Machine Learning Cheat Sheet

Course: Machine Learning
Lecturer: Dr. Rico Möckel

The following equations and algorithms will be made available to you during the closed-book exam as they are presented here.

```
function LearnRuleSet(Target, Attrs, Examples, Threshold):
    LearnedRules := ∅
    Rule := LearnOneRule(Target, Attrs, Examples)
    while performance(Rule,Examples) > Threshold, do
        LearnedRules := LearnedRules ∪ {Rule}
        Examples := Examples \ {examples classified correctly by Rule}
        Rule := LearnOneRule(Target, Attrs, Examples)
    sort LearnedRules according to performance
    return LearnedRules
```

*Sequential Covering(by approach seperate and conquer)
*Threshold : you choose by trial and error
*LearnedRule: Add new created rule
*Examples: you need to remove examples from which you learned new rule (to prevent overfitting)
*Recur

```
function LearnOneRule(Target, Attrs, Examples):
    NewRule := "IF true THEN pos"
    NewRuleNeg := Set of all negative examples
    while NewRuleNeg not empty, do
        // add a new literal to the rule
        Lit Candidates := generate candidate literals
        BestLit := argmax_{L∈Candidates} performance(Specialise(NewRule,Lit))
        NewRule := Specialise(NewRule, BestLit)
            NewRuleNeg := Set of all negative examples covered by NewRule
    return NewRule

function Specialise(Rule, Lit):
    let Rule = "IF conditions THEN pos"
    return "IF conditions and Lit THEN pos"
```

*Top-down approach
*You are starting with the most general rule.
*NewRuleNeg: all '-' examples that are falsely covered by the rule
*argmax: stands for the BEST example
*New Rule: created by a new (more) specialised rule

Learn One Rule is learning what is the best choice while learning (seraching for best cover)

This function specialise add new condition for the given rule

1: initialize V_0 arbitrarily;
2: i=0;
3: **while (bellman_error > Δ) do**
4: for all s in *S* do
5: $V_{i+1}(s) = r(s, \pi(s)) + \gamma V_i(\delta(s, \pi(s)))$
6: i++;
7: **end while**

Bellaman Policy Evaluation: Used in reinforcement learning with Q approach algortihms to find function V for each state,
we have given: (S,A,delta, r, gamma)
*bellman error: max corrections during the full backup(each iteration computes new value for V)
*DELTA: we need to choose DELTA
*gamma: discount factor
*delta: transition function - doesn't need to be known

1: Initialise *Q randomly;*
2: Initialize *s*;
3: **while (true) do**
4: Choose an action *a*;
5: Execute *a* and observe *r* and *s'*;
6: $Q(s,a) := r + \gamma \max_{a'} Q(s',a')$;
7: *s := s';*
*8:**end while***

Q learning algorithm for deterministic states with Bellman Equation
*Q(s',a') hypothesis
*Exploration independent: off policy

$$J(c^{(1)}, \ldots, c^{(m)}, \mu_1, \ldots, \mu_K) = \frac{1}{m} \sum_{i=1}^{m} ||x^{(i)} - \mu_{c^{(i)}}||^2$$

$$\min_{\substack{c^{(1)},\ldots,c^{(m)},\\ \mu_1,\ldots,\mu_K}} J(c^{(1)}, \ldots, c^{(m)}, \mu_1, \ldots, \mu_K)$$

Cost function for k-mean function: Distortion cost function
c- cluster
x- location of the example
mean- centroid
m- # of instances

min: trying to find the min value of the cost function

$$J(x^{(1)}, \ldots, x^{(n_m)}, \theta^{(1)}, \ldots, \theta^{(n_u)}) = \frac{1}{2} \sum_{(i,j):r(i,j)=1} ((\theta^{(j)})^T x^{(i)} - y^{(i,j)})^2 + \frac{\lambda}{2} \sum_{i=1}^{n_m} \sum_{k=1}^{n} (x_k^{(i)})^2 + \frac{\lambda}{2} \sum_{j=1}^{n_u} \sum_{k=1}^{n} (\theta_k^{(j)})^2$$

