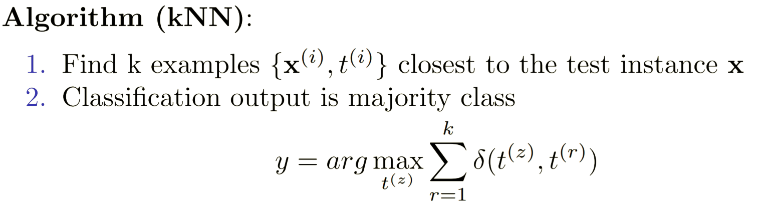
**kNN: CSC311 Material Summary. Good Luck, Have Fun!**



**For small k:** overfit. **For large k:** underfit.

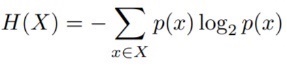
**Curse of dimensionality:** high most pts are far away & approx. Same distance (distance variance is very low)

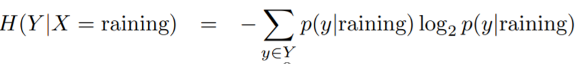
**Normalizations:** nearest neighbors can be sensitive to the ranges of different features.

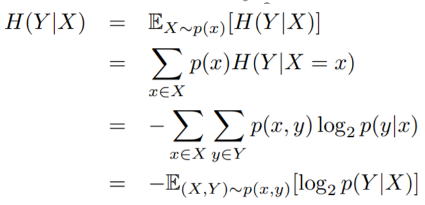
Without normalization, the performance of kNN will be reduced.

**Decision Tree:**

**Entropy:**





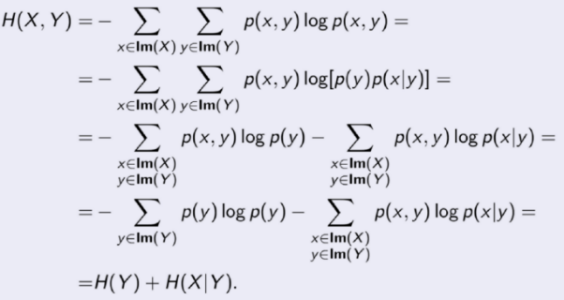


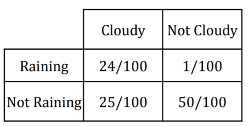
**Chain rule:**

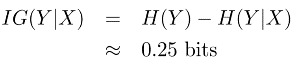
If X and Y are independent, then X doesn’t tell us anything about Y:

H(Y|X)=H(Y); But Y tell us everything about Y: H(Y|Y)=0.

By knowing X, we can only decrease uncertainty about Y: H(Y|X) <= H(Y)



Example:

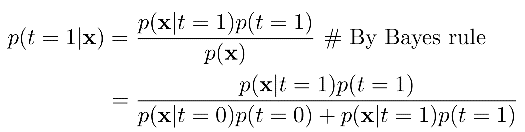
How much information about cloudiness do we get by discovering whether it is raining? 

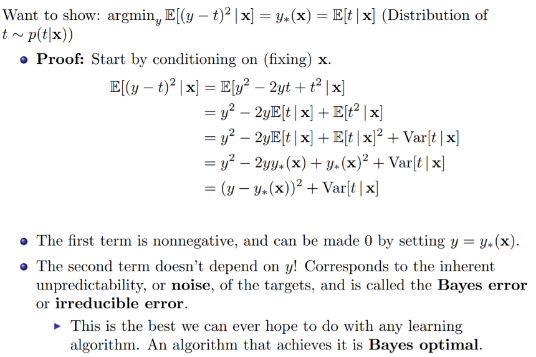
So

If X is completely uninformative about Y: **IG(Y|X) = 0**

If X is completely informative about Y: **IG(Y|X) =H(Y)** (i.e. in this case, H(Y|X) = 0)

**Probability rules:**

**Bayes:**

Correlation ,

so Var(X) = E[X^2]-(E[X])^2

Var(aX+b)=a^2Var[X]

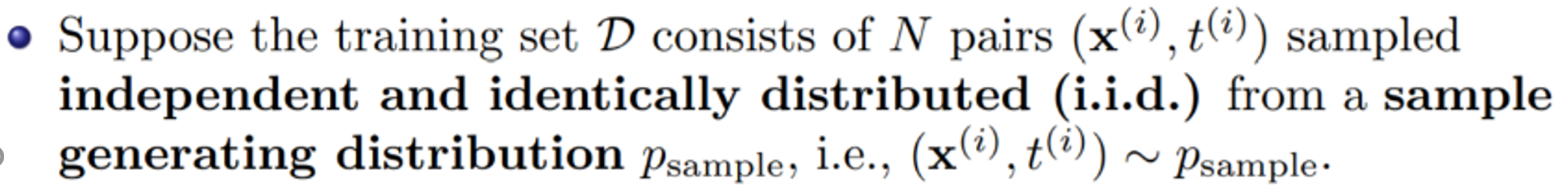
**Decision Tree Construction:** Start with empty decision tree and complete training set. – Split on the most informative attribute, partitioning dataset. – Recurse on sub partitions. **Decision Tree miscellany:** -Exponentially less data at lower levels – Too big of a tree can overfit the data -**Greedy algorithm don’t necessarily yield the global optimum** – Mistakes at top-level propagate down tree.

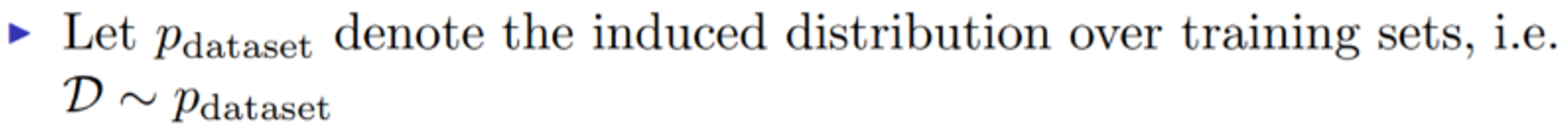
**Decision Tree’s advantage over knn: 1.** Good with discrete attributes **2.** Easily deals with missing values **3.** Robust to scale of inputs, only depends on ordering (kNN has curse of dimensionality, but D.T. doesn’t). **4.** Fast at test time **5.** More interpretable.

