CS114: Classification Problems 1

February 6, 2015

James Pustejovsky

Brandeis University

Thanks for Ray Mooney for most of these slides

Classification (Categorization)

- Given:
  - A description of an instance, $x \in X$, where X is the ***instance language*** or ***instance space***.
  - A fixed set of categories: $C=\{c_1, c_2, \ldots c_n\}$
- Determine:
  - The category of x: $c(x) \in C$, where $c(x)$ is a categorization function whose domain is X and whose range is C.
  - If $c(x)$ is a binary function $C=\{0,1\}$ ({true,false}, {positive, negative}) then it is called a ***concept***.

Learning for Categorization

- A training example is an instance $x \in X$, paired with its correct category $c(x)$: $<x, c(x)>$ for an unknown categorization function, *c*.
- Given a set of training examples, *D*.
- Find a hypothesized categorization function, $h(x)$, such that:

$$\forall < x, c(x) > \in D : h(x) = c(x)$$

Consistency

Sample Category Learning Problem

- Instance language: <size, color, shape>
  - size ∈ {small, medium, large}
  - color ∈ {red, blue, green}
  - shape ∈ {square, circle, triangle}
- *C* = {positive, negative}
- *D*:

Example	Size	Color	Shape	Category
1	small	red	circle	positive
2	large	red	circle	positive
3	small	red	triangle	negative
4	large	blue	circle	negative

Hypothesis Selection

- Many hypotheses are usually consistent with the training data.
  - red & circle
  - (small & circle) or (large & red)
  - (small & red & circle) or (large & red & circle)
  - not [(red & triangle) or (blue & circle)]
  - not [(small & red & triangle) or (large & blue & circle)]
- Bias
  - Any criteria other than consistency with the training data that is used to select a hypothesis.

Generalization

- Hypotheses must generalize to correctly classify instances not in the training data.
- Simply memorizing training examples is a consistent hypothesis that does not generalize.
- *Occam's razor*:
  - Finding a *simple* hypothesis helps ensure generalization.

Hypothesis Space

- Restrict learned functions a priori to a given ***hypothesis space***, *H*, of functions *h*(*x*) that can be considered as definitions of *c*(*x*).
- For learning concepts on instances described by *n* discrete-valued features, consider the space of conjunctive hypotheses represented by a vector of *n* constraints

  <c_1, c_2, … c_n> where each c_i is either:
  - ?, a wild card indicating no constraint on the *i*th feature
  - A specific value from the domain of the *i*th feature
  - Ø indicating no value is acceptable
- Sample conjunctive hypotheses are
  - <big, red, ?>
  - <?, ?, ?> (most general hypothesis)
  - < Ø, Ø, Ø> (most specific hypothesis)

Inductive Learning Hypothesis

- Any function that is found to approximate the target concept well on a sufficiently large set of training examples will also approximate the target function well on unobserved examples.
- Assumes that the training and test examples are drawn independently from the same underlying distribution.
- This is a fundamentally unprovable hypothesis unless additional assumptions are made about the target concept and the notion of "approximating the target function well on unobserved examples" is defined appropriately (cf. computational learning theory).

Evaluation of Classification Learning

- Classification accuracy (% of instances classified correctly).
  - Measured on an independent test data.
- Training time (efficiency of training algorithm).
- Testing time (efficiency of subsequent classification).

Category Learning as Search

- Category learning can be viewed as searching the hypothesis space for one (or more) hypotheses that are consistent with the training data.
- Consider an instance space consisting of *n* binary features which therefore has 2^n instances.
- For conjunctive hypotheses, there are 4 choices for each feature: Ø, T, F, ?, so there are 4^n syntactically distinct hypotheses.
- However, all hypotheses with 1 or more Øs are equivalent, so there are 3^n+1 semantically distinct hypotheses.
- The target binary categorization function in principle could be any of the possible $2^{2^{\wedge}n}$ functions on *n* input bits.
- Therefore, conjunctive hypotheses are a small subset of the space of possible functions, but both are intractably large.
- All reasonable hypothesis spaces are intractably large or even infinite.

Learning by Enumeration

- For any finite or countably infinite hypothesis space, one can simply enumerate and test hypotheses one at a time until a consistent one is found.

  For each *h* in *H* do:

  If *h* is consistent with the training data *D*,

  then terminate and return *h*.

- This algorithm is guaranteed to terminate with a consistent hypothesis if one exists; however, it is obviously computationally intractable for almost any practical problem.

Efficient Learning

- Is there a way to learn conjunctive concepts without enumerating them?
- How do human subjects learn conjunctive concepts?
- Is there a way to efficiently find an unconstrained boolean function consistent with a set of discrete-valued training instances?
- If so, is it a useful/practical algorithm?

Conjunctive Rule Learning

- Conjunctive descriptions are easily learned by finding all commonalities shared by all positive examples.

Example	Size	Color	Shape	Category
1	small	red	circle	positive
2	large	red	circle	positive
3	small	red	triangle	negative
4	large	blue	circle	negative

Learned rule: red & circle → positive

- Must check consistency with negative examples. If inconsistent, **no** conjunctive rule exists.

Limitations of Conjunctive Rules

- If a concept does not have a single set of necessary and sufficient conditions, conjunctive learning fails.

Example	Size	Color	Shape	Category
1	small	red	circle	positive
2	large	red	circle	positive
3	small	red	triangle	negative
4	large	blue	circle	negative
5	medium	red	circle	negative

Learned rule: red & circle → positive

Inconsistent with negative example #5!

Disjunctive Concepts

- Concept may be disjunctive.

