Masters in Computer Science : Machine Learning Concepts

\*\* concept : function or mapping from objects to membership

\*\* instance : a single set of attribute-value pairs (input space of Concept )

\*\* hypothesis : helps to predict target concept -> actual answer

\*\*\* apply candidate concepts to testing set (should include lots of examples )

\*\*\* inductive learning \*\*\* choose a hypothesis from given set of examples

What’s the Inductive Bias for Classification Function : How do we find a general rule from example ?

Occum’s Razor : Prefer simplest hypothesis that fits data

Generalization is the whole point in Machine Learning

Restriction Bias : consider only those hypothesis which can be represented by chosen algorithm

**Supervised Learning** : We take examples of inputs and outputs and based on that we consider a new input and predict the corresponding output.

**Eager Learners** : Decision trees, regression, neural networks, SVMs, Bayes nets

🡪 find a function that best fits training data i.e. spend time to learn from data , when new inputs are received the input features are fed into the function 🡪 here we consider global scale inputs and avoid local sensitivities

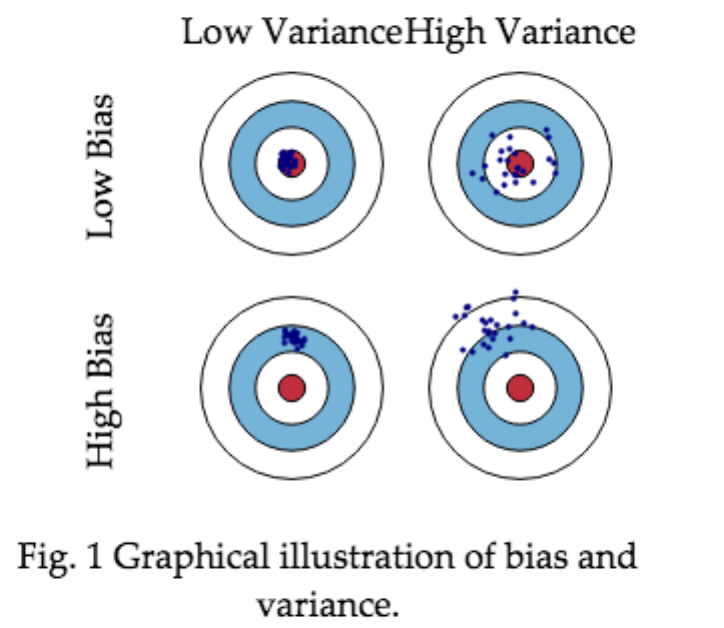
**Lazy Learners** : lazy learners do not compute a function to fit the training data before new data is received 🡪 so we save signifactnt time upfront 🡪 new instances are compared to the training data to make a classification / regression decision !!! 🡪 considers local-scale estimation