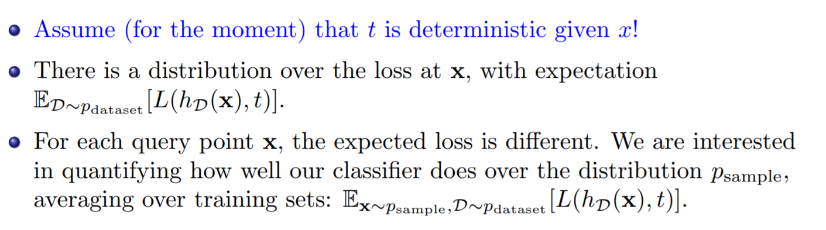
**kNN’s advantage over decision tree:** **1.** Able to handle attributes/features that interact in complex ways (e.g. pixels)

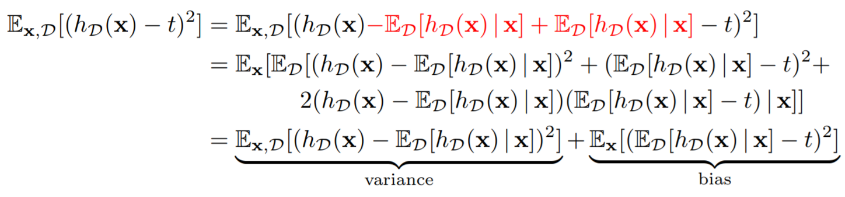
**2.** Can incorporate interesting distance measures

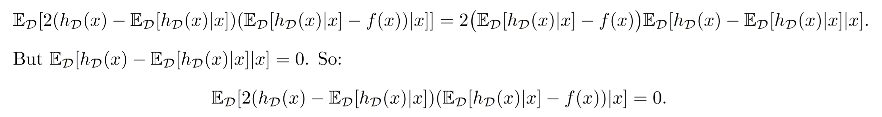
**Bias-Variance Decomposition:**



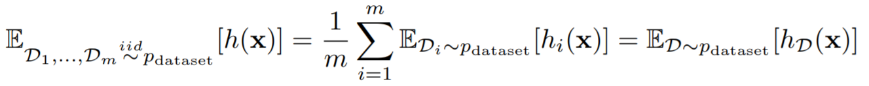




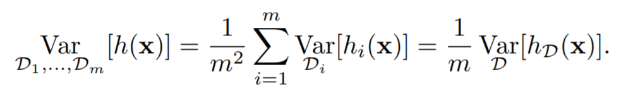




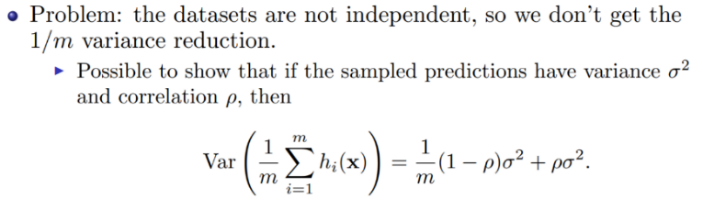
**Bagging:** Suppose we could somehow sample independent **training sets** from . We could then learn a predictor based on each one, and take the average of all the predictors.

**Bias: Unchanged** 

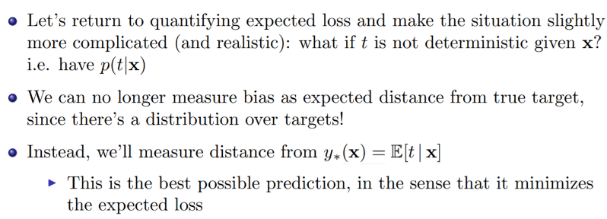
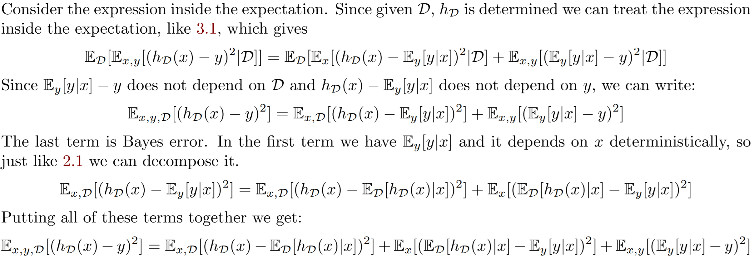
**Variance: Reduced**



**Problem of bagging:** In reality we don’t have that can randomly generate datas. Solution: Take a single dataset D with **n** examples. Then generate m new datasets, each by sampling **n** training examples from D, with replacement.



|  |  |
| --- | --- |
|  | Therefore, we use **random forest. (bagged decision trees, with when choosing each node, choose a rand set of d input features.** It reduces the variance by **reducing the correlation between the trees.** (i.e. between each model) |

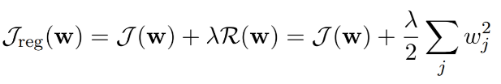
**Bayes Optimality**

**Linear Regression:** Optimal weights:

**Polynomial curve fitting:** Fit the data using a degree-M polynomial function of the form

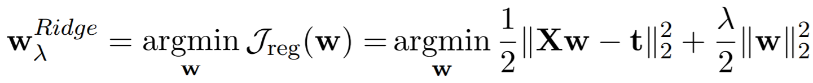
This is called **feature mapping:** where . **M to small:** underfit. **M too large:** overfit

**Regularization:** Why regularization? Because we want to reduce overfitting on complex model. Why complex model? Data might support

complex models!

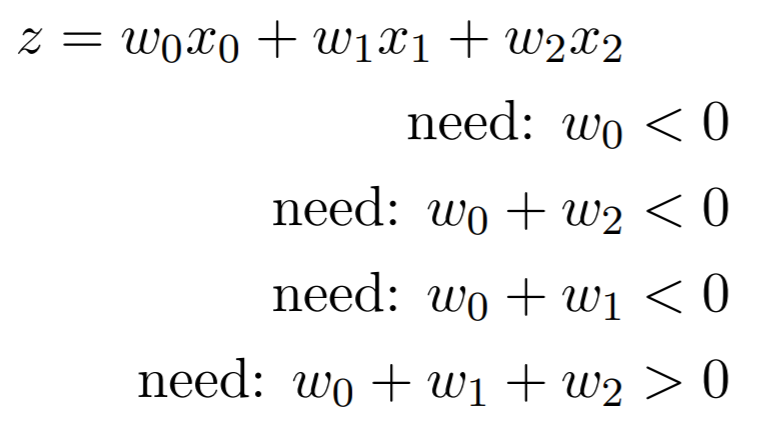
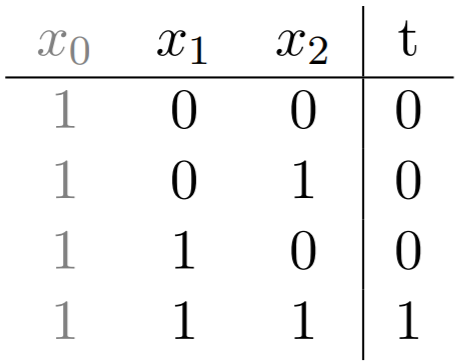
penalizes weight value here. If you fit training data poorly, J is large. If your optimal weights have high values, R is large.

