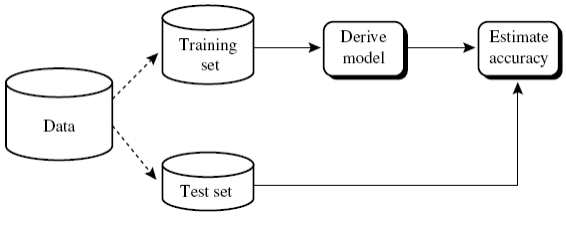
Knowledge Discovery and Data Mining

Unit # 6

Evaluating the Accuracy of a Classifier

- Holdout, random subsampling, crossvalidation, and the bootstrap are common techniques for assessing accuracy based on randomly sampled partitions of the given data.



Holdout Method

- The holdout method is what we have alluded to so far in our discussions about accuracy.
- In this method, the given data are randomly partitioned into two independent sets, a training set and a test set.
- Typically, two-thirds of the data are allocated to the training set, and the remaining one-third is allocated to the test set.
- The training set is used to derive the model, whose accuracy is estimated with the test set.
- The estimate is pessimistic because only a portion of the initial data is used to derive the model.

Performance Evaluation

- We evaluate performance of the classifier on the testing set
- With large labeled dataset, we would typically take 2/3 for training, 1/3 for testing
- What can we do if we have a small dataset?
  - Can't afford to take 1/3 for testing
  - Small testing means predicted error will be far from true error

Cross Validation

- Cross Validation helps to get a more accurate performance evaluation on "small" dataset
- K-fold Cross Validation
  - Divide the labeled data set into k subsets
  - Repeat k times:
    - Take subset i as the test data and the rest of subsets as the training data. Train the classifier and asses the testing error on subset i.
  - Average the testing error from the k iterations
- Leave one out Cross Validation
  - Cross validation with k = 1

Bootstrap

- The bootstrap method samples the given training tuples uniformly with replacement.
- That is, 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. A commonly used one is the .632 bootstrap, which works as follows.
  - Suppose we are given a data set of d tuples.
  - The data set is sampled d times, with replacement, resulting in a bootstrap sample or training set of d samples.

Bootstrap (Cont'd)

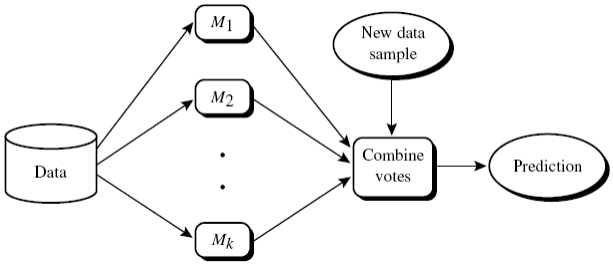
- It is very likely that some of the original data tuples will occur more than once in this sample.
- The data tuples that did not make it into the training set end up forming the test set.
- Suppose we were to try this out several times.
- As it turns out, on average, 63.2% of the original data tuples will end up in the bootstrap, and the remaining 36.8% will form the test set

Bootstrap (Cont'd)

- We can repeat the sampling procedure k times, where in each iteration, we use the current test set to obtain an accuracy estimate of the model obtained from the current bootstrap sample.
- The overall accuracy of the model is then estimated as $Acc(M) = (i=1 \text{ to } K) \Sigma (0.632 \times Acc(Mi)_{train\_set} + 0.368 \times Acc(Mi)_{test\_set})$
- where $Acc(Mi)_{test\ set}$ is the accuracy of the model obtained with bootstrap sample I when it is applied to test set i. $Acc(Mi)_{train\ set}$ is the accuracy of the model obtained with bootstrap sample i when it is applied to the original set of data tuples.
- The bootstrap method works well with small data sets.

Ensemble Models

- Bagging and boosting are examples of ensemble methods, or methods that use a combination of models.
- Each combines a series of k learned models (classifiers or predictors), M1, M2, …, Mk, with the aim of creating an improved composite model, M*.
- Both bagging and boosting can be used for classification as well as prediction



Bagging

- The name *Bagging* came from the abbreviation "Bootstrap AGGregatING" (1996). As the name implies, the two ingredients of Bagging are *bootstrap* and *aggregation*.
- Baggin applies bootstrap sampling to obtain the data subsets for training the base learners.
- Given a training data set containing *m* number of examples, a sample of *m* will be generated by *sampling with replacement*.

Bagging (Cont'd)

- By applying the process *T* times, *T* samples of *m* training examples are obtained.
- Then for each sample a base learner can be trained by applying a learning algorithm.
- Bagging adopts the most popular strategies for aggregating the outputs of the base learners, that is, voting for classification and averaging for regression.

