T

Tagging

►POS Tagging

TAN

►Tree Augmented Naive Bayes

Taxicab Norm Distance

►Manhattan Distance

TD-Gammon

Definition

TD-Gammon is a world-champion strength backgammon program developed by Gerald Tesauro. Its development relied heavily on machine learning techniques, in particular on ►Temporal-Difference Learning. Contrary to successful game programs in domains such as chess, which can easily out-search their human opponents but still trail these ability of estimating the positional merits of the current board configuration, TD-Gammon was able to excel in backgammon for the same reasons that humans play well: its grasp of the positional strengths and weaknesses was excellent. In 1998, it lost a 100-game competition against the world champion with only 8 points. Its sometimes unconventional but very solid evaluation of certain opening strategies had a strong impact on the backgammon community and was soon adapted by professional players.

Description of the Learning System

TD-Gammon is a conventional game-playing program that uses very shallow search (the first versions only searched one ply) for determining its move. Candidate moves are evaluated by a ►Neural Network, which is trained by TD(λ), a well-known algorithm for Temporal-Difference Learning (Tesauro, 1992). This evaluation function is trained on millions of games that the program played against itself. At the end of each game, a reinforcement signal that indicates whether the game has been lost or won is passed through all moves of the game. TD-Gammon developed useful concepts in the hidden layer of its network. Tesauro (1992) shows examples for two hidden units of TD-Gammon that he interpreted as a race-oriented feature detector and an attack-oriented feature detector.

TD-Gammon clearly surpassed its predecessors, in particular the Computer Olympiad champion Neurogammon, which was trained with ►Preference Learning (Tesauro, 1989). In fact, early versions of TD-Gammon, which only used the raw board information as features, already learned to play as well as Neurogammon, which used a sophisticated set of features. Adding more sophisticated features to the input representation further improved TD-Gammon's playing strength. Over time, TD-Gammon not only that increase the number of training games that it played against itself, but Tesauro also increased the search depth and changed the network architecture, so that TD-Gammon eventually reached world-championship strength (Tesauro, 1995).

Cross References

►Machine Learning and Game Playing

Recommended Reading

Tesauro, G. (1989). Connectionist learning of expert preferences by comparison training. In D. Touretzky (Ed.), *Proceedings of the advances in neural information processing systems 1 (NIPS-88)* (pp. 99–106). San Francisco: Morgan Kaufmann.

Tesauro, G. (1992). Practical issues in temporal difference learning. *Machine Learning, 8*, 257–278. http://mlis.www.wkap.nl/mach/abstracts/absv8p257.htm.

Claude Sammut & Geoffrey I. Webb (eds.), *Encyclopedia of Machine Learning*, DOI 10.1007/978-0-387-30164-8,

Tesauro, G. (1995). Temporal difference learning and TD-Gammon. *Communications of the ACM, 38*(3), 58–68. http://www.research.ibm.com/massdist/tdl.html.

TDIDT Strategy

►Divide-and-Conquer Learning

Temporal Credit Assignment

►Credit Assignment

Temporal Data

►Time Series

Temporal Difference Learning

William Uther
NICTA and the University of New South Wales

Definition

Temporal Difference Learning, also known as TD-Learning, is a method for computing the long term utility of a pattern of behavior from a series of intermediate rewards (Sutton, 1984, 1988; Sutton and Barto, 1998). It uses differences between successive utility estimates as a feedback signal for learning. The Temporal Differencing approach to model-free ►reinforcement learning was introduced by, and is often associated with, R.S. Sutton. It has ties to both the artificial intelligence and psychological theories of reinforcement learning as well as ►dynamic programming and operations research from economics (Bellman, 1957; Bertsekas & Tsitsiklis, 1996; Puterman, 1994; Samuel, 1959; Watkins, 1989).

While TD learning can be formalised using the theory of ►Markov Decision Processes, in many cases it has been used more as a heuristic technique and has achieved impressive results even in situations where the formal theory does not strictly apply, e.g., Tesauro's TD-Gammon (Tesauro, 1995) achieved world champion performance in the game of backgammon. These heuristic results often did not transfer to other domains, but over time the theory behind TD learning has expanded to cover large areas of reinforcement learning.

Formal Definitions

Consider an agent moving through a world in discrete time steps, $t_1, t_2, \ldots$. At each time step, t, the agent is informed of both the current state of the world, $s_t \in \mathcal{S}$, and its reward, or utility, for the previous time step, $r_{t-1} \in \mathbb{R}$.

As the expected long term utility of a pattern of behavior can change depending upon the state, the utility is a function of the state, $V : \mathcal{S} \to \mathbb{R}$. V is known as the *value function* or *state-value function*. The phrase "long term utility" can be formalized in multiple ways.

Undiscounted sum of reward:

The simplest definition is that long term reward is the sum of all future rewards.

$$\begin{aligned} V(s_t) &= r_t + r_{t+1} + r_{t+2} + \cdots \\ &= \sum_{\delta=0}^{\infty} r_{t+\delta} \end{aligned}$$

Unfortunately, the undiscounted sum of reward is only well defined if this sum converges. Convergence is usually achieved by the addition of a constraint that the agent's experience terminates at some, finite, point in time and all rewards after that point are zero.

Discounted sum of reward:

The *discounted* utility measure discounts rewards exponentially into the future.

$$\begin{aligned} V(s_t) &= r_t + \gamma r_{t+1} + \gamma^2 r_{t+2} + \cdots \quad \gamma \in [0,1] \\ &= \sum_{\delta=0}^{\infty} \gamma^{\delta} r_{t+\delta} \end{aligned}$$

Note that when $\gamma = 1$ the discounted and undiscounted regimes are identical. When $\gamma < 1$, the discounted reward scheme does not require that the agent experience terminates at some finite time for convergence. The *discount factor* γ can be interpreted as an inflation rate, a probability of failure for each time step, or simply as a mathematical trick to achieve convergence.

Average reward:

Rather than consider a sum of rewards, the *average reward* measure of utility estimates both the expected reward per future time step, also known as the *gain*, and the current difference from that long-term average, or *bias*.

$$G(s_t) = \lim_{n\to\infty} \frac{1}{n} \sum_{\delta=0}^{n} r_{t+\delta}$$
$$B(s_t) = \sum_{\delta=0}^{\infty} [r_{t+\delta} - G(s_{t+\delta})]$$

A system where any state has a nonzero probability of being reached from any other state is known as an ergodic system. For such a system the gain, $G(s)$, will have the same value for all states and the bias, $B(s)$, serves a similar purpose to $V(s)$ above in indicating the relative worth of different states. While average reward has a theoretical advantage in that there is no discount factor to choose, historically average reward has been considered more complex to use than the discounted reward regimes and so has been less used in practice. There is a strong theoretical relationship between average reward and discounted reward in the limit as the discount factor approaches one.

Here we focus on discounted reward.

Estimating Discounted Sum of Reward The temporal differencing estimation procedure is based on recursive reformulation of the above definitions. For the discounted case:

$$\begin{aligned} V(s_t) &= r_t + \gamma r_{t+1} + \gamma^2 r_{t+2} + \gamma^3 r_{t+3} + \cdots \\ &= r_t + \gamma[r_{t+1} + \gamma r_{t+2} + \gamma^2 r_{t+3} + \cdots] \\ &= r_t + \gamma V(s_{t+1}) \end{aligned}$$

From the recursive formulation we can see that the long term utility for one time step is closely related to the long term utility at the next time step. If there is already an estimate of the long term utility at s_t, $V(s_t)$, then we could calculate a change in that value given a new trajectory as follows:

$$\Delta_t = [r_t + \gamma V(s_{t+1})] - V(s_t)$$

If we are dealing with a stochastic system, then we may not want to update $V(s_t)$ to the new value in one jump, but rather only move part way toward the new value:

$$\Delta_t = \alpha(r_t + \gamma V(s_{t+1}) - V(s_t))$$

where α is a learning rate between 0 and 1. As an assignment, this update can be written in a number of equivalent ways, the two most common being:

$$V(s_t) \leftarrow V(s_t) + \alpha(r_t + \gamma V(s_{t+1}) - V(s_t)) \quad \text{or,}$$
$$V(s_t) \leftarrow (1-\alpha)V(s_t) + \alpha(r_t + \gamma V(s_{t+1}))$$

This *update*, *error*, *learning* or *delta* rule is the core of temporal difference learning. It is from this formulation, which computes a delta based on the difference in estimated long term utility of the world at two consecutive time steps, that we get the term temporal differencing.

Having derived this update rule, we can now apply it to finding the long term utility of a particular agent. In the simplest case we will assume that there are a finite number of Markov states of the world, and that these can be reliably detected by the agent at run time. We will store the function V as an array of real numbers, with one number for each world state.

After each time step, t, we will use the knowledge of the previous state, s_t, the instantaneous reward for the time step, r_t, and the resulting state, s_{t+1}, to update the value of the previous state, $V(s_t)$, using the delta rule above:

$$V(s_t) \leftarrow V(s_t) + \alpha(r_t + \gamma V(s_{t+1}) - V(s_t))$$

Eligibility Traces and TD(λ)

Basic temporal differencing as represented above can be quite slow to converge in many situations. Consider, for example, a simple corridor with a single reward at the end, and an agent that walks down the corridor. Assume that the value function was initialized to a uniform zero value. On each walk down the corridor, useful information is only pushed one step back toward the start of the corridor.

Eligibility traces try to alleviate this problem by pushing information further back along the trajectory of the agent with each update to V. An algorithm incorporating eligibility traces can be seen as a mixture of "pure" TD, as described above, and ▶Monte-Carlo estimation of the long term utility. In particular, the λ parameter to the TD(λ) family of algorithms

specifies where in the range from pure TD, when $\lambda = 0$, to pure Monte-Carlo, when $\lambda = 1$, a particular algorithm falls.

Eligibility traces are implemented by keeping a second function of the state space, $\epsilon : \mathcal{S} \to \mathbb{R}$. The ϵ function represents how much an experience now should affect the value of a state the agent has previously passed through. When the agent performs an update, the values of all states are changed according to their eligibility.

The standard definition of the eligibility of a particular state uses an exponential decay over time, but this is not a strict requirement and other definitions of eligibility could be used. In addition, each time a state is visited, its eligibility increases. Formally, on each time step,

$$\forall_{s \in \mathcal{S}}\ \epsilon(s) \leftarrow \gamma\lambda\epsilon(s) \qquad \text{and then,}$$
$$\epsilon(s_t) \leftarrow \epsilon(s_t) + 1$$

This eligibility is used to update all state values by first calculating the delta for the current state as above, but then applying it to all states according to the eligibility values:

$$\Delta_t = \alpha(r_t + \gamma V(s_{t+1}) - V(s_t))$$
$$\forall_{s \in \mathcal{S}}\ V(s) \leftarrow V(s) + \Delta_t \epsilon(s)$$

Convergence

TD value function estimation has been shown to converge under many conditions, but there are also well known examples where it does not converge at all, or does not converge to the correct long term reward (Tsitsiklis & Van Roy, 1997).

In particular, temporal differencing has been shown to converge to the correct value of the long term discounted reward if,

- The world is finite.
- The world state representation is Markovian.
- The rewards are bounded.
- The representation of the V function has no constraints (e.g., a tabular representation with an entry for each state).
- The learning rate, α, is reduced according to the Robbins-Monro conditions: $\sum_{t=0}^{\infty} \alpha_t = \infty$, and $\sum_{t=0}^{\infty} \alpha_t^2 < \infty$.

Much of the further work in TD learning since its invention has been in finding algorithms that provably converge in more general cases.

These convergence results require that a Markovian representation of state be available to the agent. There has been research into how to acquire such a representation from a sequence of observations. The approach of the Temporal Differencing community has been to use TD-Networks (Sutton & Tanner, 2004).

Control of Systems

Temporal Difference Learning is used to estimate the long term reward of a pattern of behavior. This estimation of utility can then be used to improve that behavior, allowing TD to help solve a reinforcement learning problem. There are two common ways to achieve this: An *Actor-Critic* setup uses value function estimation as one component of a larger system, and the *Q-learning* and *SARSA* techniques can be viewed as slight modifications of the TD method which allow the extraction of control information more directly from the value function.

First we will formalise the concept of a pattern of behavior. In the preceding text it was left deliberately vague as TD can be applied to multiple definitions. Here we will focus on discrete action spaces.

Assume there is a set of allowed actions for the agent, $\mathcal{A}$. We define a *Markov policy* as a function from world states to actions, $\pi : \mathcal{S} \to \mathcal{A}$. We also define a *stochastic* or *mixed* Markov policy as a function from world states to probability distributions over actions, $\pi : \mathcal{S} \to \mathcal{A} \to [0,1]$. The goal of the control algorithm is to find an optimal policy: a policy that maximises long term reward in each state. (When function approximation is used (see section "Approximation"), this definition of an optimal policy no longer suffices. One can then either move to average reward if the system is ergodic, or give a, possibly implicit, weighting function specifying the relative importance of different states.)

Actor-Critic Control Systems Actor-Critic control is closely related to *mixed policy iteration* from Markov Decision Process theory. There are two parts to an actor-critic system; the *actor* holds the current policy for the agent, and the *critic* evaluates the actor and suggests improvements to the current policy.

There are a number of approaches that fall under this model. One early approach stores a preference value for each world state and action pair, $p : \mathcal{S} \times \mathcal{A} \rightarrow \mathbb{R}$. The actor then uses a stochastic policy based on the Gibbs softmax function applied to the preferences:

$$\pi(s,a) = \frac{e^{p(s,a)}}{\sum_{x \in \mathcal{A}} e^{p(s,x)}}$$

The critic then uses TD to estimate the long term utility of the current policy, and also uses the TD update to change the preference values. When the agent is positively surprised it increases the preference for an action, when negatively surprised it decreases the preference for an action. The size of the increase or decrease is modulated by a parameter, β:

$$p(s_t, a_t) \leftarrow p(s_t, a_t) + \beta \Delta_t$$

Convergence of this algorithm to an optimal policy is not guaranteed.

A second approach requires the agent to have an accurate model of its environment. In this approach the critic uses TD to learn a value function for the current behavior. The actor uses model based forward search to choose an action likely to lead to a state with a high expected long term utility. This approach is common in two player, zero sum, alternating move games such as Chess or Checkers where the forward search is a deterministic game tree search.

More modern approaches which guarantee convergence are related to *policy gradient* approaches to reinforcement learning (Castro & Meir, 2010). These store a stochastic policy in addition to the value function, and then use the TD updates to estimate the gradient of the long term utility with respect to that policy. This allows the critic to adjust the policy in the direction of the negative gradient with respect to long term value, and thus improve the policy.

Other Value Functions The second class of approaches to using TD for control relies upon extending the value function to estimate the value of multiple actions. Instead of V we use a *state-action value function*, $Q : \mathcal{S} \times \mathcal{A} \rightarrow \mathbb{R}$. The update rule for this function is minimally modified from the TD update defined for V above.

Once these state-action value functions have been estimated, a policy can be selected by choosing for each state the action that maximizes the state-action value function, and then adding some exploration.

In order for this extended value function to be learned, the agent must explore each action in each state infinitely often. Traditionally this has been assured by making the agent select random actions occasionally, even when the agent believes that action would be sub-optimal. In general the choice of when to explore using a sub-optimal action, the *exploration/exploitation trade-off*, is difficult to optimize. More recent approaches to optimizing the exploration/exploitation trade-off in reinforcement learning estimate the variance of the value function to decide where they need to explore (Auer & Ortner, 2007).

The requirement for exploration leads to two different value functions that could be estimated. The agent could estimate the value function of the pattern of behavior currently being executed, which includes the exploration. Or, the agent could estimate the value function of the current best policy, excluding the exploration currently in use. These are referred to as *on-policy* and *off-policy* methods respectively.

Q-Learning is an off-policy update rule:

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha(r_t + \gamma V(s_{t+1}) - Q(s_t, a_t))$$

$$\text{where} \qquad V(s_{t+1}) = \max_{a \in \mathcal{A}} Q(s_{t+1}, a)$$

SARSA is an on-policy update rule:

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha(r_t + \gamma Q(s_{t+1}, a_{t+1}) - Q(s_t, a_t))$$

Then for both:

$$\pi(s) = argmax_{a \in \mathcal{A}} Q(s,a)$$

and some exploration.

As can be seen above, the update rules for SARSA and Q-learning are very similar – they only differ in the value used for the resulting state. Q-learning uses the value of the best action, whereas SARSA uses the value of the action that will actually be chosen.

Q-Learning converges to the best policy to use once you have converged and can stop exploring. SARSA converges to the best policy to use if you want to keep exploring as you follow the policy (Lagoudakis & Parr, 2003).

Approximation

A major problem with many state based algorithms, including TD learning, is the so-called ▶curse of dimensionality. In a factored state representation, the number of states increases exponentially with the number of factors. This explosion of states produces two problems: it can be difficult to store a function over the state space, and even if the function can be stored, so much data is required to learn the function that learning is impractical.

The standard response to the curse of dimensionality is to apply function approximation to any function of state. This directly attacks the representation size, and also allows information from one state to affect another "similar" state allowing generalisation and learning.

While the addition of function approximation can significantly speed up learning, it also causes difficulty with convergence. Some types of function approximation will stop TD from converging at all. The resulting algorithms can either oscillate forever or approach infinite values. Other forms of approximation cause TD to converge to a estimate of long term reward which is only weakly related to the true long term reward (Baird, 1995; Boyan & Moore, 1995; Gordon, 1995).

Most styles of function approximation used in conjunction with TD learning are parameterized, and the output is differentiable with respect to those parameters. Formally we have $V : \Theta \to \mathcal{S} \to \mathbb{R}$, where Θ is the space of possible parameter vectors, so that $V_\theta(s)$ is the value of V at state s with parameter vector θ, and $\nabla V_\theta(s)$ is the gradient of V with respect to θ at s. The TD update then becomes:

$$\Delta_t = \alpha \left(r_t + \gamma V_\theta(s_{t+1}) - V_\theta(s_t) \right)$$
$$\theta \leftarrow \theta + \Delta_t \nabla V_\theta(s_t)$$

We describe three styles of approximation: state abstraction, linear approximation, and smooth general approximators (e.g., neural networks).

State abstraction refers to grouping states together and thereafter using the groups, or *abstract states*, instead of individual states. This can significantly reduce the amount of storage required for the value function as only values for abstract states need to be stored. It also preserves convergence results. A slightly more advanced form of state abstraction is the tile coding or CMAC (Albus, 1981). In this type of function approximation, the state representation is assumed to be factored, i.e., each state is represented by a vector of values rather than a single scalar value. The CMAC represents the value function as the sum of separate value functions; one for each dimension of the state. Those individual dimensions can each have their own state abstraction. Again, TD has been shown to converge when used with a CMAC value function representation.

In general, any form of function approximation that forms a contraction mapping will converge when used with TD (see the entry on ▶Markov Decision Processes). Linear interpolation is a contraction mapping, and hence converges. Linear extrapolation is not a contraction mapping and care needs to be taken when using general linear functions with TD. It has been shown that general linear function approximation used with TD will converge, but only when complete trajectories are followed through the state space (Tsitsiklis & Van Roy, 1997).

It is not uncommon to use various types of back-propagation neural nets with TD, e.g., Tesauro's TD-gammon. More recently, TD algorithms have been proposed that converge for arbitrary differentiable function approximators (Maei et al., 2009; Papavassiliou and Russell, 1999). These use more complex update techniques than those shown above.

Related Differencing Systems

TD learning was originally developed for use in environments where accurate models were unavailable. It has a close relationship with the theory of Markov Decision Processes where an accurate model is assumed. Using the notation $V(s_t) \rightsquigarrow V(s_{t+1})$ for a TD-style update that moves the value at $V(s_t)$ closer to the value at $V(s_{t+1})$ (including any discounting and intermediate rewards), we can now consider many possible updates.