For least squares problem, we have

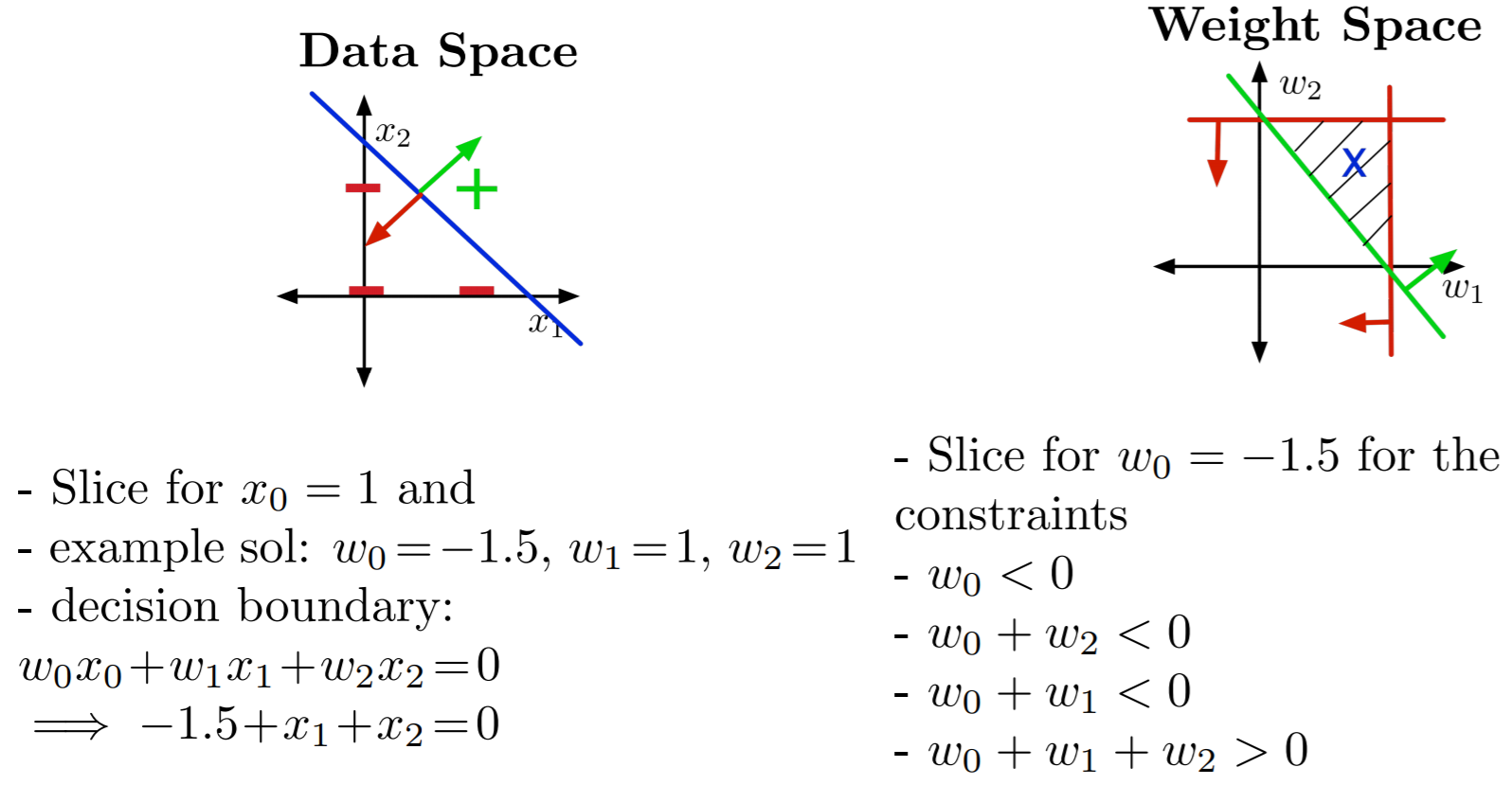


**Logistic Regression:**

Binary classification using linear model:









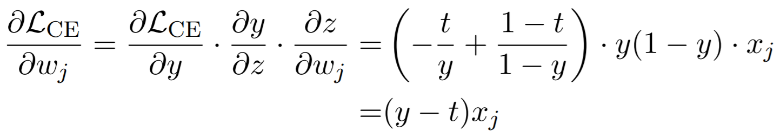
**Logistic activation function: How do we represent prediction?**

Logistic function is called **sigmoid**.

.

**Loss function for logistic regression:**

**Gradient of logistic loss**:



Problem: if t = 1 and I am really confident that it’s a negative example. i.e. z << 0. Then log(y) might be numerically 0.

Solution:

