O

Object

▸ Instance

Object Consolidation

▸ Entity Resolution

Object Identification

▸ Record Linkage

Object Matching

▸ Record Linkage

Object Space

▸ Instance Space

Objective Function

▸ Partitional Clustering

Observation Language

Hendrik Blockeel
Katholieke Universiteit Leuven, Heverlee, Leuven, Belgium
Leiden Institute of Advanced Computer Science, Heverlee, Belgium

Synonyms

Instance language

Definition

The *observation language* used by a machine learning system is the language in which the observations it learns from are described.

Motivation and Background

Most machine learning algorithms can be seen as a procedure for deriving one or more hypotheses from a set of observations. Both the input (the observations) and the output (the hypotheses) need to be described in some particular language and this language is called the observation language or the ▸ Hypothesis Language respectively. These terms are mostly used in the context of symbolic learning, where these languages are often more complex than in subsymbolic or statistical learning.

C. Sammut, G.I. Webb (eds.), *Encyclopedia of Machine Learning and Data Mining*,
DOI 10.1007/978-1-4899-7687-1

The following sections describe some of the key observation languages.

Attribute-Value Learning

Probably the most used setting in machine learning is the *attribute-value* setting (see ▸ Attribute-Value Learning). Here, an example (observation) is described by a fixed set of attributes, each of which is given a value from the domain of the attribute. Such an observation is often called a vector or, in relational database terminology, a tuple. The attributes are usually atomic (i.e., not decomposable in component values) and single-valued (i.e., an attribute has only one value, not a set of values). So we have an instance space (or space of observations)

$$\mathcal{O} = A_1 \times \cdots \times A_n,$$

elements of which are denoted using an observation language that typically has the same structure:

$$\mathcal{L}_O = \mathcal{L}_{A_1} \times \cdots \times \mathcal{L}_{A_n},$$

(the language contains tuples of objects that represent the attribute values).

The attribute-value framework easily allows for both supervised and unsupervised learning; in the supervised learning setting, the label of an instance is simply included as an attribute in the tuple, where as for unsupervised learning, it is excluded.

The attribute-value setting assumes that all instances can be represented using the same fixed set of attributes. When instances can be of different types or are variable-sized (e.g., when an instance is set-valued), this assumption may not hold, and more powerful languages may have to be used instead.

Learning from Graphs, Trees, or Sequences

We here consider the case in which a single instance is a graph, or a node in a graph. Note that trees and sequences are special cases of graphs.

A graph is defined as a pair (V, E), where V is a set of vertices and E a set of edges each edge being a pair of vertices. If the pair is ordered, the graph is directed; otherwise it is undirected. For simplicity, we restrict ourselves to undirected graphs.

A graph can, in practice, not be encoded in attribute-value format without the loss of information. That is, one could use a number of properties of graphs as attributes in the encoding, but several graphs may then still map onto the same representation, which implies loss of information. In theory, one could imagine defining a total order on (certain classes of) graphs and representing each graph by its rank in that order (which is a single numerical attribute), thus representing graphs as numbers without loss of information; but then it is not obvious how to map patterns in this numerical representation to patterns in the original representation. No such approaches have been proposed till now.

Describing the instance space is more difficult here than in the attribute value case. Consider a task of graph classification, where in observations are of the form (G, y) with G a graph and y a value for a target attribute Y. Then we can define the instance space as

$$\mathcal{O} = \{(V, E) | V \subseteq \mathbf{N} \wedge E \subseteq V^2\} \times Y,$$

where $\mathbf{N}$ is the set of all natural numbers. (For each graph, there exists a graph defined over $\mathbf{N}$ that is isomorphic with it, so $\mathcal{O}$ contains all possible graphs up to isomorphism.)

A straightforward observation language in the case of graph classification is then

$$\{(G, y) | G = (V, E) \wedge V \subseteq \mathcal{L}_V \wedge E \subseteq V^2 \wedge y \in Y\},$$

where $\mathcal{L}_V$ is some alphabet for representing nodes.

In learning from graphs, there are essentially two settings: those where a prediction is made for entire graphs, and those where a prediction is made for single nodes in a graph. In the first case, observations are of the form (G, y), where

Anne	1997
Bernard	1999
Celine	1996
Daniel	1999
Elisa	1997
Fabian	1999

Anne	Algebra	1998	A
Anne	Calculus	1998	B
Bernard	Databases	2000	A
Celine	Biology	1999	B
Celine	Databases	2000	B
Celine	Calculus	1998	A

Algebra
Biology
Calculus
Databases

Adams	Algebra	1998
Adams	Calculus	1999
Baeck	Biology	1999
Cools	Calculus	1998
Cools	Databases	1999

Adams
Baeck
Cools

Observation Language, Fig. 1 A small database of students

as, in the second case, they are of the form (G, v, y), where $G = (V, E)$ and $v \in V$. That is, a node is given together with the graph in which it occurs (its "environment"), and a prediction is to be made for this specific node, using the information about its environment.

In many cases, the set of observations one learns from is of the form (G, v_i, y_i), where each instance is a different node of exactly the same graph G. This is the case when, for instance, classifying web pages, we take the whole web as their environment.

In a labeled graph, labels are associated with each node or edge. Often these are assumed atomic, being elements of a finite alphabet or real numbers, but they can also be vectors of reals.

Relational Learning

In ▸ relational learning, it is assumed that relationships may exist between different instances of the instance space, or an instance may internally consist of multiple objects among which relationships exist.

This essentially corresponds to learning from graphs, except that in a graph only one binary relation exists (the edges E), whereas here there may be multiple relations and they may be non binary. The expressiveness of the two settings is the same, however, as any relation can be represented using only binary relations.

In the attribute-value setting, one typically uses one table where each tuple represents all the relevant information for one observation. In the relational setting, there may be multiple tables, and information on a single instance is contained in multiple tuples, possibly belonging to multiple relations.

Example 1 Assume we have a database about students, courses, and professors (see Fig. 1). We can define a single observation as all the information relevant to one student, that is: the name, year of entrance, etc. of the student and also the courses they take and the professors teaching these courses.

The most obvious link to the graph representation is as follows: create one node for each tuple, labeled with that tuple, and create a link between two nodes if the corresponding tuples are connected by a foreign key relationship.

Defining a single observation as a set of tuples that are connected through foreign keys in the database corresponds to representing each observation (G, v, y) as (G', v, y), where G' is the connected component of G that contains v. The actual links are usually not explicitly written in this representation, as they are implicit: there is an edge between two tuples if they have the same value for a foreign key attribute.

Inductive Logic Programming

In ▸ inductive logic programming, a language based on first order logic is used to represent the observations. Typically, an observation is then

represented by a *ground fact*, which basically corresponds to a single tuple in a relational database. In some settings an observation is represented by an *interpretation*, a set of ground facts, which corresponds to the set of tuples mentioned in the previous subsection.

While the target variable can always be represented as an additional attribute, ILP systems often learn from examples and counterexamples of a concept. The target variable is then implicit: it is true or false depending on whether the example is in the positive or negative set, but it is not explicitly included in the fact.

Typical for the inductive logic programming setting is that the input of a system may contain, besides the observations, background knowledge about the application domain. The advantage of the ILP setting is that no separate language is needed for such background knowledge: the same first order logic-based language can be used for representing the observations as well as the background knowledge.

Example 2 Take the following small dataset:

```
sibling(bart,lisa).
sibling(lisa,bart).
:- sibling(bart, bart).
:- sibling(lisa, lisa).
father(homer, bart).
mother(marge, bart).
father(homer, lisa).
mother(marge, lisa).
```

There are positive and negative (preceded by :-) examples of the Sibling relation. The following hypothesis might be learned:

```
sibling(X,Y) :- father(Z,X),
father(Z,Y), X ≠ Y.
sibling(X,Y) :- mother(Z,X),
mother(Z,Y), X ≠ Y.
```

If the following clauses as included as background knowledge:

```
parent(X,Y) :- father(X,Y).
parent(X,Y) :- mother(X,Y).
```

then the same ILP system might learn the following more compact definition:

```
sibling(X,Y) :- parent(Z,X),
parent(Z,Y), X ≠ Y.
```

Further Reading

Most of the literature on hypothesis and observation languages is found in the area of inductive logic programming. Excellent starting points to become familiar with this field are *Relational Data Mining* by Džeroski and Lavraè (2001) and *Logical and Relational Learning* by De Raedt (2008).

De Raedt (1998) compares a number of different observation and hypothesis languages with respect to their expressiveness, and indicates relationships between them.

Cross-References

- Hypothesis Language
- Inductive Logic Programming
- Relational Learning

Recommended Reading

De Raedt L (1998) Attribute-value learning versus inductive logic programming: the missing links (extended abstract). In: Page D (ed) Proceedings of the eighth international conference on inductive logic programming. Lecture notes in artificial intelligence, vol 1446. Springer, Berlin, pp 1–8

De Raedt L (2008) Logical and relational learning. Springer, Berlin

Džeroski S, Lavraè N (eds) (2001) Relational data mining. Springer, Berlin.

Occam's Razor

Geoffrey I. Webb
Faculty of Information Technology, Monash University, Victoria, Australia

Synonyms

Ockham's Razor

Definition

Occam's Razor is the maxim that "entities are not to be multiplied beyond necessity," or as it is often interpreted in the modern context "of two hypotheses H and H', both of which explain E, the simpler is to be preferred" (Good 1977)

Motivation and Background

Most attempts to learn a model from data confront the problem that there will be many models that are consistent with the data. In order to learn a single model, a choice must be made between the available models. The factors taken into account by a learner in choosing between models are called its learning biases (Mitchell 1980). A preference for simple models is a common learning bias and is embodied in many learning techniques including pruning, minimum message length, and minimum description length. Regularization is also sometimes viewed as an application of Occam's razor.

Occam's razor is an imperative, rather than a proposition. That is, it is neither true nor false. Rather, it is a call to act in a particular way without making any claim about the consequences of doing so. In machine learning the so-called *Occam thesis* is sometimes assumed that: given a choice between two plausible classifiers that perform identically on the training set, the simpler classifier is expected to classify correctly more objects outside the training set. (Webb 1996)

While there are many practical advantages in having a learning bias toward simple models, there remains controversy as to whether the Occam thesis is true (Webb 1996; Domingos 1999; Blumer et al. 1987).

Cross-References

- ▸ Learning Bias
- ▸ Language Bias
- ▸ Minimum Description Length Principle
- ▸ Minimum Message Length
- ▸ Pruning
- ▸ Regularization

Recommended Reading

Blumer A, Ehrenfeucht A, Haussler D, Warmuth MK (1987) Occam's razor. Inf Process Lett 24(6): 377–380

Domingos P (1999) The role of Occam's razor in knowledge discovery. Data Min Knowl Discov 3(4):409–425

Good IJ (1977) Explicativity: a mathematical theory of explanation with statistical applications. Proc R Soc Lond Ser A 354:303–330

Mitchell TM (1980) The need for biases in learning generalizations. Department of computer science, Technical report CBM-TR-117, Rutgers University

Webb GI (1996) Further experimental evidence against the utility Of occams razor. J Artif Intell Res 4:397–417. AAAI Press, Menlo Park

Ockham's Razor

▸ Occam's Razor

Offline Learning

▸ Batch Learning

One-Against-All Training

▸ Class Binarization

One-Against-One Training

▸ Class Binarization

1-Norm Distance

▸ Manhattan Distance

One-Step Reinforcement Learning

▸ Associative Reinforcement Learning

Online Controlled Experiments and A/B Testing

Ron Kohavi[1] and Roger Longbotham[2]
[1]Application Services Group, Microsoft, Bellevue, WA, USA
[2]Data and Decision Sciences Group, Microsoft, Redmond, WA, USA

Abstract

The Internet connectivity of client software (e.g., apps running on phones and PCs), websites, and online services provide an unprecedented opportunity to evaluate ideas quickly using controlled experiments, also called A/B tests, split tests, randomized experiments, control/treatment tests, and online field experiments. Unlike most data mining techniques for finding correlational patterns, controlled experiments allow establishing a causal relationship with high probability. Experimenters can utilize the scientific method to form a hypothesis of the form "If a specific change is introduced, will it improve key metrics?" and evaluate it with real users.

The theory of a controlled experiment dates back to Sir Ronald A. Fisher's experiments at the Rothamsted Agricultural Experimental Station in England in the 1920s, and the topic of offline experiments is well developed in Statistics (Box et al., Statistics for experimenters: design, innovation, and discovery. Wiley, Hoboken, 2005). Online-controlled experiments started to be used in the late 1990s with the growth of the Internet. Today, many large sites, including Amazon, Bing, Facebook, Google, LinkedIn, and Yahoo!, run thousands to tens of thousands of experiments each year testing user interface (UI) changes, enhancements to algorithms (search, ads, personalization, recommendation, etc.), changes to apps, content management system, etc. Online-controlled experiments are now considered an indispensable tool, and their use is growing for startups and smaller websites. Controlled experiments are especially useful in combination with Agile software development (Martin, Clean code: a handbook of Agile software craftsmanship. Prentice Hall, Upper Saddle River, 2008; Rubin, Essential scrum: a practical guide to the most popular Agile process. Addison-Wesley Professional, Upper Saddle River, 2012), Steve Blank's Customer Development process (Blank, The four steps to the epiphany: successful strategies for products that win. Cafepress.com., 2005), and MVPs (minimum viable products) popularized by Eric Ries's Lean Startup (Ries, The lean startup: how today's entrepreneurs use continuous innovation to create radically successful businesses. Crown Business, New York, 2011).