As noted above, one way of applying TD to control is to use forward search. Forward search can be implemented using dynamic programming, and the result is closely related to TD. Let state $c(s)$ be the best child of state s in the forward search. We can then consider an update, $V(s) \rightsquigarrow V(c(s))$. If we let $l(s)$ be the best leaf in the forward search, we could then consider an update $V(s) \rightsquigarrow V(l(s))$. Neither of these updates consider the world after an actual state transition, only simulated state transitions, and so neither is technically a TD update.

Some work has combined both simulated time steps and real time steps. The TD-Leaf learning algorithm for alternative move games uses the $V(l(s_t)) \rightsquigarrow V(l(s_{t+1}))$ update rule (Baxter et al., 1998).

An important issue to consider when using forward search is whether the state distribution where learning takes place is different to the state distribution where the value function is used. In particular, if updates only occur for states the agent chooses to visit, but the search is using estimates for states that the agent is not visiting, then TD may give poor results. To combat this, the TreeStrap(α-β) algorithm for alternating move games updates all nodes in the forward search tree to be closer to the bound information provided by their children (Veness et al., 2009).

Biological Links

There are strong relationships between TD learning and the Rescorla–Wagner model of Pavlovian conditioning. The Rescorla–Wagner model is one way to formalize the idea that learning occurs when the co-occurence of two events is surprising rather than every time a co-occurence is experienced. The Δ_t value calculated in the TD update can be viewed as a measure of surprise. These findings appear to have a neural substrate in that dopamine cells react to reward when it is unexpected and to the predictor when the reward is expected (Schultz et al., 1997; Sutton & Barto, 1990).

Cross References

►Curse of Dimensionality
►Markov Decision Processes
►Monte-Carlo Simulation
►Reinforcement Learning

Recommended Reading

Albus, J. S. (1981). *Brains, behavior, and robotics*. Peterborough: BYTE, ISBN: 0070009759.

Auer, P., & Ortner, R. (2007). Logarithmic online regret bounds for undiscounted reinforcement learning. *Neural and Information Processing Systems (NIPS)*.

Baird, L. C. (1995). Residual algorithms: reinforcement learning with function approximation. In A. Prieditis & S. Russell (Eds.), *Machine Learning: Proceedings of the Twelfth International Conference (ICML95)* (pp. 30–37). San Mateo: Morgan Kaufmann.

Baxter, J., Tridgell, A., & Weaver, L. (1998). KnightCap: a chess program that learns by combining TD(lambda) with game-tree search. In J. W. Shavlik (Ed.), *Proceedings of the Fifteenth International Conference on Machine Learning (ICML '98)* (pp. 28–36). San Francisco: Morgan Kaufmann.

Bellman, R. E. (1957). *Dynamic programming*. Princeton: Princeton University Press.

Bertsekas, D. P., & Tsitsiklis, J. (1996). Neuro-dynamic programming. Belmont: Athena Scientific.

Boyan, J. A., & Moore, A. W. (1995). Generalization in reinforcement learning: safely approximating the value function. In G. Tesauro, D. S. Touretzky, & T. K. Leen (Eds.), *Advances in neural information processing systems* (Vol. 7). Cambridge: MIT Press.

Di Castro, D., & Meir, R. (2010). A convergent online single time scale actor critic algorithm. *Journal of Machine Learning Research, 11*, 367–410. http://jmlr.csail.mit.edu/papers/v11/dicastro10a.html

Gordon, G. F. (1995). *Stable function approximation in dynamic programming* (Technical report CMU-CS-95-103). School of Computer Science, Carnegie Mellon University.

Lagoudakis, M. G., & Parr, R. (2003). Least-squares policy iteration. *Journal of Machine Learning Research 4*, 1107–1149. http://www.cs.duke.edu/~parr/jmlr03.pdf

Maei, H. R. et al. (2009). Convergent temporal-difference learning with arbitrary smooth function approximation. *Neural and Information Processing Systems (NIPS)*, pp. 1204–1212. http://books.nips.cc/papers/files/nips22/NIPS2009_1121.pdf

Mahadevan, S. (1996). Average reward reinforcement learning: foundations, algorithms, and empirical results. *Machine Learning, 22*, 159–195, doi: 10.1023/A:1018064306595.

Papavassiliou, V. A., & Russell, S. (1999). Convergence of reinforcement learning with general function approximators. *International Joint Conference on Artificial Intelligence*, Stockholm.

Puterman, M. L. (1994). *Markov decision processes: discrete stochastic dynamic programming. Wiley series in probability and mathematical statistics. Applied probability and statistics section*. New York: Wiley.

Samuel, A. L. (1959). Some studies in machine learning using the game of checkers. *IBM Journal on Research and Development, 3*(3), 210–229.

Schultz, W., Dayan, P., & Read Montague, P. (1997). A neural substrate of prediction and reward. *Science, 275*(5306), 1593–1599, doi: 10.1126/science.275.5306.1593.

Sutton, R., & Tanner, B. (2004). Temporal difference networks. *Neural and Information Processing Systems (NIPS)*.

Sutton, R. S. (1984). Temporal credit assignment in reinforcement learning. Ph.D. thesis, University of Massachusetts, Amherst.

Sutton, R. S. (1988). Learning to predict by the method of temporal differences. *Machine learning, 3*, 9–44, doi: 10.1007/BF00115009.

Sutton, R. S., & Barto, A. G. (1998). *Reinforcement learning: an introduction*. Cambridge: MIT Press.

Sutton, R. S., & Barto, A. G. (1990). Time-derivative models of Pavlovian reinforcement. In M. Gabriel & J. Moore (Eds.), *Learning and computational neuroscience: foundations of adaptive networks* (pp. 497–537). Cambridge: MIT Press.

Tesauro, G. (1995). Temporal difference learning and TD-gammon. *Communications of the ACM, 38*(3), 58–67.

Tsitsiklis, J. N., & Van Roy, B. (1997). An analysis of temporal-difference learning with function approximation. *IEEE Transactions on Automatic Control, 42*(5), 674–690.

Veness, J., et al. (2009). Bootstrapping from game tree search. *Neural and Information Processing Systems (NIPS)*.
Watkins, C. J. C. H. (1989). *Learning with delayed rewards*. Ph.D. thesis, Cambridge University Psychology Department, Cambridge.

Test Data

Synonyms
Evaluation data; Test instances

Definition
Test data are data to which a ►model is applied for the purposes of ►evaluation. When ►holdout evaluation is performed, test data are also called *out-of-sample data*, *holdout data*, or the *holdout set*.

Cross References
►Test Set

Test Instances

►Test Data

Test Set

Synonyms
Evaluation data; Evaluation set; Test data

Definition
A test set is a ►data set containing data that are used for ►evaluation by a ►learning system. Where the ►training set and the test set contain disjoint sets of data, the test set is known as a ►holdout set.

Cross References
►Data Set

Test Time

A learning algorithm is typically applied at two distinct times. Test time refers to the time when an algorithm is applying a learned model to make predictions. ►Training time refers to the time when an algorithm is learning a model from ►training data. ►Lazy learning usually blurs the distinction between these two times, deferring most learning until test time.

Test-Based Coevolution

Synonyms
Competitive coevolution

Definition
A coevolutionary system constructed to simultaneously develop solutions to a problem and challenging tests for candidate solutions. Here, individuals represent complete solutions or their tests. Though not precisely the same as *competitive coevolution*, there is a significant overlap.

Text Clustering

►Document Clustering

Text Learning

►Text Mining

Text Mining

Dunja Mladenić
Jožef Stefan Insitute, Ljubljana, Slovenia

Synonyms
Analysis of text; Data mining on text; Text learning

Definition
The term *text mining* is used analogous to ►data mining when the data is text. As there are some data specificities when handling text compared to handling data from databases, text mining has a number of specific methods and approaches. Some of these are extensions of data mining and machine learning methods, while other are rather text-specific. Text mining approaches

combine methods from several related fields, including machine learning, data mining, ►information retrieval, ►natural language processing, ►statistical learning, and the Semantic Web. Basic text mining approaches are also extended to enable handling different natural languages (►cross-lingual text mining) and are combined with methods for handling different data types, such as links and graphs (►link mining and link discovery, ►graph mining), images and video (multimedia mining).

Cross References

►Cross-Lingual Text Mining
►Feature Construction In Text Mining
►Feature Selection In Text Mining
►Semi-Supervised Text Processing
►Text Mining For Advertising
►Text Mining For News and Blogs Analysis
►Text Mining for the Semantic Web
►Text Mining For Spam Filtering
►Text Visualization

Text Mining for Advertising

Massimiliano Ciaramita
Yahoo! Research Barcelona,
Barcelona, Spain

Synonyms

Content match; Contextual advertising; Sponsored search; Web advertising

Definition

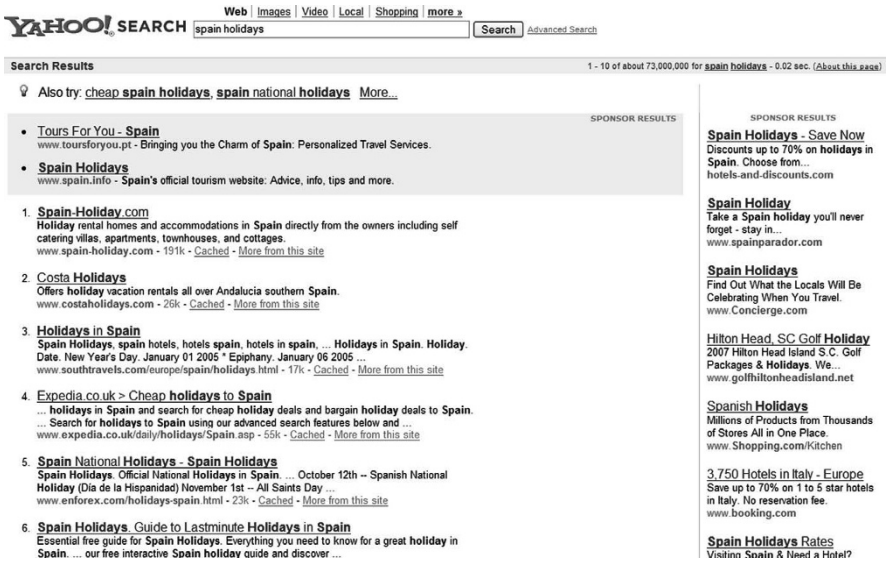
Text mining for advertising is an area of investigation and application of text mining and machine learning methods to problems such as Web advertising; e.g., automatically selecting the most appropriate ads with respect to a Web page, or query submitted to a search engine. Formally, the task can be framed as a ranking or matching problem where the unit of retrieval, rather than a Web page, is an advertisement. Most of the time ads have simple and homogeneous predefined textual structures, however, formats can vary and include audio and visual information. Advertising is a challenging problem due to several factors such as the economic nature of the transactions involved, engineering issues concerning scalability, and the inherent complexity of modeling the linguistic and multimedia content of advertisements.

Motivation and Background

The role of advertising in supporting and shaping the development of the Web has substantially increased over the past years. According to the Interactive Advertising Bureau (IAB), Internet advertising revenues in the USA totaled almost $8 billion in the first 6 months of 2006, a 36.7% increase over the same period in 2005, the last in a series of consecutive growths. Search, i.e., ads placed by Internet companies in Web pages or in response to specific queries, is the largest source of revenue, accounting for 40% of total revenue (Internet Advertising Bureau, 2006). The most important categories of Web advertising are *keyword match*, also known as *sponsored search* or *paid listing*, which places ads in the search results for specific queries (see Fain & Pedersen, 2006 for a brief history of sponsored search), and *content match*, also called *content-targeted advertising* or *contextual advertising*, which places ads in Web pages based on the page content. Figure 1 shows an example of sponsored search and ads are listed on the right side of the page.

Currently, most of the focus in Web advertising involves sponsored search, because matching based on keywords is a well-understood problem. Content match has greater potential for content providers, publishers, and advertisers, because users spend most of their time on the Web on content pages as opposed to search engine result pages. However, content match is a harder problem than sponsored search. Matching ads with query terms is to a certain degree straightforward, because advertisers themselves choose the keywords that characterize their ads that are matched against keywords chosen by users while searching. In contextual advertising, matching is determined automatically by the page content, which complicates the task considerably.

Advertising touches challenging problems concerning how ads should be analyzed, and how the accurately and efficiently systems select the best ads. This area of research is developing rapidly in information retrieval.



Text Mining for Advertising. Figure 1. Ads ranked next to a search results page for the query "Spain holidays"

How best to model the structure and components of ads, and the interaction between the ads and the contexts in that they appear are open problems. Information retrieval systems are designed to capture relevance, which is a basic concept in advertising as well. Elements of an ad such as text and images tend to be mutually relevant, and often (in print media for example) ads are placed in contexts that match a certain product at a topical level; e.g., an ad for sneakers placed on a sport news page. However, topical relevance is only one the basic parameters which determine a successful advertisement placement. For example, an ad for sneakers might be appropriate and effective on a page comparing MP3 players, because they may share a target audience (e.g., joggers) although they arguably refer to different topics, and it is possible they share no common vocabulary. Conversely, there may be ads that are topically similar to a Web page, but cannot be placed there because they are inappropriate. An example might be placing ads for a product in the page of a competitor.

The language of advertising is rich and sophisticated and can rely considerably on complex inferential processes. A picture of a sunset in an ad for life insurance carries a different implication than a picture of a sunset in an ad for beer. Layout and visual content are designed to elicit inferences, possibly hinging on cultural elements; e.g., the age, appearance, and gender of people in an ad affect its meaning. Adequate automatic modeling will likely involve, to a substantial degree, understanding the language of advertisement and the inferential processes involved (Vestergaard & Schroeder, 1985). Today this seems beyond what traditional information retrieval systems are designed to cope with. In addition, the global context can be captured only partially by modeling the text alone. As the Web evolves into an immense infrastructure for social interaction and multimedia information sharing the need for modeling structured "content" becomes more and more crucial. This applies to information retrieval and specifically to advertising. For this reason, the problem of content match is of particular interest and opens new problems and opportunities for interdisciplinary research.

Today, contextual advertising, the most interesting sub-task from a mining perspective, consists mostly in selecting ads from a pool to match the textual content of a particular Web page. Ads provide a limited amount of text: typically a few keywords, a title, and brief description. The ad-placing system needs to identify relevant ads, from huge ad inventories, quickly and efficiently based on this very limited amount of information. Current approaches have focused on augmenting the representation of the page to increase the chance of a match (Ribeiro-Neto, Cristo, Golgher, and de Moura, 2005), or by using machine learning to find complex ranking functions (Lacerda et al., 2006), or

by reducing the problem of content match to that of sponsored search by extracting keywords from the Web page (Yih et al., 2006). All these approaches are based on methods that quantify the similarity between the ad and the target page on the basis of traditional information retrieval notions such as cosine similarity and *tf.idf* features. The relevance of an ad for a page depends on the number of overlapping words, weighted individually and independently as a function of their individual distributional properties in the collection of documents or ads.

Structure of Learning Problem

The typical elements of an advertisement are a set of *keywords*, a *title*, a *textual description* and a hyperlink pointing to page, the *landing page*, relative to a product or service, etc. In addition, an ad has an *advertiser id* and can be part of a *campaign*, i.e., a subset of all the ads with same advertiser id. This latter information can be used, for example, to impose constraints on the number of ads to display relative to the campaign or advertiser. While this is possibly the most common layout, it is important to realize that ads structure can vary significantly and include relevant audio and visual content.

In general, the learning problem for an ad-placing system can be formalized as a ranking task. Let $\mathcal{A}$ be a set of ads, $\mathcal{P}$ the set of possible pages, and $\mathcal{Q}$ the set of possible queries. In keyword match, the goal is to find a function $F : \mathcal{A} \times \mathcal{Q} \to \mathbb{R}$; e.g., a function that counts the number of individual common terms or n-grams of such terms. In content match, the objective is to find a function $F : \mathcal{A} \times \mathcal{P} \to \mathbb{R}$. The keyword match problem is to a certain extent straightforward and amounts to matching small set of terms defined manually by both the user and the advertiser. The content match task shares with the former task the peculiarities of the units of retrieval (the ads), but introduces new and interesting issues for text mining and learning because of the more complex conditioning environment, the Web page content, which needs to modeled automatically.

In general terms, an ad can be represented as a feature vector $\mathbf{x} = \Phi(a_i, p_j)$ such that $a_i \in \mathcal{A}$, $p_j \in \mathcal{P}$, and given a d-dimensional feature space $\mathcal{X} \subset \mathbb{R}^d$, $\Phi : \mathcal{A} \times \mathcal{P} \to \mathcal{X}$. In the traditional machine learning setting, one introduces a weight vector $\alpha \in \mathbb{R}^d$ which quantifies each feature's contribution individually. The vector's weights can be learned from manually edited rankings (Lacerda et al., 2006; Ribeiro-Neto et al., 2005) or from click-through data as in search results optimization (Joachims, 2002). In the case of a linear classifier the score of an ad-target page pair $\mathbf{x}_i$ would be:

$$F(\mathbf{x}; \alpha) = \sum_{s=1}^{d} \alpha_s x_s. \quad (1)$$

Several methods can be used to learn similar or related models such as perceptron, SVM, boosting, etc. Constraints on the number of advertisers or campaigns could be easily implemented as post-ranking filters on the top of the ranked list of ads or included in a suitable objective function.

A basic model for ranking ads can be defined in the vector space model for information retrieval, using a ranking function based on cosine similarity, where ads and target pages are represented as vectors of terms weighted by fixed schemes such as *tf.idf*. If only one feature is used, the cosine based on *tf.idf* between the ad and the page, a standard vector space model baseline is obtained, which is at the base of the ad-placing ranking functions variants proposed by (Ribeiro-Neto et al., 2005) Recent work has shown that machine learning-based models are considerably more accurate than such baselines. However, as in document retrieval, simple feature maps which include mostly coarse-grained statistical properties of the ad-page pairs, such as *tfidf-based* cosine, are the most desirable for efficiency and bias reasons. Properties of the different components of the ad can be used and weighted in different ways, and soft or hard constraints introduced to model the presence of the ads keyword in the Web page. The design space for ad-place systems is vast and still little explored. All systems presented so far in the literature make use of manually annotated data for training and/or evaluating a model.

Structure of Learning Systems

Web advertising presents peculiar engineering and modeling challenges and has motivated research in different areas. Systems need to be able to deal in real time with huge volumes of data and transactions involving billions of ads, pages, and queries. Hence several engineering constraints need to be taken into account; efficiency and computational costs are crucial factors in the

choice of matching algorithms (The Yahoo! Research Team, 2006). Ad-placing systems might require new global architecture design; e.g., Attardi et al. (2004) proposed an architecture for information retrieval systems that need to handle large-scale targeted advertising based on an information filtering model. The ads that appear on Web pages or search results pages will ultimately be determined taking into account the expected revenues and the price of the ads. Modeling the microeconomics factors of such processes is a complex area of investigation in itself (Feng et al., 2005).

Another crucial issue is the evaluation of the effectiveness of the ad-placing systems. Studies have emphasized the impact of the quality of the matching on the success of the ad in terms of click-through rates (Gallagher et al., 2001; Sherman & Deighton, 2001). Although ►click-through rates (CTRs) provide a traditional measure of effectiveness, it has been found that ads can be effective even when they do not solicit any conscious response and that the effectiveness of the ad is mainly determined by the level of congruency between the ad and the context in which it appears (Yoo, 2006).

Keyword Extraction Approaches

Since the query-based ranking problem is better understood than contextual advertising, one way of approaching the latter would be to represent the content page as a set of keywords and then ranking the ads based on the keywords extracted from the content page. Carrasco et al. (2003) proposed clustering of bi-partite advertiser-keyword graphs for keyword suggestion and identifying groups of advertisers. Yih, Goodman, & Carvalho (2006) proposed a system for keyword extraction from content pages. The goal is to determine which keywords, or key phrases, are more relevant in a Web page. Yih et al. develop a supervised approach to this task from a corpus of pages where keywords have been manually identified. They show that a model learned with ►logistic regression outperforms traditional vector models based on fixed *tf.idf* weights. The most useful features to identify good keywords efficiently are, in this case, term frequency and document frequency of the candidate keywords, and particularly the frequency of the candidate keyword in a search engine query log. Other useful features include the similarity of the candidate with the page's URL and the length, in number of words, of the candidate keyword. In terms of feature representation thus, they propose a feature map $\Phi : \mathcal{A} \rightarrow \mathcal{Q}$, which represent a Web page as a set of keywords. The accuracy of the best learned system is 30.06%, in terms of the top predicted keyword being in the set of manually generated keywords for a page, against 13.01% of the simpler *tf.idf* based model. While this approach is simple to apply, it remains to be seen how accurate it is at identifying good ads for a page. It identifies potentially useful sources of information in automatically-generated keywords. An interesting related finding concerning keywords is that longer keywords, about four words long, lead to increased click-through rates (OneUpWeb, 2005).