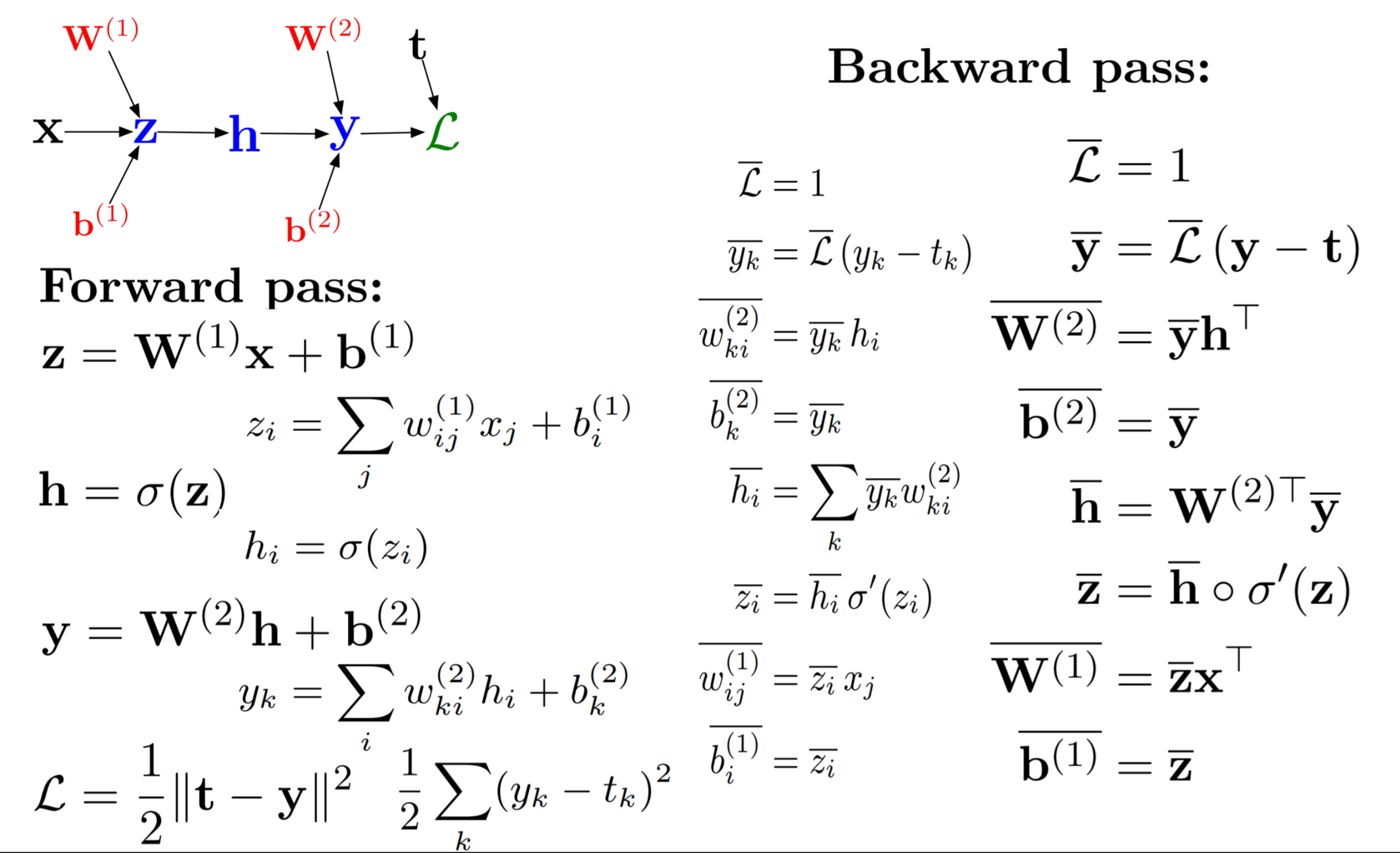
**Neural Networks:**

**Multiclass classification:** Targets form a discrete set {1,...,K}, K classes. Targets are represented as **1-hot** vectors. t=[0,0,..,1,0,..,0]

Prediction for the probability of each class

**CE for multiclass classification:**

**NN’s diagram, in both scalar and vector form:**



Computation cost: SGD < mSGD < GD

Convergence rate: SGD < mSGD < GD

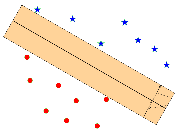
**GD Rules**: let be gradient. Then

GD:

SGD: Choose ,

Mini-batch GD: Choose a subset

**SGD’s mathematical justification:** if you sample a training example **uniformly at random**, the stochastic gradient is an **unbiased estimate** of the batch gradient.

**SVM:**

Goal: Minimize the hinge loss that leads to an optimal separating hyperplane.

What is the optimal separating hyperplane? A **hyperplane** is described by points s.t. .

We want to find the hyperplane that minimize the training loss:

(minimizing the hinge loss! is the hinge loss). Often add in the end (regularizer). **Decision boundary** is linear. Can only do binary classification.

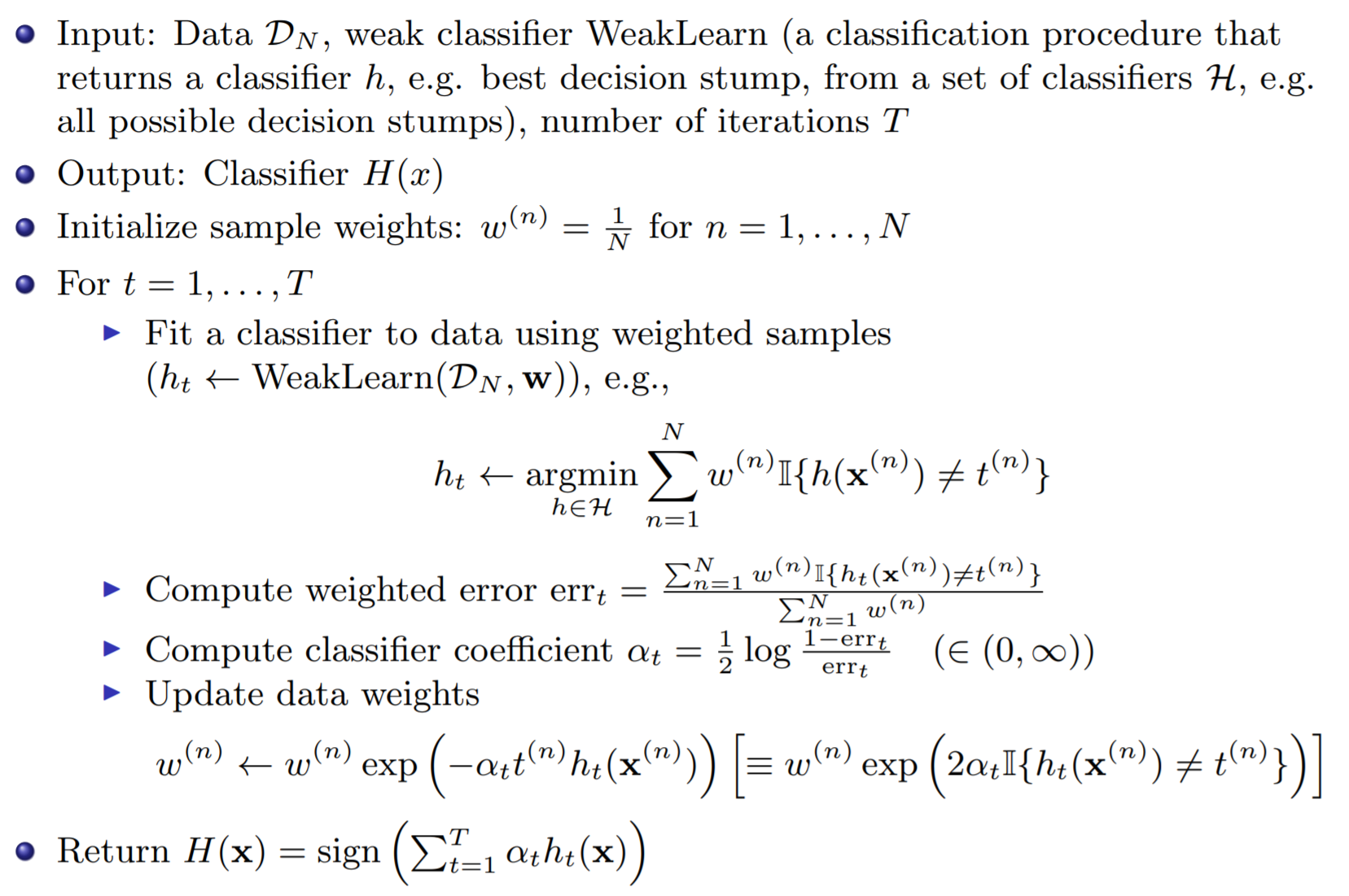
To find **w** and b: Find all the three points that are support vectors. Then, substitute their coordinates to **.** The right side equals whether it is a positive example (1) or negative example (-1). E.g. 1, , , Then solve for this system of linear equation! You will get **w** and b.