Synonyms

A/B Testing; Randomized Experiments; Split Tests

Motivation and Background

Many good resources are available with motivation and explanations about online-controlled experiments (Siroker and Koomen 2013; Goward 2012; McFarland 2012b; Schrage 2014; Kohavi et al. 2009, 2014, 2013).

We provide a motivating visual example of a controlled experiment that ran at Microsoft's Bing. The team wanted to add a feature allowing advertisers to provide links to the target site. The rationale is that this will improve ads' quality by giving users more information about what the advertiser's site provides and allows users to directly navigate to the subcategory matching their intent. Visuals of the existing ads layout (control) and the new ads layout (treatment) with site links added are shown in Fig. 1.

In a controlled experiment, users are randomly split between the variants (e.g., the two different ads layouts) in a persistent manner (a user receives the same experience in multiple visits). Their interactions with the site are instrumented

Online Controlled Experiments and A/B Testing, Fig. 1 Ads with site link experiment. Treatment (*bottom*) has site links. The difference might not be obvious at first but it is worth tens of millions of dollars

and key metrics computed. In this experiment, the Overall Evaluation Criterion (OEC) was simple: increasing average revenue per user to Bing without degrading key user engagement metrics. Results showed that the newly added site links increased revenue, but also degraded user metrics and page load time, likely because of increased vertical space usage. Even offsetting the space by lowering the average number of mainline ads shown per query, this feature improved revenue by tens of millions of dollars per year with neutral user impact, resulting in extremely high ROI (return on investment).

Running online-controlled experiments is not applicable for every organization. We begin with key tenets, or assumptions, an organization needs to adopt (Kohavi et al. 2013).

Tenet 1: The Organization Wants to Make Data-Driven Decisions and Has Formalized the Overall Evaluation Criterion (OEC)

You will rarely hear someone at the head of an organization say that they don't want to be data-driven, but measuring the incremental benefit to users from new features has costs, and objective measurements typically show that progress is not as rosy as initially envisioned. In any organization, there are many important metrics reflecting revenue, costs, customer satisfaction, loyalty, etc., and very frequently an experiment will improve one but hurt another of these metrics. Having a single metric, which we call the Overall Evaluation Criterion, or OEC, that is at a higher level than these and incorporates the trade-off among them is essential for organizational decision-making.

An OEC has to be defined, and it should be measurable over relatively short durations (e.g., 2 weeks). The hard part is finding metrics that are measurable in the short-term that are predictive of long term goals. For example, "profit" is not a good OEC, as short-term theatrics (e.g., raising prices) can increase short-term profit, but hurt it in the long run. As shown in *Trustworthy Online Controlled Experiments: Five Puzzling Outcomes Explained* (Kohavi et al. 2012), market share can be a long-term goal, but it is a terrible short-term criterion: making a search engine worse forces people to issue more queries to find an answer, but, like hiking prices, users will find better alternatives long-term. Sessions per user, or repeat visits, is a much better OEC for a search engine. Thinking of the drivers of lifetime value can lead to a strategically powerful OEC (Kohavi et al. 2009). We cannot overemphasize the importance of coming up with a good OEC that the organization can align behind.

Tenet 2: Controlled Experiments Can Be Run and Their Results Are Trustworthy

Not every decision can be made with the scientific rigor of a controlled experiment. For example, you cannot run a controlled experiment on the possible acquisition of one company by another. Hardware devices may have long lead times for manufacturing, and modifications are

hard, so controlled experiments with actual users are hard to run on a new phone or tablet. For customer-facing websites and services, changes are easy to make through software, and running controlled experiments is relatively easy.

Assuming you can run controlled experiments, it is important to ensure their trustworthiness. When running online experiments, getting numbers is easy; getting numbers you can trust is hard, and we have had our share of pitfalls and puzzling results (Kohavi et al. 2012, 2010; Kohavi and Longbotham 2010).

Tenet 3: We Are Poor at Assessing the Value of Ideas

Features are built because teams believe they are useful, yet in many domains, most ideas fail to improve key metrics. Only one third of the ideas tested on the Experimentation Platform at Microsoft improved the metric(s) they were designed to improve (Kohavi et al. 2009). Success is even harder to find in well-optimized domains like Bing. Jim Manzi (2012) wrote that at Google, only "about 10 percent of these [controlled experiments, were] leading to business changes." Avinash Kaushik wrote in his Experimentation and Testing primer (Kaushik 2006) that "80 % of the time you/we are wrong about what a customer wants." Mike Moran (2007, 240) wrote that Netflix considers 90 % of what they try to be wrong. Regis Hadiaris from Quicken Loans wrote that "in the five years I've been running tests, I'm only about as correct in guessing the results as a major league baseball player is in hitting the ball. That's right - I've been doing this for 5 years, and I can only "guess" the outcome of a test about 33 % of the time!" (Moran 2008). Dan McKinley at Etsy wrote (McKinley 2013) "nearly everything fails" and "it's been humbling to realize how rare it is for them [features] to succeed on the first attempt. I strongly suspect that this experience is universal, but it is not universally recognized or acknowledged." Finally, Colin McFarland wrote in the book *Experiment!* (McFarland 2012b, 20) "No matter how much you think it's a no-brainer, how much research you've done, or how many competitors are doing it, sometimes, more often than you might think, experiment ideas simply fail."

Not every domain has such poor statistics, but most who have run controlled experiments in customer-facing websites and applications have experienced this humbling reality: we are poor at assessing the value of ideas, and that is the greatest motivation for getting an objective assessment of features using controlled experiments.

Structure of an Experimentation System

Elements of an Experimentation System

The simplest experimental setup is to evaluate a factor with two levels, a control (version A) and a treatment (version B). The control is normally the default version, and the treatment is the change that is tested. Such a setup is commonly called an A/B test. It is commonly extended by having several levels, often referred to as A/B/n split tests. An experiment with multiple factors is referred to as multivariable (or multivariate).

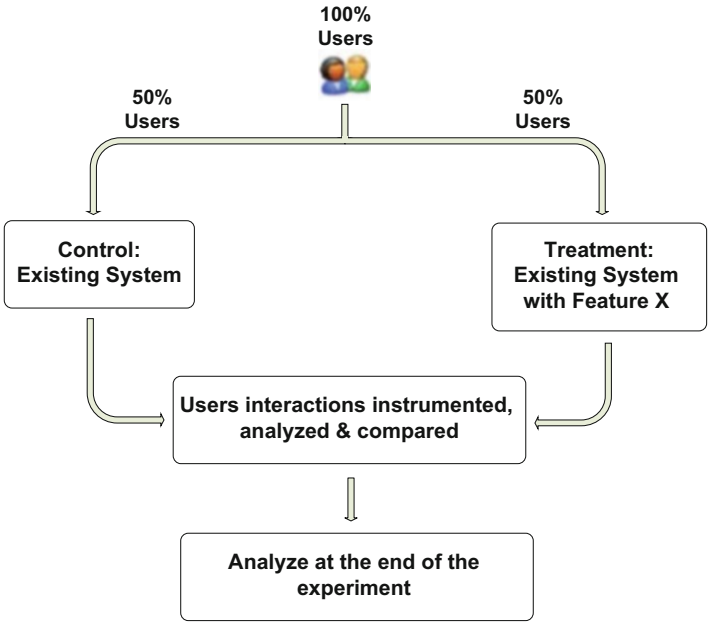
Figure 2 shows the high-level structure of an A/B experiment. In practice, one can assign any percentages to the treatment and control, but 50 % provides the experiment the maximum statistical power, and we recommend maximally powering the experiments after a ramp-up period at smaller percentages to check for egregious errors.

In a general sense, the analysis will test if the statistical distribution of the treatment is different from that of the control. In practice, the most common test is whether the two means are equal or not. For this case, the effect of version B (or treatment effect) is defined to be

$$E(B) = \bar{X}_B - \bar{X}_A \tag{1}$$

where X is a metric of interest and $\bar{X}_B$ is the mean for variant B. However, for interpretability, the percent change is normally reported with a suitable (e.g., 95 %) confidence interval. See, for example, Kohavi et al. (2009).

Control of extraneous factors and randomization are two essential elements of any experimentation system. Any factor that may affect



Online Controlled Experiments and A/B Testing, Fig. 2 High-level structure of an online experiment

an online metric is either a test factor (one you intentionally vary to determine its effect) or a non-test factor. Non-test factors could either be held fixed, blocked, or randomized. Holding a factor fixed can impact external validity and is thus not recommended. For example, if weekend days are known to be different from week days, you could run the experiment only on weekdays (or weekends), but it would be better to have complete weeks in the experiment for better external validity. Blocking (e.g., pairing) can reduce the variance relative to randomization and is recommended when experimentation units in each block are more homogenous than between blocks. For example, if the randomization unit is a user page view, then blocking by weekend/weekday can reduce the variance of the effect size, leading to higher sensitivity. Time is a critical non-test factor, and because many external factors vary with time, it is important to randomize over time by running the control and treatment(s) concurrently with a fixed percentage to each throughout the experiment. (If the relative percentage changes, you will be subject to Simpson's paradox (Malinas and Bigelow 2009; Kohavi and Longbotham 2010).) Controlling a non-test factor assures it will have equal influence on the control and treatment, hence not affecting the estimate of the treatment effect.

Experimentation Architecture Alternatives

Controlled experiments on the web: survey and practical guide (Kohavi et al. 2009) provides a review of many architecture alternatives. The main three components of an experimentation capability involve the randomization algorithm, the assignment method (i.e., how the randomly assigned experimental units are given the variants), and the data path (which captures raw observation data and processes it). Tang et al. (2010) give a detailed view of the infrastructure for experiments as carried out by Google.

To validate an experimentation system, we recommend that A/A tests be run regularly to test that the experimental setup and randomization mechanism is working properly. An A/A test, sometimes called a null test (Peterson 2004), exercises the experimentation system, assigning users to one of two groups, but exposes them to exactly the same experience. An A/A test can be used to (i) collect data and assess its variability for power calculations and (ii) test the experimentation system (the null hypothesis should be rejected about 5 % of the time when

a 95 % confidence level is used) (Kohavi et al. 2009; Martin 2008).

Planning Experiments

Several aspects of planning an experiment are important: estimating adequate sample size, gathering the right metrics, tracking the right users, and randomization unit.

Sample size. Sample size is determined by the percent of users admitted into the experiment variants (control and treatments) and how long the experiment runs. As an experiment runs longer, more visitors are admitted into the variants, so sample sizes increase. Experimenters can choose the relative percent of visitors that are in the control and treatment which affects how long you will need to run the experiment. Several authors (Deng et al. 2013; Kohavi et al. 2009) have addressed the issue of sample size and length of experiment in order to achieve adequate statistical power for an experiment, where statistical power of an experiment is the probability of detecting a given effect when it exists (technically, the probability of correctly rejecting the null hypothesis when it is false). In addition to planning an experiment for adequate power, a best practice is to run the experiment for at least one week (to capture a full weekly cycle) and then multiple weeks beyond that. When "novelty" or "primacy" effects are suspected (i.e., the initial effect of the treatment is not the same as the long-term effect), the experiment should be run long enough to estimate the asymptotic effect of the treatment. Finally, measuring the effect on high-variance metric, such as loyalty (sessions/user), will generally require more users than for other metrics (Kohavi et al. 2012).

Observations, Metrics, and the OEC. Gathering observations (i.e., logging events) so that the right metrics can be computed is critical to successful experimentation. Whenever possible and economically feasible, one should gather as many observations as possible that relate to answering potential questions of interest, whether user related or performance related (e.g., latency, utilization, crashes). We recommend computing many metrics from the observations (e.g., hundreds) because they can give rise to surprising insights, although care must be taken to correctly understand and control for the false-positive rate (Kohavi et al. 2013; Hochberg and Benjamini 1995). While having many metrics is great for insights, decisions should be made using the Overall Evaluation Criterion (OEC). See Tenet 1 earlier for a description of the OEC.

Triggering. Some treatments may be relevant to all users who come to a website. However, for many experiments, the difference introduced is relevant for a subset of visitors (e.g., a change to the checkout process, which only 10 % of visitors start). In these cases, it is best to include only those visitors who would have experienced a difference in one of the variants (this commonly requires counterfactual triggering for the control). Some architectures (Kohavi et al. 2009) trigger users into an experiment either explicitly or using lazy (or late-bound) assignment. In either case, the key is to analyze only the subset of the population that was potentially impacted. Triggering reduces the variability in the estimate of treatment effect, leading to more precise estimates. Because the diluted effect is often of interest, the effect can then be diluted (Deng and Hu 2015).

