Linear Regression

* LSM: Select *h* such that *Error* is minimized

Assume

Because math, the *a* and *b* that minimize this error is

* But there are multiple features (multiple x values), so it’s not that simple.

Here,

and and

The that minimizes this error is

* Regularization
  + Regularization reduces overfitting
  + Add to the Error term
  + Small is better for smoother curves

Bayesian Learning

* Maximize , Maximum a posteriori (MAP)
* If we do not have any prior knowledge and every *h* in H is equally likely, we can drop . This is the Maximum likelihood estimate (MLE)
* Point estimation
  + Take the *ln* of the argmax function

,

Solve for *θ*

Naïve Bayes Classifier

* Same as Bayesian learning, except additional assumption that all features are independent.
* Don’t forget to multiply by as well when deciding how to classify an instance.
* Smoothing
  + When an instance has never been observed, this method breaks. Smooth the formula:

Where and

* + For every , add the *mp / m* in the calculation so that no values are 0 anywhere.
* This assumption requires less data
* If the values get too small, use lns

Decision Tree

* Learn an unknown Boolean function *f*
* Measure: information gain
* ID3 Algorithm
  + Select the attribute that can maximally reduce the *impurity* of the instances.
  + Entropy is highest when instances are 50/50

Where *A* is the attribute to split the data by, and *i* represents each value of the attribute.

* + Continue selecting the best remaining attribute until every leaf is *pure* (all remaining instances are the same classification)
* Hypothesis space
  + Every finite discrete function can be represented by a decision tree.
  + Each *n­*-feature Boolean function can be represented by a binary decision tree with *n* depth
* Inductive bias
  + ID3 algorithm produces only one tree, though there can be many trees consistent with the training data. ID3 prefers shorter trees (preference bias).
* Overfitting
  + Noise (errors in data), imbalanced distribution
  + Control this by pruning the tree.
    - Stop growing the tree when further classification is not statistically significant
    - Prune branches and test accuracy on test-data
* Numerical attributes
  + Check the midpoint of each instance (sorted) and check the impurity of using the midpoint as a threshold
  + Can have multiple thresholds per attribute

Concept Learning

* Learn an unknown Boolean function *f*
* If a dataset has *n* discrete features and each attribute has *k* discrete possibilities, then the full hypothesis space contains

hypotheses

* Measure hypothesis by its consistency
* Prior knowledge:
  + Each attribute can be:
    - A specified value
    - ? – Any value is acceptable
    - ∅ – No value is acceptable
  + A hypothesis is represented as
    - *<* val1, val2, val3, … *>*
    - For example, the hypothesis would classify all instances where *val1* is *sunny* and *val2* is *warm* as Yes. All other instances would be No.
  + The hypothesis means each instance is a Yes instance.
  + All hypothesis containing even one ∅ classifies all instances as No.
  + Full hypothesis space contains

hypotheses

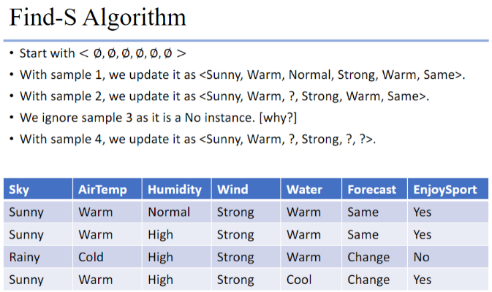
* + (more specific) is a subset of the hypothesis (more general).
  + The goal is to find one or more hypotheses and choose the most general one.
* Assumptions
  + The hypothesis space contains *the* hypothesis that describes the true concept
  + The training data has no error
* Inductive Bias
  + Restriction bias: assume the true target is a conjunction of constraints
  + No preference bias: output the whole version space

|  |  |
| --- | --- |
| Find-S | Candidate Elimination |
| Find a hypothesis consistent with training data. | Find a compact representation of all hypothesis consistent with training data. |
| Only consider the most specific one. | Consider all possible consistent hypothesis. |
| Ignore No instances. | Consider both Yes and No instances |

Sigmoid

* This is a node within a neural network
* The node performs the following calculation
* Input: a real vector
* Output: a real value (the float produced by the above function)
* Define the error:

|  |  |
| --- | --- |
| **Naïve Bayes** | **Logistic Regression** |
| Generative classifier | Discriminative classifier |
| Can generate new data; we know and | Cannot generate new data; we only know |
| Generally not a linear classifier | Always a linear classifier |
| Gaussian Naïve Bayes with class independent variance is representationally equivalent to logistic regression | |