Algorithm: Bagging. The bagging algorithm—create an ensemble of models (classifiers or predictors) for a learning scheme where each model gives an equally-weighted prediction.

Input:

- D, a set of d training tuples;
- k, the number of models in the ensemble;
- a learning scheme (e.g., decision tree algorithm, backpropagation, etc.)

Output: A composite model, $M*$.

Method:

(1) for $i = 1$ to k do // create k models:
(2) create bootstrap sample, D_i, by sampling D with replacement;
(3) use D_i to derive a model, M_i;
(4) endfor

To use the composite model on a tuple, X:

(1) if classification then
(2) let each of the k models classify X and return the majority vote;
(3) if prediction then
(4) let each of the k models predict a value for X and return the average predicted value;

Bagging (Cont'd)

- The bagged classifier often has significantly greater accuracy than a single classifier derived from D, the original training data.
- It will not be considerably worse and is more robust to the effects of noisy data.
- The increased accuracy occurs because the composite model reduces the variance of the individual classifiers.
- For prediction, it was theoretically proven that a bagged predictor will always have improved accuracy over a single predictor derived from D.

Boosting

- In boosting, weights are assigned to each training tuple.
- A series of k classifiers is iteratively learned.
- After a classifier Mi is learned, the weights are updated to allow the subsequent classifier,Mi+1, to "pay more attention" to the training tuples that were misclassified by Mi.
- The final boosted classifier, M, combines the votes of each individual classifier, where the weight of each classifier's vote is a function of its accuracy.

Algorithm: Adaboost. A boosting algorithm—create an ensemble of classifiers. Each one gives a weighted vote.

Input:

- D, a set of d class-labeled training tuples;
- k, the number of rounds (one classifier is generated per round);
- a classification learning scheme.

Output: A composite model.

Method:

(1) initialize the weight of each tuple in D to $1/d$;
(2) for $i = 1$ to k **do** // for each round:
(3) sample D with replacement according to the tuple weights to obtain D_i;
(4) use training set D_i to derive a model, M_i;
(5) compute $error(M_i)$, the error rate of M_i (Equation 6.66)
(6) if $error(M_i) > 0.5$ then
(7) reinitialize the weights to $1/d$
(8) go back to step 3 and try again;
(9) endif
(10) for each tuple in D_i that was correctly classified **do**
(11) multiply the weight of the tuple by $error(M_i)/(1 - error(M_i))$; // update weights
(12) normalize the weight of each tuple;
(13) endfor

5

Boosting Algorithm (Cont'd)

To use the composite model to classify tuple, X:

(1) initialize weight of each class to 0;
(2) for $i = 1$ to k **do** // for each classifier:
(3) $w_i = log\frac{1-error(M_i)}{error(M_i)}$; // weight of the classifier's vote
(4) $c = M_i(X)$; // get class prediction for X from M_i
(5) add w_i to weight for class c
(6) endfor
(7) return the class with the largest weight;

Idea Behind Ada Boost

- Algorithm is iterative
- Maintains distribution of weights over the training examples
- Initially distribution of weights is uniform
- At successive iterations, the weight of misclassified examples is increased, forcing the weak learner to focus on the hard examples in the training set.

Bagging vs. Boosting

- *Because of the way boosting focuses on* the misclassified tuples, it risks overfitting the resulting composite model to such data.
- Therefore, sometimes the resulting "boosted" model may be less accurate than a single model derived from the same data.
- Bagging is less susceptible to model overfitting.
- While both can significantly improve accuracy in comparison to a single model, boosting tends to achieve greater accuracy.

Random Forest

- Random Forest (2001) is a representative of the state-of-the-art ensemble methods. It is an extension of Bagging, where the major difference with Bagging is the incorporation of randomized feature selection.
- During the construction of a component decision tree, at each step of split selection, RF first randomly selects a subset of features, and then carries out the conventional split selection within the selected feature subset.

Random Forest (Cont'd)

- A parameter K controls the incorporation of randomness. When K equals the total number of features, the constructed decision tree is identical to the traditional decision tree; when K = 1, a feature will be selected randomly.
- The suggested value of K is the logarithm of the number of features.
- Notice that randomness is only introduced into the feature selection process, not into the choice of split points on the selected feature.

Stacking

- Stacking is a general procedure where a learner is trained to combine the individual learners.
- Here, the individual learners are called the *first-level learners*, while the combiner is called the *second-level learner*, or *meta-learner*.
- The basic idea is to train the first-level learners using the original training data set, and then generate a new data set for training the second-level learner, where the outputs of the first-level learners are regarded as input features while the original labels are still regarded as labels of the new training data.

Stacking (Cont'd)

- The first-level learners are often generated by applying different learning algorithms, and so, stacked ensembles are often heterogeneous, though it is also possible to construct homogeneous stacked ensembles.