Randomization Unit. Most experiments use the visitor as the randomization unit for several reasons. First, for many changes being tested, it is important to give the user a consistent online experience. Second, most experimenters evaluate metrics at the user level, such as sessions per user and clicks per user. Ideally, the randomization by the experimenter is by a true user, but in many unauthenticated sites, a cookie stored by the user's browser is used, so in effect, the randomization unit is the cookie. In this case, the same user will appear to be different users if she comes to the site using a different browser, different device, or having deleted her cookie during the experiment. The next section will discuss how the choice of randomization unit affects how the analysis of different metrics should be carried out. The randomization unit can also affect the power of the test for some metrics.

For example, Deng et al. (2011) showed that the variance of page level metrics can be greatly reduced if randomization is done at the page level, but user metrics cannot be computed in such cases. In social-network settings, spillover effects violate the standard no-interference assumption, requiring unique approaches, such as clustering (Ugander et al. 2013).

Analysis of Experiments

If an experiment is carried out correctly, the analysis should be a straightforward application of well-known statistical methods. Of course, this is much preferred than trying to recover from a poor experimental design or implementation.

Confidence Intervals. Most reporting systems will display the treatment effect (actual and percent change) along with suitable confidence intervals. For reasonably large sample sizes, generally considered to be thousands of users in each variant, the means may be considered to have normal distributions (see Kohavi et al. (2014) for detailed guidance), making the formation of confidence intervals routine. However, care must be taken to use the Fieller theorem formula (Fieller 1954) for percent effect since there is a random quantity in the denominator.

Decision-making. A common approach to deciding if the treatment is better than the control is the usual hypothesis-testing procedure, assuming the normal distribution if the sample size is sufficient (Kohavi et al. 2009). Alternatives to this when normality cannot be assumed are transformations of the data (Bickel and Doksum 1981) and nonparametric or resampling/permutation methods to determine how unusual the observed sample is under the null hypothesis (Good 2005). When conducting a test of whether the treatment had an effect or not (e.g., a test of whether the treatment and control means are equal), a p value of the statistical test is often produced as evidence. More precisely, the p value is the probability to obtain an effect equal to or more extreme than the one observed, presuming the null hypothesis of no effect is true (Biau et al. 2010).

Another alternative is to use Bayes' theorem to calculate the posterior odds that the treatment had a positive impact versus the odds it had no impact (Stone 2013).

Analysis Units. Metrics may be defined with different analysis units, such as user, session, or other appropriate bases. For example, an ecommerce site may be interested in metrics such as revenue per user, revenue per session, or revenue per purchaser. Straightforward statistical methods (e.g., the usual t-test and variants) apply to any metric that has user as its analysis unit if users are the unit of randomization since users may be considered independent. However, if the analysis unit is not the same as the randomization unit, the analysis units may not be considered independent, and other methods need to be used to calculate standard deviation or to compare treatment to control. Bootstrapping (Efron and Tibshirani 1993) and the delta method (Casella and Berger 2001) are two commonly used methods when the analysis unit is not the same as the randomization unit.

Variance Reduction. Increasing the sample size is one way to increase power. However, online researchers are continually looking for ways to increase the power of their experiments while shortening, or at least not extending, the length of the tests. One way to do this is to use covariates such as pre-experiment user metrics, user demographics, location, equipment, software, connection speed, etc. (Deng et al. 2013) gave an example where a 50 % reduction in variance for a metric could be achieved by using only the pre-experiment metric values for the users.

Diagnostics. In order to assure the experimental results are trustworthy, every experimentation system should have some diagnostic tools built in. Graphs of the number of users in each variant, metric means, and treatment effects over time will help the researcher see unexpected problems or upsets to the experiment. In addition, diagnostic tests that trigger an alarm when an expected condition is not met should be built in. One critical diagnostic test is the "sample

ratio mismatch" or SRM. A simple statistical test checks if the actual percentage for each variant is close enough to the planned percentages. We have found this one diagnostic is frequently the "canary in the coal mine" for online experiments. There are many possible ways an experiment can skew the number of visitors to one variant or another, and many of them will cause a large bias in the treatment effect. Another common useful test is that the performance, or latency, of the two versions is similar when expected to be so. It some cases, the treatment may be slower due to caching issues (e.g., cold start), or if the variant are unbalanced (e.g., 90/10 %), a shared resource like an LRU cache (Least Recently Used) will give an advantage to the larger variant (Kohavi and Longbotham 2010). When an experimentation platform allows overlapping experiments, a diagnostic to check for interactions between overlapping experiments is also helpful. Anytime an alarm or graph indicates a potential problem, the researcher should investigate to determine the source.

Robot Removal. Robots must be removed from any analysis of web data since their activity can severely bias experiment results; see Kohavi et al. (2009). Some robots may slip through robot-filtering techniques and should be considered when diagnostics suggest there may be a problem with the experiment.

Summary

The Internet and online connectivity of client software, websites, and online services provide a fertile ground for scientific testing methodology. Online experimentation is now recognized as a critical tool to determine whether a software or design change should be made. The benefit of experimenting online is the ability to set up a software platform for conducting the tests, which makes experimentation much more scalable and efficient and allows evaluating ideas quickly.

Recommended Reading

Biau DJ, Jolles BM, Porcher R (2010) P value and the theory of hypothesis testing. Clin Orthop Relat Res 468(3):885–892

Bickel PJ, Doksum KA (1981) An analysis of transformations revisited. J Am Stat Assoc 76(374):296–311. doi:10.1080/01621459.1981.10477649

Blank SG (2005) The four steps to the epiphany: successful strategies for products that win. Cafepress.com.

Box GEP, Hunter JS, Hunter WG (2005) Statistics for experimenters: design, innovation, and discovery. Wiley, Hoboken

Casella G, Berger RL (2001) Statistical inference, 2nd edn. Cengage Learning. http://www.amazon.com/Statistical-Inference-George-Casella

Deng A, Hu V (2015) Diluted treatment effect estimation for trigger analysis in online controlled experiments. In: WSDM, Shanghai 2015

Deng A, Xu Y, Kohavi R, Walker T (2013) Improving the sensitivity of online controlled experiments by utilizing pre-experiment data. In: WSDM, Rome 2013

Deng S, Longbotham R, Walker T, Xu Y (2011) Choice of randomization unit in online controlled experiment. In: Joint statistical meetings proceedings, Miami Beach, pp 4866–4877

Efron B, Tibshirani RJ (1993) An introduction to the bootstrap. Chapman & Hall, New York

Fieller EC (1954) Some problems in interval estimation. J R Stat Soc Ser B 16(2):175–185. doi:JSTOR2984043

Good PI (2005) Permutation, parametric and bootstrap tests of hypotheses, 3rd edn. Springer, New York

Goward C (2012) You should test that: conversion optimization for more leads, sales and profit or the art and science of optimized marketing. Sybex. http://www.amazon.com/You-Should-Test-That-Optimization/dp/1118301307

Hochberg Y Benjamini Y (1995) Controlling the false discovery rate: a practical and powerful approach to multiple testing Series B. J R Stat Soc 57(1):289–300

Kaushik A (2006) Experimentation and testing: a primer. Occam's razor. http://www.kaushik.net/avinash / 2006 / 05 / experimentation-and-testing-a-primer.html. Accessed 22 May 2008

Kohavi R, Deng A, Frasca B, Longbotham R, Walker T, Xu Y (2012) Trustworthy online controlled experiments: five puzzling outcomes explained. In: Proceedings of the 18th conference on knowledge discovery and data mining. http://bit.ly/expPuzzling

Kohavi R, Deng A, Frasca B, Walker T, Xu Y, Pohlmann N (2013) Online controlled experiments at large scale. In: Proceedings of the 19th ACM SIGKDD international conference on knowledge discovery and data mining (KDD 2013). http://bit.ly/ExPScale

Kohavi R, Deng A, Longbotham R, Xu Y (2014) Seven rules of thumb for web site. In: Proceedings of the 20th ACM SIGKDD international conference on knowledge discovery and data mining (KDD '14). http://bit.ly/expRulesOfThumb

Kohavi R, Longbotham R (2010) Unexpected results in online controlled experiments. In: SIGKDD Explorations. http://bit.ly/expUnexpected

Kohavi R, Longbotham R, Walker T (2010) Online experiments: practical lessons. IEEE Comput Sept:82–85. http://bit.ly/expPracticalLessons

Kohavi R, Longbotham R, Sommerfield D, Henne RM (2009) Controlled experiments on the web: survey and practical guide. Data Min Knowl Discov 18:140–181. http://bit.ly/expSurvey

Kohavi R, Crook T, Longbotham R (2009) Online experimentation at microsoft. In: Third workshop on data mining case studies and practice prize. http://bit.ly/expMicrosoft

Malinas G, Bigelow J (2009) Simpson's paradox. Stanford Encyclopedia of Philosophy. http://plato.stanford.edu/entries/paradox-simpson/

Manzi J (2012) Uncontrolled: the surprising payoff of trial-and-error for business, politics, and society. Basic Books. https://www.amazon.com/Uncontrolled-Surprising-Trial-Error-Business-ebook/dp/B007V2VEQO

Martin RC (2008) Clean code: a handbook of Agile software craftsmanship. Prentice Hall, Upper Saddle River

McFarland C (2012a) Experiment!: website conversion rate optimization with A/B and multivariate. New Riders. http://www.amazon.com/Experiment-Website-conversion-optimization-multivariate/dp/0321834607

McFarland C (2012b) Experiment!: website conversion rate optimization with A/B and multivariate testing. New Riders. http://www.amazon.com/Experiment-Website-conversion-optimization-multivariate/dp/0321834607

McKinley D (2013) Testing to cull the living flower. http://mcfunley.com/testing-to-cull-the-living-flower

Moran M (2007) Do it wrong quickly: how the web changes the old marketing rules. IBM Press. http://www.amazon.com/Do-Wrong-Quickly-Changes-Marketing/dp/0132255960/

Moran M (2008) Multivariate testing in action: Quicken Loan's Regis Hadiaris on multivariate testing. www.biznology.com/2008/12/multivariate_testing_in_action/

Peterson ET (2004) Web analytics demystified: a marketer's guide to understanding how your web site affects your business. Celilo Group Media and CafePress. http://www.amazon.com/Web-Analytics-Demystified-Marketers-Understanding/dp/0974358428/

Ries E (2011) The lean startup: how today's entrepreneurs use continuous innovation to create radically successful businesses. Crown Business, New York

Rubin KS (2012) Essential scrum: a practical guide to the most popular Agile process. Addison-Wesley Professional, Upper Saddle River

Schrage M (2014) The innovator's hypothesis: how cheap experiments are worth more than good ideas. MIT Press. http://www.amazon.com/Innovators-Hypothesis-Cheap-Experiments-Worth/dp/0262528967

Siroker D, Koomen P (2013) A/B testing: the most powerful way to turn clicks into customers. Wiley. http://www.amazon.com/Testing-Most-Powerful-Clicks-Customers/dp/1118792416

Stone JV (2013) Bayes' rule: a tutorial introduction to Bayesian analysis. Sebtel Press. http://www.amazon.com/Bayes-Rule-Tutorial-Introduction-Bayesian/dp/0956372848

Tang D, Agarwal A, O'Brien D, Meyer M (2010) Overlapping experiment infrastructure: more, better, faster experimentation. In: KDD 2010: The 16th ACM SIGKDD international conference on knowledge discovery and data mining, Washington, DC, 25–28 July

Ugander J, Karrer B, Backstrom L, Kleinberg J (2013) Graph cluster randomization: network exposure to multiple universes. In: Proceedings of the 19th ACM SIGKDD international conference on knowledge discovery and data mining (KDD '13), Chicago

Online Learning

Peter Auer
Department of Information Technology, University of Leoben, Leoben, Austria

Abstract

Online learning and its variants are one of the main models of computational learning theory, complementing statistical PAC learning and related models. An online learner needs to make predictions about a sequence of instances, one after the other, and receives feedback after each prediction. The performance of the online learner is typically compared to the best predictor from a given class, often in terms of its excess loss (the regret) over the best predictor. Some of the fundamental online learning algorithms and their

variants are discussed: weighted majority, follow the perturbed leader, follow the regularized leader, the perceptron algorithm, the doubling trick, bandit algorithms, and the issue of adaptive versus oblivious instance sequences. A typical performance proof of an online learning algorithm is exemplified for the perceptron algorithm.

Synonyms

Mistake-bounded learning; Prediction with expert advice; Sequential learning

Definition

In the online learning model, the learner needs to make predictions or choices about a sequence of instances, one after the other, and receives a loss or reward after each prediction or choice. Typically, the learner receives a description of the current instance before making a prediction. The goal of the learner is to minimize its accumulated losses (or equivalently maximize the accumulated rewards).