Example	Size	Color	Shape	Category
1	small	red	circle	positive
2	large	red	circle	positive
3	small	red	triangle	negative
4	large	blue	circle	negative
5	medium	red	circle	negative

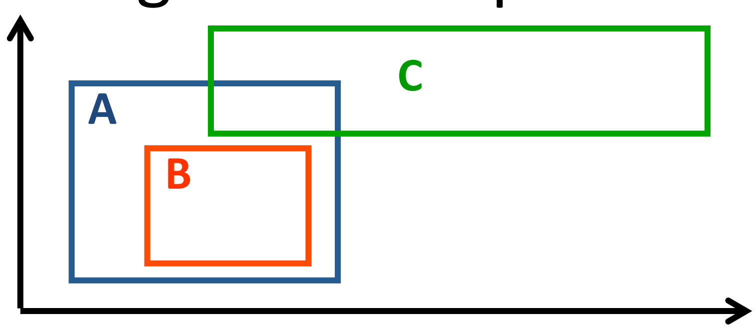
Learned rules: small & circle → positive
large & red → positive

Using the Generality Structure

- By exploiting the structure imposed by the generality of hypotheses, an hypothesis space can be searched for consistent hypotheses without enumerating or explicitly exploring all hypotheses.
- An instance, $x \in X$, is said to ***satisfy*** an hypothesis, h, iff $h(x)=1$ (positive)
- Given two hypotheses h_1 and h_2, h_1 is ***more general than or equal to*** h_2 ($h_1 \geq h_2$) iff every instance that satisfies h_2 also satisfies h_1.
- Given two hypotheses h_1 and h_2, h_1 is **(*strictly*) *more general than*** h_2 ($h_1 > h_2$) iff $h_1 \geq h_2$ and it is not the case that $h_2 \geq h_1$.
- Generality defines a partial order on hypotheses.

Examples of Generality

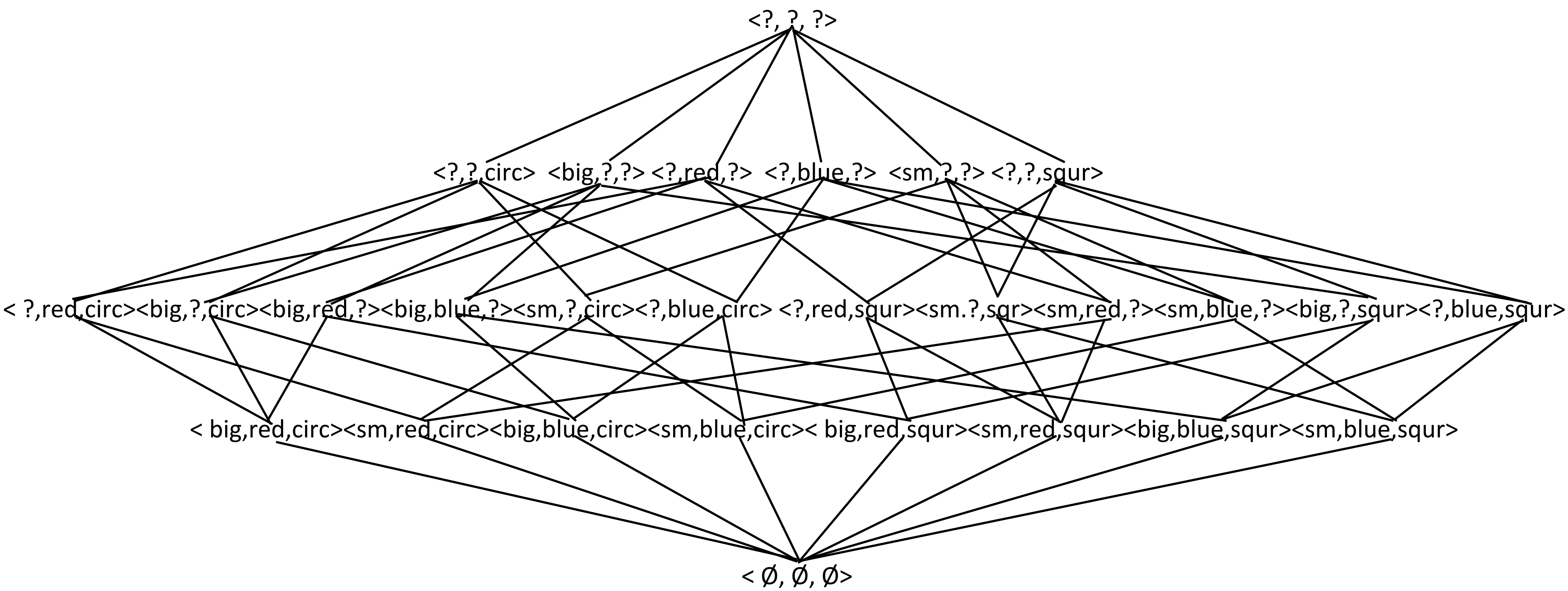
- Conjunctive feature vectors
  - <?, red, ?> is more general than <?, red, circle>
  - Neither of <?, red, ?> and <?, ?, circle> is more general than the other.
- Axis-parallel rectangles in 2-d space



  - A is more general than B
  - Neither of A and C are more general than the other.