**What is the benefit of the hinge loss compared to 0-1 loss?** Minimum of a function will be at its critical points. But the gradient of the 0-1 loss is zero everywhere it’s defines. This means that changing the weights by a very small amount probably has no effect on the loss. We can use the 0-1 loss function, but minimizing this loss is computationally difficult, and it cannot distinguish different hypotheses that achieve the same accuracy. The hinge loss is a relaxation of the 0-1 loss, it can be minimized using gradient descent.

**Claim: hinge loss = 0 iff all points are correctly classified.**

**Boosting:**

AdaBoost Algorithm:



Weak classifiers which get lower weighted error get more weight in the final classifier. (Since low => high, so when computing H(x) it gets more weight)

Also, when updating weights:

**If ,**  **high** so misclassified examples get more attention

**If ,**  **low** so misclassified examples get less attention

**(i.e. stronger classifiers misclassified => add more weight!)**

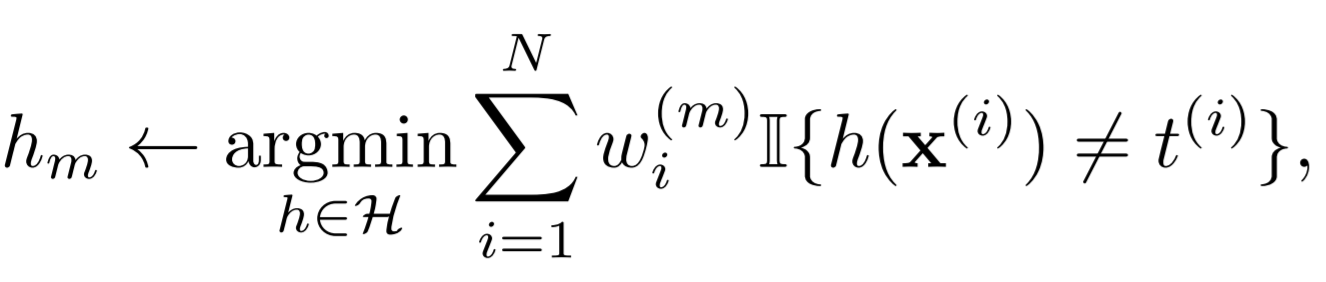
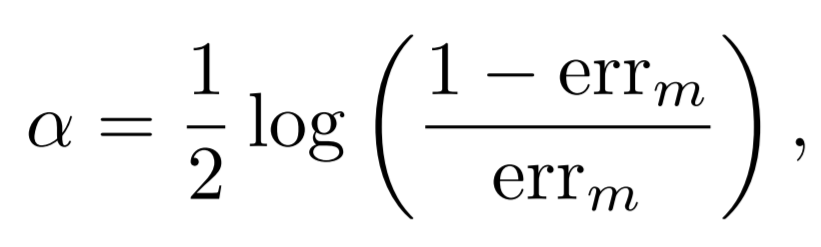
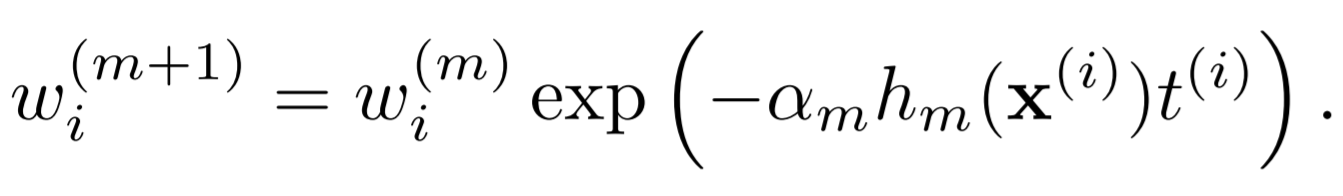
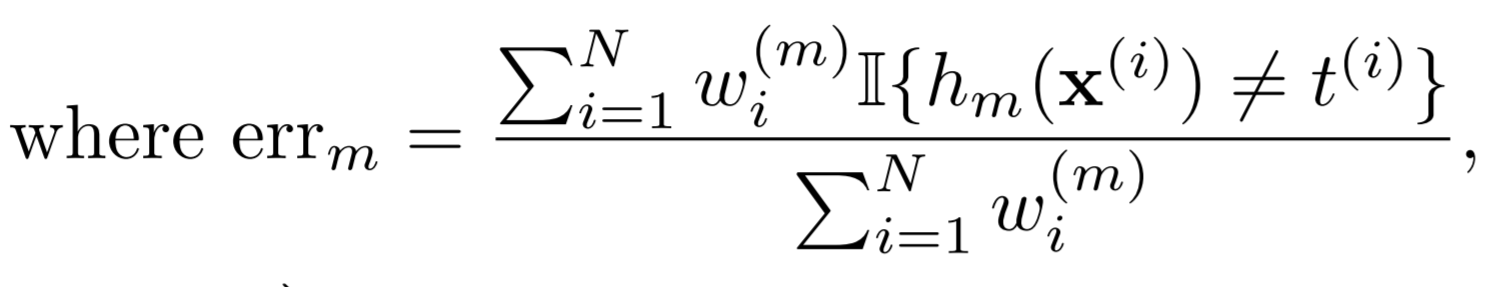
AdaBoost Minimizes the training error.

**AdaBoost penalizes more regarding error compared to logistic regression.**

**Theorem:** Assume that at each iteration of AdaBoost the WeakLearn returns a hypothesis with error for all t = 1,..,T with The training error of the output hypothesis is at most

. This is under the simplifying assumption that each weak learner is -better than a random predictor. It is called a geometric convergence. -> Fast! **AdaBoost is an additive model.**

**Q: How does Adaboost reduce bias?** By making each classifier focus on previous mistakes **Q: Sometimes the test error decreases even after the training error is zero. How does that happen?** The confidence of the classifier is not just the training error. After the training error is zero, the confidence continues to increase as each example goes further away from the decision boundary. The farther an example is, the more confident the model is in its label, or another way of thinking about it, the decision is more robust to noise in the input features.