The performance of the online learner is usually compared to the best predictor in hindsight from a given class of predictors. This comparison with a predictor in hindsight allows for meaningful performance bounds even without any assumptions on how the sequence of instances is generated. In particular, this sequence of instances may not be generated by a random process but by an adversary that tries to prevent learning.

In this sense performance bounds for online learning are typically worst-case bounds that hold for any sequence of instances. This is possible since the performance bounds are relative to the best predictor from a given class. Often these performance guarantees are quite strong, showing that the learner can do nearly as well as the best predictor from a large class of predictors.

Motivation and Background

Online learning is one of the main models of learning theory, complementing the statistical approach of the PAC learning model by allowing a more general process for generating learning instances. The distinctive properties of the online learning model are:

- Learning proceeds in trials,
- There is no designated learning phase, the performance of the learner is evaluated continuously from the start,
- No assumptions on the generation of the inputs to the learner are necessary; they may depend even adversarially on previous predictions of the learner,
- Sequential predictions model an interaction between the learner and its environment,
- Performance guarantees for learning algorithms are typically relative to the performance of the best predictor in hindsight from some given class.

The first explicit models of online learning were proposed by Angluin (1988) and Littlestone (1988), but related work on repeated games by Hannan (1957) dates back to 1957. Littlestone proposed online learning as a sequence of trials, where in each the learner receives some input, makes a prediction of the associated output, and receives the correct output. It was assumed that some function from a known class maps the inputs to correct outputs. The performance of the learner is measured by the number of mistakes made by a learner, before it converges to the correct predictor. Angluin's equivalence query model of learning is formulated differently but is essentially equivalent to Littlestone's model.

The restriction that some function from the class must predict all outputs correctly was then removed, e.g., Vovk (1990) and Littlestone and Warmuth (1994). In their setting the learner competes with the best predictor from the given class. As the class of predictors can be seen as a set of experts advising the learner about the correct predictions, this led to the term "prediction with

expert advice." A comprehensive treatment of binary predictions with expert advice can be found in Cesa-Bianchi et al. (1997). Relations of online learning to several other fields (e.g., compression, competitive analysis, game theory, and portfolio selection) are discussed in the excellent book on sequential prediction by Cesa-Bianchi and Lugosi (2006).

Structure of Learning System

The online learning model is formalized as follows. In each trial $t = 1, 2, \ldots$, the learner

1. Receives input $x_t \in X$,
2. Chooses a prediction or output $y_t \in Y$,
3. Receives response $z_t \in Z$,
4. Incurs loss $\ell_t = \ell(y_t, z_t)$,

where $\ell : Y \times Z \mapsto \mathbb{R}$ is some loss function. The performance of a learner up to trial T is measured by its accumulated loss $L_T = \sum_{t=1}^{T} \ell_t$. For now it is assumed that inputs x_t and responses z_t are independent from the learner's predictions y_t. Such sequences of instances are called *oblivious* to the learner. *Adaptive* sequences of instances will be discussed later.

Performance bounds for online learning algorithms are typically in respect to the performance of an optimal predictor (or expert) E^* in hindsight from some class $\mathcal{E}$, $E^* \in \mathcal{E}$. A predictor E maps the past given by $(x_1, y_1, z_1), \ldots, (x_{t-1}, y_{t-1}, z_{t-1})$ and the current input x_t to a prediction y_t^E. As for the learner, the performance of a predictor is measured by its accumulated loss $L_T^E = \sum_{t=1}^{T} \ell_t^E$, where $\ell_t^E = \ell(y_t^E, z_t)$. Most bounds for the loss of online algorithms are of the form

$$L_T \le a \min_{E \in \mathcal{E}} L_T^E + b\mathcal{C}(\mathcal{E}),$$

where the constants a and b depend on the loss function and $\mathcal{C}(\mathcal{E})$ measures the complexity of the class of predictors (e.g., the complexity $\mathcal{C}(\mathcal{E})$ could be $\log|\mathcal{E}|$ for a finite class $\mathcal{E}$.) Often it is possible to trade the constant a against the constant b such that bounds

$$L_T \le L_T^* + o(L_T^*)$$

can be achieved, where $L_T^* = \min_{E \in \mathcal{E}} L_T^E$ is the loss of the best predictor in hindsight up to time T. These bounds are of particular interest as they show that the loss of the learning algorithm is only little larger than the loss of the best predictor. For such bounds the regret R_T of the learning algorithm,

$$R_T = L_T - L_T^*,$$

is the relevant quantity that measures the cost of not knowing the best predictor in advance.

The next section makes this general definition of online learning more concrete by presenting some important online learning algorithms.

Theory/Solution

The Weighted Majority Algorithm

The weighted majority algorithm developed by Littlestone and Warmuth (1994) is one of the fundamental online learning algorithms, with many relatives using similar ideas. We will present it for the basic scenario with a finite set of experts $\mathcal{E}$, binary predictions $y_t \in \{0, 1\}$, binary responses $z_t \in \{0, 1\}$, and the discrete loss which just counts mistakes, $\ell(y, z) = |y - z|$, such that $\ell(y, z) = 0$ if $y = z$ and $\ell(y, z) = 1$ if $y \ne z$. (We will use the terms experts and predictors interchangeably. In the literature finite sets of predictors are often called experts.)

The weighted majority algorithm maintains a weight w_t^E for each expert $E \in \mathcal{E}$ that is initialized as $w_1^E = 1$. The weights are used to combine the predictions y_t^E of the experts by a weighted majority vote: $y_t = 1$ if $\sum_E w_t^E y_t^E \ge \frac{1}{2} \sum_E w_t^E$, and $y_t = 0$ otherwise. After receiving the response z_t, the weights of experts that made incorrect predictions are reduced by multiplying with some constant $\beta < 1$, $w_{t+1}^E = \beta w_t^E$ if $y_t^E \ne z_t$, and $w_{t+1}^E = w_t^E$ if $y_t^E = z_t$. As a performance bound for the weighted majority algorithm one can achieve

$$L_T \le 2L_T^* + 2\sqrt{2L_T^* \log|\mathcal{E}|} + 4\log|\mathcal{E}|$$

with $L_T^* = \min_{E \in \mathcal{E}} L_T^E$ and an appropriate β. (Better constants on the square root and the logarithmic term are possible.)

While in this bound the loss of the deterministic weighted majority algorithm is twice the loss of the best expert, the randomized version of the weighted majority algorithm almost achieves the loss of the best expert. Instead of using a deterministic prediction, the randomized weighted majority algorithm tosses a coin and predicts $y_t = 1$ with probability $\sum_E w_t^E y_t^E / \sum_E w_t^E$. Below we prove the following bound on the expected loss of the randomized algorithm:

$$\mathbb{E}[L_T] \le \frac{\log(1/\beta)}{1-\beta} L_T^* + \frac{1}{1-\beta} \log|\mathcal{E}|. \quad (1)$$

Approximately optimizing for β yields $\beta = 1 - \varepsilon$, where $\varepsilon = \min\{1/2, \sqrt{2(\log|\mathcal{E}|)/L_T^*}\}$, and

$$\mathbb{E}[L_T] \le L_T^* + \sqrt{2L_T^* \log|\mathcal{E}|} + 2\log|\mathcal{E}|. \quad (2)$$

The expectation in these bounds is only in respect to the randomization of the algorithm, no probabilistic assumptions on the experts or the sequence of responses are made. These bounds hold for any set of experts and any oblivious sequence of responses that does not depend on the randomization of the algorithm. It can be even shown that the following similar bound holds with probability $1 - \delta$ (in respect to the randomization of the algorithm):

$$L_T \le L_T^* + \sqrt{T \log(|\mathcal{E}|/\delta)}. \quad (3)$$

The proof of bound (1) shows many of the ideas used in the proofs for online learning algorithms. Key ingredients are a potential function and how the changes of the potential function relate to losses incurred by the learning algorithm. For the weighted majority algorithm, a suitable potential function is the sum of the weights, $W_t = \sum_E w_t^E$. Then, since the losses are 0 or 1,

$$\begin{aligned} \frac{W_{t+1}}{W_t} &= \frac{\sum_E w_{t+1}^E}{\sum_E w_t^E} = \frac{\sum_E \beta^{\ell_t^E} w_t^E}{\sum_E w_t^E} \\ &= \frac{\sum_E [1 - (1-\beta)\ell_t^E] w_t^E}{\sum_E w_t^E} \\ &= 1 - (1-\beta)\frac{\sum_E \ell_t^E w_t^E}{\sum_E w_t^E}. \end{aligned} \quad (4)$$

Since the probability that the randomized weighted majority algorithm makes a mistake is given by $\mathbb{E}[\ell_t] = \sum_E \ell_t^E w_t^E / \sum_E w_t^E$, we get by taking logarithms that

$$\begin{aligned} \log W_{t+1} - \log W_t &= \log(1 - (1-\beta)\mathbb{E}[\ell_t]) \\ &\le -(1-\beta)\mathbb{E}[\ell_t] \end{aligned} \quad (5)$$

(since $\log(1-x) \le -x$ for $x \in (0,1)$). Summing over all trials $t = 1, \ldots, T$, we find

$$\log W_{T+1} - \log W_1 \le -(1-\beta)\mathbb{E}[L_t].$$

Since $W_1 = |\mathcal{E}|$ and $W_{T+1} = \sum_E w_{T+1}^E = \sum_E \beta^{L_T^E} \ge \beta^{L_T^*}$, rearranging the terms gives (1).

Extensions and Modifications of the Weighted Majority Algorithm

Variants and improved versions of the weighed majority algorithm have been analyzed for various learning scenarios. An excellent coverage of the material can be found in Cesa-Bianchi and Lugosi (2006). In this section we mention a few of them.

General loss functions. The analysis of the weighted majority algorithm can be generalized to any convex set of predictions Y and any set of outcomes Z, as long as the loss function $\ell(y, z)$ is bounded and convex in the first argument. Typically it is possibly to derive a learning algorithm with loss bound

$$L_T \le aL_T^* + b\log|\mathcal{E}|$$

for suitable values a and b. Of particular interest is the smallest b for which a loss bound with $a =$

1 can be achieved. Some algorithms for convex prediction sets Y will be discussed later.

Tracking the best expert and other structured experts. For a large number of experts, the loss bound of the weighted majority algorithm is still interesting since it scales only logarithmically with the number of experts. Nevertheless, the weighted majority algorithm and other online learning algorithms become computationally demanding as they need to keep track of the performance of all experts (computation time scales linearly with the number of experts). If the experts exhibit a suitable structure, then this computational burden can be avoided.

As an example we consider the problem of tracking the best expert. Let $\mathcal{E}_0$ be a small set of base experts. The learning algorithm is required to compete with the best sequence of at most S experts from $\mathcal{E}_0$: the trials are divided into S periods, and in each period another expert might predict optimally. Thus the minimal loss of a sequence of S experts is given by

$$L^*_{T,S} = \min_{0=T_0\le T_1\le T_2\le\cdots\le T_S=T} \sum_{i=1}^{S} \min_{E\in\mathcal{E}_0} \sum_{t=T_{i-1}+1}^{T_i} \ell_t^E,$$

where the trials are optimally divided into S periods $[T_{i-1}+1, T_i]$, and the best base expert is chosen for each period. Such sequences of base experts can be seen as experts themselves, but the number of such compound experts is $\binom{T-1}{S-1}|\mathcal{E}_0|^S$ and thus computationally prohibitive. Fortunately, a slightly modified weighted majority algorithm applied to the base experts achieves almost the same performance as the weighted majority algorithm applied to the compound experts. The modification of the weighted majority algorithm just lower bounds the relative weight of each base expert. This allows the relative weight of a base expert to grow large quickly if this expert predicts best in the current period. Hence, also the learning algorithm will predict almost optimally in each period. A recent and improved version of the weighted majority algorithm with many related references is given in Luo and Schapire (2015).

Other examples of structured experts include tree experts and shortest path problems (see Cesa-Bianchi and Lugosi (2006) for further references). For the shortest path problem in a graph, the learner has to compete with the single best path in hindsight, while edge costs may change at each time t. In principle the weighted majority algorithm could be employed with one expert for each path, but since the number of paths is usually exponential in the size of the graph, this might be computationally infeasible. Instead, the *follow the perturbed leader* strategy can be used as an alternative to the weighted majority algorithm.

Follow the perturbed leader. This is a simple prediction strategy that was originally proposed by Hannan (1957). In each trial t, it generates identically distributed random values ψ_t^E for every expert E, adds these random values to the losses of the experts so far, and predicts with the expert that achieves the minimum sum,

$$\hat{E}_t = \arg\min_{E\in\mathcal{E}} L_{t-1}^E + \psi_t^E,$$
$$y_t = y_t^{\hat{E}_t}.$$

For carefully chosen distributions of the ψ_t^E, this prediction strategy achieves loss bounds similar to the weighted majority like algorithms.