*Collaborative Filtering: you esimate X and THETA simultaneously to end up with good estimation, for recommender systems.
*E - combines all pairs from x and THETA *x- feature vector for given movie * y rating given by user * THETA: parameter vector
*lambda: optimalization criterion (if too hight =not good accuracy)

$$x_k^{(i)} := x_k^{(i)} - \alpha \left(\sum_{j:r(i,j)=1} ((\theta^{(j)})^T x^{(i)} - y^{(i,j)}) \theta_k^{(j)} + \lambda x_k^{(i)} \right)$$

$$\theta_k^{(j)} := \theta_k^{(j)} - \alpha \left(\sum_{i:r(i,j)=1} ((\theta^{(j)})^T x^{(i)} - y^{(i,j)}) x_k^{(i)} + \lambda \theta_k^{(j)} \right)$$

Collaborative Filtering Algorithm
1. Initialize x1...xm and THETA1...THETAm to small random variables
2. Minimize J(x,THETA) using gradient descent <----- formulas for it
3. For a user with (learned) parameters THETA and a movie with learned features x, predict a star rating of THETA transposeX

$$J(\theta) = \frac{1}{2m} \left[\sum_{i=1}^{m} (h_\theta(x^{(i)}) - y^{(i)})^2 + \lambda \sum_{j=1}^{n} \theta_j^2 \right]$$

Cost function of linear regresion with the regularization
*h stands for hypothesis and its create by THETAs (h=THETA0+THETA1)
* y output/taget
*m # of instances
*lambda: regularization parameter, too big underfitting

Gradient descent:

$$\theta_0 := \theta_0 - \alpha \frac{1}{m} \sum_{i=1}^{m} (h_\theta(x^{(i)}) - y^{(i)}) x_0^{(i)}$$

$$\theta_j := \theta_j - \alpha \left[\frac{1}{m} \sum_{i=1}^{m} (h_\theta(x^{(i)}) - y^{(i)}) x_j^{(i)} + \frac{\lambda}{m} \theta_j \right]$$

Linear regresion gradient descent :
for THETA 0 x0=1 because of the definition

Training set $\{(x^{(1)}, y^{(1)}), \ldots, (x^{(m)}, y^{(m)})\}$

Set $\triangle_{ij}^{(l)} = 0$ (for all l, i, j).

For $i = 1$ to m

Set $a^{(1)} = x^{(i)}$

Perform forward propagation to compute $a^{(l)}$ for $l = 2, 3, \ldots, L$

Using $y^{(i)}$, compute $\delta^{(L)} = a^{(L)} - y^{(i)}$

Compute $\delta^{(L-1)}, \delta^{(L-2)}, \ldots, \delta^{(2)}$

$\triangle_{ij}^{(l)} := \triangle_{ij}^{(l)} + a_j^{(l)} \delta_i^{(l+1)}$

$D_{ij}^{(l)} := \frac{1}{m}[\triangle_{ij}^{(l)} + \lambda \Theta_{ij}^{(l)}]$ if $j \neq 0$

$D_{ij}^{(l)} := \frac{1}{m}\triangle_{ij}^{(l)}$ if $j = 0$

$$\frac{\partial}{\partial \Theta_{ij}^{(l)}} J(\Theta) = D_{ij}^{(l)}$$

Artificial Neutral Networks :
This algorithm is for learning weights
DELTA: activation & error
a-activation of unit layer
l-layer
y- output of foreward propagation
j-dimention
delta- backpropagation computation

$$d_W(\mathbf{x}, \mathbf{y}) = \sqrt{\sum_{j=1}^{D} w_j (x_j - y_j)^2}$$

Weighted distances: Instanc Based Learning with noisy data
Solution : Give each attribute a different weight in the distance computation

$$f(x) = w_0 + \sum_{u=1} w_u K_u(d(x_u, x))$$

Radial basis function networks:
Build global approximation as linear combination of local approximation

Influence of each local approximation u goes down quickly with the distance

$$\min_\theta C \sum_{i=1}^{m} y^{(i)} cost_1(\theta^T f^{(i)}) + (1 - y^{(i)}) cost_0(\theta^T f^{(i)}) + \frac{1}{2} \sum_{j=1}^{n} \theta_j^2$$