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| ML Algo | Type | Preference Bias | Learning Function | Performance | Enhancements | Usage |
| Bayesian  (**Eager Learner**) | Classification | Prior Domain Knowledge  ~ Pr (h) prior prob for each candidate h  ~ Pr(D) – prob dist. Over observed data for each h  Occum’s Razor ?  - select h with min length  \*\* at least one maximally probable hypothesis 🡪 hmap -> argmaxP(h|D)  -> argmaxP(D|h)  (for uniform prior) | Posterior Prob  P(h|D) = P(D|h).P(h) / P(D)  Key assumption : every hi equally probably a priori => p(hi) = p(hj)  \* Noise Free Uniformly Dist. Hypothesis in V(s) \*  P(h) = 1 / |H| ,  P(D|h) = { 1 if di = h(x) , 0 otherwise }  P(h|D) = 1 / |V(s)|  \* Noisy Data\*  di = k.xi  hmc = argmax P (h|D)  = argmax P(D|h)  = argmax π P(di|h)  \* di = f(xi) + e  ln (hmc) = argmin Sum (di – hi(x))2  \* Vmap = argmaxv Sumh P(v|h).P(h|D) | **Cons :**  \* significant computational cost to find Bayes optimal hypothesis  \* sometimes huge no of hypothesis need to be surveyed | Pros : No need to be aware of given hypothesis | \* Use Bayesian Learning to represent misclassification , pruning , fitting errors  \* spam  / | \  Lottery Bank College  P(spam | lottery , not bank , not college) = p(vi). Πi P (ai | v) |
| Decision Tree :  (**Eager Learner**)  ID3 , C4.5  🡪 approximate discrete values functions  🡪 disjunction of conjunction of constraints on attr values | Classification  : for discrete input data  : for cont. input data (consider Range selection as condition - >20% ) | Occum’s Razor ?  : shorter tree  Other Biases :  : prefer attributes with many possible values  : prefer trees that places high info gain attrs close to root (attr with best answers NOT best splits) | Info Gain (S,A) = Entropy(S) – Sumv (|Sv| / |S|)\*Entropy(Sv)  \*\* wtd sum of entropies of partitions  \* Entropy(s) =  -Sum(Pv log(Pv)) | Usual problem of Dtree : for N variables 🡪 2N combinations of rows 🡪  (2)2-to-the-power-N outputs  \*\* so instead of iterating on all rows , first work upon only the attributes which have highest info gain.    \*\* handles noise , handles missing values  =============  Scope of improvement :  Decision trees, however, **often achieve lower generalization accuracy**, compared to other learning methods, such as support vector machines and neural networks. One common way to improve their accuracy is boosting | pros : computes best attribute in one move  cons :  \* does not look ahead or behind ( this problem is solved by Hill-Climbing …)  \* tends to overfit as it looks into many diferent combinations of features  \* logistic regression avoids overfitting more elegantly  \*\* Overfitting soln for DTree :  >> stop growing tree before it grows too large  >> prune after certain threshold  \* consider interdependency betn attributes P(Y=y | X=x)  \* consider GainRatio , SplitInfo | - restaurant selection decision based on cost, menu , appetite, weather, and other features. |
| Decision Tree : Regression | Classification  : for cont. output data | Lazy Distance-based learning func :  For each training sample sl -> Sl  Dl = dist(sl, Sl)=root-sum-sqr(diff)  Wj = dmax – dj | Advantages of decision trees include:  ●  computational scalability  ●  handling of messy data ­ missing values, various feature types  ● ability to deal with irrelevant features ­ the algorithm selects  “relevant” features first, and generally ignores irrelevant features.  ● If the decision tree is short, it is easy for a human to interpret it:  decision trees do not produce a black box model. |
| Linear Regression :  (**Eager Learner**)  Model a linear relationship between a dependent variable (y) and independent variables (x1,x2..) | Classification :  Scalar input , Cont. output  Vector input, Cont. outputp  \*\* Vector Input -> combinations of multiple features into a single feature | Regress to mean  Gradient :  \* for one variable derivative is slope of tangent line  \* for several variables, gradient is the direction of the fastest increase of function | y^ = θ0 + θ1x1 + θ2x2  yi = observed value  minimize the Sum of Squared Error :  ½ Sum (y^-yi)2  θ1 = θ0 - α∇J(θ)  θ1 ->next pos  θ0 ->current pos  α is the learning rate so that function takes small step towards the direction opposite to that of ∇J (direction of fastest increase) |  | Cons :  Function should be differentiable  Caution :  Learning rate must not be very small or very large |  |
| Polynomial Regression |  |  |  |  |  |  |