To apply this strategy to the shortest path problem described above, it is assumed that all paths have an equal number of edges (by possibly adding dummy edges). Then instead of generating random values ψ_t^E for each path E, a random value for each edge is generated, and the value ψ_t^E for a path is given by the sum of the random values for its edges. This allows to find the best path $\hat{E}_t$ efficiently by a shortest path calculation according to the accumulated and randomly modified edge costs.

The doubling trick. The optimal choice of β in the performance bound (1) requires knowledge about the loss of the best expert L_T^*. If such knowledge is not available, the doubling trick can be used. The idea is to start with an initial

O

guess $\hat{L}^*$ and choose β according to this guess. When the loss of the best expert exceeds this guess, the guess is doubled, β is modified, and the learning algorithm is restarted. The bound (2) increases only slightly when L_T^* is not known and the doubling trick is used. It can be shown that still

$$\mathbb{E}[L_T] \leq L_T^* + c_1\sqrt{L_T^* \log|\mathcal{E}|} + c_2 \log|\mathcal{E}|$$

for suitable constants c_1 and c_2. A thorough analysis of the doubling trick can be found in Cesa-Bianchi et al. (1997). Variations of the doubling trick can be used for many online learning algorithms to "guess" unknown quantities. A drawback of the doubling trick is that it restarts the learning algorithm and forgets about all previous trials. An alternative approach is an iterative adaptation of the parameter β, which can be shown to give better bounds than the doubling trick. The advantage of the doubling trick is that its analysis is quite simple.

Prediction with limited feedback and the multiarmed bandit problem. In the setting considered so far, the learner has full information of the past, as all past outcomes $z_1, \ldots, z_{t-1} \in \{0, 1\}$ and all predictions of the experts $y_1^E, \ldots, y_t^E$, $E \in \mathcal{E}$, are available, before prediction y_t is made. In some learning scenarios, the learner might not have such full information. One example is the multiarmed bandit problem, and a more general case is *prediction with partial monitoring*.

In the multiarmed bandit problem the learner chooses to follow one of K experts, observes the loss of this expert, and also incurs the loss of this expert. Formally, the learner chooses an expert $y_t = E_t \in \mathcal{E} = \{1, \ldots, K\}$, receives the loss of the chosen prediction $z_t = \ell_t(E_t)$, and incurs loss $\ell(y_t, z_t) = z_t = \ell_t(E_t)$. (Here $\ell_t(E)$ denotes the loss of expert E at time t.) The losses of the other experts, $\ell_t(E)$, $E \neq E_t$, are not revealed to the learner. The goal of the learner is to compete with the loss of the single best expert, $L_T^* = \min_{E \in \mathcal{E}} L_T^E$, $L_T^E = \sum_{t=1}^T \ell_t(E)$. The multiarmed bandit problem looks very much like the original online learning problem with the predictions $y \in Y$ as experts.

Since at each time t the learner observes only the loss of the chosen expert, it needs to estimate the unseen losses of the other experts and use these estimates when choosing an expert. Since accurate estimates need a sufficient amount of data, this leads to a trade-off between choosing the (apparently) best expert to minimize losses and choosing a different expert for which more data are needed. This exploration-exploitation trade-off also appears elsewhere in online learning, but it is most clearly displayed in the bandit problem. An algorithm that deals well with this trade-off is again a simple variant of the weighted majority algorithm. This algorithm does exploration trials with some small probability, and in such exploration trials, it chooses an expert uniformly at random. This algorithm has been analyzed in Auer et al. (2002) for gains instead of losses. For losses $\ell \in [-1, 0]$ the accumulated loss of the algorithm can be bounded as

$$\mathbb{E}[L_T] \leq L_T^* + 3\sqrt{K|L_T^*| \log K}.$$

Compared with (2), the regret increases only by a factor of $\sqrt{K}$. Further results, including lower bounds and results for stochastic bandit problems, are summarized in Bubeck and Cesa-Bianchi (2012). For the stochastic multiarmed bandit problem, it is assumed that the losses of the experts are generated independently at random by some distribution for each expert. This allows for specialized algorithms with substantially improved regret bounds.

A generalization of bandit problems are partial monitoring games (Bartók 2014), where the learner receives only indirect feedback about the losses of the experts. Depending on how much the feedback reveals about the incurred losses, partial monitoring games can be classified as games with either 0, $\Theta(T^{1/2})$, $\Theta(T^{2/3})$, or $\Theta(T)$ regret.

The Perceptron Algorithm

In this section we consider an example for an online learning algorithm that competes with a *continuous* set of experts, in contrast to the *finite*

sets of experts we have considered so far. This algorithm—the perceptron algorithm (Rosenblatt 1958)—was among the first online learning algorithms developed. Another of this early online learning algorithms with a continuous set of experts is the Winnow algorithm by Littlestone (1988). A unified analysis of these algorithms can be found in Cesa-Bianchi and Lugosi (2006). This analysis covers a large class of algorithms, in particular the p-norm perceptrons, which smoothly interpolate between the perceptron algorithm and Winnow.

The perceptron algorithm aims at learning a linear classification function. Thus inputs are from a Euclidean space, $X = \mathbb{R}^d$, the predictions and responses are binary, $Y = Z = \{0, 1\}$, and the discrete misclassification loss is used. Each expert is a linear classifier, represented by its weight vector $v \in \mathbb{R}^d$, whose linear classification is given by $\Phi_v : X \to \{0, 1\}$, $\Phi_v(x) = 1$ if $v \cdot x \geq 0$ and $\Phi_{v,\theta}(x) = 0$ if $v \cdot x < 0$.

The perceptron algorithm maintains a weight vector $w_t \in \mathbb{R}^d$ that is initialized as $w_1 = (0, \ldots, 0)$. After receiving input x_t, the perceptron's prediction is calculated using this weight,

$$y_t = \Phi_{w_t}(x_t),$$

and the weight vector is updated,

$$w_{t+1} = w_t + \eta(z_t - y_t)x_t,$$

where $\eta > 0$ is a learning rate parameter. Thus, if the prediction is correct, $y_t = z_t$, then the weights are not changed. Otherwise, the product $w_{t+1} \cdot x_t$ is moved into the correct direction: since $w_{t+1} \cdot x_t = w_t \cdot x_t + \eta(z_t - y_t)||x_t||^2$, $w_{t+1} \cdot x_t > w_t \cdot x_t$ if $y_t = 0$ but $z_t = 1$, and $w_{t+1} \cdot x_t < w_t \cdot x_t$ if $y_t = 1$ but $z_t = 0$.

We may assume that the inputs are normalized, $||x_t|| = 1$, otherwise a normalized x_t can be used in the update of the weight vector. Furthermore, we note that the learning rate η is irrelevant for the performance of the perceptron algorithm, since it scales only the size of the weights but does not change the predictions. Nevertheless, we keep the learning rate since it will simplify the analysis.

Analysis of the perceptron algorithm. To compare the perceptron algorithm with a fixed (and optimal) linear classifier v, we again use a potential function, $||w_t - v||^2$. For the change of the potential function when $y_t \neq z_t$, we find

$$\begin{aligned}
&||w_{t+1} - v||^2 - ||w_t - v||^2 \\
&= ||w_t + \eta(z_t - y_t)x_t - v||^2 - ||w_t - v||^2 \\
&= ||w_t - v||^2 + 2\eta(z_t - y_t)(w_t - v) \cdot x_t \\
&+ \eta^2(z_t - y_t)^2 \, ||x_t||^2 - ||w_t - v||^2 \\
&= 2\eta(z_t - y_t)(w_t \cdot x_t - v \cdot x_t) + \eta^2.
\end{aligned}$$

Since $w_t \cdot x_t < 0$ if $y_t = 0$ and $w_t \cdot x_t \geq 0$ if $y_t = 1$, we get $(z_t - y_t)(w_t \cdot x_t) \leq 0$ and

$$||w_{t+1} - v||^2 - ||w_t - v||^2 \leq -2\eta(z_t - y_t)(v \cdot x_t) + \eta^2.$$

Analogously, the linear classifier v makes a mistake in trial t if $(z_t - y_t)(v \cdot x_t) < 0$, and in this case $-(z_t - y_t)(v \cdot x_t) \leq ||v||$. Hence, summing over all trials (where $y_t \neq z_t$) gives

$$\begin{aligned}
&||w_{T+1} - v||^2 - ||w_1 - v||^2 \leq -2\eta \\
&\sum_{t:\ell_t=1,\ell_t^v=0} |v \cdot x_t| + 2\eta||v||L_T^v + \eta^2 L_T, \quad (6)
\end{aligned}$$

where the sum is over all trials where the perceptron algorithm makes a mistake but the linear classifier v makes no mistake. To proceed, we assume that for the correct classifications of the linear classifier v, the product $v \cdot x_t$ is bounded away from 0 (which describes the decision boundary). We assume $|v \cdot x_t| \geq \gamma_v > 0$. Then

$$\begin{aligned}
||w_{T+1} - v||^2 - ||w_1 - v||^2 &\leq -2\eta\gamma_v(L_T - L_T^v) \\
&+ 2\eta||v||L_T^v + \eta^2 L_T, \quad (7)
\end{aligned}$$

and

$$L_T(2\eta\gamma_v - \eta^2) \leq ||v||^2 + L_T^v(2\eta\gamma_v + 2\eta||v||),$$

since $||w_{T+1} - v||^2 \geq 0$ and $w_1 = (0, \ldots, 0)$. For $\eta = \gamma_v$ the following loss bound for the perceptron algorithm is achieved:

$$L_T \leq ||v||^2/\gamma_v^2 + 2L_T^v(1 + ||v||/\gamma_v).$$

Thus the loss of the perceptron algorithm does not only depend on the loss of an (optimal) linear classifier v but also on the gap by which the classifier can separate the inputs with $z_t = 0$ from the inputs with $z_t = 1$. The size of this gap is essentially given by $\gamma_v/||v||$.

Relation between the perceptron algorithm and support vector machines. The gap $\gamma_v/||v||$ is the quantity maximized by support vector machines, and it is the main factor determining the prediction accuracy (in a probabilistic sense) of a support vector machine. It is not coincidental that the same quantity appears in the performance bound of the perceptron algorithm, since it measures the difficulty of the classification problem.

As for support vector machines, kernels $K(\cdot, \cdot)$ can be used in the perceptron algorithm. For that, the dot product $w_t \cdot x_t$ is replaced by the kernel representation $\sum_{\tau=1}^{t-1}(z_\tau - y_\tau)K(x_\tau, x)$. Obviously this has the disadvantage that all previous inputs for which mistakes were made must be kept available.

Online Convex Optimization

For online convex optimization, the learner has to choose a prediction y_t from some convex set Y, receives as feedback a convex loss function $L_t : Y \to \mathbb{R}$, and suffers loss $\ell_t = L_t(y_t)$. An excellent exposition of online convex optimization is given in Shalev-Shwartz (2011).

Many online learning problems and algorithms can be cast in the framework of online convex optimization, in particular also the weighted majority algorithm and the perceptron algorithm. While both algorithms make binary predictions $y_t \in \{0, 1\}$ and suffer a discrete loss, they both can be *convexified*: For the weighted majority algorithm we can consider the probability p_t for $y_t = 1$ as the prediction of the algorithm, such that the expected loss is $\mathbb{E}\ell(y_t, z_t) = |z_t - p_t|$ which is a convex function in p_t. For the perceptron algorithm, the discrete loss can be upper bounded by a convex *surrogate loss function*, and the loss analysis can be done in respect to these surrogate loss functions.

Two simple but often effective strategies for online convex optimization are *follow the leader* and *follow the regularized leader*. The *follow the leader* strategy chooses the prediction that would minimize the accumulated loss so far,

$$y_t = \arg\min_{y \in Y} \sum_{i=1}^{t-1} L_i(y).$$

For some online convex optimization problems, this gives very good regret bounds, in particular for online quadratic optimization, where $Y = \mathbb{R}^d$ and $L_t(y) = ||y - z_t||^2$ is the squared Euclidean distance to some $z_t \in \mathbb{R}^d$. It can be shown that in this case, the regret is bounded by

$$\min_y \sum_{t=1}^{T} ||y - z_t||^2 - \sum_{t=1}^{T} ||y_t - z_t||^2 \leq 4Z^2(1 + \log T), \tag{8}$$

where $Z = \max_{1 \leq t \leq T} ||z_t||$. This strategy fails, though, for other loss functions, for example, for online linear optimization with losses $L_t(y) = y \cdot z_t$, when the regret might be as large as $\Omega(T)$. This problem can be avoided by the *follow the regularized leader* strategy which chooses predictions

$$y_t = \arg\min_{y \in Y} \left[\sum_{i=1}^{t-1} L_i(y) + R(y) \right]$$

for some regularization function $R : Y \to \mathbb{R}$. For online linear optimization with quadratic regularization $R(y) = \frac{Z}{B}\sqrt{\frac{T}{2}}||y||^2$, *follow the regularized leader* achieves regret

$$\min_{y : ||y|| \leq B} \sum_{t=1}^{T} (y \cdot z_t) - \sum_{t=1}^{T} (y_t \cdot z_t) \leq ZB\sqrt{2T},$$

if all $||z_t|| \leq Z$. Online convex optimization is a very active field of research, and a good starting point is the exposition (Shalev-Shwartz 2011).