**Additive models:** We obtain the additive model with   

**Bagging vs Boosting:**

**Bagging:** Train classifiers independently on random subsamples of training data. 1. reduces variance 2. bias is not changed much 3. parallel 4. want to minimize correlation between ensemble elements **Boosting:** Train classifier sequentially, each time focusing on training data points that were previously misclassified. 1. reduces bias 2. increases variance 3.sequential 4. high dependency between ensemble elements

**Generative model vs Discriminative model:**

**Discriminative approach:** estimate parameters of decision boundary/class separator directly from labeled examples. **1.** Tries to solve: How do I separate the classes? **2.** Learn p(t|**x**) directly (logistic regression models) **3.** Learn mappings from inputs to classes (linear/logistic regression, decision tress etc)

**Generative approach:** model the distribution of inputs characteristic of the class (Bayes classifier) **1.** Tries to solve: What does each class “look” like? **2.** Build a model of  **3.** Apply Bayes rule i.e. p(t, **x**) = …

**Key difference:** Is there a distributional assumption over inputs?

**Bayes Classifier:** Given features we want to compute class probabilities using Bayes Rule:

More formally:

To compute we need: and

**Naïve Bayes** assumption: Naïve Bayes assumes that the world features are **conditionally independent** given the class .

The log-likelihood can be decomposed into:

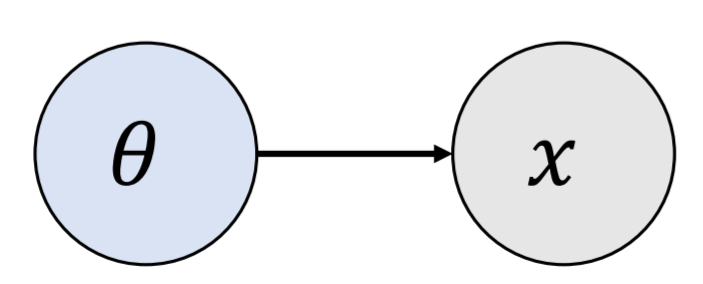
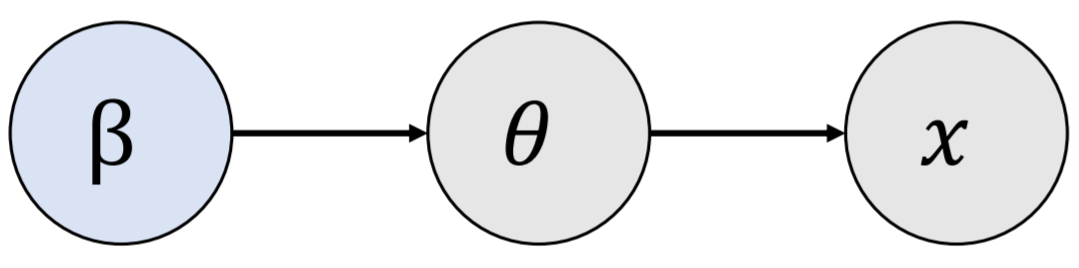
**To do learning:** (during training) Find MLE for and (Since good values of should assign high probability to the observed data.)

For the prior we maximize: . Then

For the class likelihood we maximize: Then

**To do prediction:** (during test time)We predict the category by performing inference in the model. Apply **Bayes’ Rule:**

**MLE’s disadvantage:** Not good with sparse data! e.g., What if you flip the coin **only twice** (so data is sparse) and get H both times? Then , i.e. the probability of getting head is **1**, according the model’s belief! Therefore, it assigned getting tail’s outcome probability 0. : ( Solution: Add prior distribution to theta!

(MLE) vs (MAP)

**MAP Estimation:**

**Gaussian Discriminative Analysis (GDA)**:

Assumes that is distributed according to multivariate Gaussian distribution.

Where denotes the determinant of the matrix, and is dimension of **x**. Each class has a mean vector and a covariance matrix

Use MLE to find empirical mean and empirical covariance matrix: . has parameters – could be hard to estimate

To find log-likelihood of GDA: Assume data **x** are drawn from gaussian, with and . We calculate log-likelihood , then take derivative w.r.t and , evaluate at zero to find the MLE   and  .

**Decision boundary of GDA: usually quadratic.** It is based on class posterior.

GDA makes decisions by comparing class probabilities.

Now, suppose I have two classes, and . Then the decision boundary equals the follow: decision boundary is , which is equivalent to

Quadratic function in **x**  **quadratic decision boundary!**

What is What if  **-> can be derived. Easy.**

**Simplifying GDA model:** What if **x** is high-dimensional?

For Gaussian Bayes Classifier, if input **x** is high-dimensional, then covariance matrix has many parameters . We can save some parameters by using a shared covariance for the classes, i.e. . **If so, we have a linear decision boundary!**

**Comparing Naïve Bayes vs GDA:**

Both Naïve Bayes and GDA are generative models. But:

|  |  |
| --- | --- |
| Naïve Bayes | GDA |
| Works on a set of discrete inputs  Assume conditional independence given class, so p(x1,x2|c)=p(x1|c)p(x2|c). (i.e. Covariance is **0**!)  Objective is to figure out, for each and each c, the that maximizes the likelihood of the observed data | Works on continuous inputs  Assume the underlying data is generated from Gaussian distribution. This let us specify covariance through covariance matrices, instead of being 0.  Also, see the following sentence: |

GDA’s objective is then to figure out the Gaussian parameters (mean and covariance matrix) that provide the best explanation (highest likelihood) for the observed data.

**Comparing GDA vs** **Logistic Regression:** GDA makes stronger modeling assumption: assumes class-conditional data is multivariate Gaussian. If this is true, GDA is asymptotically efficient. But LR is more robust, less sensitive to incorrect modeling assumptions. When the class-conditional distributions are non-Gaussian (true almost always), LR usually beats GDA. Also, GDA has quadratic (when not shared covariance), LR has linear decision boundary.

**PCA:**

PCA is a linear model. Goal: Dimensionality reduction -> saves computation!

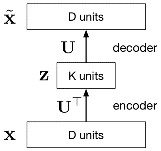
Need to choose matrix  **with orthonormal columns.**

Two criteria: **1.** Minimize the **reconstruction error:** Find vectors in a subspace that are closest to data points. **2.** Maximize the variance of reconstructions: Find a subspace where data has the most variability. . **Proof the data and its reconstruction has the same means: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**To find the best   (I.e. to find the best projection):**

Let **U** be a matrix with columns then **.**

Also,  . Here, is the projector on subspace, and  **(!!!)**

**Proof that dimensions of z are decorrelated:** **.** Since the covariance matrix is diagonal, this means the features are uncorrelated. **PCA can be viewed as a linear autoencoder.** (as shown on the left). If K < D, then **W1** maps **x** to a K-dimensional space, so it’s doing dimensionality reduction.