The Vocabulary Impedance Problem

(Ribeiro-Neto et al., 2005) introduced an approach to content match which focuses on the vocabulary mismatch problem. They notice that there tends to be not enough overlap in the text of the ad and the target page to guarantee good accuracy; they call this the *vocabulary impedance problem*. To overcome this limitation they propose to generate an augmented representation of the target page by means of a Bayesian model previously applied to document retrieval (Ribeiro-Neto & Muntz, 1996). The expanded vector representation of the target page includes a significant number of additional words which can potentially match some of the terms in the ad. They find that such a model improves over a standard vector space model baseline, evaluated by means of 11-point average precision on a test bed of 100 Web pages, from 0.168 to 0.253. One possible shortcoming of such an approach is that generating the augmented representation involves crawling a significant number of additional related pages. It has also been argued (Yih et al., 2006) that this model complicates pricing of the ads because the keywords chosen by the advertisers might not be present in the content of the matching page.

Learning with Genetic Programming

Lacerda et al. (2006) proposed to use machine learning to find good ranking functions for contextual advertising. They use the same data-set described in Ribeiro-Neto et al. (2005), but use part of the data for training a model and part for evaluation purposes. They use a genetic programming algorithm to select a

ranking function which maximizes the average precision on the training data. The resulting ranking function is a nonlinear combination of simple components based on the frequency of ad terms in the target page, document frequencies, document length, and size of the collections. Thus, in terms of the feature representation defined earlier, they choose a feature map which extracts traditional features from the ad-page pair, but then apply then genetic programming methods to select complex nonlinear combinations of such features that maximize a fitness function based on average precision. Lacerda et al. (2006) find that the ranking functions selected in this way are considerably more accurate than the baseline proposed in Ribeiro-Neto et al. (2005); in particular, the best function selected by genetic programming achieves an average precision at position three of 0.508, against 0.314 of the baseline.

Semantic Approaches to Contextual Advertising

The most common approaches to contextual advertising are based on matching terms between the ad and the content page. Broder, Fontoura, Josifovski, and Riedel (2007) notice that this approach (which they call the "syntactic—" model), can be improved by adopting a matching model which additionally takes into account topical proximity; i.e., a "semantic" model. In their model the target page and the ad are classified with respect to a taxonomy of topics. The similarity of ad and target page estimated by means of the taxonomy provides an additional factor in the ads ranking function. The taxonomy, which has been manually built, contains approximately 6,000 nodes, where each node represents a set of queries. The concatenation of all queries at each node is used as a meta-document, ads and target pages are associated with a node in the taxonomy using a nearest neighbor classifier and *tf.idf* weighting. The ultimate score of an ad a_i for a page p is a weighted sum of the taxonomy similarity score and the similarity of a_i and p based on standard syntactic measures (vector cosine). On evaluation, Broder et al. (2007) report a 25% improvement for mid-range recalls of the syntactic-semantic model over the pure syntactic one.

Cross References

►Boosting
►Genetic Programming
►Information Retrieval
►Perceptron
►SVM
►TF–IDF
►Vector Space Model

Recommended Reading

Attardi, G., Esuli, A., & Simi, M. (2004). Best bets, thousands of queries in search of a client. In *Proceedings of the 13th international conference on World Wide Web, alternate track papers & posters*. New York: ACM Press.

Broder, A., Fontoura, M., Josifovski, V., & Riedel, L. (2007). A semantic approach to contextual advertising. In *Proceedings of the 30th annual international ACM SIGIR conference on research and development in information retrieval*. New York: ACM Press.

Carrasco, J. J., Fain, D., Lang, K., & Zhukov, L. (2003). Clustering of bipartite advertiser-keyword graph. In *Workshop on clustering large datasets, IEEE conference on data mining*. New York: IEEE Computer Society Press.

Fain, D., & Pedersen, J. (2006). Sponsored search: A brief history. In *Proceedings of the 2nd workshop on sponsored search auctions*. Web Publications.

Feng, J., Bhargava, H., & Pennock, D. (2005). Implementing sponsored search in Web search engines: Computational evaluation of alternative mechanisms. *Informs Journal on Computing* (forthcoming).

Gallagher, K., Foster, D., & Parsons, J. (2001). The medium is not the message: Advertising effectiveness and content evaluation in print and on the Web. *Journal of Advertising Research, 41*(4), 57–70.

Internet Advertising Bureau (IAB). (2006). *IAB Internet Advertising Revenue Report*. http://www.iab.net/resources/adrevenue/pdf/IAB_PwC%202006Q2.pdf

Joachims, T. (2002). Optimizing search engines using clickthrough data. In *Proceedings of the ACM conference on knowledge discovery and data mining (KDD)*. New York: ACM Press.

Lacerda, A., Cristo, M., Gonçalves, M. A., Fan, W., Ziviani, N., & Ribeiro-Neto, B. (2006). Learning to advertise. In *Proceedings of the 29th annual international ACM SIGIR conference on research and development in information retrieval* (pp. 549–556). New York: ACM Press.

OneUpWeb (2005). *How keyword length affects conversion rates*. http://www.oneupweb.com/landing/keywordstudy_landing.htm.

Ribeiro-Neto, B., Cristo, M., Golgher, P. B., & de Moura, E. S. (2005). Impedance coupling in content-targeted advertising. In *Proceedings of the 28th annual international ACM SIGIR conference on research and development in information retrieval* (pp. 496–503). New York: ACM Press.

Ribeiro-Neto, B., & Muntz, R. (1996). A belief network model for IR. In *Proceedings of the 19th annual international ACM SIGIR conference on research and development in information retrieval* (pp. 253–260). New York: ACM Press.

Sherman, L., & Deighton, J. (2001). Banner advertising: Measuring effectiveness and optimizing placement. *Journal of Interactive Marketing, 15*(2), 60–64.

The Yahoo! Research Team. (2006). Content, metadata, and behavioral information: Directions for Yahoo! Research. *IEEE Data Engineering Bulletin, 29*(4), 10–18.

Vestergaard, T., & Schroeder, T. (1985). *The language of advertising.* Oxford: Blackwell.

Yih, W., Goodman, J., & Carvalho, V. R. (2006). Finding advertising keywords on web pages. In *Proceedings of the 15th international conference on World Wide Web* (pp. 213–222). New York: ACM Press.

Yoo, C. Y. (2006). *Preattentive processing of web advertising.* PhD thesis, University of Texas at Austin.

Text Mining for News and Blogs Analysis

Bettina Berendt
Katholieke Universiteit Leuven, Heverlee, Belgium

Definition

News and blogs are two types of media that generate and offer informational resources. *News* is any information whose revelation is anticipated to have an intellectual or actionable impact on the recipient. The dominant type of news in text analysis is that pertaining to current events. Originally referring to print-based news from press agencies or end-user news providers (like individual newspapers or serials), it now increasingly refers to Web-based news in the online editions of the same providers or in online-only news media. The term is generally understood to denote only the reports in news media, not opinion or comment pieces. A *blog* is a (more or less) frequently updated publication on the Web, sorted in (usually reverse) chronological order of the constituent *blog posts*. The content may reflect any interest including personal, journalistic, or corporate. Blogs were originally called weblogs. To avoid confusion with web server log files that are also known by this term, the abbreviation "blog" was coined and is now commonly used.

News and blogs consist of textual and (in some cases) pictorial content, and, when Web-based, may contain additional content in any other format (e.g., video, audio) and hyperlinks. They are indexed by time and structured into smaller units: news media into articles, blogs into blog posts. In most news and blogs, textual content dominates. Therefore, text analysis is the most often applied form of knowledge discovery. This comprises tasks and methods from data/text mining, ►information retrieval, and related fields. In accordance with the increasing convergence of these fields, this article refers to all of them as ►*text mining*.

Motivation and Background

News and blogs are today's most common sources for learning about current events and also, in the case of blogs, for uttering opinions about current events. In addition, they may deal with topics of more long-term interest. Both reflect and form societies', groups' and individuals' views of the world, fast or even instantaneous with the events triggering the reporting.

However, there are differences between these two types of media regarding authoring, content, and form. News is generally authored by people with journalistic training who abide by journalistic standards regarding the style and language of reporting. Topics and ways of reporting are circumscribed by general societal consensus and the policies of the news provider. In contrast, everybody with Internet access can start a blog, and there are no restrictions on content and style (beyond the applicable types of censorship). Thus, blogs offer end users a wider range of topics and views on them. On the one hand, this implies that journalistic blogs, which correspond most closely to news, are only one type of blogs. Other frequent types are diary-like personal blogs, corporate blogs for public relations, and blogs focusing on special-interest topics. On the other hand, their comparative lack of restrictions has helped to establish blogs as an important alternative source of information, as a form of grassroots journalism that may give rise to a counterpublic. An example are the *warblogs* published during the early years of the Iraq War (2003+) by independent sources (often civilian individuals) both in the West and in the Middle East.

These application characteristics lead to various linguistic and computational challenges for text-mining analyses of news and blogs:

- *Indexing, taxonomic categorization, partial redundancy, and data streams:* News is indexed by time

and by source (news agency or provider). In a multisource corpus, many articles published at about the same time (in the same or in other languages) describe the same events. Over time, a story may develop in the articles. Such multiple reporting and temporal structures are also observed for popular topics in blogs.

- *Language and meaning:* News is written in clear, correct, "objective," and somewhat schematized language. Usually, the start of a news article summarizes the whole article (feeds are a partial analogue of this in blogs). Information from external sources such as press agencies is generally reused rather than referenced. In sum, news makes fewer assumptions about the reader's background and context knowledge than many other texts.
- *Nonstandard language and subjectivity:* The language in blogs ranges from high-quality "news-like" language through poor-quality, restricted-code language with many spelling and grammatical errors to creative, sometimes literary, language. Jargon is very common in blogs, and new linguistic developments are adopted far more quickly than could be reflected in external resources such as lexica. Many blog authors strive not for objectivity, but for subjectivity and emotionality.
- *Thematic diversity and new forms of categorization:* News are generally categorized by topic area ("politics," "business," etc.). In contrast, a blog author may choose to write about differing, arbitrary topics. When blogs are labeled, it is usually not with reference to a stable, taxonomic system, but with an arbitrary number of *tags*: free-form, often informal labels chosen by the user.
- *Social structure and its impact on content and meaning:* The content of a blog (post) is often not contained in the text alone. Rather, blog software supports "Social Web" behavior, and bloggers practice it: multiway communication rather than broadcasting, and semantics-inducing referencing of both content and people. Specifically, hyperlinks to other resources provide not only context but also content; "blogrolls" (hyperlinks to other blogs) supply context in terms of other blogs/bloggers recommended by the author; comments to blog posts are integral part of the communication that the post triggered. "Trackback" links, indicating hyperlinks set to the blog, may be automatically added by blogging software and thus, create a dynamic citation context.

Structure of the Learning System

Tasks

News and blogs may serve many different interests, for example, those of:

- End users who want to know what is happening in given universes of discourse, to follow developments within these areas, or to identify sources that are of long-term interest to them. These users differ by their preferences, their educational level, the purposes of their searches, and other factors. This calls for search engines, temporal analyses, topic identification, personalization, and related functionalities.
- Companies that want to learn about their target groups' views and opinions of their products and activities, detect trends and make predictions. Similar market research may be carried out by nonprofit organizations or politicians.
- People who use blogs to gain insights about specific blog author(s) as background knowledge for decisions on befriending, hiring, or insuring these individuals (see Nowson & Oberlander 2007) on the textual analysis of blogs for determining personality features).

The literature on news and blogs analysis reflects these and other possible uses. A number of standard tasks are emerging, furthered by the competitions at events such as the Topic Detection and Tracking (TDT) research program and workshops (http://www.itl.nist.gov/iad/mig/tests/tdt, Allan, 2002), Text Retrieval Conference (TREC, http://trec.nist.gov/, e.g., MacDonald, Ounis, & Soboroff, 2007), and Document Understanding/Text Analysis Conference (DUC/TAC, http://www.nist.gov/tac/). Other initiatives also provide and encourage the usage of standardized real-world ►datasets, but instigate research on novel questions by standardizing neither tasks nor ►algorithm evaluation. Prominent examples are the Reuters-21578 dataset, which is not only a collection of newswire articles but also the most classical dataset for text mining in general (http://kdd.ics.uci/edu/databases/reuters21578/reuters21578.html), and the blog datasets provided by

International Conference on Weblogs and Social Media (ICWSM, http://www.icwsm.org) and its precursors.

Tasks can be grouped by different criteria:

- *Use case and type of result:* description and prediction (supervised or unsupervised, may include topic identification, tracking, and/or novelty detection); search (ad hoc or filtering); recommendation (of blogs, blog posts, or tags); summarization
- *Higher-order characterization to be extracted:* topic; opinion
- *Time dimension:* nontemporal; temporal (stream mining); multiple streams (e.g., in different languages, see ▶cross-lingual text mining)
- *User adaptation:* none (no explicit mention of user issues and/or general audience); customizable; personalized

Since the beginning of news mining in the 1990s and of blog mining in the early 2000s, more complex combinations of these dimensions have been explored. Examples include (a) the TDT research program (1997/1998–2004) required an explicit focus on temporal analyses and called for topic description and prediction in a news stream; (b) "bursty" topics in a stream of blogs were used to predict peaks in a stream of online sales (Gruhl, Guha, Kumar, Novak, & Tomkins, 2005); (c) the role of opinion mining as a key question in blog analysis was manifested by the first TREC blog track in 2006 (see also MacDonald et al., 2007); it is now a standard task, also for analysing microblogs (Jansen, Zhang, Sobel, & Chowdury, 2009); (d) a recommendation method on a document stream based on tracking multiple topics over time, personalized to a user whose interests may change over time was developed in (Pon, Cardenas, Buttler, & Critchlow, 2007); (e) in (Lu & Zhai, 2008), opinions were summarized in a set of non–time-indexed texts, for a general audience; and (f) in (Subašić & Berendt, 2008), bursty topics in a news stream were summarized into graph patterns that can be interactively explored and customized.

Another important task is spam detection and blocking (Kolari, Java, Finin, Oates, & Joshi, 2006). While basically nonexistent in news mining (news are identified by their sources, which are "white-listed" and thus credible), spamming has become a severe problem in the blogosphere, ranging from comment spam via "flogs" (e.g., ghostwritten by a marketing department but pretending to be an end user), to "splogs" (artificially created blogs used to increase the visibility and search engine rankings of associated sites). (cf. ▶text mining for spam detection).

Solution Approaches

Solution approaches are based on general ▶data-mining methods and adapted to the conceptual specifics of news and blogs and their mining tasks (see list of tasks above). Methods include (▶document) classification and ▶clustering, latent-variable techniques such as (P)LSA or LDA (cf. ▶feature construction), ▶mixture models, ▶time series, and ▶stream mining methods. Named-entity recognition may be an important part or companion of topic detection (cf. ▶information extraction). Opinion mining often relies on word class identification and ▶part-of-speech tagging, and it generally employs lexica (e.g., of typical opinionated words and their positive or negative polarity). Data cleaning is similar to that of other Web documents; in particular, it requires the provision or learning of wrappers for removing markup elements.

In addition, many solution approaches exploit the specific formatting and/or linguistic features of blogs. For example, to improve the retrieval of blogs about a queried event, the format elements "timestamp" and "number of comments" can be treated as indicators of increased topical relevance and likelihood of being opinionated, respectively (Mishne, 2007). Structural elements of blogs such as length and representation in post title versus post body have been used for blog distillation (filtering out those blogs that are principally devoted to a topic rather than just mentioning it in passing) (Weerkamp, Balog, & de Rijke, 2008). Text-based statistical topic modeling can be enhanced by ▶regularizing it with the (e.g., social) network structure associated with blog data (Mei, Cai, Zhang, & Zhai, 2008) (cf. ▶link mining and link discovery). However, many blogs are not strongly hyperlinked – but tags also carry "Social Web" information: A combination of text clustering and tag analysis can serve to identify topics as well as the blogs that are on-topic and likely to retain this focus over time (Hayes, Avesani, & Bojars, 2007).

Due to blog writing style, standard indicators of relevance may not be applicable. For example, a term's

TF.IDF score, which is commonly used as a ►weight in a ►feature vector representing the document, assumes that important terms are mentioned (frequently) in the document and infrequently elsewhere. However, blogs often rely on implicit context – established by hyperlinks or by the history of the discussion. Solution proposals include the integration of the text from previous blog posts with the same tag (Hayes et al., 2007); in addition, terms from hyperlinked documents could be taken into account. In addition, while blogs may be more opinionated than news texts, their language may make it more difficult to extract topics and argumentation vis-à-vis that topic. Specifically, blogs often contain irony and other indirect uses of language for expressing appreciation or discontent. The "emotional charge" of a text has, therefore, been proposed as a better target for blog classification (Gamon et al., 2008).

Viewed in relation to each other, news and blogs pose some additional challenges for automated analysis and text mining. Several studies (e.g., Adamic & Glance 2005) address questions such as: How are blogs linked to news media (and possibly vice versa)? Do they form a coherent whole, "the blogosphere," or rather a loose connection of mutually unrelated, political, national, linguistic, etc., blogospheres? What are the topics investigated in blogs versus news? Are stories reported by news or blogs first, and how does the other side follow up reporting? In general, how do blogs and news refer to and contextualize each other (e.g., Gamon et al. 2008; Berendt & Trümper 2009; Leskovec, Backstrom, & Kleinberg 2009)?

Finally, text mining faces a further challenge: while news are always meant to be read, many blogs are not (e.g., because they are a personal journal) – even if they are accessible over the Web. This raises the question of whether text mining could or should become privacy-aware (cf. ►privacy-related aspects and techniques).

Recommended Reading

Adamic, L., & Glance, N. (2005). The political blogosphere and the 2004 U.S. election: Divided they blog. In E. Adar, N. Glance, & M. Hurst (Eds.), *Proceedings of the WWW 2005 2nd annual workshop on the weblogging ecosystem: Aggregation, analysis and dynamics.* Chiba, Japan. http://doi.acm.org/10.1145/1134271.1134277 New York, NY, 36–43.

Allan, J. (Ed.). (2002). *Topic detection and tracking: Event-based information organization*. Norwell, MA: Kluwer Academic Publishers.

Berendt, B., & Trümper, D. (2009). Semantics-based analysis and navigation of heterogeneous text corpora: The *Porpoise* news and blogs engine. In I.-H. Ting & H.-J. Wu (Eds.), *Web mining applications in e-commerce and e-services* Berlin: Springer.

Gamon, M., Basu, S., Belenko, D., Fisher, D., Hurst, M., & König, A. C. (2008). BLEWS: Using blogs to provide context for news articles. In E. Adar, M. Hurst, T. Finin, N. Glance, N. Nicolov, B. Tseng, & F. Salvetti (Eds.), *Proceedings of the second international conference on weblogs and social media (ICWSM'08).* Seattle, WA. Menlo Park, CA. http://www.aaai.org/Papers/ICWSM/2008/ICWSM08-015.pdf

Gruhl, D., Guha, R., Kumar, R., Novak, J., & Tomkins, A. (2005). The predictive power of online chatter. In R. Grossman, R. J. Bayardo, & K. P. Bennett (Eds.), *Proceedings of the 11th ACM SIGKDD international conference on knowledge discovery and data mining.* Chicago, IL. New York, NY.