Oblivious Versus Adaptive Instance Sequences

So far we have assumed that the sequence of instances is not influenced by the predictions of the learner. If the sequence of instances is adaptive and depends on the predictions of the learner, additional care is necessary. In particular the definition of regret is subtle: since the instances depend on the predictions, the instances encountered by the learner may be very different from the instances encountered when following the prediction of a single expert. Therefore, it is in general not possible to bound the loss of a learner by the losses the experts would have incurred when making their predictions. This notion of regret is called *policy regret* (Dekel 2012), and it is easy to construct examples where any learning algorithm suffers $\Omega(T)$ loss while the predictions of the best expert suffer zero loss. To obtain nontrivial bounds on the policy regret, the adaptiveness of the instance sequence needs to be restricted, for example, by a bounded memory assumption: the instance at time t may depend only on the last m predictions of the learner.

In contrast, the loss of the learner can often be bounded by the loss of the best expert *on the sequence generated in response to the predictions of the learner.* The difference between the loss of the learner and the loss of the best expert on the same sequence of instances is called the *external regret*. As explained above, the notion of external regret is not fully satisfactory for adaptive sequences, but it allows to carry over many result for oblivious sequences to adaptive sequences. For an example, the high probability bound (3) for the weighted majority algorithm,

$$L_T \leq L_T^* + \sqrt{T \log(|\mathcal{E}|/\delta)},$$

holds also for any adaptive instance sequence that depends on the past predictions of the *learner*.

Recommended Reading

Angluin D (1988) Queries and concept learning. Mach Learn 2:319–342

Auer P, Cesa-Bianchi N, Freund Y, Schapire R (2002) The nonstochastic multiarmed bandit problem. SIAM J Comput 32:48–77

Bartók G, Foster D, Pál D, Rakhlin A, Szepesvári C (2014) Partial monitoring—classification, regret bounds, and algorithms. Math Oper Res 39: 967–997

Bubeck S, Cesa-Bianchi N (2012) Regret analysis of stochastic and nonstochastic multi-armed bandit problems. Found Trends Mach Learn 5:1–122

Cesa-Bianchi N, Freund Y, Haussler D, Helmbold D, Schapire R, Warmuth M (1997) How to use expert advice. JACM 44:427–485

Cesa-Bianchi N, Lugosi G (2006) Prediction, learning, and games. Cambridge University Press, Cambridge/New York

Dekel O, Tewari A, Arora R (2012) Online bandit learning against an adaptive adversary: from regret to policy regret. In: Proceedings of the 29th international conference on machine learning, Edinburgh

Hannan J (1957) Approximation to Bayes risk in repeated play. Contrib Theory Games 3:97–139

Littlestone N (1988) Learning quickly when irrelevant attributes abound: a new linear-threshold algorithm. Mach Learn 2:285–318

Littlestone N, Warmuth M (1994) The weighted majority algorithm. Inf Comput 108:212–261

Luo H, Schapire RE (2015) Achieving all with no parameters: Adanormalhedge. In: Proceedings of the 28th conference on learning theory, Paris, pp 1286–1304

Rosenblatt F (1958) The perceptron: a probabilistic model for information storage and organization in the brain. Psychol Rev 65:386–408

Shalev-Shwartz S (2011) Online learning and online convex optimization. Found Trends Mach Learn 4: 107–194

Vovk V (1990) Aggregating strategies. In: Proceedings of 3rd annual workshop on computational learning theory, Rochester. Morgan Kaufmann, pp 371–386

Ontology Learning

Different approaches have been used for building ontologies, most of them to date mainly using manual methods (▸ Text Mining for the Semantic Web). An approach to building ontologies was set up in the CYC project, where the main step involved manual extraction of common sense knowledge from different sources. Ontology construction methodologies usually involve several phases including *identifying the purpose of the ontology* (why to build it, how will it be used, the range of the users), *building the ontology*, *evalu-*

ation and documentation. Ontology learning relates to the phase of building the ontology using semiautomatic methods based on text mining or machine learning.

Opinion Extraction

▸ Sentiment Analysis and Opinion Mining

Opinion Mining

▸ Sentiment Analysis and Opinion Mining

Opinion Stream Mining

Myra Spiliopoulou[1], Eirini Ntoutsi[2,3], and Max Zimmermann[4]
[1]Otto-von-Guericke University-Magdeburg, Magdeburg, Germany
[2]Leibniz Universität Hannover, Hannover, Germany
[3]Ludwig Maximilians Universität München, Munich, Germany
[4]Swedish Institute of Computer Science (SICS Swedish ICT), Kista, Sweden

Abstract

Opinion stream mining aims at learning and adaptation of a polarity model over a stream of opinionated documents, i.e., documents associated with a polarity. They comprise a valuable tool to analyze the huge amounts of opinions generated nowadays through the social media and the Web. In this chapter, we overview methods for polarity learning in a stream environment focusing especially on how these methods deal with the challenges imposed by the stream nature of the data, namely the nonstationary data distribution and the single pass constraint.

Synonyms

Mining a Stream of Opinionated Documents; Polarity Learning on a Stream

Definition

Opinion stream mining is a variant of stream mining, of text mining and of opinion mining. Its goal is learning and adaptation of a polarity model over a stream of opinionated documents. An "opinionated document" is a text associated with a "polarity." Polarity is a value that represents the "strength" and the "direction" of an opinion. The strength can be a categorical value (e.g., $+$, $-$) or a ranking value (e.g., zero to five stars) or a continuous value (e.g., in the interval $[0, 1]$). The direction refers to whether the opinion is positive, negative, or neutral. Strength and direction are often mixed. For example, in a ranking using stars, five stars may stand for a very positive opinion, zero stars for a very negative one, and three stars for a neutral one.

As a variant of stream mining, opinion stream mining is subject to challenges of learning on a stream: adapting to changes in the data generating distribution – a phenomenon often called *concept drift* and processing the data as they arrive (in a single pass), since they cannot be retained permanently.

As a variant of text mining, opinion stream mining is subject to challenges of learning from texts: identifying the parts of speech that are in the text (e.g., verbs, adjectives, etc.); bringing the individual words into stem form (e.g., "opinions"→"opinion"); deciding which words will constitute the feature space and which are not informative and should be ignored; modeling the similarity between texts, taking (among other issues) differences in the length of texts into account; extracting the "entities" from

Work partially done while with the Ludwig-Maximilians University, Munich.

the text (e.g., persons, products); and detecting the "topics" of discourse in the texts.

As a variant of opinion mining, opinion stream mining faces further challenges: distinguishing between words that bear sentiment (e.g., "nice," "ugly") and those referring to facts (e.g., "sauna," "phone") and discerning different forms of sentiment (e.g., anger, joy). For static data, these challenges are addressed with techniques of natural language processing (NLP), text mining, and **Sentiment Analysis and Opinion Mining** (cf. lemma).

The aforementioned challenges are exacerbated in the stream context. Opinion stream mining provides solutions for learning and adapting a polarity model in a volatile setting: the topics in the opinionated documents may change; the attitude of people toward an entity (e.g., person, product, event) may change; the words used by people to express polarity may change; and even the words used by people, i.e., the vocabulary, may also evolve over time.

Motivation, Main Tasks, and Challenges

With the rise of WEB 2.0, more and more people use social media to upload opinions on essentially every subject – on products, persons, institutions, events, and topics of discourse. These accumulating opinionated data are valuable sources of information that can deliver valuable insights on the popularity of events; on the properties of products that are deemed important; on the positive or negative perception people have toward a product, person, or institution; on their attitude toward a specific subject of discourse; etc.

Background: The analysis of opinionated data is investigated in the research areas of *sentiment analysis* and *opinion mining*. These two areas overlap, whereby research on sentiment analysis puts more emphasis in understanding different types of "sentiment" (e.g., irony, anger, sadness, etc.), while opinion mining focuses more on learning models and discerning trends from data that simply have positive or negative "polarity" (or are neutral). For an extensive discussion of the subject, the reader is referred to the lemma **Sentiment Analysis and Opinion Mining**.

In Liu (2012), Bing Liu defines four opinion mining tasks as follows:

1. *Entity extraction:* "Extract all entity expressions in a document, and categorize or group synonymous entity expressions into entity clusters. Each entity expression cluster indicates a unique entity e_i."
2. *Property extraction:* "Extract all property expressions of the entities, and categorize these property expressions into clusters. Each property expression cluster of entity e_i represents a unique property a_{ij}."
3. *Opinion holder extraction:* "Extract opinion holders for opinions from text or structured data and categorize them. The task is analogous to the above two tasks."
4. *Sentiment classification:* "Determine whether an opinion on a property a_{ij} is positive, negative, or neutral, or assign a numeric sentiment rating to the property."

Among these tasks, the first one is not peculiar to opinion mining: *entity extraction* (EEX) is a subtask of document analysis. A widespread special case of EEX is named-entity recognition (NER); a minister is an entity, and a specific minister is a named entity. The goal of EEX and NER is to identify and annotate all entities in a document. To this purpose, NLP techniques are used, as well as collections of "named entities"; a list of the towns in a country is an example of such a collection.

The second task can be generalized in two ways. First, the properties need not be associated to an explicitly defined entity (e.g., a person or city); they may also be topics or subtopics under a subject of discourse (e.g., air pollution as a subtopic of environment pollution). Further, clustering is not the only way of identifying properties/topics: aspect-based opinion mining is a subdomain of topic modeling (cf. lemma **Topic Models for NLP Applications** for the general domain). In this subdomain, a document is perceived as a mixture of topics and sentiments.

In opinion *stream* mining, the collection of opinionated documents is not perceived as a static set but as an ongoing stream. While the first and third of the aforementioned tasks remain largely unchanged, the second and forth task must be redefined in the stream context. The task of property extraction on the stream is addressed with methods of *dynamic topic modeling* (see Blei and Lafferty (2006) for the core concepts) and with methods of text stream clustering (Aggarwal and Yu 2006).

The task of sentiment classification becomes a stream classification problem for an evolving text stream. Hereafter, we denote this task as "learning a polarity model" or simpler "polarity model learning," without referring explicitly to the fact that the model is learned on a stream.

Challenges of opinion stream mining: The challenges faced in opinion stream mining for property extraction and polarity learning emanate from the different aspects of volatility in the opinionated stream:

(a) *The data evolve with respect to the target variable:* The attitude of people toward a subject of discourse, a person, a product, or some property of this product may change over time. This corresponds to a change in the priors of the polarity class.
(b) *The topics evolve:* New subjects of discourse emerge, some product properties become uninteresting while others gain momentum. The learning algorithm must recognize that people discuss different topics.
(c) *The vocabulary evolves:* New words show up, some words fall out of use, and the polarity of some words may change. This means that the high-dimensional feature space used by the learning algorithm changes during the process of learning and adaption.
(d) *Labels are scarce:* In conventional stream classification, it is assumed that fresh labels are timely available for classifier adaption. Opinionated streams are fast and the inspection of opinions is a tedious task. So, the demand for human intervention/supervision for document labeling must be limited.

Main tasks of opinion stream mining: In response to challenges (a) and (c), opinion stream mining encompasses solutions for polarity model learning and adaption and also when the class priors change and when the vocabulary evolves. Next to fully supervised solutions, there are also semi-supervised learning methods and active learning methods, in response to challenge (d). In the following, we elaborate on supervised, semi-supervised, and active stream mining approaches for the classification of opinionated streams.

For challenge (b), we refer the reader to literature on text stream clustering, starting, e.g., with Aggarwal and Yu (2006), and to literature on dynamic topic modeling, starting with Blei and Lafferty (2006) and Wang and McCallum (2006). Dynamic topic modeling for opinionated document streams gained momentum in the last years, resulting in several works on dynamic topic mixture models that capture both aspects (properties) and sentiment. An example is Fu et al. (2015) on dynamic nonparametric hierarchical Dirichlet process topic modeling. An important characteristic of this work is that the number of topics can be determined automatically and adjusted over time. Further, an aging (time-decay) component is incorporated into the learning process; this allows for forgetting old topics (Fu et al. 2015). As we discuss in the next section, the issue of forgetting is also essential in supervised learning over the stream, as means of adaptation to concept drift.

Polarity Learning in an Opinionated Stream

Polarity learning is a supervised task that involves model learning and model adaption over an opinionated stream, i.e., an infinite sequence D of arriving *instances* $d_1, \cdots, d_i, \cdots$. An instance/opinionated document is a vector over a *word vocabulary* V, which is built up and changes over time.