Sample Generalization Lattice

Size: {sm, big} Color: {red, blue} Shape: {circ, squr}



Number of hypotheses = $3^3 + 1 = 28$

Most Specific Learner (Find-S)

- Find the most-specific hypothesis (least-general generalization, LGG) that is consistent with the training data.
- Incrementally update hypothesis after every positive example, generalizing it just enough to satisfy the new example.
- For conjunctive feature vectors, this is easy:

Initialize *h* = <Ø, Ø,… Ø>
For each positive training instance *x* in *D*
  For each feature f_i
    If the constraint on f_i in *h* is **not** satisfied by *x*
      If f_i in *h* is Ø
        then set f_i in *h* to the value of f_i in *x*
        else set f_i in *h* to "?"
If *h* is consistent with the negative training instances in *D*
  then return *h*
  else no consistent hypothesis exists

Time complexity: O(|D| n) if n is the number of features

Properties of Find-S

- For conjunctive feature vectors, the most-specific hypothesis is unique and found by Find-S.
- If the most specific hypothesis is not consistent with the negative examples, then there is no consistent function in the hypothesis space, since, by definition, it cannot be made more specific and retain consistency with the positive examples.
- For conjunctive feature vectors, if the most-specific hypothesis is inconsistent, then the target concept must be disjunctive.

Another Hypothesis Language

- Consider the case of two *unordered* objects each described by a fixed set of attributes.
  - {<big, red, circle>, <small, blue, square>}
- What is the most-specific generalization of:
  - Positive: {<big, red, triangle>, <small, blue, circle>}
  - Positive: {<big, blue, circle>, <small, red, triangle>}
- LGG is not unique, two incomparable generalizations are:
  - {<big, ?, ?>, <small, ?, ?>}
  - {<?, red, triangle>, <?, blue, circle>}
- For this space, Find-S would need to maintain a continually growing set of LGGs and eliminate those that cover negative examples.
- Find-S is no longer tractable for this space since the number of LGGs can grow exponentially.

Issues with Find-S

- Given sufficient training examples, does Find-S converge to a correct definition of the target concept (assuming it is in the hypothesis space)?
- How de we know when the hypothesis has converged to a correct definition?
- Why prefer the most-specific hypothesis? Are more general hypotheses consistent? What about the most-general hypothesis? What about the simplest hypothesis?
- If the LGG is not unique
  - Which LGG should be chosen?
  - How can a single consistent LGG be efficiently computed or determined not to exist?
- What if there is noise in the training data and some training examples are incorrectly labeled?

Effect of Noise in Training Data

- Frequently realistic training data is corrupted by errors (noise) in the features or class values.
- Such noise can result in missing valid generalizations.
  - For example, imagine there are many positive examples like #1 and #2, but out of many negative examples, only one like #5 that actually resulted from a error in labeling.

Example	Size	Color	Shape	Category
1	small	red	circle	positive
2	large	red	circle	positive
3	small	red	triangle	negative
4	large	blue	circle	negative
5	medium	red	circle	negative

Version Space

- Given an hypothesis space, *H*, and training data, *D*, the ***version space*** is the complete subset of *H* that is consistent with *D*.
- The version space can be naively generated for any finite *H* by enumerating all hypotheses and eliminating the inconsistent ones.
- Can one compute the version space more efficiently than using enumeration?

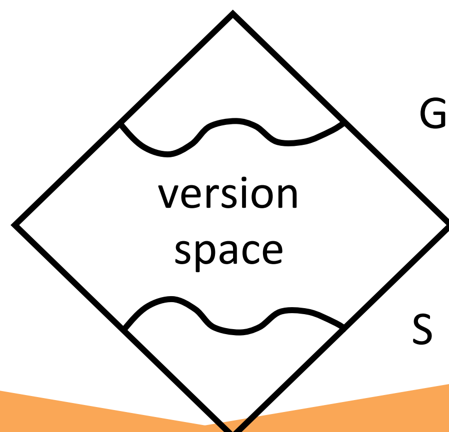
Version Space with S and G

- The version space can be represented more compactly by maintaining two boundary sets of hypotheses, *S*, the set of most specific consistent hypotheses, and *G*, the set of most general consistent hypotheses:

$$S = \{s \in H \mid Consistent(s, D) \wedge \neg \exists s' \in H[s > s' \wedge Consistent(s', D)]\}$$

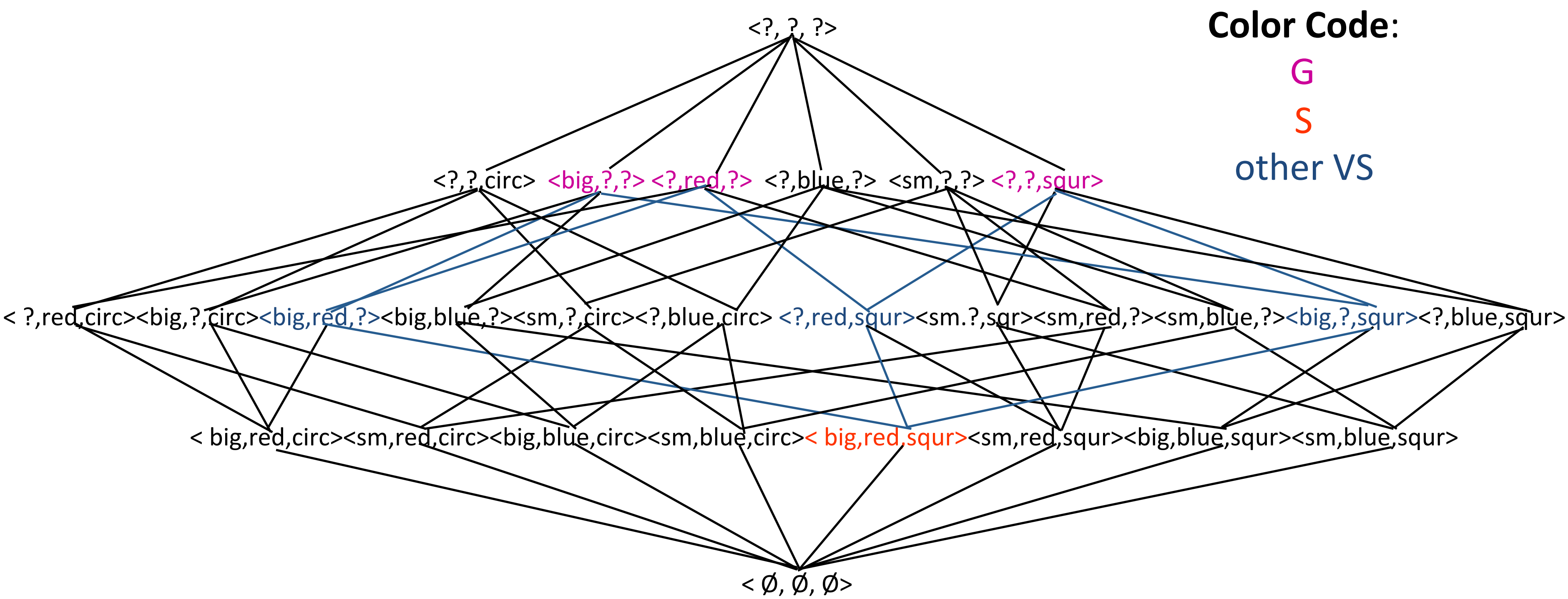
$$G = \{g \in H \mid Consistent(g, D) \wedge \neg \exists g' \in H[g' > g \wedge Consistent(s', D)]\}$$