Hayes, C., Avesani, P., & Bojars, U. (2007). An analysis of bloggers, topics and tags for a blog recommender system. In B. Berendt, A. Hotho, D. Mladenič, & G. Semeraro (Eds.), *From web to social web: Discovering and deploying user and content profiles. LNAI 4737.* Berlin: Springer.

Jansen, B.J., Zhang, M., Sobel, K., & Chowdury, A. (2009). Twitter Power: Tweets as electronic word of mouth. *Journal of the American Society for Information Science and Technology, 60*(11): 2169–2188.

Kolari, P., Java, A., Finin, T., Oates, T., & Joshi, A. (2006). Detecting spam blogs: A machine learning approach. In *Proceedings of the 21st national conference on artificial intelligence.* Boston: AAAI.

Leskovec, J., Backstrom, L., & Kleinberg, J. (2009). Meme-tracking and the dynamics of the news cycle. In J.F. Elder IV, F. Fogelman-Soulié, P.A. Flach, & M.J. Zaki (Eds.), *Proceedings of the 15th ACM SIGKDD international conference on knowledge discovery and data mining,* Paris, France. New York, NY.

Lu, Y., & Zhai, C. (2008). Opinion integration through semi-supervised topic modeling. In J. Huai & R. Chen (Eds.), *Proceeding of the 17th international conference on world wide web (WWW'08).* Beijing, China. New York, NY.

MacDonald, C., Ounis, I., & Soboroff, I. (2007). Overview of the TREC-2007 blog track. In E. M. Voorhees & L. P. Buckland (Eds.), *NIST special publication: SP 500-274: The sixteenth text REtrieval conference (TREC 2007) Proceedings.* Gaithersburg, MD. http://trec.nist.gov/pubs/trec16/papers/BLOG.OVERVIEW16.pdf

Mei, Q., Cai, D., Zhang, D., & Zhai, C. (2008). Topic modeling with network regularization. In J. Huai & R. Chen (Eds.), *Proceeding of the 17th international conference on world wide web (WWW'08)* Beijing, China. New York, NY.

Mishne, G. (2007). Using blog properties to improve retrieval. In N. Glance, N. Nicolov, E. Adar, M. Hurst, M. Liberman, & F. Salvetto (Eds.), *Proceedings of the international conference on weblogs and social media (ICWSM).* Boulder, CO. http://www.icwsm.org/papers/paper25.html

Nowson, S., & Oberlander, J. (2007). Identifying more bloggers: Towards large scale personality classification of personal weblogs. In N. Glance, N. Nicolov, E. Adar, M. Hurst, M. Liberman, & F. Salvetto (Eds.), *Proceedings of the international conference on weblogs and social media (ICWSM).* Boulder, CO. http://www.icwsm.org/papers/paper4.html.

Pon, R. K., Cardenas, A. F., Buttler, D., & Critchlow, T. (2007). Tracking multiple topics for finding interesting articles. In P. Berkhin,

R. Caruana, & X. Wu (Eds.), *Proceedings of the 13th ACM SIGKDD international conference on knowledge discovery and data mining*. San Jose, CA. New York, NY.

Subašić, I., & Berendt, B. (2008). Web mining for understanding stories through graph visualisation. In F. Giannotti, D. Gunopoulos, F. Turini, C. Zaniolo, N. Ramakrishnan, & X. Wu (Eds.), *Proceedings of the IEEE international conference on data mining (ICDM 08)*. Pisa, Italy. Los Alamitos, CA.

Weerkamp, W., Balog, K., & de Rijke, M. (2008). Finding key bloggers, one post at a time. In M. Ghallab, C. D. Spyropoulos, N. Fakotakis, & N. Avouris (Eds.), *Proceedings of the 18th European conference on artificial intelligence (ECAI 2008)*. Greece: Patras. Amsterdam, The Netherlands.

Text Mining for Spam Filtering

Aleksander Kołcz
Microsoft One Microsoft Way,
Redmond, WA, USA

Synonyms

Commerical email filtering; Junk email filtering; Spam detection; Unsolicited commercial email filtering

Definition

Spam filtering is the process of detecting unsolicited commercial email (UCE) messages on behalf of an individual recipient or a group of recipients. Machine learning applied to this problem is used to create discriminating models based on labeled and unlabeled examples of spam and nonspam. Such models can serve populations of users (e.g., departments, corporations, ISP customers) or they can be personalized to reflect the judgments of an individual. An important aspect of spam detection is the way in which textual information contained in email is extracted and used for the purpose of discrimination.

Motivation and Background

Spam has become the bane of existence for both Internet users and entities providing email services. Time is lost when sifting through unwanted messages and important emails may be lost through omission or accidental deletion. According to various statistics, spam constitutes the majority of emails sent today and a large portion of emails actually delivered. This translates to large costs related to bandwidth and storage use. Spam detection systems help to alleviate these issues, but they may introduce problems of their own, such as more complex user interfaces, delayed message delivery, and accidental filtering of legitimate messages. It is not clear if any one approach to fighting spam can lead to its complete eradication and a multitude of approaches have been proposed and implemented. Among existing techniques are those relying on the use of supervised and unsupervised machine learning techniques, which aim to derive a model differentiating spam from legitimate content using textual and nontextual attributes. These methods have become an important component of the antispam arsenal and draw from the body of related research such as text classification, fraud detection and cost-sensitive learning. The text mining component of these techniques is of particular prominence given that email messages are primarily composed of text. Application of machine learning and data mining to the spam domain is challenging, however, due, among others, to the adversarial nature of the problem (Dalvi, Domingos, Sanghai, & Verma, 2004; Fawcett, 2003).

Structure of the Learning System

Overview

A machine-learning approach to spam filtering relies on the acquisition of a learning sample of email data, which is then used to induce a classification or scoring model, followed by tuning and setup to satisfy the desired operating conditions. Domain knowledge may be injected at various stages into the induction process. For example, it is common to *a priori* specific features that are known be highly correlated with the spam label, e.g., certain patterns contained in email headers or certain words or phrases. Depending on the application environment, messages classified as spam are prevented from being delivered (e.g., are blocked or "bounced"), or are delivered with a mechanism to alert users to their likely spam nature. Filter deployment is followed by continuous evaluation of its performance, often accompanied by the collection of error feedback from its users.

Data Acquisition

A spam filtering system relies on the presence of labeled training data, which are used to induce a model of what constitutes spam and what is legitimate email. Spam detection represents a two-class problem, although it may sometimes be desired to introduce special handling

of messages for which a confident decision, either way, cannot be made. Depending on the application environment, the training data may represent emails received by one individual or a group of users. Ideally, the data should correspond to a uniform sample acquired over some period of time preceding filter deployment. Typical problems with data collection revolve around privacy issues, whereby users are unwilling to donate emails of personal or sensitive nature. Additionally, labeling mistakes are common where legitimate emails may be erroneously marked as spam or vice versa. Also, since for certain types of emails, the spam/legitimate distinction is personal, one may find that the same message content is labeled in a conflicting manner by different users (or even by the same user at different times). Therefore, data cleaning and conflict resolution techniques may need to be deployed, especially when building filters that serve a large and diverse user population.

Due to privacy concerns, few large publicly email corpora exist. The ones created for the TREC Spam Track (TREC data is available from: http://plg.uwaterloo.ca/~gvcormac/treccorpus/). stand out in terms of size and availability of published comparative results.

Content Encoding and Deobfuscation

Spam has been evolving in many ways over the course of time. Some changes reflect the shift in content advertised in such messages (e.g., from pornography and pharmaceuticals to stock schemes and *phish*). Others reflect the formatting of content. While early spam was sent in the form of plain text, it subsequently evolved into more complex HTML, with deliberate attempts to make extraction of meaningful textual features as difficult as possible. Typically, obfuscation (a list of obfuscation techniques is maintained at http://www.jgc.org/tsc.html) aims at

(a) Altering the text extracted from the message for words visible to the user (e.g., by breaking up the characters in message source by HTML tags, encoding the characters in various ways, using character look-alikes, wrapping the display of text using script code executed by the message viewer). This tactic is used to hide the message "payload."
(b) Adding content that is not visible to the user (e.g., using the background color or zero-width font to render certain characters/words). This tactic typically attempts to add "legitimate content."
(c) Purposeful misspelling of words known to be fairly incriminating (e.g., *Viagra* as *V1agr@*), in a way that allows the email recipient to still understand the spammer's message.

The line of detection countermeasures aiming at preventing effective content extraction continues in the form of image spam, where the payload message is encoded in the form of an image that is easily legible to a human but poses challenges to an automatic content extraction system. To the extent that rich and multimedia content gets sent out by legitimate users in increasing proportions, spammers are likely to use the complexity of these media to obfuscate their messages even further. The very fact that obfuscation is attempted, however, provides an opportunity for machine learning techniques to use obfuscation presence as a feature. Thus, even if payload content cannot be faithfully decoded, the very presence of elaborate encoding may help in identifying spam.

Feature Extraction and Selection

An email message represents a semistructured document, commonly following the rfc822 standard (www.faqs.org/rfcs/rfc822.html). Its header consists of fields indicative of formatting, authorship, and delivery information, while its body contains the actual content being sent. There can be little correctness enforcement of the header fields and spamming techniques often rely on spoofing and forging of the header data, although this may provide evidence of tempering. Many early approaches to detect spam depended predominantly on hand-crafted rules identifying inconsistencies and peculiarities of spam email headers. Manually or automatically generated header features continue to be relevant even when other features (e.g., message text) are considered.

Given that an email message tends to be primarily text, features traditionally useful in text categorization have also been found useful in spam detection. These include individual words, phrases, character n-grams, and other textual components (Siefkes, Assis, Chhabra, & Yerazunis, 2004). Natural language processing (NLP) techniques such as stemming, stop-word removal, and case folding are also sometimes applied to

normalize the features further. Text extraction is often nontrivial due to the application of content obfuscation techniques. For example, standard lexical feature extractors may need to be strengthened to correctly identify word boundaries (e.g., in cases where groups of characters within a word are separated by zero-width HTML tags).

Extraction of features from nontextual attachments (e.g., images, audio, and video) is also possible but tends to be more computationally demanding. Other types of features capture the way a message if formatted, encoded in HTML, composed of multiple parts, etc.

Although nontextual features have different properties than text, it is common practice to combine them with textual features and present a single unified representation to the classifier. Indeed, some approaches make no distinction between text and formatting even during the process of feature extraction, and apply pattern discovery techniques to identifying complex features automatically (Rigoutsos & Huynh, 2004). The advantage of such techniques is that they do not require rich domain knowledge and can discover new useful patterns. Due to the large space of possible patterns they can potentially be computationally expensive. However, even the seemingly simplistic treatment of an email message as a plain-text document with "words" delimited by white space often leads to very good results.

Even though typical text documents are already very sparse, the problem is even more pronounced for the email medium due to frequent misspelling and deliberate randomization performed by spammers. Insisting on using all such variations may lead to overfitting for some classifiers, and it leads to large filter memory footprints that are undesirable from an operational standpoint. However, due to the constantly changing distribution of content, it may be dangerous to rely on very few features. Traditional approaches to feature selection based on measures such as Information Gain have been reported as successful in the spam filtering domain, but even simple rudimentary attribute selection based on removing very rare and/or very frequent features tends to work well.

There are a number of entities that can be extracted from message text and that tend to be of relevance in spam detection. Among others, there are telephone numbers and URLs. In commercial email and in spam, these provide a means of ordering products and services and thus, offer important information for vendor and campaign tracking. Detection of signature and mailing address blocks can also be of interest, even if only to indicate their presence or absence.

Learning Algorithms

A variety of learning algorithms have been applied in the spam filtering domain. These range from linear classifiers such as Naive Bayes (Metsis, Androutsopoulos, & Paliouras, 2006), logistic regression (Goodman & Yih, 2006), or linear support vector machines (Drucker, Wu, & Vapnik, 1999; Kołcz & Alspector, 2001; Sculley & Wachman, 2007) to nonlinear ones such as boosted decision trees (Carreras & Màrquez, 2001). Language modeling and statistical compression techniques have also been found quite effective (Bratko, Cormack, Filipic, Lynam, & Zupan, 2006). In general, due to the high dimensionality of the feature space, the classifier chosen should be able to handle tens of thousand, or more, attributes without overfitting the training data.

It is usually required that the learned model provides a scoring function, such that for email message x $\text{score}(x) \in R$, with higher score values corresponding to higher probability of the message being spam. The score function can also be calibrated to represent the posterior probability $P(\text{spam}|x) \in [0,1]$, although accurate calibration is difficult due to constantly changing class and content distributions. The scoring function is used to establish a decision rule:

$$\text{score}(x) \geq th \rightarrow \text{spam}$$

where the choice of the decision threshold th is driven by the minimization of the expected cost. In the linear case, the scoring function takes the form

$$\text{score}(x) = w \cdot x + b$$

where w is the weight vectors, and x is a vector representation of the message. Sometimes scores are normalized with a monotonic function, e.g., to give an estimate of the probability of x being spam.

Linear classifiers tend to provide sufficiently high accuracy, which is also consistent with other application domains involving the text medium. In particular,

many variants of the relatively simple Naive Bayes classifier have been found successful in detecting spam, and Naive Bayes often provides a baseline for systems employing more complex classification algorithms (Metsis et al., 2006).

One Model versus Multiple Models

It often pays off to combine different types of classifiers (even different linear ones) in a sequential or parallel fashion to benefit from the fact that different classifiers may provide an advantage in different regions of the feature space. Stacking via ▶linear regression has been reported to be effective for this purpose (Sakkis et al., 2001; Segal, Crawford, Kephart, & Leiba, 2004). One can generally distinguish between cases where all classifiers are induced over the same data and cases where several different datasets are used. In the former case, the combination process exploits the biases of different learning algorithms. In the latter case, one can consider building a multitude of detectors, each targeting a different subclass of spam (e.g., phish, pharmaceutical spam, "Nigerian" scams, etc.). Datasets can also be defined on a temporal basis, so that different classifiers have shorter or longer memory spans. Other criteria of providing different datasets are also possible (e.g., based on the language of the message).

Additional levels of complexity in the classifier combination process can be introduced by considering alternative feature representations for each dataset. For example, a single data collection and a single learning method can be used to create several different classifiers, based upon alternative representations of the same data (e.g., using just the header features or just the message text features).

The method of classifier combination will necessarily depend on their performance and intended area of applications. The combination regimes can range from simple logical-OR through linear combinations to complex nonlinear rules, either derived automatically to maximize the desired performance or specified manually with the guidance of expert domain knowledge.

Off-line Adaptation Versus Online Adaptation

A spam filtering system can be configured to receive instant feedback from its users, informing it whenever certain messages get misdelivered (this necessarily does not include cases where misclassified legitimate messages are simply blocked). In the case of online filters, the feedback information may be immediately used to update the filtering profile. This allows a filter to adjust to the changing distribution of email content and to detection countermeasures employed by spammers. Not all classifiers are easily amenable to the online learning update, although online versions of well-known learners such as logistic regression (Goodman & Yih, 2006) and linear SVMs (Sculley & Wachman, 2007) have been proposed. The distinguishing factor is the amount of the original training data that needs to be retained in addition to the model itself to perform future updates. In this respect, Naive Bayes is particularly attractive since it does not require any of the original data for adaptation, with the model itself providing all the necessary information.

One issue with the user feedback signal, however, is its bias toward current errors of the classifier, which for learners depending on the training data being an unbiased sample drawn from the underlying distribution may lead to overcompensation rather than an improvement in filtering accuracy. As an alternative, unbiased feedback can be obtained by either selectively querying users about the nature of uniformly sampled messages or by deriving the labels implicitly.

In the case where off-line adaptation is in use, the feedback data is collected and saved for later use, whereby the filtering models are retrained periodically or only as needed using the data collected. The advantage of off-line adaptation is that it offers more flexibility in terms of the learning algorithm and its optimization. In particular, model retraining can take advantage of a larger quantity of data, and does not have to be constrained to be an extension of the current version of the model. As a result, it is, e.g., possible to redefine the features from one version of the spam filter to the next. One disadvantage is that model updates are likely to be performed less frequently and may be lagging behind the most recent spam trends.

User-specific Versus User-independent Spam Detection

What constitutes a spam message tends to be personal, at least for some types of spam. Various commercial messages, such as promotions and advertisements, e.g., may be distributed in a solicited or unsolicited manner, and sometimes only the end recipient may be able to

judge which. In that sense, user-specific spam detection has the potential of being most accurate, since a user's own judgments are used to drive the training process. Since the nonspam content received by any particular user is likely to be more narrowly distributed when compared a larger user population, this makes the discrimination problem much simpler. Additionally, in the adversarial context, a spammer should find it more difficult to measure the success of penetrating personalized filter defenses, which makes it more difficult to craft a campaign that reaches sufficiently many mail inboxes to be profitable.

One potential disadvantage of such solutions is the need for acquiring labeled data on a user by user basis, which may be challenging. For some users historical data may not yet exist (or has already been destroyed), for others even if such data exist, labeling may seem too much of a burden for the users. Aside from the data collection issues, personal spam filtering faces maintainability issues, as the filter is inherently controlled by its user. This may result in less-than-perfect performance, e.g., if the user misdirects filter training.

From the perspective of institutions and email service providers, it is often more attractive to maintain just one set of spam filters that service a larger user population. This makes them simpler to operate and maintain, but their accuracy may depend on the context of any particular user. The advantage of centralized filtering when serving large user populations is that global trends can be more readily spotted and any particular user may be automatically protected against spam, affecting other users. Also, the domain knowledge of the spam-filtering analysts can be readily injected into the filtering pipeline.

To the extent that a service provider maintains personal filters for its population of users, there are potential large system costs to account for, so that a complete cost-benefit analysis needs to be performed to assess the suitability of such as solution as opposed to a user-independent filtering complex. More details on the nature of such trade-offs can be found in (Kołcz, Bond, & Sargent, 2006).

Clustering and Volumetric Techniques

Content clustering can serve as an important data understanding technique in spam filtering. For example, large clusters can justify the use of specialized classifiers and/or the use of cost-sensitive approaches in classifier learning and evaluation (e.g., where different costs are assigned to different groups of content within each class (Kołcz & Alspector, 2001).

Both spam and legitimate commercial emails are often sent in large campaigns, where the same or highly similar content is sent to a large number of recipients, sometimes over prolonged periods of time. Detection of email campaigns can therefore play an important role in spam filtering. Since individual messages of a campaign are highly similar to one another, this can be considered a variant of near-replica document detection (Kołcz, 2005). It can also be seen as relying on identification of highly localized spikes in the content density distribution. As found in (Yoshida et al., 2004), density distribution approaches can be highly effective, which is especially attractive given that they do not require the explicitly labeled training data. Tracking of spam campaigns may be made difficult due to content randomization, and some research has been directed at making the detection methods robust in the presence such countermeasures (Kołcz, 2005; Kołcz & Chowdhury, 2007).

Misclassification Costs and Filter Evaluation

An important aspect of spam filtering is that the costs of misclassifying spam as legitimate email are not the same as the costs of making the opposite mistake. It is thus commonly assumed that the costs of a false positive mistake (i.e., a legitimate email being misclassified as spam) are much higher than the cost of mistaking spam for legitimate email. Given the prevalence of spam π and the false-spam (FS) and false-legitimate (FL) rates of the classifier, the misclassification cost c can be expressed as

$$c = C_{FS} \cdot (1 - \pi) \cdot FS + C_{FL} \cdot \pi \cdot FL$$

where C_{FS} and C_{FL} are the costs of making a false-spam and false-legitimate mistake, respectively (there is no penalty for making the correct decision). Since actual values of C_{FS} and C_{FL} are difficult to quantify, one typically sees them combined in the form of a ratio, $\lambda = C_{FS}/C_{FL}$, and the overall cost can be expressed as relative to the cost of a false-legitimate misclassification e.g.,

$$c_{\text{rel}} = \lambda \cdot (1 - \pi) \cdot \text{FS} + \pi \cdot \text{FL}$$

Practical choices of λ tend to range from 1 to 1,000. Nonuniform misclassification costs can be used during the process of model induction or in postprocessing when setting up the operating parameters of a spam filter, e.g., using the receiver operating characteristic (ROC) analysis.