An instance has a polarity label c. We denote the class attribute by C. Much of the research on opinion stream mining considers streams where documents have positive or negative polarity and are mixed with neutral documents. We use this convention in the following, i.e., we assume that the polarity label is one of positive (+), negative (−), or neutral (Ø).

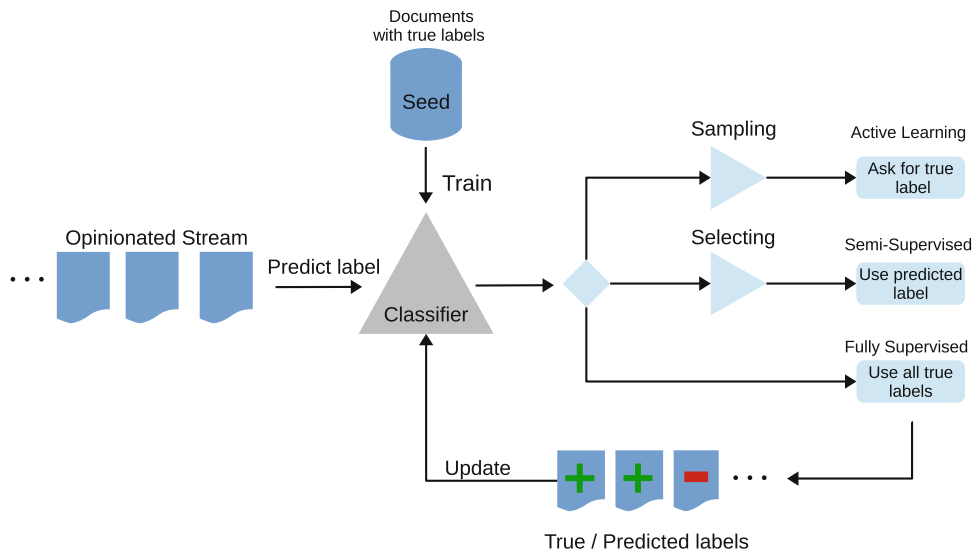
Workflow

The fully supervised stream learning scenario implies that the model is continuously learned on arriving, labeled instances. To deal with the label scarcity challenge, opinion stream mining research also contributes semi-supervised methods that learn with only an initial seed of labeled instances and active learning methods that request a label for only some of the arriving instances. An abstract workflow of the learning tasks is depicted in Fig. 1, distinguishing among supervised, semi-supervised, and active learning.

As can be seen in the figure, an initial classifier is trained on a starting set of manually labeled instances *Seed*. This set can be a small corpus of carefully selected opinionated documents that are representative of the stream, at least at the beginning, or the *Seed* can consist solely of the first arriving documents in the stream. Labels delivered by a human expert are denoted in the figure as "true labels," as opposed to the "predicted labels" that are assessed by the classifier.

In each subsequent step, the classifier predicts the labels of the arriving documents. For supervised learning, a human expert immediately delivers the true labels, which are then used for model adaption. In semi-supervised learning, the classifier adapts by using (a selection of) instances with predicted labels. In active learning, the expert is asked to deliver true labels only for some of the arriving documents which are then used for model adaption. These three ways of polarity learning are discussed hereafter.

The instances of the stream may be processed one by one as they arrive, or they may be stored into "chunks" (also called "blocks" or "batches"). In the first case, i.e., in "instance-based" processing, the classifier is adapted after seeing each new instance. In "chunk-based" processing, the classifier adapts after each chunk. A chunk may be a fixed-sized



Opinion Stream Mining, Fig. 1 Polarity learning on a stream of opinionated documents – fully supervised, semi-supervised, and active learning options

block of documents or it may be defined at different levels of temporal granularity, e.g., hourly, daily, or weekly. Instance-based processing allows for fast adaption; however, the processing cost is higher as the model is updated after each instance. Chunk-based processing is more appropriate for streams where changes in the topics and/or vocabulary are manifested gradually. A detailed discussion of instance- vs chunk-based methods can be found in the lemma **Stream Classification**.

Fully Supervised Opinion Stream Classification

Fully supervised polarity learning on an opinionated stream is performed in the same way as stream classification in a conventional stream. The reader is referred to the lemma **Stream Classification** for a detailed elaboration on the interaction between the classifier and the stream, the detection of drift, and the adaption of the model. For opinionated streams, two aspects are of particular interest: how to choose a classification algorithm for polarity learning and how to deal with changes in the vocabulary.

Stream classification algorithms for polarity learning. Since there are many stream classification algorithms, it is reasonable to investigate how appropriate they are for learning on an opinionated stream. Several comparative studies have emerged at the beginning of the decade, including Bifet and Frank (2010) and Gokulakrishnan et al. (2012). In Gokulakrishnan et al. (2012), Gokulakrishnan et al. study a Twitter stream (i.e., a stream of short texts) and evaluate multinomial Naive Bayes (MNB), support vector machines (SVM), Bayesian logistic regression, sequential minimal optimization (SMO), and random forests (RF); they show that Bayesian classifiers, RF, and SMO outperform the other methods. In Bifet and Frank (2010), Bifet et al. compare MNB, stochastic gradient descend (SGD), and a Hoeffding tree (HT) algorithm; they report that MNB and SGD perform comparably when the stream is stable, but MNB has difficulties in adapting to drifts. In terms of efficiency, MNB is the fastest and HT is the slowest.

In their survey on concept drift adaption (Gama et al. 2014), Gama et al. elaborate on how *forgetting* of old data can be used to adjust a model to drift, and they discuss different forgetting strategies. The Hoeffding tree variant AdaHT (Bifet and Gavaldà 2009) forgets subtrees if performance degrades. In an opinionated stream, it is reasonable to also forget *words*, i.e., parts of the feature space, since the choice of words used in the data (here: documents!) may also change. The MNB variant proposed in Wagner et al. (2015) quantifies the contribution of a word to the polarity model by considering the number of documents containing this word and the recency of these documents; this variant is shown to adapt well to changes in the stream.

Stream classification algorithms for an evolving vocabulary. The problem of vocabulary evolution is rarely investigated in the context of stream mining. There are studies on online topic modeling and clustering on text streams, in which the model is adapted when the vocabulary – the feature space – changes (AlSumait et al. 2008; Gohr et al. 2009; Zimmermann et al. 2016), but most studies assume that all words are known in advance, and only their contribution to the model may change over time.

Among the stream classification algorithms, adaption to an evolving vocabulary is possible for some algorithms. The Hoeffding tree variant AdaHT (Bifet and Gavaldà 2009) can forget deprecated words when it forgets parts of the model (subtrees) and may be able to include new words when it builds new subtrees. The multinomial Naive Bayes variant proposed in Wagner et al. (2015) does modify the vocabulary, by considering at each timepoint only words that appear often in recent documents.

Adaption to an evolving vocabulary is an open problem. Currently, only few stream classification algorithms can deal with changes in the feature space. How to employ other classification algorithms over the opinionated stream? The fallback solution is to extend the workflow by a task that regularly recomputes the vocabulary/feature space from the most recent documents and then re-initializes the polarity model. This solution has

the disadvantage that the old model is completely forgotten, but the advantage that any stream classification algorithm can be used for learning.

Semi-supervised Opinion Stream Classification

Goal of semi-supervised stream learning is to learn a model on an initial set of manually labeled documents, sometimes called the "seed set" or "initial seed," and then adapt the model by using the arriving unlabeled instances. Semi-supervised methods have the inherent advantage of not demanding human intervention after the initialization of the model.

For this family of methods, the initial seed is the only available ground truth. Hence, it is essential that the instances comprising the seed set are a representative sample. Evidently, this sample ceases being representative, as soon as concept drift occurs. Semi-supervised learning algorithms adapt to drift by building a training set that consists of the initial seed and arriving unlabeled instances, to which they themselves assign the labels. There are two strategies for the selection of unlabeled instances to be labeled by the classifier and added to the training set. The first strategy chooses instances on the grounds of the classifier's confidence to the predicted labels. The second strategy chooses instances by considering their similarity to previously labeled instances.

First strategy. Chapelle et al. point out that "Probably the earliest idea about using unlabeled data in classification is self-learning, which is also known as self-training, self-labeling, or decision-directed learning. This is a wrapper-algorithm that repeatedly uses a supervised learning method. It starts by training on the labeled data only. In each step a part of the unlabeled points is labeled according to the current decision function; then the supervised method is retrained using its own predictions as additional labeled points . . . " (Chapelle et al. 2006). However, self-training may lead to performance deterioration, because erroneous predictions of the classifier lead to erroneous labels in the training set.

Another approach is the "co-training" of several independent classifiers (Blum and Mitchell 1998). In the context of text classification, Aggarwal and Zhai propose to split the feature space into subsets and train an independent classifier on each subset (Aggarwal and Zhai 2014); then, high-confidence predictions of each single classifier are used to feed the other classifiers with new labels, so that no classifier is trained on its own predictions.

An example of co-training on a stream of tweets is in Liu et al. (2013): the complete feature space encompasses both text features (such as adjectives) and non-text features (e.g., emoticons). Views are built over this feature space, and a classifier (multiclass SVM) is trained on each view, using a small set of labeled instances only.

Second strategy. As an alternative to self-training and co-training, the second semi-supervised strategy adds to the training set those instances that are most similar to already labeled instances. One way to capitalize on labeled instances under this strategy is to cluster labeled and unlabeled instances together, then determine the label of each cluster from the labeled instances in it, and finally select for training some unlabeled instances per cluster (e.g., those closest to the cluster center).

In the context of opinionated semi-supervised stream learning, a clustering-based strategy brings two advantages. First, text stream clustering algorithms can be used, whereupon the clusters are updated gradually, as new unlabeled instances arrive. Further, these clusters reflect the properties/topics in the opinionated stream, thus addressing challenge (b) of task 2 on opinionated streams (cf. section on "Motivation, Main Tasks, and Challenges"). Example methods have been proposed by Gan et al. (2013) and by Zimmermann et al. (2015a).

In the previous section on fully supervised learning, we point out that forgetting (old data, part of the model, part of the feature space) may be beneficial for model adaption (cf. Gama et al. 2014). When learning in a semi-supervised way, though, forgetting may have negative side effects: since the seed set is the only ground

truth provided by the human expert, forgetting those "precious" data labels is likely to lead to performance deterioration.

Active Learning for Opinion Stream Classification

Similarly to semi-supervised approaches, active learning methods attempt to learn and adapt to the ongoing stream without demanding a label for each arriving instance. Instead of re-acting to the labels that become available, active methods *proactively* (thereof the name "active") request labels for the instances expected to be most informative for learning.

In active stream learning, there are two ways of requesting labels for some of the arriving instances. In the pool-based scenario, unlabeled instances are collected into a pool; the active learning algorithm chooses a subset of them and asks for their labels. In the sequential scenario, the algorithm decides for each arriving instance whether it will request a label for it. An overview of active learning methods for conventional streams is in Zliobaite et al. (2011).

Active learning is often used for various text mining tasks, including sentiment classification (Zhou et al. 2013). Active algorithms for opinionated streams also gain momentum. CloudFlows is a cloud-based platform for opinion stream mining that adheres to the pool-based scenario (Saveski and Grcar 2011; Kranjc et al. 2015): a first model of the stream is learned from a large corpus of tweets that contain emoticons; after initialization, the stream is partitioned into chunks, and an active learning algorithm is used to select instances and store them in a pool. The instances in the pool are ranked, and the top-ranked positions are shown to human experts. This approach has the advantage that human experts (e.g., in crowdsourcing) label the opinionated documents shown to them offline, whereupon these newly labeled instances are used for classifier adaption.

The algorithm ACOSTREAM (Zimmermann et al. 2015b) adheres to the sequential scenario, in the sense that sampling is done for each instance individually at its arriving time. This algorithm uses a variant of multinomial naive Bayes for classification, which (as in Wagner et al. 2015) deals with changes in the vocabulary of the arriving documents.

The multiclass active learning algorithm of Cheng et al. (2013) combines uncertainty and likelihood sampling to choose instances that are close to the current decision boundary, as well as instances from a yet unseen part of the data space. This algorithm (which adheres to the sequential scenario) is particularly interesting for learning on text streams, where some of the most recent instances may belong to an area of the data space that did not contain any instances in the past.

Recent Developments

Opinion stream mining builds upon advances in opinion mining, stream classification, active stream learning, and semi-supervised stream learning. Traditional methods in this domain have not been designed with big data in mind. However, opinionated streams have big data characteristics: volume, variability, variety, and veracity.

Volume refers to the huge number of opinions uploaded daily in social media and to the high dimensionality of the opinionated documents.

Variability refers to changes in the data flow rate and to changes in the data distribution, i.e., to concept drift.

Variety refers to the heterogeneous data types, including plain texts, images, and videos. The graph structure of the social networks, in which opinion holders are linked to each other, also adds to the variety of the data relevant to opinion mining.

Veracity refers to the uncertainty of the polarity labels provided by the human experts: labeling an opinionated stream is an inherently difficult task, since some opinionated documents (e.g., documents containing subtle irony) may be perceived differently by different people.