Support Vector Machines with Kernels :
Hypothesis: Given x, compute features f E R m+1
This is a training function , we got rid of 1/m because it is a CONSTANT

$$Q_\lambda^\pi(s_t, a_t) = (1 - \lambda) \sum_{n=1}^{N-t-1} \lambda^{n-1} Q_{(n)}^\pi + \lambda^{(N-t-1)} Q_{MC}^\pi$$

Sarsa/True Online Its combining estimates

Gradient descent for logistic regressions with regularization

$$\frac{\partial}{\partial\theta_0}J(\theta) = \frac{1}{m}\sum_{i=1}^{m}(h_\theta(x^{(i)}) - y^{(i)})x_0^{(i)}$$

$$\frac{\partial}{\partial\theta_1}J(\theta) = \frac{1}{m}\sum_{i=1}^{m}(h_\theta(x^{(i)}) - y^{(i)})x_1^{(i)} + \frac{\lambda}{m}\theta_1$$

$$\frac{\partial}{\partial\theta_2}J(\theta) = \frac{1}{m}\sum_{i=1}^{m}(h_\theta(x^{(i)}) - y^{(i)})x_2^{(i)} + \frac{\lambda}{m}\theta_2$$

$$h_\theta(\mathbf{x}) = g(\theta^T\mathbf{x}) = \frac{1}{1+e^{-\theta^T\mathbf{x}}}$$

Sigmoid function, predict only values [0;1]
convex shape for the minimum

$$\min_{\boldsymbol{\theta}} \frac{1}{2}[\mathbf{X}\boldsymbol{\theta} - \mathbf{y}]^T[\mathbf{X}\boldsymbol{\theta} - \mathbf{y}]$$

Normal equation to solve THETA analytically for linear regresion

$$\mathbf{X}^T\mathbf{X}\boldsymbol{\theta} = \mathbf{X}^T\mathbf{y}$$

Setting the gradient to zero

$$\boldsymbol{\theta} = (\mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^T\mathbf{y}$$

$$\theta = \left(X^TX + \lambda\begin{bmatrix}0 & & & \\ & 1 & & \\ & & \ddots & \\ & & & 1\end{bmatrix}\right)^{-1}X^Ty$$

Normal eqution under regularization
Two advantages:
* fights over-fitting
*guarantees matrix of full rank, and thus invertable

$$N_j(x_i) = \frac{1}{\sqrt{(2\pi\sigma^2)^n}}e^{-\frac{1}{2}\left((x_i-\mu_j)^T\frac{I}{\sigma^2}(x_i-\mu_j)\right)}$$

EM algorithm: representation of the clusters with a Gaussian
This counts probability of x|mean
n is the dim of examples

$$h_{ij} = \frac{P_j * N_j(x_i)}{\sum_l P_l * N_l(x_i)}$$

E-step in EM algortihm
Recall that z is a hidden variables which is 1 to N generated x and 0 otherwise\\
In the E-step, we compute h the expected value of the hidden variable
l- goes over the clusters

$$\mu_j = \frac{\sum_i h_{ij} * x_i}{\sum_i h_{ij}} \quad P_j = \frac{\sum_j h_{ij}}{n}$$

M-step
Given the expected values h , we re-estimate the mean of the Gaussian and the cluster probabilities
i- goes over examples, j over clusters

Reinforcement learning:

$$V_{i+1}(s) = r(s,\pi(s)) + \gamma V_i(\delta(s,\pi(s)))$$

Bellman equation:
r-reward; pi-policy, gamma -discount factor ;delta: transition function - doesn't need to be known

$$\pi'(s) = \arg\max_a[r(s,a) + \gamma V^\pi(\delta(s,a))]$$

<- value function policy improvement for Bellman, greedy action selection & guarantees

$$Q(s,a) \leftarrow r + \gamma max_{a'}Q(s',a')$$

Q-learning algorithm; copmutes q-values for all state action pairs with bellam equation, where Q' is current hypothesis // deterministic world

$$Q(s,a) = (1-\alpha)\cdot Q(s,a) + \alpha\cdot(r + \gamma Q(s',a'))$$

Q-learning with non deterministic enviroment ,
this one is stabilizing by taking an average of new and past Qvalues , giving increasingly higer values weight to the past with alfa : 1/1+visits(s,a)