Since the costs and cost ratios are sometimes hard to define, some approaches to evaluation favor direct values of the false-spam and false-legitimate error rates. This captures the intuitive requirement that an effective spam filter should provide high detection rate at a close-to-zero false-spam rate. Alternatively, threshold independent metrics such as the area under the ROC (AUC) can be used (Bratko et al., 2006; Cormack & Lynam, 2006), although other measures have also been proposed (Sakkis et al., 2001).

Adaptation to Countermeasures

Spam filtering is an inherently adversarial task, where any solution deployed on a large scale is likely to be met with a response on the part of the spammers. To that extent that the success of a spam filter can be pinpointed to any particular component (e.g., the type of features used), that prominent component is likely to be attacked directly and may become a victim of its own success. For example, the use of word features in spam filtering encourages countermeasures in the form of deliberate misspellings, word fragmentation, and "invisible ink" in HTML documents. Also, since some words are considered by a model inherently more legitimate than others, "word stuffing" has been used to inject large blocks of potentially legitimate vocabulary into an otherwise spammy message in the hope that this information outweighs the evidence provided by the spam content (Lowd & Meek, 2005).

Some authors have attempted to put the adversarial nature of spam filtering in the formal context of game theory (Dalvi et al., 2004). One difficulty of drawing broad conclusion based on such analyses is the breadth of the potential attack/defense front, of which only small sections have been successfully captured in the game-theory formalism. The research on countering the countermeasures points at using multiple diverse filtering components, normalization of features to keep them invariant to irrelevant alterations. A key point is that frequent filter retraining is likely to help in keeping up with the shifts in content distribution, both natural and due to countermeasures.

Future Directions

Reputation Systems and Social Networks

There has been a growing interest in developing reputation systems capturing the trustworthiness of a sender with respect to a particular user or group of users. To this end however, the identity of the sender needs to be reliably verified, which poses challenges and presents a target for potential abuses of such systems. Nevertheless, reputation systems are likely to grow in importance, since they are intuitive from the user perspective in capturing the communication relationships between users. Sender reputation can be hard or soft. In the hard variant, the recipient always accepts or declines messages from a given sender. In the soft variant, the reputation reflects the level of trustworthiness of the sender in the context of the given recipient. When sender identities resolve to individual email addresses, the reputation system can be learned via analysis of a large social network that documents who exchanges email with whom. The sender identities can also be broader however, e.g., assigning reputation to a particular mail server or all mail servers responsible for handling the outbound traffic for a particular domain. On the recipient side, reputation can also be understood globally to represent the trustworthiness of the sender with respect to all recipients hosted by the system. Many open questions remain with regard to computing and maintaining reputations as well as using them effectively to improve spam detection. In the context of text mining, one such question is the extent to which email content analysis can be used to aid the process of reputation assessment.

Cross References

►Cost-Sensitive Learning
►Logistic Regression
►Naive Bayes
►Support Vector Machines
►Text Categorization

Recommended Reading

Bratko, A., Cormack, G. V., Filipic, B., Lynam, T. R., & Zupan, B. (2006). Spam filtering using statistical data compression models. *Journal of Machine Learning Research, 7*, 2673–2698.

Carreras, X., & Màrquez, L. (2001). Boosting trees for anti-spam email filtering. In *Proceedings of RANLP-01, the 4th international conference on recent advances in natural language processing*. New York: ACM.

Cormack, G. V., & Lynam, T. R. (2006). On-line supervised spam filter evaluation. *ACM Transactions on Information Systems, 25(3)*, 11.

Dalvi, N., Domingos, P., Sanghai, M. S., & Verma, D. (2004). Adversarial classification. In *Proceedings of the tenth international conference on knowledge discovery and data mining* (Vol. 1, pp. 99–108). New York: ACM.

Drucker, H., Wu, D., & Vapnik, V. N. (1999). Support vector machines for spam categorization. *IEEE Transactions on Neural Networks*, *5*(10), 1048–1054.

Fawcett, T. (2003). In vivo' spam filtering: A challenge problem for data mining. *KDD Explorations*, *5*(2), 140–148.

Goodman, J., & Yih, W. (2006). Online discriminative spam filter training. In *Proceedings of the third conference on email and anti-spam*.Mountain View, CA. (CEAS-2006).

Kołcz, A. (2005). Local sparsity control for naive bayes with extreme misclassification costs. In *Proceedings of the eleventh ACM SIGKDD international conference on knowledge discovery and data mining*. New York: ACM.

Kołcz, A., & Alspector, J. (2001). *SVM-based filtering of e-mail spam with content-specific misclassification costs*. TextDM'2001 (IEEE ICDM-2001 workshop on text mining), San Jose, CA.

Kołcz, A., Bond, M., & Sargent, J. (2006). The challenges of service-side personalized spam filtering: Scalability and beyond. In *Proceedings of the first international conference on scalable information systems (INFOSCALE)*. New York: ACM.

Kołcz, A. M., & Chowdhury, A. (2007). Hardening fingerprinting by context. In *Proceedings of the fourth international conference on email and anti-spam*.

Lowd, D., & Meek, C. (2005). Good word attacks on statistical spam filters. In *Proceedings of the second conference on email and anti-spam*. Mountain View, CA. (CEAS-2005).

Metsis, V., Androutsopoulos, I., & Paliouras, G. (2006). Spam filtering with naive bayes - which naive bayes? In *Proceedings of the third conference on email and anti-spam*. (CEAS-2006).

Rigoutsos, I., & Huynh, T. (2004). Chung-Kwei: a pattern-discovery-based system for the automatic identification of unsolicited e-mail messages (SPAM). In *Proceedings of the first conference on email and anti-spam*. (CEAS-2004).

Sahami, M., Dumais, S., Heckerman, D., & Horvitz, E. (1998). *A Bayesian approach to filtering junk email*. AAAI workshop on learning for text categorization, Madison, Wisconsin. AAAI Technical Report WS-98-05.

Sakkis, G., Androutsopoulos, I., Paliouras, G., Karkaletsis, V., Spyropoulos, C. D., & Stamatopoulos, P. (2001). Stacking classifiers for anti-spam filtering of e-mail. In L. Lee & D. Harman (Eds.). *Proceedings of empirical methods in natural language processing (EMNLP 2001)* (pp. 44–50). http://www.cs.cornell.edu/home/llee/emnlp/proceeding.html.

Sculley, D., & Wachman, G. (2007). Relaxed online support vector machines for spam filtering. In *Proceedings of the 30th annual international ACM SIGIR conference on research and development in information retrieval*. New York: ACM.

Segal, R., Crawford, J., Kephart, J., & Leiba, B. (2004). SpamGuru: An enterprise anti-spam filtering system. In *Proceedings of the first conference on email and anti-spam*. (CEAS-2004).

Siefkes, C., Assis, F., Chhabra, S., & Yerazunis, W. (2004). Combining winnow and orthogonal sparse bigrams for incremental spam filtering. In *Proceedings of the european conference on principle and practice of knowledge discovery in databases*. New York: Springer.

Yoshida, K., Adachi, F., Washio, T., Motoda, H., Homma, T., Nakashima, A., et al. (2004). Densitiy-based spam detection. In *Proceedings of the tenth ACM SIGKDD international conference on knowledge discovery and data mining* (pp. 486–493). New York: ACM.

Text Mining for the Semantic Web

Marko Grobelnik[1], Dunja Mladenić[1],
Michael Witbrock[2]
[1]Jožef Stefan Institute
[2]Cycorp Inc
Executive Center Drive, Austin, TX, USA

Definition

►Text mining methods allow for the incorporation of textual data within applications of semantic technologies on the Web. Application of these techniques is appropriate when some of the data needed for a Semantic Web use scenario are in textual form. The techniques range from simple processing of text to reducing vocabulary size, through applying shallow natural language processing to constructing new semantic features or applying information retrieval to selecting relevant texts for analysis, through complex methods involving integrated visualization of semantic information, semantic search, semiautomatic ontology construction, and large-scale reasoning.

Motivation and Background

Semantic Web applications usually involve deep structured knowledge integrated by means of some kind of ontology. Text mining methods, on the other hand, support the discovery of structure in data and effectively support semantic technologies on data-driven tasks such as, (semi)automatic ontology acquisition, extension, and mapping. Fully automatic text mining

approaches are not always the most appropriate because often it is too difficult or too costly to fully integrate the available background domain knowledge into a suitable representation. For such cases semiautomatic methods, such as ►active learning and ►semisupervised text processing (►Semisupervised Learning), can be applied to make use of small pieces of human knowledge to provide guidance toward the desired ontology or other model. Application of these semiautomated techniques can reduce the amount of human effort required to produce training data by an order of magnitude while preserving the quality of results.

To date, Semantic Web applications have typically been associated with data, such as text documents and corresponding metadata that have been designed to be relatively easily manageable by humans. Humans are, for example, very good at reading and understanding text and tables. General semantic technologies, on the other hand, aim more broadly at handling data modalities including multimedia, signals from emplaced or remote sensors, and the structure and content of communication and transportation graphs and networks. In handling such multimodal data, much of which is not readily comprehensible by unaugmented humans, there must be significant emphasis on fully- or semiautomatic methods offered by knowledge discovery technologies whose application is not limited to a specific data representation (Grobelnik & Mladenic, 2005).

Data and the corresponding semantic structures change over time, and semantic technologies also aim at adapting the ontologies that model the data accordingly. For most such scenarios extensive human involvement in building models and adapting them according to the data is too costly, too inaccurate, and too slow. Stream mining (Gaber, Zaslavsky, & Krishnaswamy, 2005) techniques (►Data Streams: Clustering) allow text mining of dynamic data (e.g., notably in handling a stream of news or of public commentary).

Structure of Learning System

Ontology is a fundamental method for organizing knowledge in a structured way, and is applied, along with formalized reasoning, in areas from philosophy to scientific discovery to knowledge management and the Semantic Web. In computer science, an ontology generally refers to a graph or network structure consisting of a set of concepts (vertices in a graph), a set of relationships connecting those concepts (directed edges in a graph) and, possibly, a set of distinguished instance concepts assigned to particular class concepts (data records assigned to vertices in a graph). In many cases, knowledge is structured in this way to allow for automated inference based on a logical formalism such as the predicate calculus (Barwise & Etchemendy, 2002); for these applications, an ontology often further comprises a set of rules or produces new knowledge within the representation from existing knowledge. An ontology containing instance data and rules is often referred to as a knowledge base (KB) (e.g., Lenat, 1995).

Machine learning practitioners refer to the task of constructing these ontologies as *ontology learning*. From this point of view, an ontology is seen a class of models – somewhat more complex than most used in machine learning – which need to be expressed in some ►hypothesis language. This definition of ontology learning (from Grobelnik & Mladenic, 2005) enables a decomposition into several machine learning tasks, including learning concepts, identifying relationships between existing concepts, populating an existing ontology/structure with instances, identifying change in dynamic ontologies, and inducing rules over concepts, background knowledge, and instances.

Following this scheme, text mining methods have been applied to extending existing ontologies based on Web documents, learning semantic relations from text based on collocations, semiautomatic data driven ontology construction based on document clustering and classification, extracting semantic graphs from text, transforming text into RDF triples (a commonly used Semantic Web data representation), and mapping triplets to semantic classes using several kinds of lexical and ontological background knowledge. Text mining is also intensively used in the effort to produce a Semantic Web for annotation of text with concepts from ontology. For instance, a text document is split into sentences, each sentence is represented as a word-vector, sentences are clustered, and each cluster is labeled by the most characteristic words from its sentences and mapped upon the concepts of a general ontology. Several approaches that integrate ontology management, knowledge discovery, and human language technologies are described in (Davies, Grobelnik, & Mladenić, 2009).

Extending the text mining paradigm, current efforts are beginning to aim at giving machines an approximation of the full human ability to acquire knowledge from text. Machine reading aims at full text understanding by integrating knowledge-based construction and use into syntactically sophisticated natural language analysis, leading to systems that autonomously improve their ability to extract further knowledge from text (e.g., Curtis et al., 2009; Etzioni, Banko, & Cafarella, 2007; Mitchell, 2005).

Cross References

►Active Learning
►Classification
►Document Clustering
►Semisupervised Learning
►Semisupervised Text Processing
►Text Mining
►Text Visualization

Recommended Reading

Barwise, J., & Etchemendy, J. (2002). *Language proof and logic.* Center for the Study of Language and Information. ISBN, 157586374X.

Buitelaar, P., Cimiano, P., & Magnini, B. (2005). *Ontology learning from text: Methods, applications and evaluation, frontiers in artificial intelligence and applications*. The Netherlands: IOS Press.

Curtis, J., Baxter, D., Wagner, P., Cabral, J., Schneider, D., & Witbrock, M. (2009). Methods of rule acquisition in the TextLearner system. In *Proceedings of the 2009 AAAI spring symposium on learning by reading and learning to read* (pp. 22–28). Palo Alto, CA: AAAI Press.

Davies, J., Grobelnik, M., & Mladenić, D. (2009). *Semantic knowledge management*. Berlin: Springer.

Etzioni, O., Banko, M., & Cafarella, M. J. (2007). Machine Reading. In Proceedings of the 2007 AAAI spring symposium on machine reading.

Gaber, M. M., Zaslavsky, A., & Krishnaswamy, S. (2005).Mining data streams: A review. *ACM SIGMOD Record, 34*(1), 18–26. ISSN: 0163-580

Grobelnik, M., & Mladenic, D. (2005). Automated knowledge discovery in advanced knowledge management. *Journal of Knowledge Management*, *9*, 132–149.

Lenat, D. B. (1995). Cyc: A large-scale investment in knowledge infrastructure. *Communications of the ACM, 38*(11), 33–38.

Mitchell, T. (2005). Reading the web: A breakthrough goal for AI. celebrating twenty-five years of AAAI: Notes from the AAAI-05 and IAAI-05 conferences. *AI Magazine*, *26*(3), 12–16.

Text Spatialization

►Text Visualization

Text Visualization

John Risch[1], Shawn Bohn[1], Steve Poteet[2], Anne Kao[2], Lesley Quach[2], Jason Wu[2]
[1]Pacific Northwest National Laboratory
[2]Boeing Phantom Works, Seattle, WA, USA

Synonyms

Semantic mapping; Text spatialization; Topic mapping

Definition

The term *text visualization* describes a class of knowledge discovery techniques that use interactive graphical representations of textual data to enable knowledge discovery via recruitment of human visual pattern recognition and spatial reasoning capabilities. It is a subclass of *information visualization*, which more generally encompasses visualization of nonphysically based (or "abstract") data of all types. Text visualization is distinguished by its focus on the unstructured (or *free text*) component of information. While the term "text visualization" has been used to describe a variety of graphical methods for deriving knowledge from text, it is most closely associated with techniques for depicting the semantic characteristics of large document collections. Text visualization systems commonly employ unsupervised machine learning techniques as part of broader strategies for organizing and graphically representing such collections.

Motivation and Background

The Internet enables universal access to vast quantities of information, most of which (despite admirable efforts (Berners-Lee, Hendler, & Lassila, 2001)) exists in the form of unstructured and unorganized text. Advancements in search technology make it possible to retrieve large quantities of this information with reasonable precision; however, only a tiny fraction of the information available on any given topic can be effectively exploited.

Text visualization technologies, as forms of computer-supported knowledge discovery, aim to improve our ability to understand and utilize the wealth of text-based information available to us. While the term "text visualization" has been used to describe a variety of techniques for graphically depicting the characteristics of free-text data (Havre, Hetzler, Whitney, & Nowell, 2002; Small, 1996), it is most closely associated with the so-called *semantic clustering* or *semantic mapping* techniques (Chalmers & Chitson, 1992; Kohonen et al., 2000; Lin, Soergel, & Marchionini, 1991; Wise et al., 1995). These methods attempt to generate summary representations of document collections that convey information about their general topical content and similarity structure, facilitating general domain understanding and analytical reasoning processes.

Text visualization methods are generally based on vector-space models of text collections (Salton, 1989), which are commonly used in information retrieval, clustering, and categorization. Such models represent the text content of individual documents in the form of vectors of frequencies of the terms (*text features*) they contain. A document collection is therefore represented as a collection of vectors. Because the number of unique terms present in a document collection generally is in the range of tens of thousands, a dimensionality reduction method such as singular value decomposition (SVD) (Deerwester, Dumais, Furnas, Landauer, & Harshman, 1990) or other matrix decomposition method (Kao, Poteet, Ferng, Wu, & Quach, 2008; Booker et al., 1999) is typically used to eliminate noise terms and reduce the length of the document vectors to a tractable size (e.g., 50–250 dimensions). Some systems attempt to first identify discriminating features in the text and then use mutual probabilities to specify the vector space (York, Bohn, Pennock, & Lantrip, 1995).

To enable visualization, the dimensions must be further reduced to two or three. The goal is a graphical representation that employs a "spatial proximity means conceptual similarity" metaphor where topically similar text documents are represented as nearby points in the display. Various regions of the semantic map are subsequently labeled with descriptive terms that convey the primary concepts described by nearby documents. The text visualization can thus serve as a kind of graphical "table of contents" depicting the conceptual similarity structure of the collection.

Text visualization systems therefore typically implement four key functional components, namely,

1. A *tokenization* component that characterizes the lexical content of text units via extraction, normalization, and selection of key terms
2. A *vector-space modeling* component that generates a computationally tractable vector space representation of a collection of text units
3. A *spatialization* component that uses the vector space model to generate a 2D or 3D spatial configuration that places the points representing conceptually similar text units in near spatial proximity
4. A *labeling* component that assigns characteristic text labels to various regions of the semantic map

Although machine learning techniques can be used in several of these steps, their primary usage is in the spatialization stage. An unsupervised learning algorithm is typically used to find meaningful low-dimensional structures hidden in high-dimensional document feature spaces.

Structure of Learning System

Spatialization is a term generically used in ▶information visualization to describe the process of generating a spatial representation of inherently nonspatial information. In the context of text visualization, this term generally refers to the application of a nonlinear dimensionality reduction algorithm to a collection of text vectors in order to generate a visually interpretable two- or three-dimensional representation of the similarity structure of the collection. The goal is the creation of a *semantic similarity map* that positions graphical features representing text units (e.g., documents) conceptually similar to one another near one another in the visualization display. These maps may be further abstracted to produce more general summary representations of text collections that do not explicitly depict the individual text units themselves (Wise et al., 1995).

A key assumption in text visualization is that text units which express similar concepts will employ similar word patterns, and that the existence of these

word correlations creates coherent structures in high-dimensional text feature spaces. A further assumption is that text feature spaces are nonlinear, but that their structural characteristics can be approximated by a smoothly varying low-dimensional manifold. The text spatialization problem thus becomes one of finding an embedding of the feature vectors in a two- or three-dimensional manifold that best approximates this structure. Because the intrinsic dimensionality of the data is invariably much larger than two (or three), significant distortion is unavoidable. However, because the goal of text visualization is not necessarily the development of an accurate representation of interdocument similarities, but rather the depiction of broad (and ambiguously defined) semantic relationships, this distortion is generally considered acceptable.

Text vector spatialization therefore involves the fitting of a model into a collection of observations. Most text visualization systems developed to date have employed some type of unsupervised learning algorithm for this purpose. In general, the desired characteristics of an algorithm used for text spatialization include that it (1) preserves global properties of the input space, (2) preserves the pairwise input distances to the greatest extent possible, (3) supports out-of-sample extension (i.e., the incremental addition of new documents), and (4) has low computational and memory complexity. Computational and memory costs are key considerations, as a primary goal of text visualization is the management and interpretation of extremely large quantities of textual information.