- S and G represent the entire version space via its boundaries in the generalization lattice:



Version Space Lattice

Size: {sm, big}   Color: {red, blue}   Shape: {circ, squr}



<<big, red, squr> positive>
<<sm, blue, circ> negative>

Candidate Elimination (Version Space) Algorithm

Initialize G to the set of most-general hypotheses in H
Initialize S to the set of most-specific hypotheses in H
For each training example, d, do:

- If d is a positive example then:
  - Remove from G any hypotheses that do not match d
  - For each hypothesis s in S that does not match d
    - Remove s from S
    - Add to S all minimal generalizations, h, of s such that:
      1) h matches d
      2) some member of G is more general than h
  - Remove from S any h that is more general than another hypothesis in S
- If d is a negative example then:
  - Remove from S any hypotheses that match d
  - For each hypothesis g in G that matches d
    - Remove g from G
    - Add to G all minimal specializations, h, of g such that:
      1) h does not match d
      2) some member of S is more specific than h
  - Remove from G any h that is more specific than another hypothesis in G

Required Subroutines

- To instantiate the algorithm for a specific hypothesis language requires the following procedures:
  - equal-hypotheses(h_1, h_2)
  - more-general(h_1, h_2)
  - match(h, i)
  - initialize-g()
  - initialize-s()
  - generalize-to(h, i)
  - specialize-against(h, i)

Minimal Specialization and Generalization

- Procedures generalize-to and specialize-against are specific to a hypothesis language and can be complex.
- For conjunctive feature vectors:
  - generalize-to: unique, see Find-S
  - specialize-against: not unique, can convert each "?" to an alernative non-matching value for this feature.
    - Inputs:
      - *h* = <?, red, ?>
      - *i* = <small, red, triangle>
    - Outputs:
      - <big, red, ?>
      - <medium, red, ?>
      - <?, red, square>
      - <?, red, circle>

Sample VS Trace

S= {< Ø, Ø, Ø>}; G= {<?, ?, ?>}

Positive: <big, red, circle>
Nothing to remove from G
Minimal generalization of only S element is <big, red, circle> which is more specific than G.
S={<big, red, circle>}; G={<?, ?, ?>}

Negative: <small, red, triangle>
Nothing to remove from S.
Minimal specializations of <?, ?, ?> are <medium, ?, ?>, <big, ?, ?>, <?, blue, ?>, <?, green, ?>, <?, ?, circle>, <?, ?, square> but most are not more general than some element of S
S={<big, red, circle>}; G={<big, ?, ?>, <?, ?, circle>}

Sample VS Trace (cont)

S={<big, red, circle>}; G={<big, ?, ?>, <?, ?, circle>}

Positive: <small, red, circle>
Remove <big, ?, ?> from G
Minimal generalization of <big, red, circle> is <?, red, circle>
S={<?, red, circle>}; G={<?, ?, circle>}

Negative: <big, blue, circle>
Nothing to remove from S
Minimal specializations of <?, ?, circle> are <small, ? circle>, <medium, ?, circle>, <?, red, circle>, <?, green, circle> but most are not more general than some element of S.
S={<?, red, circle>}; G={<?, red, circle>}

S=G; Converged!

Properties of VS Algorithm

- *S* summarizes the relevant information in the positive examples (relative to *H*) so that positive examples do not need to be retained.
- *G* summarizes the relevant information in the negative examples, so that negative examples do not need to be retained.
- Result is not affected by the order in which examples are processes but computational efficiency may.
- Positive examples move the *S* boundary up; Negative examples move the *G* boundary down.
- If *S* and *G* converge to the same hypothesis, then it is the only one in *H* that is consistent with the data.
- If *S* and *G* become empty (if one does the other must also) then there is no hypothesis in *H* consistent with the data.

Sample Weka VS Trace 1

```
java weka.classifiers.vspace.ConjunctiveVersionSpace -t figure.arff -T figure.arff -v -P

Initializing VersionSpace
S = [[#,#,#]]
G = [[?,?,?]]

Instance: big,red,circle,positive
S = [[big,red,circle]]
G = [[?,?,?]]

Instance: small,red,square,negative
S = [[big,red,circle]]
G = [[big,?,?], [?,?,circle]]

Instance: small,red,circle,positive
S = [[?,red,circle]]
G = [[?,?,circle]]

Instance: big,blue,circle,negative
S = [[?,red,circle]]
G = [[?,red,circle]]

Version Space converged to a single hypothesis.
```

Sample Weka VS Trace 2

java weka.classifiers.vspace.ConjunctiveVersionSpace -t figure2.arff -T figure2.arff -v -P

Initializing VersionSpace
S = [[#,#,#]]
G = [[?,?,?]]