| Multi-Layer Perceptron  (**Eager Learner**) | Classification | prefer small random values for the initial weights | Perceptron is a linear function that offers a hyperplane in n dimensions, perpendicular to the vector *w* = (*w*1, *w*2, . . . , *wn*) . The perceptron classifies things on one side of the hyperplane as positive and things on the other side as negative.  Best function to update weights |  | Addition of hidden layers help map continuous functions (change in input changes output very smoothly)  Multiply weights only if we don’t get better errors |  |
| K Nearest Neighbors | Classification | Preference Bias :  Why consider KNN over other ?  \* near points are similar to one another (locality)  \* smoothly changing behavior from one neighborhood to another neighborhood.  \* so we can choose best distance function | Choose best distance function | Performance :  Problem : curse of dimensionality :  … as the number of features grow, the amount of data required for accurate generalization grows exponentially .  Reducing weights will help curb the effect of dimensionality.  When *k* is small, models have high bias, fitting on a strongly local level. Larger *k* creates models with lower bias but higher variance | Enhancements : |  |
| K Nearest Neighbors | Regression  LWR (locally weighted regression) …. |  | It combines the traditional regression with instance­based learning’s sensitivity to training items with high similarity to the test point | -- reduce the pull effect of far-away points through Kernels  -- the squared deviations are weighted by a kernel function that decreases with distance, such that for a new test instance, a regression function is found for that specific point that emphasizes fitting close­by points and ignoring the pull of far­away points… |  |  |
| Ensemble Learning | Classification | Preference Bias :  - Individual rule (result of learning over a subset of data) does not provide answer but when combined , the complex rule works well . | The Key Algo :  Pr**D**[h(x) <> c(x)]  \*\* boost up the distribution ….  h1 h2 h3  x1 +1 -1 +1  x2 -1 -1 +1  x3 +1 -1 +1  \*\* find hypothesis at each time-step Ht with small error , (Weak Classifier) constantly creating new distributions … (Boosting)  \*\* Final Hypothesis : sgn (sign) function of the weighted sum of all of the rules. | Performance :  Why Boosting does so well ?  >> if there are some samples which do not provide good result, then boosting can re-rate the samples so that some of ‘past underperformers’ become more important.  >>  Use Grad Boost to handle noisy data in DTree :  <https://en.wikipedia.org/wiki/Gradient_boosting>  >> Boosting does overfit if Weak Learners uses NN with many layers of nodes | Advantages:  ●  Computationally efficient.  ●  No difficult parameters to set.  ●  Versatile ­ a wide range of base learners can be used with  AdaBoost.Caveats:  ●  Algorithm seems susceptible to uniform noise.  ●  Weak learner should not be too complex ­ to avoid overfitting.  ●  There needs to be enough data so that the weak learning  requirement is satisfied ­ the base learner should perform consistently better than random guessing, with generalization error < 0.5 for binary classification problems. | email :  body -> manly +  from -> spouse -  image -> +  only urls -> +  find which Wiki pages can recommended for extended period of time (feature set a combination of binary , text , nemerics)  Ref : <http://statweb.stanford.edu/~tibs/ElemStatLearn/>  <http://media.nips.cc/Conferences/2007/Tutorials/Slides/schapire-NIPS-07-tutorial.pdf> |
| SVM  The classifier is greater than or equal to 1 for the positive examples and less than or equal to -1 for the negative examples ….….  …… difference between the vector x1 and the vector x2 projected | Classification |  | >> finding the line of least commitment in the linear separable set of data, is the basis behind support vector machines  >> a line that leaves as much space as possible from the boundaries.  yi (wTxj + b) >= 1 w – params of hyperplane , maximize the equation … | >> : similar to KNN , but here instead of being completely lazy , spend upfront efforts to do complicated quadratic programs 🡪 to consider required points .  >> For classification tasks involving more than two groups, a common strategy is to use multiple binary classifiers to decide on a single­best class for new instances |  | Mostly binary classification (linear and non-linear) |