A leading approach is to iteratively adapt the nodes of a fixed topology mesh to the high-dimensional feature space via adaptive refinement. This is the basis of the well-known Kohonen feature mapping algorithm, more commonly referred to as the ▶self-organizing map (SOM) (Kohonen, 1997). In a competitive learning process, text vectors are presented one at a time to a (typically triangular) grid, the nodes of which have been randomly initialized to points in the term space. The Euclidean distance to each node is computed, and the node closest to the sample is identified. The position of the winning node, along with those of it's topologically nearest neighbors, is incrementally adjusted toward the sample vector. The magnitude of the adjustments is gradually decreased over time. The process is generally repeated using every vector in the set for many hundreds or thousands of cycles until the mesh converges on a solution. At the conclusion, the samples are assigned to their nearest nodes, and the results are presented as a uniform grid. In the final step, the nodes of the grid are labeled with summary terms which describe the key concepts associated with the text units that have been assigned to them.

Although self-organizing maps can be considered primarily a clustering technique, the grid itself theoretically preserves the topological properties of the input feature space. As a consequence, samples that are nearest neighbors in the feature space generally end up in topologically adjacent nodes. However, while SOMs are topology-preserving, they are not distance-preserving. Vectors that are spatially distant in the input space may be presented as proximal in the output, which may be semantically undesirable. SOMs have a number of attractive characteristics, including straightforward out-of-sample extension and low computational and memory complexity. Examples of the use of SOMs in text visualization applications can be found in (Lin et al., 1991; Kaski, Honkela, Lagus, & Kohonen, 1998; Kohonen et al., 2000).

Often, it is considered desirable to attempt to preserve the distances among the samples in the input space to the greatest extent possible in the output. The rationale is that the spatial proximities of the text vectors capture important and meaningful characteristics of the associated text units: spatial "nearness" corresponds to conceptual "nearness." As a consequence, many text visualization systems employ distance-preserving dimensionality reduction algorithms. By far the most commonly used among these is the class of algorithms known as *multidimensional scaling* (MDS) algorithms.

Multidimensional scaling is "a term used to describe any procedure which starts with the 'distances' between a set of points (or individuals or objects) and finds a configuration of the points, preferably in a smaller number of dimensions, usually 2 or 3" ((Chatfield & Collins, 1980), quoted in Chalmers & Chitson, 1992). There are two main subclasses of MDS algorithms. Metric (quantitative, also known as classical) MDS algorithms attempt to preserve the pairwise input distances to the greatest extent possible in the output

configuration, while nonmetric (qualitative) techniques attempt only to preserve the rank order of the distances. Metric techniques are most commonly employed in text visualization.

Metric MDS maps the points in the input space to the output space while maintaining the pairwise distances among the points to the greatest extent possible. The quality of the mapping is expressed in a stress function which is minimized using any of a variety of optimization methods, e.g., via eigen decomposition of a pairwise dissimilarity matrix, or using iterative techniques such as generalized Newton–Raphson, simulated annealing, or genetic algorithms. A simple example of a stress function is the raw stress function (Kruskal, 1964) defined by

$$\phi(Y) = \sum_{ij} (||x_i - x_j|| - ||y_i - y_j||)^2$$

in which $||x_i - x_j||$ is the Euclidean distance between points x_i and x_j in the high-dimensional space, and $||y_i - y_j||$ is the distance between the corresponding points in the output space. A variety of alternative stress functions have been proposed (Cox & Cox, 2001). In addition to its distance-preserving characteristics, MDS has the added advantage of preserving the global properties of the input space. A major disadvantage of MDS, however, is its high computational complexity, which is approximately $O(kN^2)$, where N is the number of data points and k is the dimensionality of the embedding. Although computationally expensive, MDS can be used practically on data sets of up to several hundred documents in size. Another disadvantage is that out-of-core extension requires reprocessing of the full data set if an optimization method which computes the output coordinates all at once is used.

The popularity of MDS methods has led to the development of a range of strategies for improving on its computational efficiency to enable scaling of the technique to text collections of larger size. One approach is to use either cluster centroids or a randomly sampled subset of input vectors as surrogates for the full set. The surrogates are down-projected independently using MDS, then the remainder of the data is projected relative to this "framework" using a less expensive algorithm, e.g., distance-based triangulation. This is the basis for the *anchored least stress* algorithm used in the SPIRE text visualization system (York et al., 1995), as well as the more recently developed Landmark MDS (de Silva & Tenenbaum, 2003) algorithm.

While self-organizing maps and multidimensional scaling techniques have received the most attention to date, a number of other machine learning techniques have also been used for text spatialization. The Starlight system (Risch et al., 1999) uses *stochastic proximity embedding* (Agrafiotis, 2003), a high-speed nonlinear manifold learning algorithm. Other approaches have employed methods based on graph layout techniques (Fabrikant, 2001). Generally speaking, any of a number of techniques for performing dimensionality reduction in a correlated system of measurements (classified under the rubric of factor analysis in statistics) may be employed for this purpose.

Machine learning algorithms can also be used in text visualization for tasks other than text vector spatialization. For example, generation of descriptive labels for semantic maps requires partitioning of the text units into related sets. Typically, a partitioning-type ►clustering algorithm such as K-means is used for this purpose (see ►Partitional Clustering), either as an element of the spatialization strategy (see York et al., 1995), or as a postspatialization step. The labeling process itself may also employ machine learning algorithms. For instance, the TRUST system (Booker et al., 1999; Kao et al., 2008) employed by Starlight generates meaningful labels for document clusters using a kind of ►unsupervised learning. By projecting a cluster centroid defined in the reduced dimensional representation (e.g., 50–250 dimensions) back into the full term space, terms related to the content of the documents in the cluster are identified and used as summary terms. Machine learning techniques can also be applied indirectly during the tokenization phase of text visualization. For example, information extraction systems commonly employ rule sets that have been generated by a supervised learning algorithm (Mooney & Bunescu, 2006). Such systems may be used to identify tokens that are most characteristic of the overall topic of a text unit, or are otherwise of interest (e.g., the names of people or places). In this way, the dimensionality of the input space can be drastically reduced, accelerating downstream processing while

simultaneously improving the quality of the resulting visualizations.

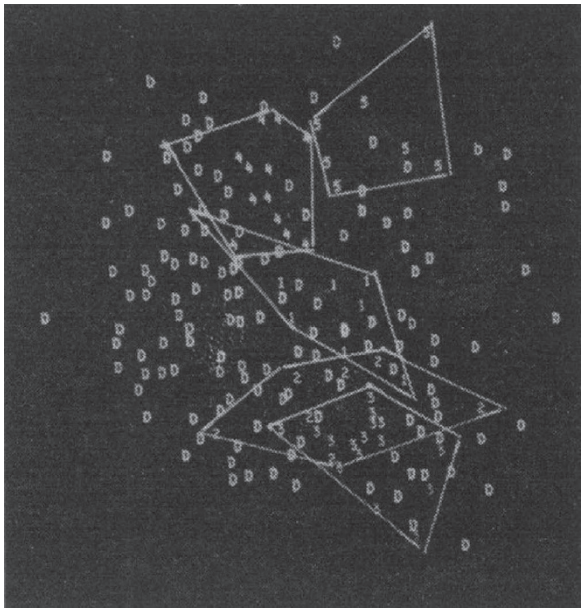
Applications

Sammon

The first text visualization system based on a text vector space model was likely a prototype developed in the 1960s by John Sammon's "nonlinear mapping," or *NLM*, algorithm (today referred to as organizing text data). The configuration depicted here is the result of applying Sammon's algorithm to a collection of 188 documents represented as 17-dimensional vectors determined according to document relevance to 1,125 key words and phrases. Among other interesting and prescient ideas, Sammon describes techniques for interacting with text visualizations depicted on a "CRT display" using a light pen (Fig. 1).

Lin

Lin's 1991 prototype (Lin et al., 1991) was one of the first to demonstrate the use of self-organizing maps for organizing text documents. Lin formed a 25-dimensional vector space model of a 140 document collection using 25 key index terms extracted from the text. The document vectors were used to train a 140 node feature map, generating the result shown here (the fact that the number of nodes matches the number of documents is coincidental). Lin was also among the first to assign text labels to various regions of the resulting map to improve the interpretability and utility of the resulting product (Fig. 2).



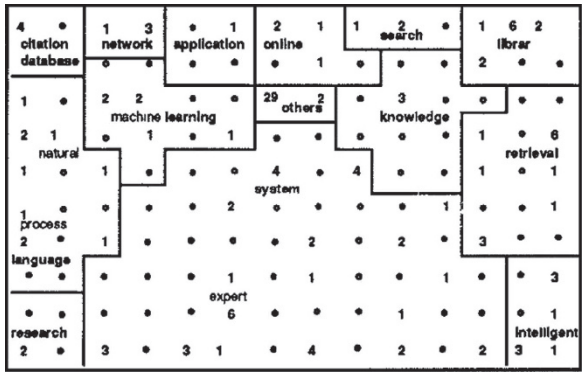
Text Visualization. Figure 1.

BEAD

The BEAD system (Chalmers & Chitson, 1992) was a text visualization prototype developed during the early 1990s at Rank Xerox EuroPARC. BEAD employed a vector space model constructed using document keywords and a hybrid MDS algorithm based on an optimized form of simulated annealing. Although it did not include a region labeling component, BEAD did support highlighting of visualization features in response to query operations, a now standard text visualization system feature. The BEAD project also pioneered a number of now common interaction techniques, and was among the first to explore 3D representations of document collections (Fig. 3).

IN-SPIRE

IN-SPIRE (formerly SPIRE, Spatial Paradigm for Information Retrieval and Exploration) (Wise et al., 1995), was originally developed in 1995 at Pacific Northwest National Laboratory (PNNL). Over the years, IN-SPIRE has evolved from using MDS, to Anchored Least Stress, to a hybrid clustering/PCA projection scheme. The SPIRE/IN-SPIRE system introduced several new concepts, including the use of a 3D "landscape" abstraction (called a *ThemeView*) for depicting the general characteristics of large text collections. A recently developed parallelized version of the software is capable of



Text Visualization. Figure 2.

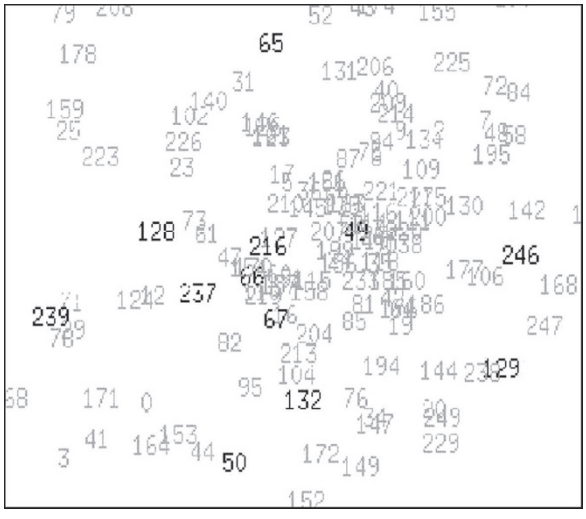
generating visualizations of document collections containing millions of items (Fig. 4).

WEBSOM

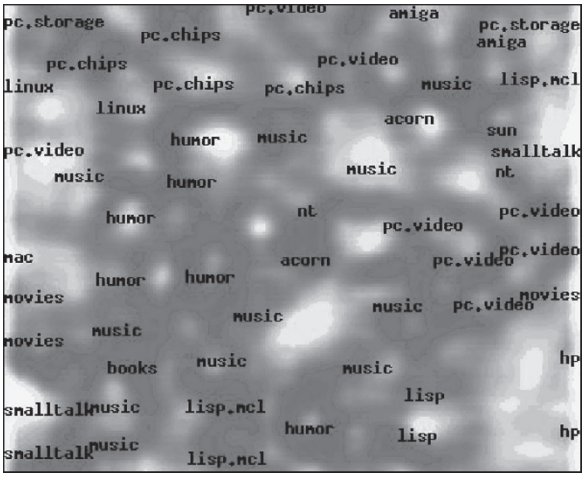
WEBSOM (Kaski et al., 1998) was another early application of Kohonnen self-organizing maps to text data. Early versions of WEBSOM used an independent SOM to generate reduced dimensionality text vectors which were then mapped with a second SOM for visualization purposes. More recent SOM-based text visualization experiments have employed vectors constructed via random projections of weighted word histograms (Kohonen et al., 2000). SOMs have been used to generate semantic maps containing millions of documents (Fig. 5).

Starlight

Starlight (Risch et al., 1999) is a general-purpose information visualization system developed at PNNL that includes a text visualization component. Starlight's text visualization system uses the Boeing *Text Representation Using Subspace Transformation* (TRUST) text engine for vector space modeling and text summarization. Text vectors generated by TRUST are clustered, and the cluster centroids are down-projected to 2D and 3D using a nonlinear manifold learning algorithm. Individual document vectors associated with each cluster are likewise projected within a local coordinate system established at the projected coordinates of their associated cluster centroid, and TRUST is used to generate topical labels for each cluster. Starlight is unique in that

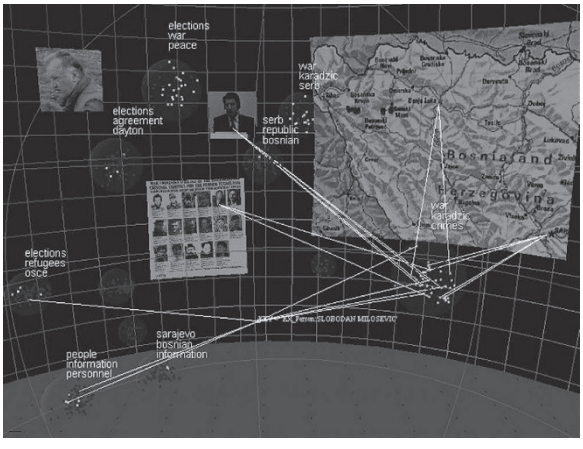


Text Visualization. Figure 3.



Text Visualization. Figure 5.

Text Visualization. Figure 4.



Text Visualization. Figure 6.

it couples text visualization with a range of other information visualization techniques (such as link displays) to depict multiple aspects of information simultaneously (Fig. 6).

Cross References

►Dimensionality Reduction
►Document Classification/Clustering
►Feature Selection/Construction
►Information Extraction/Visualization
►Self-Organizing Maps
►Text Preprocessing

Recommended Reading

Agrafiotis, D. K. (2003). Stochastic proximity embedding. *Journal of Computational Chemistry, 24*(10), 1215–1221.

Berners-Lee, T., Hendler, J., & Lassila, O. (2001). The semantic web. *Scientific American, 284*(5), 34–43.

Booker, A., Condliff, M., Greaves, M., Holt, F. B., Kao, A., Pierce, D. J., et al. (1999). Visualizing text data sets. *Computing in Science & Engineering, 1*(4), 26–35.

Chalmers, M., & Chitson, P. (1992). Bead: Explorations in information visualization. In *SIGIR '92: Proceedings of the 15th annual international ACM SIGIR conference on research and development in information retrieval* (pp. 330–337). New York: ACM.

Chatfield, C., & Collins, A. (1980). *Introduction to multivariate analysis*. London: Chapman & Hall.

Cox, M. F., & Cox, M. A. A. (2001). *Multidimensional scaling*. London: Chapman & Hall.

Crouch, D. (1986). The visual display of information in an information retrieval environment. In *Proceedings of the ACM SIGIR conference on research and development in information retrieval* (pp. 58–67). New York: ACM.

Deerwester, S., Dumais, S., Furnas, G., Landauer, T., & Harshman, R. (1990). Indexing by latent semantic analysis. *Journal of American Society for Information Science, 41*(6), 391–407.

de Silva, V., & Tenenbaum, J. B. (2003). Global versus local methods in nonlinear dimensionality reduction. In S. Becker, S. Thrun, & K. Obermayer (Eds.), *Proceedings of the NIPS* (Vol. 15, pp. 721–728).

Doyle, L. (1961). Semantic roadmaps for literature searchers. *Journal of the Association for Computing Machinery, 8*(4), 367–391.

Fabrikant, S. I. (2001). Visualizing region and scale in information spaces. In *Proceedings of the 20th international cartographic conference, ICC 2001* (pp. 2522–2529). Beijing, China.

Havre, S., Hetzler, E., Whitney, P., & Nowell, L. (2002). ThemeRiver: Visualizing thematic changes in large document collections. *IEEE Transactions on Visualization and Computer Graphics, 8*(1), 9–20.

Huang, S., Ward, M., & Rundensteiner, E. (2003). *Exploration of dimensionality reduction for text visualization* (Tech. Rep. TR-03-14). Worcester, MA: Worcester Polytechnic Institute, Department of Computer Science.

Kao, A., Poteet, S., Ferng, W., Wu, J., & Quach, L. (2008). Latent semantic indexing and beyond, to appear. In M. Song & Y. F. Wu (Eds.), *Handbook of research on text and web mining technologies*. Hershey, PA: Idea Group Inc.

Kaski, S., Honkela, T., Lagus, K., & Kohonen, T. (1998). WEBSOM-self-organizing maps of document collections. *Neurocomputing, 21*, 101–117.

Kohonen, T. (1997). *Self-organizing maps. Series in information sciences* (2nd ed., Vol. 30). Heidelberg: Springer.

Kohonen, T., Kaski, S., Lagus, K., Salojärvi, J., Honkela, J., Paatero, V., et al. (2000). Self organization of a massive document collection. *IEEE Transactions on Neural Networks, 11*(3), 574–585.

Kruskal, J. B. (1964). Multidimensional scaling by optimizing goodness of fit to a nonmetric hypothesis. *Psychometrika, 29*(1), 1–27.

Lin, X., Soergel, D., & Marchionini, D. A. (1991). Self-organizing semantic map for information retrieval. In *Proceedings of the fourteenth annual international ACM/SIGIR conference on research and development in information retrieval* (pp. 262–269). Chicago.

Mooney, R. J., & Bunescu, R. (2006). Mining knowledge from text using information extraction. In K. Kao & S. Poteet (Eds.), *SigKDD explorations* (pp. 3–10).

Paulovich, F. V., Nonato, L. G., & Minghim, R. (2006). Visual mapping of text collections through a fast high precision projection technique. In *Proceedings of the tenth international conference on information visualisation (IV'06)* (pp. 282–290).

Risch, J. S., Rex, D. B., Dowson, S. T., Walters, T. B., May, R. A., & Moon, B. D. (1999). The STARLIGHT information visualization system. In S. Card, J. Mackinlay, & B. Shneiderman (Eds.), *Readings in information visualization: Using vision to think* (pp. 551–560). San Francisco: Morgan Kaufmann.

Salton, G. (1989). *Automatic text processing*. Reading, MA: Addison-Wesley.

Sammon, J. W. (1969). A nonlinear mapping for data structure analysis. *IEEE Transactions on Computer, 18*(5), 401–409.

Small, D. (1996). Navigating large bodies of text. *IBM Systems Journal, 35*(3&4), 514–525.

Wise, J. A., Thomas, J. J., Pennock, K., Lantrip, D., Pottier, M., Schur, A., et al. (1995). Visualizing the non-visual: Spatial analysis and interaction with information from text documents. In *Proceedings of the IEEE information visualization symposium '95* (pp. 51–58). Atlanta, GA.

York, J., Bohn, S., Pennock, K., & Lantrip, D. (1995). Clustering and dimensionality reduction in SPIRE. In *Proceedings, symposium on advanced information processing and analysis, AIPA95*. Tysons Corner, VA.

TF–IDF

TF–IDF (*term frequency–inverse document frequency*) is a term weighting scheme commonly used to represent textual documents as vectors (for purposes of classification, clustering, visualization, retrieval, etc.). Let $T = \{t_1,\ldots, t_n\}$ be the set of all terms occurring in the document corpus under consideration. Then a document

d_i is represented by a n-dimensional real-valued vector $\mathbf{x}_i = (x_{i1},\ldots,x_{in})$ with one component for each possible term from T.