Instance: big,red,circle,positive
S = [[big,red,circle]]
G = [[?,?,?]]

Instance: small,blue,triangle,negative
S = [[big,red,circle]]
G = [[big,?,?], [?,red,?], [?,?,circle]]

Instance: small,red,circle,positive
S = [[?,red,circle]]
G = [[?,red,?], [?,?,circle]]

Instance: medium,green,square,negative
S = [[?,red,circle]]
G = [[?,red,?], [?,?,circle]]

Sample Weka VS Trace 3

```
java weka.classifiers.vspace.ConjunctiveVersionSpace -t figure3.arff -T figure3.arff -v -P
Initializing VersionSpace
S = [[#,#,#]]
G = [[?,?,?]]

Instance: big,red,circle,positive
S = [[big,red,circle]]
G = [[?,?,?]]

Instance: small,red,triangle,negative
S = [[big,red,circle]]
G = [[big,?,?], [?,?,circle]]

Instance: small,red,circle,positive
S = [[?,red,circle]]
G = [[?,?,circle]]

Instance: big,blue,circle,negative
S = [[?,red,circle]]
G = [[?,red,circle]]
Version Space converged to a single hypothesis.

Instance: small,green,circle,positive
S = []
G = []
Language is insufficient to describe the concept.
```

Sample Weka VS Trace 4

java weka.classifiers.vspace.ConjunctiveVersionSpace -t figure4.arff -T figure4.arff -v -P

Initializing VersionSpace
S = [[#,#,#]]
G = [[?,?,?]]

Instance: small,red,square,negative
S = [[#,#,#]]
G = [[medium,?,?], [big,?,?], [?,blue,?], [?,green,?], [?,?,circle], [?,?,triangle]]

Instance: big,blue,circle,negative
S = [[#,#,#]]
G = [[medium,?,?], [?,green,?], [?,?,triangle], [big,red,?], [big,?,square], [small,blue,?], [?,blue,square], [small,?,circle], [?,red,circle]]

Instance: big,red,circle,positive
S = [[big,red,circle]]
G = [[big,red,?], [?,red,circle]]

Instance: small,red,circle,positive
S = [[?,red,circle]]
G = [[?,red,circle]]

Version Space converged to a single hypothesis.

Correctness of Learning

- Since the entire version space is maintained, given a continuous stream of noise-free training examples, the VS algorithm will eventually converge to the correct target concept if it is in the hypothesis space, *H*, or eventually correctly determine that it is not in *H*.
- Convergence is correctly indicated when S=G.

Computational Complexity of VS

- Computing the *S* set for conjunctive feature vectors is linear in the number of features and the number of training examples.
- Computing the *G* set for conjunctive feature vectors is exponential in the number of training examples in the worst case.
- In more expressive languages, both *S* and *G* can grow exponentially.
- The order in which examples are processed can significantly affect computational complexity.

Active Learning

- In ***active learning***, the system is responsible for selecting good training examples and asking a teacher (oracle) to provide a class label.
- In ***sample selection***, the system picks good examples to query by picking them from a provided pool of unlabeled examples.
- In ***query generation***, the system must generate the description of an example for which to request a label.
- Goal is to minimize the number of queries required to learn an accurate concept description.

Active Learning with VS

- An ideal training example would eliminate half of the hypotheses in the current version space regardless of its label.
- If a training example matches half of the hypotheses in the version space, then the matching half is eliminated if the example is negative, and the other (non-matching) half is eliminated if the example is positive.
- Example:
  - Assume training set
    - Positive: <big, red, circle>
    - Negative: <small, red, triangle>
  - Current version space:
    - {<big, red, circle>, <big, red, ?>, <big, ?, circle>, <?, red, circle> <?, ?, circle> <big, ?, ?>}
  - An optimal query: <big, blue, circle>
- Given a ceiling of $\log_2|VS|$ such examples will result in convergence. This is the best possible guarantee in general.

Using an Unconverged VS

- If the VS has not converged, how does it classify a novel test instance?
- If all elements of *S* match an instance, then the entire version space much (since it is more general) and it can be confidently classified as positive (assuming target concept is in *H*).
- If no element of *G* matches an instance, then the entire version space must not (since it is more specific) and it can be confidently classified as negative (assuming target concept is in *H*).
- Otherwise, one could vote all of the hypotheses in the VS (or just the *G* and *S* sets to avoid enumerating the VS) to give a classification with an associated confidence value.
- Voting the entire VS is probabilistically optimal assuming the target concept is in *H* and all hypotheses in *H* are equally likely *a priori*.

Learning for Multiple Categories

- What if the classification problem is not concept learning and involves more than two categories?
- Can treat as a series of concept learning problems, where for each category, C_i, the instances of C_i are treated as positive and all other instances in categories C_j, $j \neq i$ are treated as negative (***one-versus-all***).
- This will assign a unique category to each training instance but may assign a novel instance to zero or multiple categories.
- If the binary classifier produces confidence estimates (e.g. based on voting), then a novel instance can be assigned to the category with the highest confidence.
- Other approaches exist, such as learning to discriminate all pairs of categories (***all-pairs***) and combining decisions appropriately during test.