Stacking (Cont'd)

- In the training phase of stacking, a new data set needs to be generated from the first-level classifiers. If the exact data that are used to train the first level learner are also used to generate the new data set for training the second-level learner, there will be a high risk of overfitting.
- Hence, it is suggested that the instances used for generating the new data set are excluded from the training examples for the first-level learners.

KNIME Demo

Ensemble Model

Eager Learners

- The classification methods discussed so far in this course—decision tree induction, Bayesian classification and classification by back-propagation—are all examples of *eager learners.*
- *Eager learners,* when given a set of training tuples, will construct a generalization (i.e., classification) model before receiving new (e.g., test) tuples to classify.
- We can think of the learned model as being ready and eager to classify previously unseen tuples.

Lazy Learners

- Imagine a contrasting lazy approach, in which the learner instead waits until the last minute before doing any model construction in order to classify a given test tuple.
- That is, when given a training tuple, a lazy learner simply stores it (or does only a little minor processing) and waits until it is given a test tuple.
- Only when it sees the test tuple does it perform generalization in order to classify the tuple based on its similarity to the stored training tuples.

Instance-based Learning

- In instance-based learning the training examples are stored verbatim, and a distance function is used to determine which member of the training set is closest to an unknown test instance.
- Once the nearest training instance has been located, its class is predicted for the test instance.
- The only remaining problem is defining the distance function, and that is not very difficult to do, particularly if the attributes are numeric.

Distance Function

- Although there are other possible choices, most instance-based learners use Euclidean distance.
- When comparing distances it is not necessary to perform the square root operation; the sums of squares can be compared directly.
- One alternative to the Euclidean distance is the Manhattan or city-block metric, where the difference between attribute values is not squared but just added up (after taking the absolute value).

Normalization

- Different attributes are measured on different scales, so if the Euclidean distance formula were used directly, the effects of some attributes might be completely dwarfed by others that had larger scales of measurement.
- Consequently, it is usual to normalize all attribute values to lie between 0 and 1,

Distance for Categorical Attributes

- For categorical attributes, a simple method is to compare the corresponding value of the attribute in tuple *X1 with that in tuple X2.*
- *If the two are identical (e.g., tuples X1 and X2 both have the color blue), then the difference between the two is taken as 0.*
- If the two are different (e.g., tuple *X1 is blue but tuple X2 is red), then the difference* is considered to be 1.

K-Nearest Neighbor

- The *k-nearest-neighbor* method was first described in the early 1950s.
- The method is labor intensive when given large training sets, and did not gain popularity until the 1960s when increased computing power became available.
- It has since been widely used in the area of pattern recognition.

Right Value for K?

- A good value for K can be determined experimentally.
- Starting with k = 1, we use a test set to estimate the error rate of the classifier.
- This process can be repeated each time by incrementing k to allow for one more neighbor.
- The k value that gives the minimum error rate may be selected.
- In general, the larger the number of training tuples is, the larger the value of k will be (so that classification and prediction decisions can be based on a larger portion of the stored tuples)

Low Training Time

- Unlike eager learning methods, lazy learners do less work when a training tuple is presented and more work when making a classification or prediction.
- Because lazy learners store the training tuples or "instances," they are also referred to as instance based learners, even though all learning is essentially based on instances.

High Prediction Time

- Although instance-based learning is simple and effective, it is often slow.
- The obvious way to find which member of the training set is closest to an unknown test instance is to calculate the distance from every member of the training set and select the smallest.
- This procedure is linear in the number of training instances: in other words, the time it takes to make a single prediction is proportional to the number of training instances.

High Prediction Time (Cont'd)

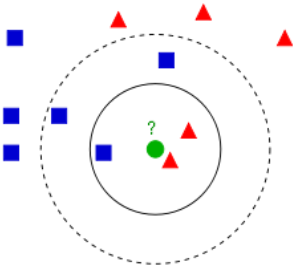
- When making a classification or prediction, lazy learners can be computationally expensive.
- They require efficient storage techniques and are well-suited to implementation on parallel hardware.
- They offer little explanation or insight into the structure of the data.
- Lazy learners, however, naturally support incremental learning.
- They are able to model complex decision spaces having hyper polygonal shapes that may not be as easily describable by other learning algorithms (such as hyper-rectangular shapes modeled by decision trees).

Prediction using Nearest Neighbor

- Nearest neighbor classifiers can also be used for prediction, that is, to return a real-valued prediction for a given unknown tuple.
- In this case, the classifier returns the average value of the real-valued labels associated with the *k nearest neighbors of the unknown* tuple.