Challenges associated to these four Vs are not always peculiar to opinion stream mining: while challenges associated to variability are exacerbated in the opinion stream mining context, challenges associated to, e.g., volume can benefit

from general-purpose big data solutions. These include, among others, scalable machine learning and online NLP algorithms, crowdsourcing approaches for data labeling, visualization advances, and visual analytics for the monitoring and interpretation of activities on social platforms.

Open Problems

Opinion stream mining is a rather young area. Open problems include:

- How to extend the traditional notion of "concept drift" so that it also cover changes in the feature space? How to design algorithms that detect such changes and adapt to them in an efficient way?
- How to distinguish between concept drift and "virtual drift" (Gama et al. 2014), i.e., between changes that do affect the decision boundary and changes that do not?
  Especially in an opinionated stream, many changes occur at each moment, e.g., new words appear, and the number of postings changes with the hour of the day, but not all of them require model adaption. How to design algorithms that recognize virtual drift and only adapt the model when true concept drift occurs?
- How to capture changes in the semantics and polarity of words?
  If a word's semantics or polarity change, how to inform existing resources (e.g., lexica like SentiWordNet) that a word's meaning and polarity are different for old documents than for recent ones?
- How to deal with label veracity in the stream? A promising approach is crowdsourcing, s is done, e.g., in CloudFlows (Kranjc et al. 2015). *Amazon Mechanical Turk* is a popular platform, where one can upload tasks for crowdsourcing. However, crowdsourcing has not been designed for learning and adaption on a fast stream, so solutions that also deal with stream velocity are necessary.

An associated open issue that can also be found in text stream mining, e.g., in the analysis of news streams, concerns the description of *bursts*. A burst is a rapid increase in social activity and may also be associated with a rapid change in the class priors and in the words being used to express polarity and to express facts. Do these changes disappear after the burst fades out, or do people take up the new words/expressions and use them also when they express opinions on other subjects? Does a burst lead to (more) permanent changes in the way people express opinions, on their perception toward a given entity, or on the topics they discuss?

Impact

Opinions have been always important for decision making. The opinion deluge we encounter nowadays mainly due to the WWW and the widespread usage of social networks is transforming business, society, and our own decisions on, e.g., what product to buy, which movie to watch, etc. Opinion (stream) mining offers solutions for automatically exploiting such sort of data for decision making, through, e.g., prediction models. Beyond its usage as a "standalone tool" for, e.g., polarity prediction, opinion (stream) mining has an impact on other areas of research, an example of which is the area of *recommenders:* next to the ratings typically used by recommenders, it is possible to also capitalize on the user reviews as more and more users also provide reviews on the rated items. These reviews are rich in information: they typically describe the aspects of the items that the users like/dislike. Further, if there are no ratings, they may be inferred from the reviews. A recent work in this area is McAuley and Leskovec (2013).

Cross-References

- ▸ Active Learning
- ▸ Concept Drift
- ▸ Co-training
- ▸ Incremental Learning

- Online Learning
- Semi-supervised Learning
- Sentiment Analysis and Opinion Mining

Recommended Reading

Some of the publications cited thus far elaborate on issues that were only briefly touched in this lemma. In Liu (2012), Bing Liu gives a thorough overview of sentiment analysis and opinion mining. For text classification methods, readers are referred to the recent book chapter of Aggarwal and Zhai (2014).

References

Aggarwal CC, Yu PS (2006) A framework for clustering massive text and categorical data. In: Proceedings of 6th SIAM international conference on data mining (SDM'06), Bethesda. SIAM, pp 479–483

Aggarwal C, Zhai C (2014) Text classification. In: Aggarwal C (ed) Data classification: algorithms and applications, chapter 11. Chapman & Hall/CRC, Boca Raton, pp 287–336

AlSumait L, Barbara D, Domeniconi C (2008) On-line LDA: adaptive topic models for mining text streams with applications to topic detection and tracking. In: Proceedings of 2008 IEEE conference on data mining (ICDM'08), Pisa. IEEE, pp 373–382

Bifet A, Frank E (2010) Sentiment knowledge discovery in Twitter streaming data. In: Proceedings of the 13th international conference on discovery science (DS'10), Canberra. Springer, pp 1–15

Bifet A, Gavaldà R (2009) Adaptive learning from evolving data streams. In: Proceedings of the 8th international symposium on intelligent data analysis: advances in intelligent data analysis VIII (IDA), Lyon. Springer, pp 249–260

Blei DM, Lafferty JD (2006) Dynamic topic models. In: Proceedings of 23rd international conference on machine learning (ICML'06), Pittsburgh, pp 113–120

Blum A, Mitchell T (1998) Combining labeled and unlabeled data with co-training. In: Proceedings of 11th conference on computational learning theory, Madison. ACM, pp 92–100

Chapelle O, Schölkopf B, Zien A (2006) Semi-supervised learning. MIT, Cambridge

Cheng Y, Chen Z, Liu L, Wang J, Agrawal A, Choudhary A (2013) Feedback-driven multiclass active learning for data streams. In: Proceedings of 22nd international conference on information and knowledge management (CIKM'13), San Fransisco, pp 1311–1320

Fu X, Yang K, Huang JZ, Cui L (2015) Dynamic non-parametric joint sentiment topic mixture model. Know-Based Syst 82(C):102–114

Gama J, Žliobaitė I, Bifet A, Pechenizkiy M, Bouchachia A (2014) A survey on concept drift adaptation. ACM Comput Surv 46(4):44:1–44:37

Gan H, Sang N, Huang R, Tong X, Dan Z (2013) Using clustering analysis to improve semi-supervised classification. Neurocomputing 101:290–298

Gohr A, Hinneburg A, Schult R, Spiliopoulou M (2009) Topic evolution in a stream of documents. In: SIAM data mining conference (SDM'09), Reno, pp 378–385

Gokulakrishnan B, Priyanthan P, Ragavan T, Prasath N, Perera A (2012) Opinion mining and sentiment analysis on a Twitter data stream. In: Proceedings of the 2012 international conference on advances in ICT for emerging regions (ICTer), Colombo, pp 182–188

Kranjc J, Smailovic J, Podpecan V, Grcar M, Znidarsic M, Lavrac N (2015) Active learning for sentiment analysis on data streams: methodology and workflow implementation in the ClowdFlows platform. Inf Process Manag 51(2):187–203

Liu B (2012) Sentiment analysis and opinion mining. Synth Lect Hum Lang Technol 5(1):1–167

Liu S, Li F, Li F, Cheng X, Shen H (2013) Adaptive co-training SVM for sentiment classification on tweets. In: Proceedings of 22nd international conference on information and knowledge management (CIKM'13), San Fransisco, pp 2079–2088

McAuley J, Leskovec J (2013) Hidden factors and hidden topics: understanding rating dimensions with review text. In: Proceedings of 7th ACM conference on recommender systems (RecSys'13), Hong Kong. ACM, pp 165–172

Saveski M, Grcar M (2011) Web services for stream mining: a stream-based active learning use case. In: Proceedings of the workshop "Planning to Learn and Service-Oriented Knowledge Discovery" at ECML PKDD 2011, Athens

Wagner S, Zimmermann M, Ntoutsi E, Spiliopoulou M (2015) Ageing-based multinomial naive bayes classifiers over opinionated data streams. In: European conference on machine learning and principles and practice of knowledge discovery in databases (ECMLPKDD'15), Porto, 07–11 Sept 2015. Volume 9284 of lecture notes in computer science. Springer International Publishing

Wang X, McCallum A (2006) Topics over time: a non-Markov continuous-time model of topical trends. In: Proceedings of 12th ACM SIGKDD international conference on knowledge discovery and data mining (KDD'06), Philadelphia, pp 424–433

Zhou S, Chen Q, Wang X (2013) Active deep learning method for semi-supervised sentiment classification. Neurocomputing 120:536–546

Zimmermann M, Ntoutsi E, Spiliopoulou M (2015a) Discovering and monitoring product features and

the opinions on them with OPINSTREAM. Neurocomputing 150:318–330
Zimmermann M, Ntoutsi E, Spiliopoulou M (2015b) Incremental active opinion learning over a stream of opinionated documents. In: WISDOM'15 (workshop on issues of sentiment discovery and opinion mining) at KDD'15, Sydney
Zimmermann M, Ntoutsi E, Spiliopoulou M (2016) Extracting opinionated (sub)features from a stream of product reviews using accumulated novelty and internal re-organization. Inf Sci 329:876–899
Zliobaite I, Bifet A, Pfahringer B, Holmes G (2011) Active learning with evolving streaming data. In: Proceedings of ECML PKDD 2011, Athens. Volume 6913 of LNCS. Springer

Optimal Learning

▸ Bayesian Reinforcement Learning

Ordered Rule Set

▸ Decision List

Ordinal Attribute

An *ordinal attribute* classifies data into categories that can be ranked. However, the differences between the ranks cannot be calculated by arithmetic. See ▸ Attribute and ▸ Measurement Scales.

Out-of-Sample Data

Out-of-sample data are data that were not used to learn a model. ▸ Holdout evaluation uses out-of-sample data for evaluation purposes.

Out-of-Sample Evaluation

Definition

Out-of-sample evaluation refers to ▸ algorithm evaluation whereby the learned model is evaluated on ▸ out-of-sample data, which are data that were not used in the process of learning the model. Out-of-sample evaluation provides a less biased estimate of learning performance than ▸ in-sample evaluation. ▸ Cross validation, ▸ holdout evaluation and ▸ prospective evaluation are three main approaches to out-of-sample evaluation. Cross validation and holdout evaluation run risks of overestimating performance relative to what should be expected on future data, especially if the data set used is not a true random sample of the distribution on which the learned models are to be applied in the future.

Cross-References

▸ Algorithm Evaluation

Overall and Class-Sensitive Frequencies

The underlying idea for learning strategies processing ▸ missing attribute values relies on the class distribution; i.e., the class-sensitive frequencies are utilized. As soon as we substitute a missing value by a suitable one, we take the desired class of the example into consideration in order not to increase the noise in the data set. On the other hand, the overall (class-independent) frequencies are applied within classification.

Overfitting

Geoffrey I. Webb
Faculty of Information Technology, Monash University, Clayton, Melbourne, VIC, Australia

Synonyms

Overtraining

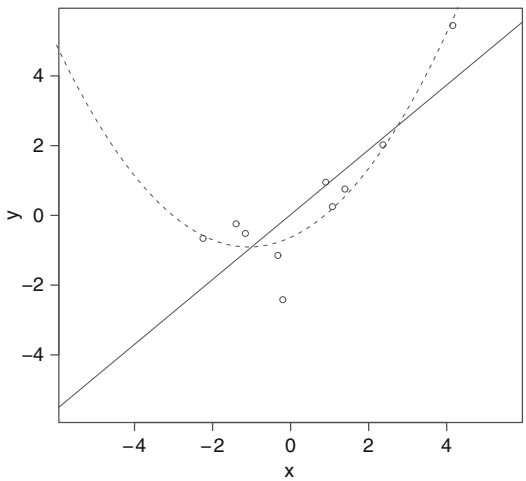
Definition

A model *overfits* the ▸ training data when it describes features that arise from noise or variance in the data, rather than the underlying distribution from which the data were drawn. Overfitting usually leads to loss of ▸ accuracy on ▸ out-of-sample data.

Discussion

In general there is a trade-off between the size of the space of distinct models that a learner can produce and the risk of overfitting. As the space of models between which the learner can select increases, the risk of overfitting will increase. However, the potential for finding a model that closely fits the true underling distribution will also increase. This can be viewed as one facet of the ▸ bias and variance trade-off.

Figure 1 illustrates overfitting. The points are drawn randomly from a distribution in which $y = x + \varepsilon$, where ε is random noise. The best single line fit to this distribution is $y = x$. ▸ Linear regression finds a model $y = 0.02044 + 0.92978 \times x$, shown as the solid line in Fig. 1. In contrast, second degree polynomial regression finds the model $-0.6311 + 0.5128 \times x + 0.2386 \times x^2$, shown as the dashed line. The space of second degree polynomial models is greater than that of linear models, and so the second degree polynomial more closely fits the example data, returning the lower ▸ squared error. However, the linear model more closely fits the true distribution and is more likely to obtain lower squared error on future samples.



Overfitting, Fig. 1 Linear and polynomial models fitted to random data drawn from a distribution for which the linear model is a better fit

While this example relates to ▸ regression, the same effect also applies to classification problems. For example, an overfitted ▸ decision tree may include splits that reflect noise rather than underlying regularities in the data.

The many approaches to avoiding overfitting include

- Using low variance learners;
- ▸ Minimum Description Length and ▸ Minimum Message Length techniques
- ▸ Pruning
- ▸ Regularization

Cross-References

▸ Bias Variance Decomposition
▸ Minimum Description Length Principle
▸ Minimum Message Length
▸ Pruning
▸ Regularization

Overtraining

▸ Overfitting