Inductive Bias

- A hypothesis space that does not include all possible classification functions on the instance space incorporates a bias in the type of classifiers it can learn.
- Any means that a learning system uses to choose between two functions that are both consistent with the training data is called ***inductive bias***.
- Inductive bias can take two forms:
  - ***Language bias***: The language for representing concepts defines a hypothesis space that does not include all possible functions (e.g. conjunctive descriptions).
  - ***Search bias***: The language is expressive enough to represent all possible functions (e.g. disjunctive normal form) but the search algorithm embodies a preference for certain consistent functions over others (e.g. syntactic simplicity).

Unbiased Learning

- For instances described by *n* features each with *m* values, there are m^n instances. If these are to be classified into *c* categories, then there are $c^{m\wedge n}$ possible classification functions.
  - For *n*=10, *m*=*c*=2, there are approx. $3.4\text{x}10^{38}$ possible functions, of which only 59,049 can be represented as conjunctions (an incredibly small percentage!)
- However, unbiased learning is futile since if we consider all possible functions then simply memorizing the data without any real generalization is as good an option as any.
- Without bias, the version-space is always trivial. The unique most-specific hypothesis is the disjunction of the positive instances and the unique most general hypothesis is the negation of the disjunction of the negative instances:

$$S = \{(p_1 \vee p_2 \vee \ldots \vee p_k)\}$$

$$G = \{\neg(n_1 \vee n_2 \vee \ldots \vee n_j)\}$$

Futility of Bias-Free Learning

- A learner that makes no *a priori* assumptions about the target concept has no rational basis for classifying any unseen instances.
- Inductive bias can also be defined as the assumptions that, when combined with the observed training data, logically entail the subsequent classification of unseen instances.
  - Training-data + inductive-bias **|—** novel-classifications
- The bias of the VS algorithm (assuming it refuses to classify an instance unless it is classified the same by all members of the VS), is simply that *H* contains the target concept.
- The rote learner, which refuses to classify any instance unless it has seen it during training, is the least biased.
- Learners can be partially ordered by their amount of bias
  - Rote-learner < VS Algorithm < Find-S

No Panacea

- No Free Lunch (NFL) Theorem (Wolpert, 1995)
  Law of Conservation of Generalization Performance (Schaffer, 1994)
  - One can prove that improving generalization performance on unseen data for some tasks will always decrease performance on other tasks (which require different labels on the unseen instances).
  - Averaged across all possible target functions, no learner generalizes to unseen data any better than any other learner.
- There does not exist a learning method that is uniformly better than another for all problems.
- Given any two learning methods *A* and *B* and a training set, *D*, there always exists a target function for which *A* generalizes better (or at least as well) as *B*.
  - Train both methods on *D* to produce hypotheses h_A and h_B.
  - Construct a target function that labels all unseen instances according to the predictions of h_A.
  - Test h_A and h_B on any unseen test data for this target function and conclude that h_A is better.

Logical View of Induction

- Deduction is inferring sound specific conclusions from general rules (axioms) and specific facts.
- Induction is inferring general rules and theories from specific empirical data.
- Induction can be viewed as inverse deduction.
  - Find a hypothesis *h* from data *D* such that
    - $h \cup B \vdash D$

      where *B* is optional background knowledge
- ***Abduction*** is similar to induction, except it involves finding a specific hypothesis, *h*, that best *explains* a set of evidence, *D,* or inferring cause from effect. Typically, in this case *B* is quite large compared to induction and *h* is smaller and more specific to a particular event.

Induction and the Philosophy of Science

- Bacon (1561-1626), Newton (1643-1727) and the sound deductive derivation of knowledge from data.
- Hume (1711-1776) and the ***problem of induction***.
  - Inductive inferences can never be proven and are always subject to disconfirmation.
- Popper (1902-1994) and ***falsifiability***.
  - Inductive hypotheses can only be falsified not proven, so pick hypotheses that are most subject to being falsified.
- Kuhn (1922-1996) and ***paradigm shifts***.
  - Falsification is insufficient, an alternative paradigm must be available that is clearly elegant and more explanatory must be available.
    - Ptolmaic epicycles and the Copernican revolution
    - Orbit of Mercury and general relativity
    - Solar neutrino problem and neutrinos with mass
- Postmodernism: Objective truth does not exist; relativism; science is a social system of beliefs that is no more valid than others (e.g. religion).

Ockham (Occam)'s Razor

- William of Ockham (1295-1349) was a Franciscan friar who applied the criteria to theology:
  - "Entities should not be multiplied beyond necessity" (Classical version but not an actual quote)
  - "The supreme goal of all theory is to make the irreducible basic elements as simple and as few as possible without having to surrender the adequate representation of a single datum of experience." (Einstein)
- Requires a precise definition of simplicity.
- Acts as a bias which assumes that nature itself is simple.
- Role of Occam's razor in machine learning remains controversial.

Naïve Bayes Classification

Axioms of Probability Theory

- All probabilities between 0 and 1

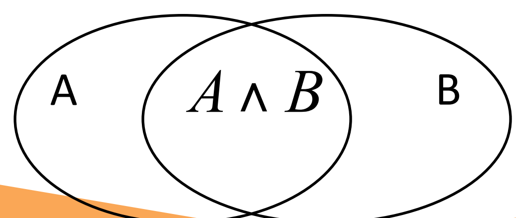
$$0 \le P(A) \le 1$$

- True proposition has probability 1, false has probability 0.

$$P(\text{true}) = 1 \qquad P(\text{false}) = 0.$$

- The probability of disjunction is:

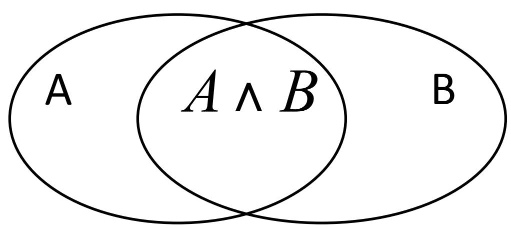
$$P(A \vee B) = P(A) + P(B) - P(A \wedge B)$$