If given a distribution

The MLE for is

Take the ln of both sides

Perceptron

* This is a node within a neural network
* This is basically just a weighted sum of all the inputs

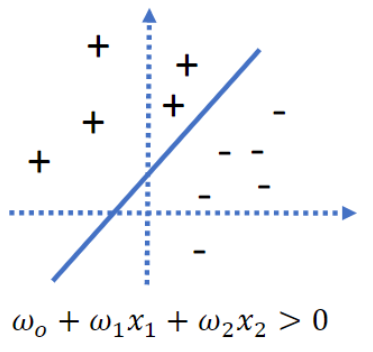
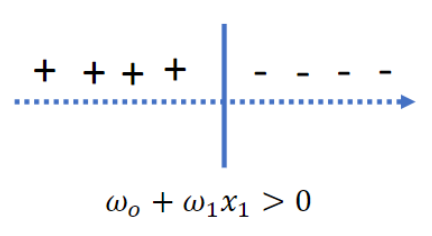
If

A value of 1 is returned and passed onto the next node

If

A value of -1 is returned and passed onto the next node

* Decision surface of Perceptron
  + Linearly separable



* Training a perceptron
  + Method 1: Perceptron training rule
    - *t*: target value
    - *o*: current prediction
    - : learning rate, small value
    - Continue updating until it converges
    - It will converge if the data is linearly separable and the learning rate is small enough. This is important because the training data may not be linearly separable, even if the underlying truth is linear
  + Method 2: LSM + Gradient Descent
    - Linear unit:
    - Error
    - Least squares method
    - Batch Mode: define the error for the whole training data, and update the parameters by minimizing the total error.
    - Incremental Mode: consider the error for each instance and update the parameters by minimizing the error for a single attribute. Then, move on to the next attribute and repeat.

Neural Networks

* A neural network is a series of interconnected units that map input values to an output.
* Input units: usually a real vector
* Processing units
  + Node containing some function (perceptron, sigmoid, etc.)
  + receive input from other units
  + output result to other units
* Edges
  + These are the branches that connect the nodes
  + They are weights (the parameters)
* Train a neural network
  + Define the error E
  + Calculate
  + Batch and incremental mode apply

Logistic Regression

* Classify an instance based on the higher probability
* This is still a linear classifier, as the classification is still ultimately determined by the same rule as a linear regression
* Learn using Bayesian learning

Take

For all

where is the learning rate

is the difference between the observed value and the predicted probability

* Regularization
  + Method 1: Add term and maximize
  + Method 2: Assume follows a Gaussian distribution with

\*\*\*cry\*\*\*

Notes

* If the data is not linearly separable, the gradient descent algorithm *can* still converge because the objective function is concave.
* If a decision tree classifier has 100% accuracy on the training data, this does *not* mean logistic regression will also have 100% accuracy (think linearly separable).
* A decision tree is *not* the smallest possible representation of the target function. Some Boolean functions may generate very large trees.
* If values are missing for regression, you can:
  + Ignore the instances with missing values
  + Take the most likely value for the missing values.
* A data set that is linearly separable is trained two ways, using *gradient descent* and *perceptron*. They will not have the same accuracy on the training and test data, even though they are linearly separable.
* Given infinite training data over *n* Boolean attributes, it is **true** that we can always learn the target concept using a decision tree but not using Naïve Bayes or Logistic Regression. NB and LR are linear classifiers, but the target concept may not be linear.

|  |  |  |
| --- | --- | --- |
| **Perceptron** | **Sigmoid** | **Linear** |
| Not differentiable | Differentiable | Differentiable |
| Output a classification (-1 or 1) | Output a bounded float/value (0,1) | Output a float/value (unbounded) |
|  |  |  |

Instance Based Learning

* Instance based learning is when step 1) acquire training data but do nothing and sept 2) once a new instance is received, create an approach customized to the new instance to classify/predict it.
* Parametric means a particular form is assumed. Non-parametric means the distribution is completely data-driven and few assumptions are made a priori
* K-nearest neighbor algorithm,
  + Define the similarity between instances as , where the distances can be Euclidean, Manhattan, etc.
  + Weight each instance equally
  + Weight more-similar instances higher
  + When using weights, all points can be considered, but this makes the method slower
* The scale of a variable affects the Euclidean distance. Not all points may be on the same scale, therefore, the data should be normalized by dividing all attribute values by the standard deviation
* If it is known that some instance are irrelevant, you can set their weights to zero. If it is *not* known which attributes are good, choose random small weights and learn the best parameters by updating the weights and minimizing error.
* More features ~ higher dimensionality ~ increased sparsity
  + To achieve the same density, the # of instances required is exponential as features are added