Example

- The test sample (green circle) should be classified either to the first class of blue squares or to the second class of red triangles.
- If *k = 3* it is assigned to the second class because there are 2 triangles and only 1 square inside the inner circle.
- If *k = 5* it is assigned to the first class (3 squares vs. 2 triangles inside the outer circle).



Example: Continuous Attributes

X	Y	C
0.3	0.7	A
0.2	0.9	B
0.6	0.6	A
0.5	0.5	A
0.7	0.7	B
0.4	0.9	B

0.2	0.6	?

Example: Mammals vs. Non-mammals
K = 3, K = 5

Name	Give Birth	Can Fly	Live in Water	Have Legs	Class
human	yes	no	no	yes	mammals
python	no	no	no	no	non-mammals
salmon	no	no	yes	no	non-mammals
whale	yes	no	yes	no	mammals
frog	no	no	sometimes	yes	non-mammals
komodo	no	no	no	yes	non-mammals
bat	yes	yes	no	yes	mammals
pigeon	no	yes	no	yes	non-mammals
cat	yes	no	no	yes	mammals
leopard shark	yes	no	yes	no	non-mammals
turtle	no	no	sometimes	yes	non-mammals
penguin	no	no	sometimes	yes	non-mammals
porcupine	yes	no	no	yes	mammals
eel	no	no	yes	no	non-mammals
salamander	no	no	sometimes	yes	non-mammals
gila monster	no	no	no	yes	non-mammals
platypus	no	no	no	yes	mammals
owl	no	yes	no	yes	non-mammals
dolphin	yes	no	yes	no	mammals
eagle	no	yes	no	yes	non-mammals

Give Birth	Can Fly	Live in Water	Have Legs	Class
yes	no	yes	no	?

Example: Play Tennis
K = 3, K = 5

Outlook	Temperature	Humidity	Windy	Class
sunny	hot	high	false	N
sunny	hot	high	true	N
overcast	hot	high	false	P
rain	mild	high	false	P
rain	cool	normal	false	P
rain	cool	normal	true	N
overcast	cool	normal	true	P
sunny	mild	high	false	N
sunny	cool	normal	false	P
rain	mild	normal	false	P
sunny	mild	normal	true	P
overcast	mild	high	true	P
overcast	hot	normal	false	P
rain	mild	high	true	N

Outlook	Temperature	Humidity	Windy	Class
rain	hot	high	false	?

Limitations

- It tends to be slow for large training sets, because the entire set must be searched for each test instance
- It performs badly with noisy data, because the class of a test instance is determined by its single nearest neighbor without any “averaging” to help to eliminate noise.
- It performs badly when different attributes affect the outcome to different extents—in the extreme case, when some attributes are completely irrelevant—because all attributes contribute equally to the distance formula.
- It does not perform explicit generalization

Reducing the Number of Exemplars

- The plain nearest-neighbor rule stores a lot of redundant exemplars: it is almost always completely unnecessary to save all the examples seen so far.
- A simple variant is to classify each example with respect to the examples already seen and to save only ones that are misclassified.
- Discarding correctly classified instances reduces the number of exemplars and proves to be an effective way to prune the exemplar database.

Reducing the Number of Exemplars (Cont'd)

- Unfortunately, the strategy of only storing misclassified instances does not work well in the face of noise.
- Noisy examples are very likely to be misclassified, and so the set of stored exemplars tends to accumulate those that are least useful.
- This effect is easily observed experimentally. Thus this strategy is only a stepping-stone on the way toward more effective instance-based learners.

Pruning Noisy Exemplars

- Noisy exemplars inevitably lower the performance of any nearest-neighbor scheme that does not suppress them because they have the effect of repeatedly misclassifying new instances.
- A solution is to monitor the performance of each exemplar that is stored and discard ones that do not perform well.
- This can be done by keeping a record of the number of correct and incorrect classification decisions that each exemplar makes.

Supervised Learning Process

- Data Collection/Preparation
  - Data Cleaning
  - Discretization
    - Supervised/Unspervised
  - Identification of right number of discrete states for nominal attributes
- Feature Selection
  - Supervised/Unsupervised
- Classification Techniques
  - Decision Tree, Neural Networks, Naïve Bayes, Nearest Neighbor
- Model Testing and Evaluation
  - Accuracy, Recall/Precision/F-Measure, Bagging, Boosting, Cross-Validation, ROC Curve, Lift Charts
- The whole activity is not a sequential process. More importantly, you would always be required to make good use of common sense.

The No Free Lunch Theorem

- The No Free Lunch Theorem (1996;1997) implies that it is hopeless to dream for a learning algorithm which is consistently better than other learning algorithms.
- It is important to notice, however, that the NFL Theorem considers the whole problem space, that is, all possible learning tasks; while in real practice, we are usually only interested in a given task, and in such situation, the effort of trying to find the best algorithm is valid.