training and test sets out of a dataset?
  - We don’t want to make unreasonable assumptions about our population
- Many approaches
  - Holdout
  - Cross-validation
  - Bootstrap
  - ….

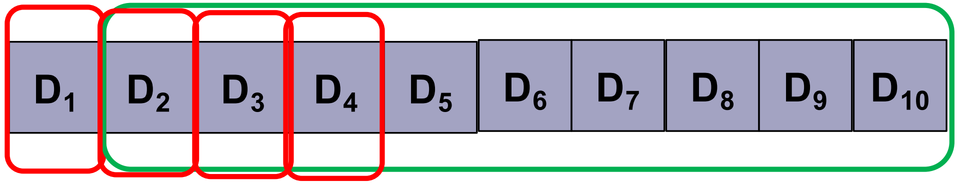
Evaluation setup 1/5

- 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
  - (+) It takes no longer to compute
  - (-) it depends on how data are divided

- Random sampling: a variation of holdout
  - Repeat holdout *k* times, accuracy is the *avg* accuracy obtained

Evaluation setup 2/5

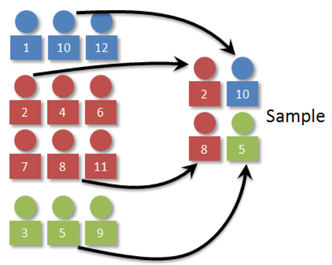
- Cross-validation (*k*-fold cross validation, k = 10 usually)
  - Randomly partition the data into *k mutually exclusive* subsets D_1, …, D_k each approximately equal size
  - Training and testing is performed k times
    - At the *i*-th iteration, use D_i as test set and rest as training set



  - Each point is in a test set 1 time and in a training set k-1 times
  - Accuracy is the avg accuracy over all iterations
  - (+) Does not rely so much on how data are divided
  - (-) The algorithm should re-run from scratch k times
- Leave-one-out: k-folds with k = #of tuples, so only one sample is used as a test set at a time;
    - 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
    - Stratified 10 fold cross-validation is recommended!!!

Evaluation setup 3/5

- Stratified sampling vs random sampling
  - Stratified sampling creates a mini-reproduction of the population in terms of the class labels. E.g., if 25% of the population belongs to the class “blue”, 25% to class “green” and 50% to class “red” then 25% of the sample is drawn randomly from class “blue”, 25% from class “green” and 50% from class “red”.



Source: https://faculty.elgin.edu/dkernler/statistics/ch01/images/strata-sample.gif

- Stratified cross-validation: folds are stratified so that class distribution in each fold is approximately the same as that in the initial data
  - Stratified 10 fold cross-validation is recommended!!!

Evaluation setup 4/5

- Bootstrap: Samples the given training data 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
  - Works well with small data sets
- Several boostrap methods, and a common one is .632 boostrap
  - Suppose we are given a data set of #d tuples. The data set is sampled #d times, with replacement, resulting in a training set of #d samples (known also as *bootstrap sample*):
  - The data tuples that did not make it into the training set end up forming the test set.
  - Each sample has a probability 1/d of being selected and (1-1/d) of not being chosen. We repeat d times, so the probability for a tuple to not be chosen during the whole period is $(1-1/d)^d$.
    - For large d: $\left(1-\frac{1}{n}\right)^n \approx e^{-1} \approx 0.368$
  - So on average, 36.8% of the tuples will not be selected for training and thereby end up in the test set; the remaining 63.2% will form the train set.

Evaluation setup 5/5

- Repeat the sampling procedure k times → k bootstrap datasets
- Report the overall accuracy of the model:

$$acc_{boot}(M) = \frac{1}{k}\sum_{i=1}^{k}(0.632 \times acc(M_i)_{testSet_i} + 0.368 \times acc(M_i)_{train\_set})$$

Accuracy of the model obtained by bootstrap sample i when it is applied on test set i.

Accuracy of the model obtained by bootstrap sample i when it is applied over all labeled data

Evaluation summary

- Evaluation measures
  - accuracy, error rate, sensitivity, specificity, precision, F-score, F_β…
- Train – test splitting
  - Holdout, cross-validation, bootstrap,…
- Other parameters
  - Speed (model building time, model testing time)
  - Robustness to noise, outliers and missing values
  - Scalability for large data sets
  - Interpretability (by humans)