**Unsupervised Learning:**

**K-means algorithm:** Objective: Find cluster centers and assignments to minimize the sum of squared distances of data points to their assigned cluster centers. **1.** Data sample (observed) **2.** Cluster center (not observed) **3.** Responsibilities: Cluster assignment for sample 1-of-K encoding (not observed)

Mathematically, we want:

The algorithm: Initialization: Set K cluster means to random values.

Repeat the following until convergence (i.e. until assignments do not change):

**Assignment:** Optimize w.r.t. : Each data point assigned to nearest center

And **Responsibilities** (1 hot, if not using soft K-means):

for

**Refitting:** Optimize *J* w.r.t. : Each center is set to mean of data assigned to it.

K-means algorithm reduces the cost **at each iteration.**

**Soft K-means:** different assignment step.

Soft K-means becomes K-means. **Proof:** when beta is approaching infinity, the assignment step will be much more sensitive to the distance between each point and centers. Thus, if any center is slightly closer to the data point, it will immediately become the choice.

**EM Algorithm:**

Initialize the means   and mixing coefficients

Iterate until convergence:

E-step: Evaluate the responsibilities given current parameters.

M-step: Re-estimate the parameters given the current responsibilities.

with

Evaluate the log likelihood and check for convergence.

-> When this converges, stop!

**What happened in the EM algorithm: 1**. The maximum likelihood objective was hard to optimize. **2.** However, the likelihood for is easy to compute. . **3.** We don’t know about , so we replaced with resibilties . That is, we replaced with its **expectation** under . Then, this is now easy to optimize, we can get

with (E-step)

**Both k-means and EM suffer from getting stuck at local minima.**

**Claim:** For EM algorithm with shared covariance**,** it becomes soft k-means algorithm. **Proof: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**K-means vs EM:**

|  |  |
| --- | --- |
| K-means | EM |
| Assignment Step: Assign each data point to the closest cluster.  Refitting Step: Move each luster center to the average of the data assigned to it. | E-step: Compute the posterior probability over z given our current model. ()  M-step: Maximize the probability that it would generate the data it is currently responsible for. |

**Matrix Factorization:**

Can view the following as Matrix Factorization.

**PCA:  .** SVD of the data matrix , with .

The eigen-decomposition of follows directly from the eigen-decomposition of **:** . The SVD gives **U**, which is equivalent to the learned basis of PCA. First K principal components corresponds first K columns of **U**. Ultimately, PCA with *K* principal components finds the *optimal* rank-K approximation of , in terms of error .

. Note that the case K = D corresponds to the entire SVD of **X**.

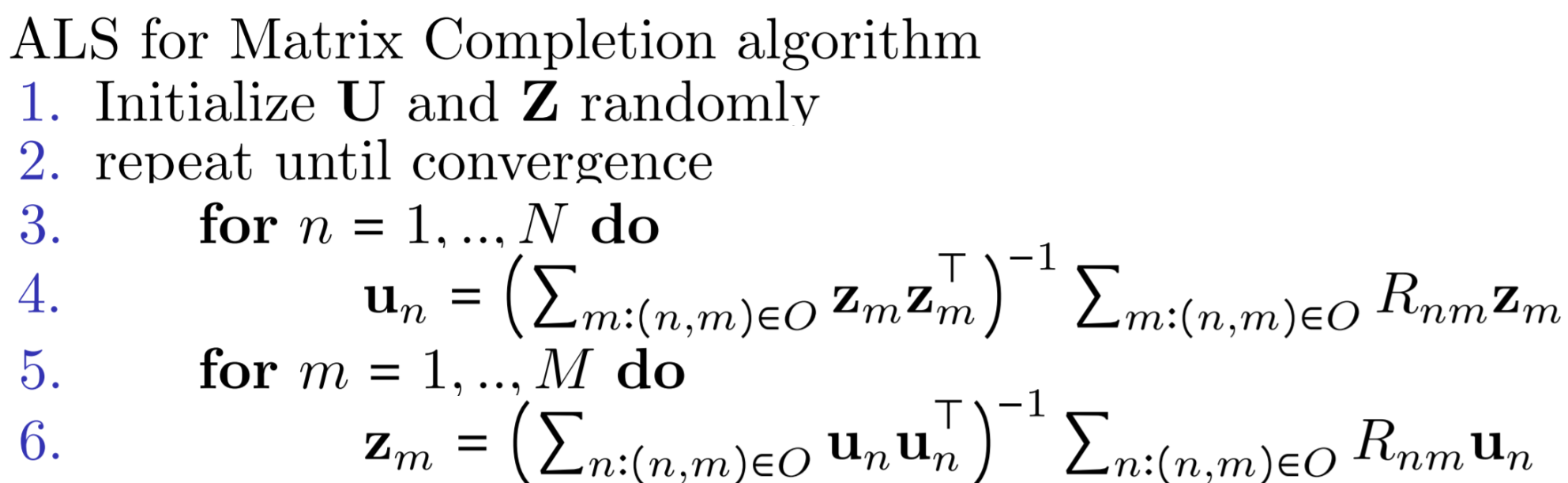
**Recommender Systems:**

Sparse ratings matrix: **R,** Let the representation of user be the representation of movie be . n = 1,…,K. We want to enforce **.** It is impossible to do , since most data entries are **missing!**

**Solution to missing data:** Let

Then we use square error loss to solve:

Problem: this equation is very hard to minimize. It is non-convex. Solution is to use alternating least squares, A.k.a. ALS: fix **Z** and optimize **U,** then followed by fix **U** and optimize **Z**, and so on until convergence.



**K-means**: Can view K-means as a matrix factorization too. **1.** Stack 1-of K vectors for assignments into a matrix **R**, and stack the cluster centers  **into a matrix**  matrix **M**. We want to let , through reconstruction!

K-means distortion function in matrix form: . We can do co-clustering so that **X** will give a block structure.

**Sparse Coding:** **,**where **s** is the sparse vector. **1.** We would like to choose **s** to accurately reconstruct the image, but encourage sparsity in **s**.

**2.** We want to minimize the following function:

Here, is a hyperparameter that trades off reconstruction error vs sparsity.

We can learn a dictionary by optimizing both **A** (aka the dictionary) and to trade off reconstruction error and sparsity

Can fit using ALS over A and S, just like anything mentioned before.

**Reinforcement Learning:**

Goal: Find a policy that maximizes the value function.

Optimal value function:

Given , the optimal policy can be obtained as

The goal of an RL agent is to find a policy that is close to optimal, i.e., .

The value function satisfy the following recursive relationship:

is called the **Bellman operator.** We define the bellman optimality operator :

Claim: , . The solutions of these fixed-pt equations are unique.