The weight x_{ij} corresponding to term t_j in document d_i is usually a product of three parts: one which depends on the presence or frequency of t_j in d_i, one which depends on t_j's presence in the corpus as a whole, and a normalization part which depends on d_j. The most common TF–IDF weighting is defined by $x_{ij} = \mathrm{TF}_i \cdot \mathrm{IDF}_j \cdot (\sum_j (\mathrm{TF}_{ij}\mathrm{IDF}_j)^2)^{-1/2}$, where TF_{ij} is the *term frequency* (i.e., number of occurrences) of t_j in d_i, and IDF_j is the IDF of t_j, defined as $\log(N/\mathrm{DF}_j)$, where N is the number of documents in the corpus and DF_j is the document frequency of t_j (i.e., the number of documents in which t_j occurs). The normalization part ensures that the vector has a Euclidean length of 1.

Several variations on this weighting scheme are also known. Possible alternatives for TF_{ij} include $\min\{1, \mathrm{TF}_{ij}\}$ (to obtain binary vectors) and $(1 + \mathrm{TF}_{ij}/\max_j \mathrm{TF}_{ij})/2$ (to normalize TF within the document). Possible alternatives for IDF_j include 1 (to obtain plain TF vectors instead of TF–IDF vectors) and $\log(\sum_i \sum_k \mathrm{TF}_{ik}/\sum_i \mathrm{TF}_{ij})$. The normalization part can be omitted altogether or modified to use some other norm than the Euclidean one.

Threshold Phenomena in Learning

►Phase Transitions in Machine Learning

Time Sequence

►Time Series

Time Series

Eamonn Keogh
University of California
Riverside, CA, USA

Synonyms

Temporal data; Time sequence; Trajectory data

Definition

A *Time Series* is a sequence $T = (t_1, t_2,\ldots,t_n)$ which is an ordered set of n real-valued numbers. The ordering is typically temporal; however, other kinds of data such as color distributions (Hafner, Sawhney, Equitz, Flickner, & Niblack, 1995), shapes (Ueno, Xi, Keogh, & Lee, 2006), and spectrographs also have a well-defined ordering and can be fruitfully considered "time series" for the purposes of machine learning algorithms.

Motivation and Background

The special structure of time series produces unique challenges for machine learning researchers.

It is often the case that each individual time series object has a very high dimensionality. Whereas classic algorithms often assume a relatively low dimensionality (for example, a few dozen measurements such as "height, weight, blood sugar," etc.), time series learning algorithms must be able to deal with dimensionalities in hundreds or thousands. The problems created by high-dimensional data are more than mere computation time considerations; the very meaning of normally intuitive terms, such as "similar to" and "cluster forming," become unclear in high-dimensional space. The reason for this is that as dimensionality increases, all objects become essentially equidistant to each other and thus classification and clustering lose their meaning. This surprising result is known as the ►*curse of dimensionality* and has been the subject of extensive research. The key insight that allows meaningful time series machine learning is that although the actual dimensionality may be high, the *intrinsic* dimensionality is typically much lower. For this reason, virtually all time series data mining algorithms avoid operating on the original "raw" data; instead, they consider some higher level representation or abstraction of the data. Such algorithms are known as ►*dimensionality reduction* algorithms. There are many general dimensionality reduction algorithms, such as singular value decomposition and random projections, in addition to many reduction algorithms specifically designed for time series, including piecewise liner approximations, Fourier transforms, wavelets, and symbol approximations (Ding, Trajcevski, Scheuermann, Wang, & Keogh, 2008).

In addition to the high dimensionality of individual time series objects, many time series datasets have very

high numerosity, resulting in a large volume of data. One implication of high numerosity combined with the high dimensionality of this is that the entire dataset may not fit in main memory. This requires an efficient disk-aware learning algorithm or a careful *sampling* approach.

A final consideration due to the special nature of time series is the fact that individual datapoints are typically highly correlated with their neighbors (a phenomenon known as *autocorrelation*). Indeed, it is this correlation that makes most time series excellent candidates for dimensionality reduction. However, for learning algorithms that assume the independence of features (i.e., ►Naïve Bayes), this lack of independence must be countered or mitigated in some way.

While virtually every machine learning method has been used to classify time series, the current state-of-the-art method is the nearest neighbor algorithm (Ueno et al., 2006) with a suitable distance measure (Ding et al., 2008). This simple method outperforms neutral networks and Bayesian classifiers.

The major database (SIGMOD, VLDB, PODS) and data mining (SIGKDD, ICDM, SDM) conferences typically feature several time series machine learning/data mining papers each year. In addition, because of the ubiquity of time series, several other communities have active subgroups that conduct research on time series; for example, the SIGGRAPH conference typically has papers on learning or indexing or motion capture time series, and most medical conferences have tracks devoted to medical time series, such as electrocardiograms and electroencephalograms.

The UCR Time Series Archive has several dozen time series datasets which are widely used to test classification and clustering algorithms, and the UCI Data Mining archive has several additional datasets.

Recommended Reading

Ding, H., Trajcevski, G., Scheuermann, P., Wang, X., & Keogh, E. A. (2008). Querying and mining of time series data: Experimental comparison of representations and distance measures. *In Proceeding of the VLDB*. VLDB Endowment.

Hafner, J., Sawhney, H., Equitz, W., Flickner, M., & Niblack, W. (1995). Efficient color histogram indexing for quadratic form distance functions. *IEEE Transactions on Pattern Analysis and Machine Intelligence, 17*(7), 729–736.

Ueno, K., Xi, X., Keogh, E., & Lee, D. (2006). Anytime classification using the nearest neighbor algorithm with applications to stream mining. In Proceedings of IEEE international conference on data mining (ICDM).

Topic Mapping

►Text Visualization

Topology of a Neural Network

Risto Miikkulainen
The University of Texas at Austin
Austin, TX, USA

Synonyms

Connectivity; Neural network architecture; Structure

Definition

Topology of a neural network refers to the way the ►Neurons are connected, and it is an important factor in network functioning and learning. A common topology in unsupervised learning is a direct mapping of inputs to a collection of units that represents categories (e.g., ►Self-organizing maps). The most common topology in supervised learning is the fully connected, three-layer, feedforward network (see ►Backpropagation, Radial Basis Function Networks). All input values to the network are connected to all neurons in the hidden layer (hidden because they are not visible in the input or the output), the outputs of the hidden neurons are connected to all neurons in the output layer, and the activations of the output neurons constitute the output of the whole network. Such networks are popular partly because theoretically they are known to be universal function approximators (with e.g., a sigmoid or gaussian nonlinearity in the hidden layer neurons), although in practice networks with more layers may be easier to train (see ►Cascade Correlation, Deep Belief Networks). Layered networks can be extended to processing sequential input and/or output by saving a copy of the hidden layer activations and using it as additional input to the hidden layer in the next time step (see ►Simple Recurrent Networks).

Fully recurrent topologies, where each neuron is connected to all other neurons (and possibly to itself) can also be used to model time-varying behavior, although such networks may be unstable and difficult to train (e.g., with backpropagation; but see also ►Boltzmann Machines). Modular topologies, where different parts of the networks perform distinctly different tasks, can improve stability and can also be used to model high-level behavior (e.g., ►Reservoir Computing, ►Adaptive Resonance Theory). Whatever the topology, in most cases, learning involves modifying the ►Weights on the network connections. However, arbitrary network topologies are possible as well and can be constructed as part of the learning (e.g. with backpropagation or ►Neuroevolution) to enhance feature selection, recurrent memory, abstraction, or generalization.

Trace-Based Programming

Pierre Flener[1,2], Ute Schmid[3]
[1]Sabancı University, Orhanlı, Tuzla, İstanbul, Turkey
[2]Uppsala University
Uppsala, Sweden
[3]University of Bamberg
Bamberg, Germany

Synonyms

Programming from traces; Trace-based programming

Definition

Trace-based programming addresses the inference of a program from a small set of example computation traces. The induced program is typically a recursive program. A computation *trace* is a nonrecursive expression that describes the transformation of some specific input into the desired output with help of a predefined set of primitive functions. While the construction of traces is highly dependent on background knowledge or even on knowledge about the program searched for, the inductive ►generalization is based on syntactical methods of detecting regularities and dependencies between traces, as proposed in classical approaches to ►inductive programming (see Example 5 of that encyclopedia entry) or ►explanation-based learning (EBL). As an alternative to providing traces by hand-simulation, AI planning techniques or ►programming by demonstration (PBD) can be used.

Cross References

►Inductive Programming
►Programming by Demonstration

Recommended Reading

Biermann, A. W. (1972). On the inference of Turing machines from sample computations. *Artificial Intelligence, 3*(3), 181–198.

Schmid, U., & Wysotzki, F. (1998). Induction of recursive program schemes. In *Proceedings of the tenth european conference on machine learning (ECML 1998): Lecture notes in artificial intelligence* (Vol. 1398, pp. 214–225). Berlin: Springer.

Schrödl, S., & Edelkamp, S. (1999). Inferring flow of control in program synthesis by example. In *Proceedings of the 23rd annual german conference on artificial intelligence (KI 1999): Lecture notes in artificial intelligence* (Vol. 1701, pp. 171–182). Berlin: Springer.

Shavlik, J. W. (1990). Acquiring recursive and iterative concepts with explanation-based learning. *Machine Learning, 5*, 39–70.

Wysotzki, F. (1983). Representation and induction of infinite concepts and recursive action sequences. In *Proceedings of the eighth international joint conference on artificial intelligence (IJCAI 1983)* (pp. 409–414). San Mateo, CA: Morgan Kaufmann.

Training Curve

►Learning Curves in Machine Learning

Training Data

Synonyms

Training examples; Training instances; Training set

Definition

Training data are data to which a ►learner is applied.

Training Examples

►Training Data

Training Instances

►Training Data

Training Set

Synonyms
Training data

Definition
A training set is a ►data set containing data that are used for learning by a ►learning system. A training set may be divided further into a ►growing set and a ►pruning set.

Cross References
►Data Set
►Training Data

Training Time

A learning algorithm is typically applied at two distinct times. Training time refers to the time when an algorithm is learning a model from ►training data. ►Test time refers to the time when an algorithm is applying a learned model to make predictions. ►Lazy learning usually blurs the distinction between these two times, deferring most learning until test time.

Trait

►Attribute

Trajectory Data

►Time Series

Transductive Learning

►Semi-Supervised Learning
►Semi-Supervised Text Processing

Transfer of Knowledge Across Domains

►Inductive Transfer

Transition Probabilities

In a ►Markov decision process, the *transition probabilities* represent the probability of being in *state* s' at time $t+1$, given you take *action* a from state s at time t for all s, a and t.

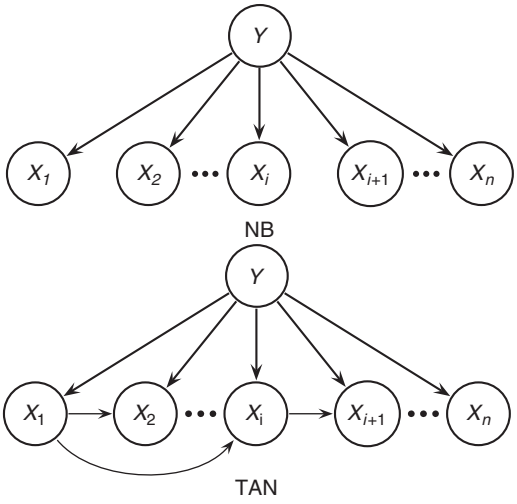
Tree Augmented Naive Bayes

Fei Zheng, Geoffrey I. Webb
Monash University

Synonyms
TAN

Definition
Tree augmented naive Bayes is a ►semi-naive Bayesian Learning method. It relaxes the ►naive Bayes attribute independence assumption by employing a tree structure, in which each attribute only depends on the class and one other attribute. A maximum weighted spanning tree that maximizes the likelihood of the training data is used to perform classification.

Classification with TAN
Interdependencies between attributes can be addressed directly by allowing an attribute to depend on other non-class attributes. However, techniques for learning unrestricted ►Bayesian networks often fail to deliver lower zero-one loss than naive Bayes (Friedman, Geiger, & Goldszmidt, 1997). One possible reason for this is that full Bayesian networks are oriented toward optimizing the likelihood of the training data rather than the conditional likelihood of the class attribute given a full set of other attributes. Another possible reason is that full Bayesian networks have high variance due



Tree Augmented Naive Bayes. Figure 1. Bayesian network examples of the forms of model created by NB and TAN

to the large number of parameters estimated. An intermediate alternative technique is to use a less restrict structure than naive Bayes. Tree augmented naive Bayes (TAN) (Friedman et al., 1997) employs a tree structure, allowing each attribute to depend on the class and at most one other attribute. Figure 1 shows Bayesian network representations of the types of model that NB and TAN respectively create.

Chow and Liu (1968) proposed a method that efficiently constructs a maximum weighted spanning tree which maximizes the likelihood that the training data was generated from the tree. The weight of an edge in the tree is the mutual information of the two attributes connected by the edge. TAN extends this method by using conditional mutual information as weights. Since the selection of root does not affect the log-likelihood of the tree, TAN randomly selects a root attribute and directs all edges away from it. The parent of each attribute X_i is indicated as $\pi(X_i)$ and the parent of the class is $\varnothing$. It assumes that attributes are independent given the class and their parents and classifies the test instance $\mathbf{x} = \langle x_1, \ldots, x_n \rangle$ by selecting

$$\underset{y}{\operatorname{argmax}} \hat{P}(y) \prod_{1 \le i \le n} \hat{P}(x_i \mid y, \pi(x_i)), \qquad (1)$$

where $\pi(x_i)$ is a value of $\pi(X_i)$ and y is a class label.

Due to the relaxed attribute independence assumption, TAN considerably reduces the ▶bias of naive Bayes at the cost of an increase in ▶variance. Empirical results (Friedman et al., 1997) show that it substantially reduces zero-one loss of naive Bayes on many data sets and that of all data sets examined it achieves lower zero-one loss than naive Bayes more often than not.

Cross References

▶Averaged One-Dependence Estimators
▶Bayesian Network
▶Naive Bayes
▶Semi-Naive Bayesian Learning

Recommended Reading

Chow, C. K., & Liu, C. N. (1968). Approximating discrete probability distributions with dependence trees. *IEEE Transactions on Information Theory, 14*, 462–467.

Friedman, N., Geiger, D., & Goldszmidt, M. (1997). Bayesian network classifiers. *Machine Learning, 29*(2), 131–163.

Tree Mining

Siegfried Nijssen
Katholieke Universiteit Leuven,
Belgium

Definition

Tree mining is an instance of constraint-based pattern mining and studies the discovery of tree patterns in data that is represented as a tree structure or as a set of trees structures. Minimum frequency is the most studied constraint.

Motivation and Background

Tree mining is motivated by the availability of many types of data that can be represented as tree structures. There is a large variety in tree types, for instance, ordered trees, unordered trees, rooted trees, unrooted (free) trees, labeled trees, unlabeled trees, and binary trees; each of these has its own application areas. An example are trees in tree banks, which store sentences annotated with parse trees. In such data, it is

not only of interest to find commonly occurring sets of words (for which frequent itemset miners could be used), but also to find commonly occurring parses of these words. Tree miners aim at finding patterns in this structured information. The patterns can be interesting in their own right, or can be used as features in classification algorithms.

Structure of Problem

All tree miners share a similar problem setting. Their input consists of a set of trees and a set of constraints, usually a minimum frequency constraint, and their output consists of all subtrees that fulfill the constraints.

Tree miners differ in the constraints that they are able to deal with, and the types of trees that they operate on. The following types of trees can be distinguished:

Free trees, which are graphs without cycles, and no order on the nodes or edges;

Unordered trees, which are free trees in which one node is chosen to be the root of the tree;

Ordered trees, which are rooted trees in which the nodes are totally ordered.

For each of these types of tree, we can choose to have labels on the nodes, or on the edges, or on both.

The differences between these types of trees are illustrated in Fig. 1. Every graph in this figure can be interpreted as a free tree F_i, an unordered tree U_i, or an ordered tree T_i. When interpreted as ordered trees, none of the trees are equivalent. When we interpret them as unordered trees, U_1 and U_2 are equivalent representations of the same unordered tree that has B as its root and C and D as its children. Finally, as free trees, not only F_1 and F_2 are equivalent, but also F_5 and F_7.

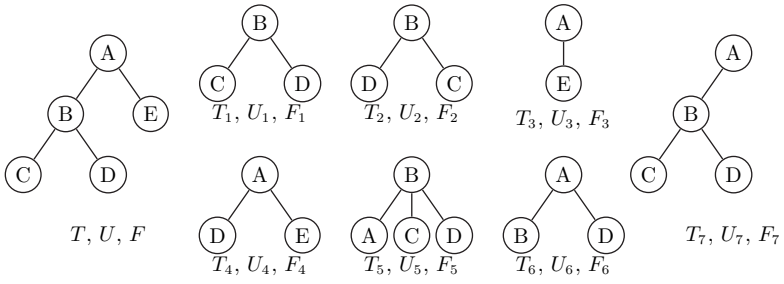
Intuitively, a free tree requires less specification than an ordered tree. The number of possible free trees is smaller than the number of possible ordered trees. On the other hand, to test if two trees are equivalent we need a more elaborate computation for free trees than for ordered trees.

Assume that we have data represented as (a set of) trees, then the data mining problem is to find patterns, represented as trees, that fulfill constraints based on this data. To express these constraints, we need a coverage relation that expresses when one tree can be considered to occur in another tree. Different coverage relations can be expressed for free trees, ordered trees, and unordered trees. We will introduce these relations through operations that can be used to transform trees. As an example, consider the operation that removes a leaf from a tree. We can repeatedly apply this operation to turn a large tree into a smaller one. Given two trees A and B, we say that A occurs in B as

Induced subtree, if A can be obtained from B by repeatedly removing leaves from B. When dealing with rooted trees, the root is here also considered to be a leaf if it has one child;

Root-induced subtree, if A can be obtained from B by repeatedly removing leaves from B. When dealing with rooted trees, the root is not allowed to be removed;

Embedded subtree, if A can be obtained from B by repeatedly either (1) removing a leaf or (2) removing an internal node, reconnecting the children of



Tree Mining. Figure 1. The leftmost tree is part of the data, the other trees could be patterns in this tree, depending on the subtree relation that is used

the removed node with the parent of the removed node;

Bottom-up subtree, if there is a node v in B such that if we remove all nodes from B that are not a descendant of v, we obtain A;

Prefix, if A can be obtained from B by repeatedly removing the last node from the ordered tree B;

Leaf set, if A can be obtained from B by selecting a set of leaves from B, and all their ancestors in B.

For free trees, only the induced subtree relation is well-defined. A prefix is only well-defined for ordered trees, the other relations apply both to ordered and unordered trees. In the case of unordered trees, we assume that each operation maintains the order of the original tree B. The relations are also illustrated in Fig. 2.

Intuitively, we can speak of *occurrences* (also called *embeddings* by some authors) of a small tree in a larger tree. Each such occurrence (or embedding) can be thought of as a function φ that maps every node in the small tree to a node in the large tree.

Using an occurrence relation, we can define frequency measures. Assume given a forest $\mathcal{F}$ of trees, all ordered, unordered, or free. Then the frequency of a tree A can be defined

Transaction-based, where we count the number of trees $B \in \mathcal{F}$ such that A is a subtree of B;

Node-based, where we count the number of nodes v in $\mathcal{F}$ such that A is a subtree of the bottom-up subtree below v.

Node-based frequency is only applicable in rooted trees, in combination with the root-induced, bottom-up, prefix, or leaf set subtree relations.

Given a definition of frequency, constraints on trees of interest can be expressed:

Minimum frequency, to specify that only trees with a certain minimum number of occurrences are of interest;

Closedness, to specify that a tree is only of interest if its frequency is different from all its supertrees;

Maximality, to specify that a tree is only of interest if none of its supertrees is frequent.

Observe that in all of these constraints, the subtree relation is again important. The subtree relation is not only used to compare patterns with data, but also patterns among themselves.

The tree mining problem can now be stated as follows. Given a forest of trees $\mathcal{F}$ (ordered, unordered, or free) and a set of constraints, based on a subtree relation, the task is to find all trees that satisfy the given constraints.