Clustering

In the K-means algorithm, the update step for the center is

Once the centers for all clusters have been updates, data points will be re-classified based on their distance to each center

Using batch gradient descent:

Mixture of K-gaussians (EM Algorithm)

E-step : Estimate by finding its expected value

M-step: Estimate mixture weights (

M-step (derivation): Calculate the new maximum likelihood hypothesis using estimated E

Support Vector Machine

The decision boundary is the perpendicular bisector of the SVMs. The SVMs are the points that are closest to the decision boundary.

Optimization

Given training data,find ω and b by solving

This gets turned into (step 1)

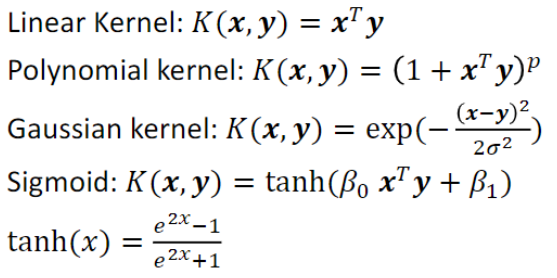
Subject to and

The minima is reached when

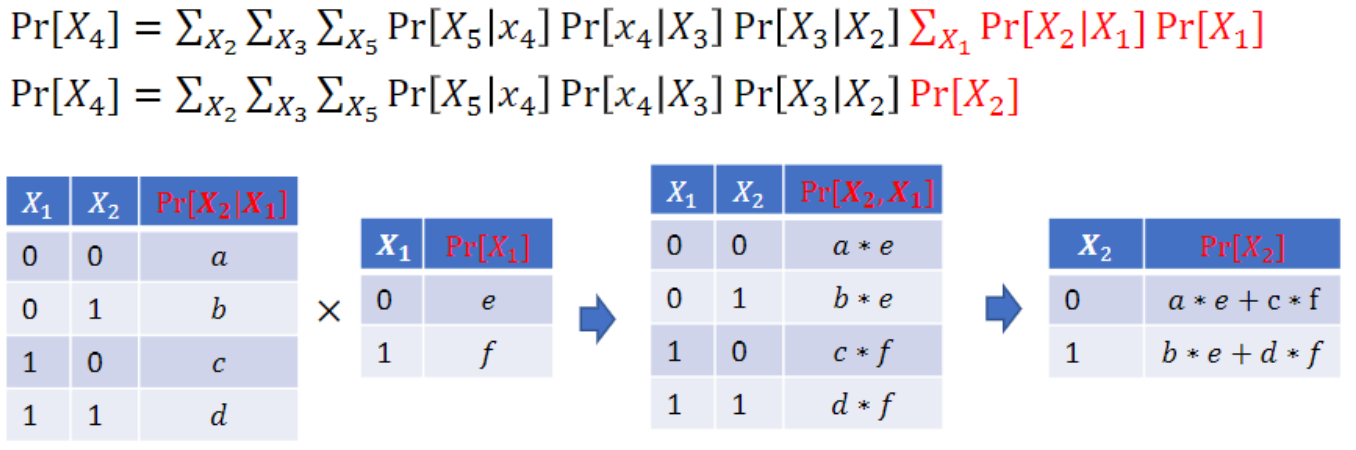
Step 2: Dual Problem

Subject to and

When data is not linearly separable, we can map it to a higher-dimensional space using a kernel. Every set of n-dimensional data is linearly separable after properly mapping to a dimensional space.



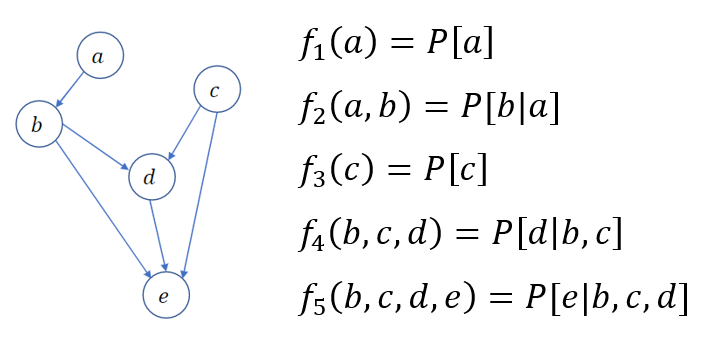
Graphical Models



When multiplying factors out using their table, do the first step. The second step shows *X1* being summed out and thus eliminated.