Its very important to calculate ‘Bias Errors’ and ‘Variance Errors’ while comparing various algorithms.

Error due to Bias => when a prediction model is built multiple times then Bias Error is the difference between ‘Expected Prediction value’ and Correct value. Bias provides a deviation of **prediction ranges** from real values .

Example of low bias ==> tendency of mean of all the sample points to converge towards mean of real values



Error due to Variance => how much the predictions for a **given point** vary between different implementations of the model.

Example of high variability ==> sample points tend to be dispersed away from each other.

Reference : <http://scott.fortmann-roe.com/docs/BiasVariance.html>

**Ensemble Learning (Boosting)**

Important difference of **Ensemble Learners** from other types of **Learners** :

-- NN already knows the Network and tries to learn the weights

-- DTree gradually builds the rules

Ensemble Learner 🡪 finds the best combination of rules .

1. Initialize the importance weights *wi* = 1/*N* for all training examples *i*. 2. For m = 1 to M:

a) Fit a classifier *Gm*(*x*) to the training data **using the weights** *wi*.

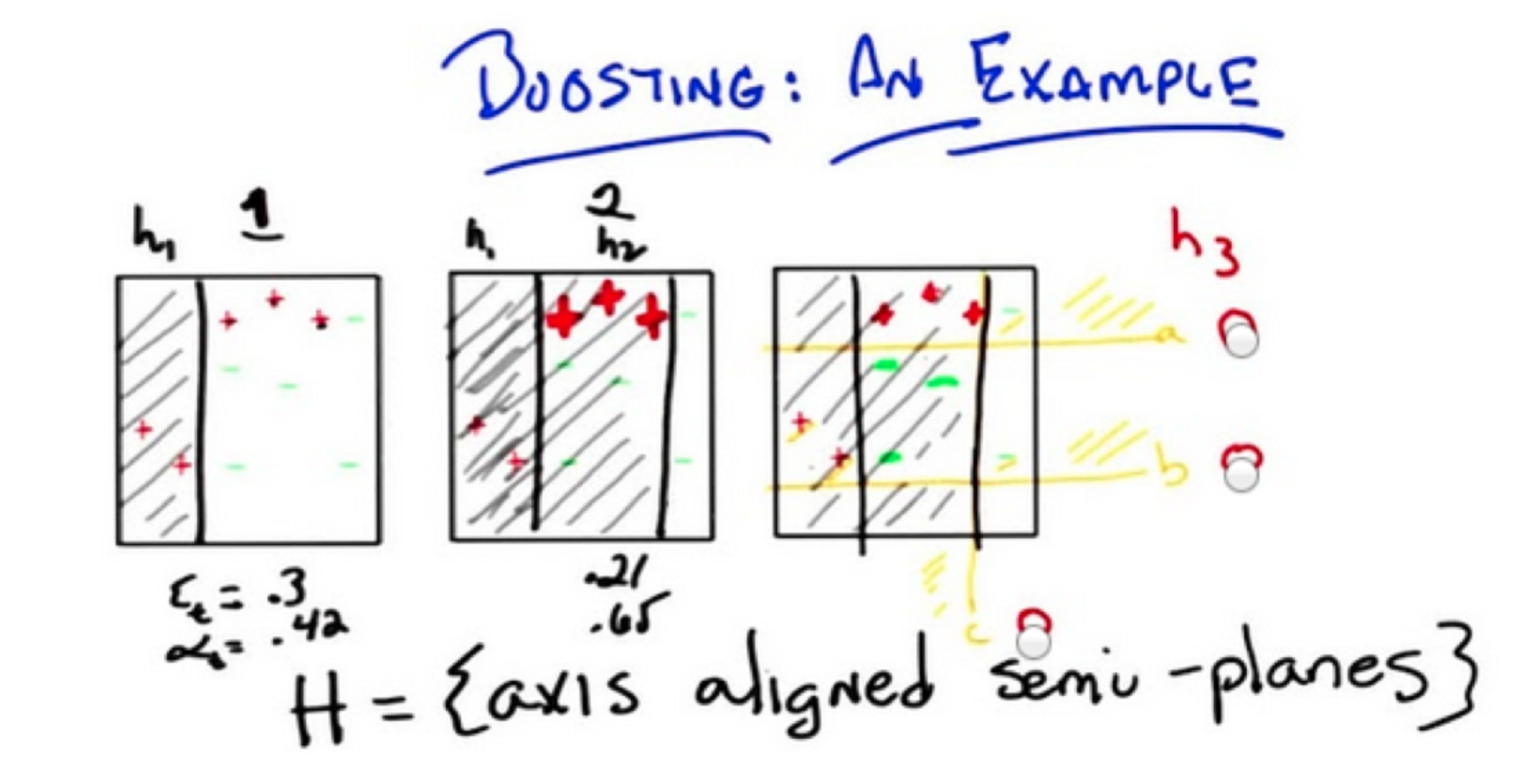
b) Compute the error: *errm* =  ∑ *wi I*(*yi* =/ *Gm*(*xi*)) / ∑ *wi*

c) Compute α*m* = *log*((1 − *errm*)/*errm*)