**Value-based approaches** try to find a   such that   (thus  )

Then, we apply greedy policy of   in practice.

**Greedy policy of**

**Finding the optimal value function using Value Iteration:**

Assume we know the model P and R, and the state-action space is small, Then we can use the bellman optimality equation. We start from an initial function . For each , apply .

(continuous)

(discrete)

**Claim**: The VI converges to the optimal value function. This is because of the contraction property of the Bellman (optimality) operator.

i.e.,

**Problem for VI**: **1.** when state-action space is large, you cannot integrate exactly.

**2.** We often do not know the dynamics and the reward function , so we cannot calculate the Bellman operators. Solution Batch RL!

**Batch RL:** Suppose we are **only** given the following dataset.

We can estimate using the data given.

**1.** A batch, i.d.

**2.**  ( is a distribution over )

**3.** ; **4.**

**Claim:** is a noisy version of .

**Proof:**

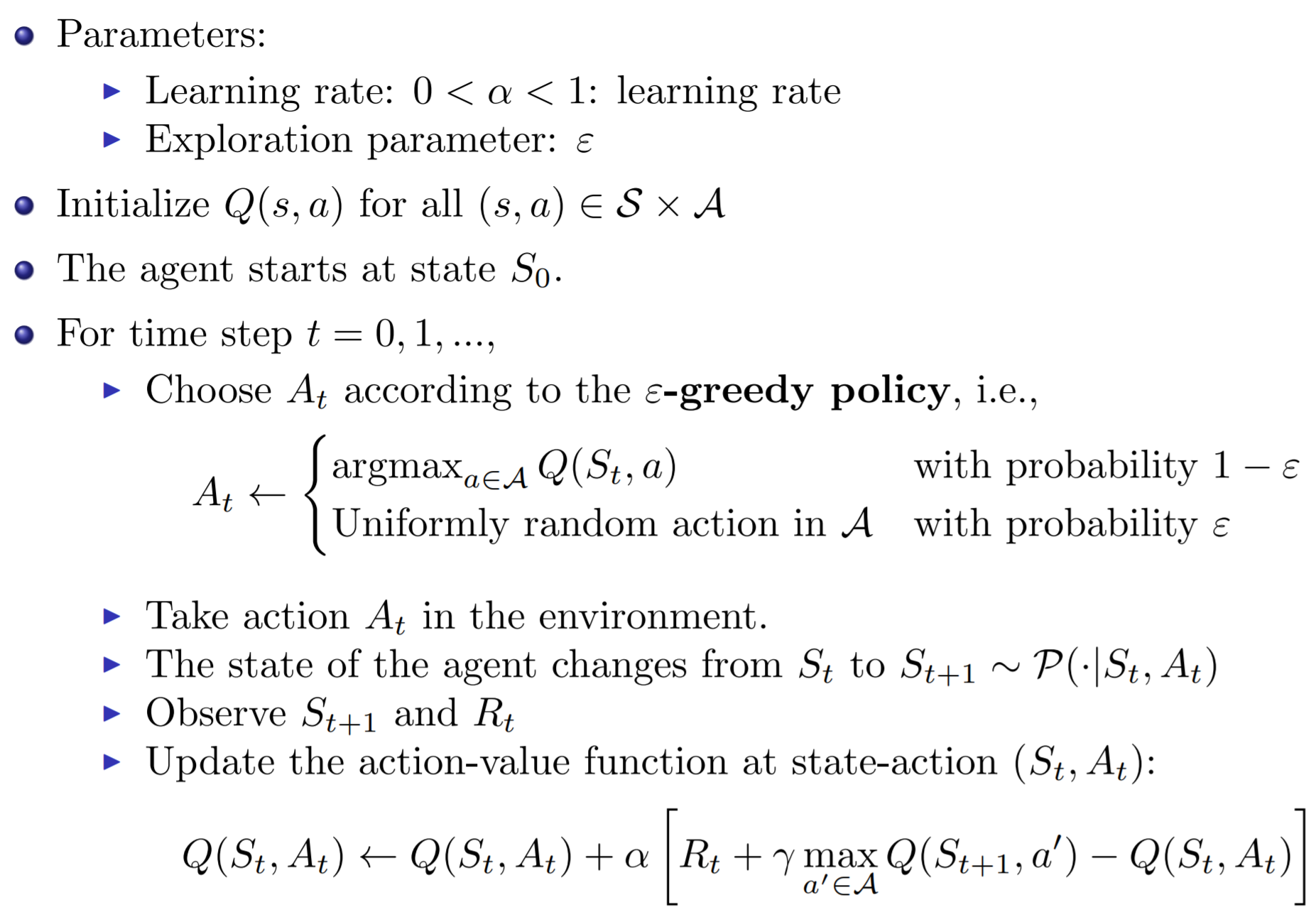
**The Algorithm of Batch RL:** Given the dataset and action value function estimate , we firstly construct the dataset with and By the proof above we can treat the problem of estimating as a regression problem with noisy data. We solve a regression problem. We minimize the squared error.

The policy of the agent is selected to be the **greedy** policy w.r.t. the final estimate of the value function: At state , the agent chooses

This method is called **Approximate Value Iteration (AVI).**

**Online Learning:** Motivation: RL problems are often interactive: The agent interacts with the environment and updates its knowledge of the world and its policy, with the goal of achieving as much rewards as possible.

**Q-Learning with epsilon-greedy policy:**



**RL miscellaneous:**

**Q:** What property does Transition probability follow? **S:** Markov property: the future depends on the past only through the current state

**Q:** Why do we need the discount factor? (Why cannot we always use the cumulative rewards, i.e. = 1?) **S:** The cumulative reward is problematic for continuing tasks because the final time step would be T = ∞, and the return could easily be infinite.

**Q: VI vs AVI:** S: VI: Finding the optimal policy / value function when P and R known (planning problem) AVI: We are given dataset of size N, we can approximately perform VI using these data by regression, minimizing MSE.

**Q:** What is the **difference between Batch RL and online RL?** S: In batch RL, we have a batch of data coming from the previous interaction of the agent with the environment. This allowed us to use tools from the supervised learning literature. But RL problems are often interactive: the agent continually interacts with the environment, updates its knowledge of the world and its policy, with the goal of achieving as much rewards as possible.

**Q:** What is the difference between **exploration** and **exploitation** and how are they related in the Q-Learning algorithm? S: The RL agent should collect as much information about the environment as possible (exploration), while benefiting from the knowledge that has been gathered so far in order to obtain a lot of rewards (exploitation). In Q-Learning, the -greedy policy ensures that most of the time (probability 1 - ) the agent exploits its incomplete knowledge of the world by choosing the best action, but occasionally (probability ) it explores other actions.