Tree	Induced	Embedded	Root-Induced	Bottom-up	Prefix	Leaf-set
T_1	yes	yes	no	yes	no	no
T_2	no	no	no	no	no	no
T_3	yes	yes	yes	no	no	yes
T_4	no	yes	no	no	no	no
T_5	no	no	no	no	no	no
T_6	no	no	no	no	no	no
T_7	yes	yes	yes	no	yes	yes

Tree	Induced	Embedded	Root-Induced	Bottom-up	Leaf-set
U_1	yes	yes	no	yes	no
U_2	yes	yes	no	yes	no
U_3	yes	yes	yes	no	yes
U_4	no	yes	no	no	no
U_5	no	no	no	no	no
U_6	no	no	no	no	no
U_7	yes	yes	yes	no	yes

Tree	Induced
F_1	yes
F_2	yes
F_3	yes
F_4	no
F_5	yes
F_6	no
F_7	yes

Tree Mining. Figure 2. Relations between the trees of Fig. 1

Theory/Solution

The tree mining problem is an instance of the more general problem of constraint-based pattern mining under constraints. For more information about the general setting, see the sections on constraint-based mining, itemset mining, and graph mining.

All algorithms iterate a process of generating candidate patterns, and testing if these candidates satisfy the constraints. Essential is to avoid that every possible tree is considered as a candidate. To this purpose, the algorithms exploit that many frequency measures are anti-monotonic. This property states that for two given

trees A and B, where A is a subtree of B, if A is infrequent, then also B is infrequent, and therefore, we do not need to consider it as a candidate.

This observation can make it possible to find all trees that satisfy the constraints, if these requirements are fulfilled:

- We have an algorithm to enumerate candidate subtrees, which satisfies these properties:
  - It should be able to enumerate all trees in the search space;
  - It should avoid that no two equivalent subtrees are listed;
  - It should only list a tree after at least one of its subtrees has been listed, to exploit the anti-monotonicity of the frequency constraint;
- We have an algorithm to efficiently compute in how many database trees a pattern tree occurs.

The algorithmic solutions to these problems depend on the type of tree and the subtree relation.

Encoding and Enumerating Trees

We will first consider how tree miners internally represent trees. Two types of encodings have been proposed, both of which are string-based. We will illustrate these encodings for node-labeled trees, and start with *ordered* trees.

The first encoding is based on a *preorder* listing of nodes: (1) for a rooted ordered tree T with a single vertex r, the *preorder string* of T is $S_{T,r} = l_r - 1$, where l_r is the label for the single vertex r, and (2) for a rooted ordered tree T with more than one vertex, assuming the root of T is r (with label l_r) and the children of r are $r_1, \ldots, r_K$ from left to right, then the preorder string for T is $S_{T,r} = l_r S_{T,r_1} \cdots S_{T,r_K} - 1$, where $S_{T,r_1}, \ldots, S_{T,r_K}$ are the preorder strings for the bottom-up subtrees below nodes $r_1, \ldots, r_K$ in T.

The second encoding is based on listing the *depths* of the nodes together with their labels in prefix-order. The depth of a node v is the length of the path from the root to the node v. The code for a tree is $S_{T,r} = d_r l_r S_{T,r_1} \cdots S_{T,r_K}$, where d_r is the depth of the node r in tree T.

Both encodings are illustrated in Fig. 3.

Tree	Depth-sequence	Preorder string
T_6	1*A*2*B*2*D*	*AB*-1*D*-1
T_7	1*A*2*B*3*C*3*D*	*ABC*-1*D*-1-1-1
T	1*A*2*B*3*C*3*D*2*E*	*ABC*-1*D*-1-1*E*-1
T_4	1*A*2*D*2*E*	*AD*-1*E*-1-1
T_3	1*A*2*E*	*AE*-1-1
T_5	1*B*2*A*2*C*2*D*	*BA*-1*C*-1*D*-1-1
T_1	1*B*2*C*2*D*	*BC*-1*D*-1-1
T_2	1*B*2*D*2*C*	*BD*-1*C*-1-1

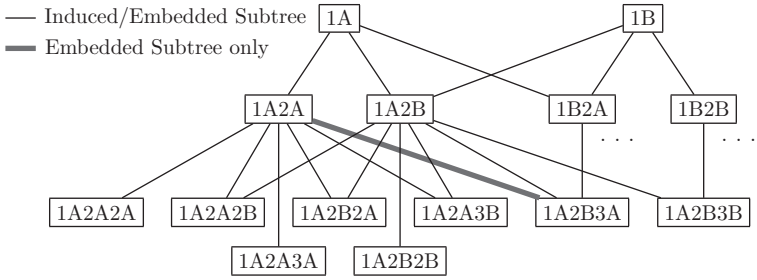
Tree Mining. Figure 3. Depth sequences for all the trees of Fig. 1, sorted in lexicographical order. Tree T_2 is the canonical form of unordered tree U_2, as its depth sequence is the highest among equivalent representations

A search space of trees can be visualized as in Fig. 4. In this figure, every node corresponds to the depth encoding of a tree, while the edges visualize the partial order defined by the subtree relation. It can be seen that the number of induced subtree relations between trees is smaller than the number of embedded subtree relations.

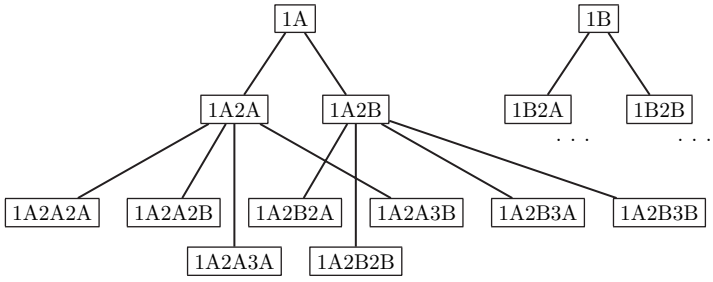
The task of the enumeration algorithm is to traverse this search space starting from trees that contain only one node. Most algorithms perform the search by building an enumeration tree over the search space. In this enumeration tree every pattern should have a single parent. The children of a pattern in the enumeration tree are called its *extensions* or its *refinements*. An example of an enumeration tree for the induced subtree relation is given in Fig. 5.

In the enumeration tree that is given here, the parent of a tree is its prefix in the depth encoding. An alternative definition is that the parent of a tree can be obtained by removing the last node in a prefix order traversal of the ordered tree. Every refinement in the enumeration has one additional node that is connected to the *rightmost path* of the parent.

The enumeration problem is more complicated for *unordered trees*. In this case, the trees represented by the strings 1*A*2*A*2*B* and 1*A*2*B*2*A* are equivalent, and we



Tree Mining. Figure 4. A search space of ordered trees, where edges denote subtree relationships



Tree Mining. Figure 5. Part of an enumeration tree for the search space of Fig. 4

only wish to enumerate one of these strings. This can be achieved by defining a total order on all strings that represent trees, and to define that only the highest (or lowest) string of a set of equivalent strings should be considered.

For depth encodings, the ordering is usually lexicographical, and the highest string is chosen to be the *canonical* encoding. In our example, 1*A*2*B*2*A* would be canonical. This code has the desirable property that every prefix of a canonical code is also a canonical code. Furthermore it can be determined in polynomial time which extensions of a canonical code lead to a canonical code, such that it is not necessary to consider any code that is not canonical.

Alternative codes have also been proposed, which are not based on a preorder, depth-first traversal of a tree, but on a level-wise listing of the nodes in a tree.

Finally, for *free trees* we have the additional problem that we de not have a root for the tree. Fortunately, it is known that every free tree either has a uniquely determined *center* or a uniquely determined *bicenter*. This (bi)center can be found by determining the longest path between two nodes in a free tree: the node(s) in the middle of this path are the center of the tree. It can be shown that if multiple paths have the same maximal length, they will have the same (bi)center. By appointing one center to be the root, we obtain a rooted tree, for which we can compute a code.

To avoid that two codes are listed that represent equivalent free trees, several solutions have been proposed. One is based on the idea of first enumerating paths (thus fixing the center of a tree), and for each of these paths enumerating all trees that can be grown around them. Another solution is based on enumerating all rooted, unordered trees under the constraint that at least two different children of the root have a bottom-up subtree of equal, maximal depth. In the first approach, a preorder depth encoding was used; in the second approach a level-wise encoding was used.

Counting Trees

To evaluate the frequency of a tree the subtree relation between a candidate pattern tree and all trees in the database has to be computed. For each of our subtree relations, polynomial algorithms are known to decide the relation, which are summarized in Table 1.

Tree Mining. Table 1 Worst case complexities of the best known algorithms that determine whether a tree relation holds between two trees; *m* is the number of nodes in the pattern tree, *l* is the number of leafs in the pattern tree, *n* the number of nodes in the database tree

	Ordered	Unordered
Embedding	$O(nl)$	NP-complete
Induced	$O(nm)$	$O(nm^{1\frac{1}{2}}/\log m)$
Root-induced	$O(n)$	$O(nm^{1\frac{1}{2}}/\log m)$
Leaf-set	$O(n)$	$O(nm^{1\frac{1}{2}}/\log m)$
Bottom-up	$O(n)$	$O(n)$
Prefix	$O(m)$	

Even though a subtree testing algorithm and an algorithm for enumerating trees are sufficient to compute all frequent subtrees correctly, in practice fine-tuning is needed to obtain an efficient method. There are two reasons for this:

- In some databases, the number of candidates can by far exceed the number of trees that are actually frequent. One way to reduce the number of candidates is to only generate a particular candidate after we have encountered at least one occurrence of it in the data (this is called *pattern growth*); another way is to require that a candidate is only generated if at least two of its subtrees satisfy the constraints (this is called *pattern joining*).
- The trees in the search space are very similar to each other: a parent only differs from its children by the absence of a single node. If memory allows, it is desirable to *reuse* the subtree matching information, instead of starting the matching from scratch.

A large number of data structures have been proposed to exploit these observations. We will illustrate these ideas using the FreqT algorithm, which mines induced, ordered subtrees, and uses a depth encoding for the trees.

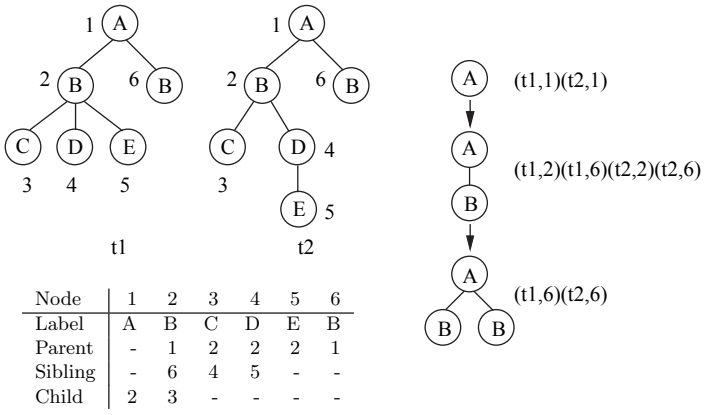
In FreqT, for a given pattern tree A, a list of (database tree, database node) pointers is stored. Every element (B, v) in this list corresponds to an occurrence of tree A in tree B in which the last node (in terms of the preorder) of A is mapped to node v in database tree B. For a database and three example trees this is illustrated in Fig. 6.

Every tree in the database is stored as follows. Every node is given an index, and for every node, we store the index of its parent, its righthand sibling, and its first child.

Let us consider how we can compute the occurrences of the subtree 1*A*2*B*2*B* from the occurrences of the tree 1*A*2*B*. The first occurrence of 1*A*2*B* is $(t1, 2)$, which means that the *B* labeled node can be mapped to node 2 in $t1$. Using the arrays that store the database tree, we can then conclude that node 6, which is the right-hand sibling of node 2, corresponds to an occurence of the subtree 1*A*2*B*2*B*. Therefore, we add $(t1, 6)$ to the occurrence list of 1*A*2*B*2*B*. Similarly, by scanning the data we find out that the first child of node 2 corresponds to an occurrence of the subree 1*A*2*B*3*C*, and we add $(t1, 3)$ to the occurrence list of 1*A*2*B*3*C*.

Overall, using the parent, sibling and child pointers we can scan every node in the data that could correspond to a valid expansion of the subtree 1*A*2*B*, and update the corresponding lists. After we have done this for every occurrence of the subtree, we know the occurrence lists of all possible extensions.

From an occurrence list we can determine the frequency of a tree. For instance, the transaction-based frequency can be computed by counting the number of different database trees occurring in the list.



Node	1	2	3	4	5	6
Label	A	B	C	D	E	B
Parent	-	1	2	2	2	1
Sibling	-	6	4	5	-	-
Child	2	3	-	-	-	-

Tree Mining. Figure 6. A tree database (left) and three ordered trees with their occurrence lists according to the FreqT algorithm (right). The datastructure that stores *t*1 in FreqT is given in the table (right)

As we claimed, this example illustrates two features that are commonly seen in tree miners: first, the occurrence list of one tree is used to compute the occurrence list of another tree, thus reusing information; second, the candidates are collected from the data by scanning the nodes that connect to the occurrence of a tree in the data. Furthermore, this example illustrates that a careful design of the datastructure that stores the data can ease the frequency evaluation considerably.

FreqT does not perform pattern joining. The most well-known example of an algorithm that performs tree joining is the embedded TreeMiner (Zaki, 2002). Both the FreqT and the TreeMiner perform the search depth-first, but also tree miners that use the traditional level-wise approach of the Apriori algorithm have been proposed. The FreqT and the TreeMiner have been extended to unordered trees.

Other Constraints

As the number of frequent subtrees can be very large, approaches have been studied to reduce the number of trees returned by the algorithm, of which closed and maximal trees are the most popular. To find closed or maximal trees, two issues need to be addressed:

- How do we make sure that we only output a tree if it is closed or maximal, that is, how do we determine that none of its supertrees has the same support, or is frequent?
- Can we conclude that some parts of the search space will never contain a closed or maximal tree, thus making the search more efficient?

Two approaches can be used to address the first issue:

- All closed patterns can be stored, and every new pattern can be compared with the stored set of patterns;
- When we evaluate the frequency of a pattern in the data, we also (re)evaluate the frequency of all its possible extensions, and only output the pattern if its support is different.

The second approach requires less memory, but in some cases requires more computations.

To prune the search space, a common approach is to check all occurrences of a tree in the data. If every occurrence of a tree can be extended into an occurrence of another tree, the small tree should not be considered, and the search should continue with the tree that contains all common edges and nodes. Contrary to graph mining, it can be shown that this kind of pruning can safely be done in most cases.

Applications

Examples of databases to which tree mining algorithms have been applied are

Parse tree analysis: Since the early 1990s large *Treebank* datasets have been collected consisting of

sentences and their grammatical structure. An example is the Penn TreeBank (Marcus, Santorini, & Marcinkiewicz, 1993). These databases contain rooted, ordered trees. To discover differences in domain languages it is useful to compare commonly occurring grammatical constructions in two different sets of parsed texts, for which tree miners can be used (Sekine, 1998).

Computer network analysis: IP *multicast* is a protocol for sending data to multiple receivers. In an IP multicast session a webserver sends a packet once; routers copy a packet if two different routes are required to reach multiple receivers. During a multicast session rooted trees are obtained in which the root is the sender and the leaves are the receivers. Commonly occurring patterns in the routing data can be discovered by analyzing these unordered rooted trees (Chalmers & Almeroth, 2003).

Webserver access log analysis: When users browse a website, this behavior is reflected in the access log files of the webserver. Servers collect information such as the webpage that was visited, the time of the visit, and the webpage that was clicked to reach the webpage. The access logs can be transformed into a set of ordered trees, each of which corresponds to a visitor. Nodes in these trees correspond to webpages; edges are inserted if a user browses from one webpage to another. Nodes are ordered in viewing order. A tool was developed to perform this transformation in a sensible way (Punin, Krishnamoorthy, & Zaki, 2002).

Phylogenetic trees: One of the largest tree databases currently under construction is the TreeBASE database, which is comprised of a large number of phylogenetic trees (Morell, 1996). The trees in the TreeBASE database are submitted by researchers and are collected from publications. Originating from multiple sources, they can disagree on parts of the phylogenetic tree. To find common agreements between the trees, tree miners have been used (Zhang & Wang, 2005). The phylogenetic trees are typically unordered; labels among siblings are unique.

Hypergraph mining: Hypergraphs are graphs in which one edge can have more than two endpoints. Those hypergraphs in which no two nodes share the same label can be transformed into unordered trees, as follows. First, an artificial root is inserted. Second, for each edge of the hypergraph a child node is added to the root, labeled with the label of the hyperedge. Finally, the labels of nodes within hyperedges are added as leaves to the tree. An example of hypergraph data is bibliographic data: if each example corresponds to a paper, nodes in the hypergraph correspond to authors cited by the paper, and hyperedges connect coauthors of cited papers.

Multi-relational data mining: Many multi-relational databases are tree shaped, or a tree-shaped view can be created. For instance, a transaction database in which every transaction is associated with customers and their information, can be represented as a tree (Berka, 1999).

XML data mining: Several authors have stressed that tree mining algorithms are most suitable for mining XML data. XML is a tree–shaped data format, and tree miners can be helpful when trying to (re)construct Document Type Definitions (DTDs) for such documents.

Cross References

►Constraint-based Mining
►Graph Mining

Further Reading

The FreqT algorithm was introduced in (Asai, Abe, Kawasoe, Arimura, Satamoto, & Arikawa, 2002; Wang & Liu, 1998; Zaki, 2002). The most popular tree miner is the embedded tree miner by Zaki (2002). A more detailed overview of tree miners can be found in Chi, Nijssen, Muntz, and Kok (2005). Most implementations of tree miners are available on request from their authors.

Recommended Reading

Asai, T., Abe, K., Kawasoe, S., Arimura, H., Satamoto, H., & Arikawa, S. (2002). Efficient substructure discovery from large semi-structured data. In *Proceedings of the second SIAM international conference on data mining* (pp. 158–174). SIAM.

Berka, P. (1999). *Workshop notes on discovery challenge PKDD-99* (Tech. Rep.). Prague, Czech Republic: University of Economics.

Chalmers, R., & Almeroth, K. (2003). On the topology of multicast trees. In *IEEE/ACM transactions on networking* (Vol. 11, pp. 153–165). IEEE Press/ACM Press.

Chi, Y., Nijssen, S., Muntz, R. R., & Kok, J. N. (2005). Frequent subtree mining—An overview. In *Fundamenta Informaticae* (Vol. 66, pp. 161–198). IOS Press.

Marcus, M. P., Santorini, B., & Marcinkiewicz, M. A. (1993). Building a large annotated corpus of English: The Penn Treebank. In *Computational linguistics* (Vol. 19, pp. 313–330). MIT Press.

Morell, V. (1996). TreeBASE: The roots of phylogeny. In *Science* (Vol. 273, p. 569).

Punin, J., Krishnamoorthy, M., & Zaki, M. J. (2002). LOGML—log markup language for web usage mining. In *WEBKDD 2001—mining web log data across all customers touch points. Third international workshop. Lecture notes in artificial intelligence* (Vol. 2356, pp. 88–112). Springer.

Sekine, S. (1998). *Corpus-based parsing and sublanguages studies*. Ph.D. dissertation. New York University, New York.

Wang, K., & Liu, H. (1998). Discovering typical structures of documents: A road map approach. In *Proceedings of the 21st annual international ACM SIGIR conference on research and development in information retrieval* (pp. 146–154). ACM Press.

Zaki, M. J. (2002). Efficiently mining frequent trees in a forest. In *Proceedings of the 8th international conference knowledge discovery and data mining* (*KDD*) (pp. 71–80). ACM Press.

Zhang, S., & Wang, J. (2005). Frequent agreement subtree mining. http://aria.njit.edu/mediadb/fast/.

Tree-Based Regression

►Regression Trees

True Negative

True negatives are the negative examples that are correctly classified by a classification model. See ►confusion matrix for a complete range of related terms.

True Negative Rate

►Specificity

True Positive

True positives are the positive examples that are correctly classified by a classification model. See ►confusion matrix for a complete range of related terms.

True Positive Rate

►Sensitivity

Type

►Class

Typical Complexity of Learning

►Phase Transitions in Machine Learning