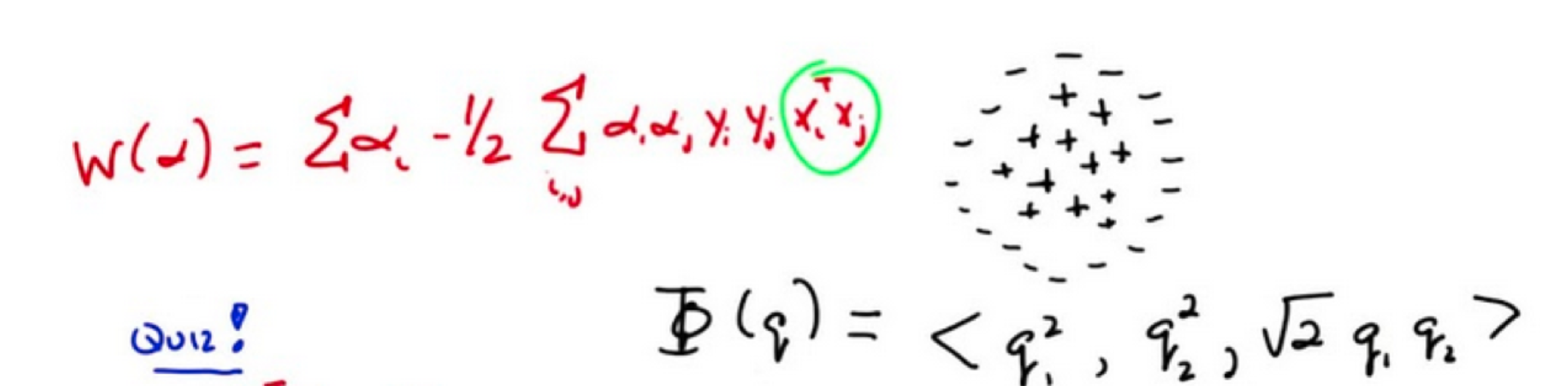
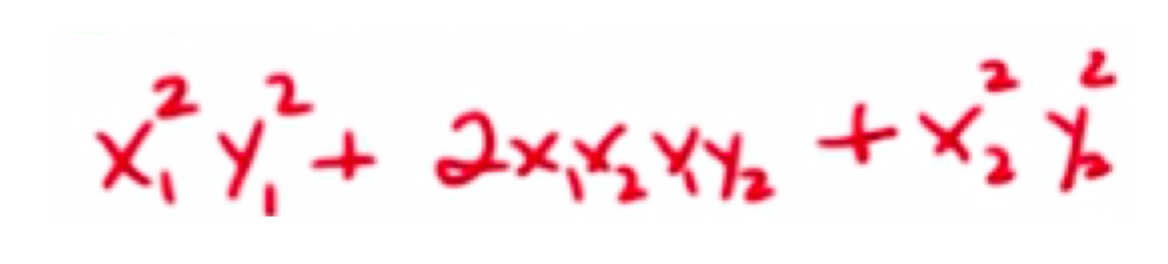
d) Update weights: *wi* ← *wi* . *exp*[α*m*. *I*(*yi* =/ *Gm*(*xi*))] for *i* = 1, 2, ... *N*

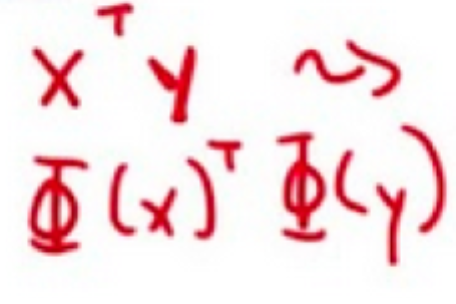
3. Return *G*(*x*) = *sign*[ ∑ α*mGm*(*x*)].

We can see that for error < 0.5, the α*m* parameter is positive



**SVM**

 >>> 

Here instead of Polynomial Regression we consider Polynomial Kernel 🡪 kernel represents domain knowledge

…. projecting into some higher dimensional space

For data that is separable, but not linearly, we can use a kernel function to capture a nonlinear dividing curve. The kernel function should capture some aspect of similarity in our data.