Conditional Probability

- P(*A* | *B*) is the probability of *A* given *B*
- Assumes that *B* is all and only information known.
- Defined by:

$$P(A \mid B) = \frac{P(A \wedge B)}{P(B)}$$



Independence

- *A* and *B* are *independent* iff:

$$P(A \mid B) = P(A)$$

These two constraints are logically equivalent

$$P(B \mid A) = P(B)$$

- Therefore, if *A* and *B* are independent:

$$P(A \mid B) = \frac{P(A \wedge B)}{P(B)} = P(A)$$

$$P(A \wedge B) = P(A)P(B)$$

Joint Distribution

- The joint probability distribution for a set of random variables, $X_1,\ldots,X_n$ gives the probability of every combination of values (an n-dimensional array with v^n values if all variables are discrete with v values, all v^n values must sum to 1): $P(X_1,\ldots,X_n)$

positive

	circle	square
red	0.20	0.02
blue	0.02	0.01

negative

	circle	square
red	0.05	0.30
blue	0.20	0.20

- The probability of all possible conjunctions (assignments of values to some subset of variables) can be calculated by summing the appropriate subset of values from the joint distribution.

$$P(red \wedge circle) = 0.20 + 0.05 = 0.25$$
$$P(red) = 0.20 + 0.02 + 0.05 + 0.3 = 0.57$$

- Therefore, all conditional probabilities can also be calculated.

$$P(positive \mid red \wedge circle) = \frac{P(positive \wedge red \wedge circle)}{P(red \wedge circle)} = \frac{0.20}{0.25} = 0.80$$

Probabilistic Classification

- Let Y be the random variable for the class which takes values $\{y_1,y_2,\ldots y_m\}$.
- Let X be the random variable describing an instance consisting of a vector of values for n features $<X_1,X_2\ldots X_n>$, let x_k be a possible value for X and x_{ij} a possible value for $X_{i.}$
- For classification, we need to compute $P(Y=y_i \mid X=x_k)$ for $i=1\ldots m$
- However, given no other assumptions, this requires a table giving the probability of each category for each possible instance in the instance space, which is impossible to accurately estimate from a reasonably-sized training set.
  - Assuming Y and all X_i are binary, we need 2^n entries to specify $P(Y=pos \mid X=x_k)$ for each of the 2^n possible x_k's since $P(Y=neg \mid X=x_k) = 1 - P(Y=pos \mid X=x_k)$
  - Compared to $2^{n+1} - 1$ entries for the joint distribution $P(Y,X_1,X_2\ldots X_n)$

Bayes Theorem

$$P(H \mid E) = \frac{P(E \mid H)P(H)}{P(E)}$$

Simple proof from definition of conditional probability:

$$P(H \mid E) = \frac{P(H \wedge E)}{P(E)} \quad \text{(Def. cond. prob.)}$$

$$P(E \mid H) = \frac{P(H \wedge E)}{P(H)} \quad \text{(Def. cond. prob.)}$$

$$P(H \wedge E) = P(E \mid H)P(H)$$

QED: $$P(H \mid E) = \frac{P(E \mid H)P(H)}{P(E)}$$

Bayesian Categorization

- Determine category of x_k by determining for each y_i

$$P(Y = y_i \mid X = x_k) = \frac{P(Y = y_i)P(X = x_k \mid Y = y_i)}{P(X = x_k)}$$

- $P(X{=}x_k)$ can be determined since categories are complete and disjoint.

$$\sum_{i=1}^{m} P(Y = y_i \mid X = x_k) = \sum_{i=1}^{m} \frac{P(Y = y_i)P(X = x_k \mid Y = y_i)}{P(X = x_k)} = 1$$

$$P(X = x_k) = \sum_{i=1}^{m} P(Y = y_i)P(X = x_k \mid Y = y_i)$$

Bayesian Categorization (cont.)

- Need to know:
  - Priors: $P(Y=y_i)$
  - Conditionals: $P(X=x_k \mid Y=y_i)$
- $P(Y=y_i)$ are easily estimated from data.
  - If n_i of the examples in D are in y_i then $P(Y=y_i) = n_i / |D|$
- Too many possible instances (e.g. 2^n for binary features) to estimate all $P(X=x_k \mid Y=y_i)$.
- Still need to make some sort of independence assumptions about the features to make learning tractable.

Generative Probabilistic Models

- Assume a simple (usually unrealistic) probabilistic method by which the data was generated.
- For categorization, each category has a different parameterized generative model that characterizes that category.
- **Training**: Use the data for each category to estimate the parameters of the generative model for that category.
  - **Maximum Likelihood Estimation (MLE)**: Set parameters to maximize the probability that the model produced the given training data.
  - If M_λ denotes a model with parameter values λ and D_k is the training data for the *k*th class, find model parameters for class *k* (λ_k) that maximize the likelihood of D_k:

$$\lambda_k = \underset{\lambda}{\operatorname{argmax}}\, P(D_k \mid M_\lambda)$$

- **Testing**: Use Bayesian analysis to determine the category model that most likely generated a specific test instance.

Naïve Bayesian Categorization

- If we assume features of an instance are independent **given the category** (***conditionally independent***).

$$P(X \mid Y) = P(X_1, X_2, \cdots X_n \mid Y) = \prod_{i=1}^{n} P(X_i \mid Y)$$

- Therefore, we then only need to know P(X_i | Y) for each possible pair of a feature-value and a category.
- If Y and all X_i and binary, this requires specifying only 2n parameters:
  - P(X_i=true | Y=true) and P(X_i=true | Y=false) for each X_i
  - P(X_i=false | Y) = 1 – P(X_i=true | Y)
- Compared to specifying 2^n parameters without any independence assumptions.