To convert a factor to a probability, normalize the column first so that all values add up to 1

Graphical Models



Independence (d separation)

* means “are *a* and *e* independent, given *b* and *c*)
* Plot the variables mentioned and all their ancestors
* “Moralize” by connecting all nodes with a common child (marry the parents)
* Make all edges undirected
* Delete all the given nodes
* If the two variables in question are still connected, they are *not* necessarily independent. If there is no path between them, they are independent.

Inferring P[d] along the order (*b, c, a, e*)

Start with:

Multipy and together to get 2Val(f\*)

Sum out *b* to get Val(f\*)

Multiply and together to get Val(f\*)

Sum out *c* to get Val(f\*)

Multiply and together to get Val(f\*)

Sum out *a* to get Val(f\*)

Sum out *e* to get Val(f\*)

Complexity:

Cut-Based Clustering

Find a clustering that minimizes

Graphical Models

With probability at least 95% output a hypothesis with error at most 0.15

A training sample must have at least *m* training instances to meet the requirements

PAC Learnability

For a certain target, can it be reliably learned from a reasonable number of randomly drawn training examples with a reasonable amount of computation.

* Small error
* No infinite training data
* No infinite runtime

-exhausted means that every hypothesis in the version space is good (meets a certain threshold).

To learn :

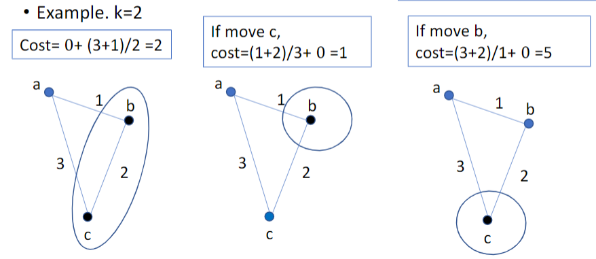
because there are 3 variables

Cross Validation and Model Selection

* Training-Validation-Test method
  + Partition the data into 3 subsets
  + Use training data to create tree/find parameters
  + Use validation data to tune the model
  + Use test data to determine the performance of the model
    - Test data not used anywhere in the training process
* Leave one out cross validation
  + For each point/instance in your database
    - Create a model based on the entire dataset excluding that single point
    - Test the model on the single point and note the error
  + Note the average error as performance
  + Training is long, but testing is quick
* K-fold Cross Validation
  + Partition data into *k* parts
  + For each subgroup k:
    - Create a model based on the entire data set excluding the subgroup
    - Test the model on the excluded subgroup and note the error
  + Choose the model with the smallest error
  + This is a compromise between TVT and LOOCV

|  |  |  |  |
| --- | --- | --- | --- |
|  | TTV | LOOCV | KFCV |
| # of trainings | 1 |  | *k* (3, 5, 10) |
| Training time | 67% |  |  |
| Testing size | 33% | 1 |  |

* No Free Lunch Theorem: if non prior knowledge is known, nothing is better than a random guess



Model Combining

* Bagging: take the average prediction of a set of individual models
* Bootstrap Aggregation (bagging): a bootstrap dataset is a dataset with *N* points that was created by randomly sampling *N* instances **with replacement**. Then do bagging with the bootstrap datasets. Note: you can use bagging even if you only have one classifier because the datasets are different
* A random forest is a bootstrap aggregation implemented in decision trees. However, for each bootstrap tree, also randomly select *p* attributes (so not all of them) to build the tree with. This will avoid having similar trees.
* If bagging does not work, try Boosting
  + Boosting is a technique that combines multiple ‘base’ classifiers (i.e. linear classifier, quadratic, sin, etc) for different regions of the dataset
  + Define the base classifiers to best tested, and do the following for each (in sequence)
    - Train the entire dataset using the base classifier. Then go back and check accuracy. For any instance that is incorrectly classified, assign a higher weight to it based on the error
  + For each instance, determine how it is classified by picking the base classifier with the smallest weight associated with it.

Step 1: Minimize the weighted error function

Step 2: Set and

Step 3: Update weights by

Prediction:

Notes

* Bias measures the accuracy of the algorithm—high bias means it is a poor match
* Variance measures the precision of the match—high variance means a weak match