Naïve Bayes Example

Probability	positive	negative
P(*Y*)	0.5	0.5
P(small | *Y*)	0.4	0.4
P(medium | *Y*)	0.1	0.2
P(large | *Y*)	0.5	0.4
P(red | *Y*)	0.9	0.3
P(blue | *Y*)	0.05	0.3
P(green | *Y*)	0.05	0.4
P(square | *Y*)	0.05	0.4
P(triangle | *Y*)	0.05	0.3
P(circle | *Y*)	0.9	0.3

Test Instance:
<medium ,red, circle>

Naïve Bayes Example

Probability	positive	negative
P(*Y*)	0.5	0.5
P(medium | *Y*)	0.1	0.2
P(red | *Y*)	0.9	0.3
P(circle | *Y*)	0.9	0.3

Test Instance:
<medium ,red, circle>

P(positive | *X*) = P(positive)*P(medium | positive)*P(red | positive)*P(circle | positive) / P(*X*)
0.5 * 0.1 * 0.9 * 0.9
= 0.0405 / P(*X*) = 0.0405 / 0.0495 = 0.8181

P(negative | *X*) = P(negative)*P(medium | negative)*P(red | negative)*P(circle | negative) / P(*X*
0.5 * 0.2 * 0.3 * 0.3
= 0.009 / P(*X*) = 0.009 / 0.0495 = 0.1818

P(positive | *X*) + P(negative | *X*) = 0.0405 / P(*X*) + 0.009 / P(*X*) = 1

P(*X*) = (0.0405 + 0.009) = 0.0495

Estimating Probabilities

- Normally, probabilities are estimated based on observed frequencies in the training data.
- If D contains n_k examples in category y_k, and n_{ijk} of these n_k examples have the jth value for feature X_i, x_{ij}, then:

$$P(X_i = x_{ij} \mid Y = y_k) = \frac{n_{ijk}}{n_k}$$

- However, estimating such probabilities from small training sets is error-prone.
- If due only to chance, a rare feature, X_i, is always false in the training data, $\forall y_k$:P(X_i=true | Y=y_k) = 0.
- If X_i=true then occurs in a test example, X, the result is that $\forall y_k$: P(X | Y=y_k) = 0 and $\forall y_k$: P(Y=y_k | X) = 0

Probability Estimation Example

Ex	Size	Color	Shape	Category
1	small	red	circle	positive
2	large	red	circle	positive
3	small	red	triangle	negative
4	large	blue	circle	negative

Probability	positive	negative
P(*Y*)	0.5	0.5
P(small | *Y*)	0.5	0.5
P(medium | *Y*)	0.0	0.0
P(large | *Y*)	0.5	0.5
P(red | *Y*)	1.0	0.5
P(blue | *Y*)	0.0	0.5
P(green | *Y*)	0.0	0.0
P(square | *Y*)	0.0	0.0
P(triangle | *Y*)	0.0	0.5
P(circle | *Y*)	1.0	0.5

Test Instance *X*:
<medium, red, circle>

P(positive | *X*) = 0.5 * 0.0 * 1.0 * 1.0 / P(X) = 0

P(negative | *X*) = 0.5 * 0.0 * 0.5 * 0.5 / P(X) = 0

Smoothing

- To account for estimation from small samples, probability estimates are adjusted or *smoothed*.
- Laplace smoothing using an *m*-estimate assumes that each feature is given a prior probability, *p*, that is assumed to have been previously observed in a "virtual" sample of size *m*.

$$P(X_i = x_{ij} \mid Y = y_k) = \frac{n_{ijk} + mp}{n_k + m}$$

- For binary features, *p* is simply assumed to be 0.5.

Laplace Smothing Example

- Assume training set contains 10 positive examples:
  - 4: small
  - 0: medium
  - 6: large
- Estimate parameters as follows (if *m*=1, *p*=1/3)
  - P(small | positive) = (4 + 1/3) / (10 + 1) = 0.394
  - P(medium | positive) = (0 + 1/3) / (10 + 1) = 0.03
  - P(large | positive) = (6 + 1/3) / (10 + 1) = 0.576
  - P(small or medium or large | positive) = 1.0

Continuous Attributes

- If X_i is a continuous feature rather than a discrete one, need another way to calculate P(X_i | Y).
- Assume that X_i has a Gaussian distribution whose mean and variance depends on Y.
- During training, for each combination of a continuous feature X_i and a class value for Y, y_k, estimate a mean, μ_{ik} , and standard deviation σ_{ik} based on the values of feature X_i in class y_k in the training data.
- During testing, estimate P(X_i | $Y=y_k$) for a given example, using the Gaussian distribution defined by μ_{ik} and σ_{ik} .

$$P(X_i \mid Y = y_k) = \frac{1}{\sigma_{ik}\sqrt{2\pi}} \exp\left(\frac{-(X_i - \mu_{ik})^2}{2\sigma_{ik}^{\;2}} \right)$$

Comments on Naïve Bayes

- Tends to work well despite strong assumption of conditional independence.
- Experiments show it to be quite competitive with other classification methods on standard UCI datasets.
- Although it does not produce accurate probability estimates when its independence assumptions are violated, it may still pick the correct maximum-probability class in many cases.
  - Able to learn conjunctive concepts in any case
- Does not perform any search of the hypothesis space. Directly constructs a hypothesis from parameter estimates that are easily calculated from the training data.
  - Strong bias
- Not guarantee consistency with training data.
- Typically handles noise well since it does not even focus on completely fitting the training data.