CS229 ML

1. Intro √
2. Linear regression & gradient descent √
3. Locally weighted & logistic regression √
4. Perception & generalized linear model √
5. GDA & Naive Bayes √😊
6. SVM √
7. Kernels √
8. Data splits & model & cross-validation √
9. Approx/estimation error & ERM ?
10. Decision trees & ensemble methods √😊
11. Neural networks √
12. Backprop & improving neural networks √
13. Debugging ML models & error analysis ?
14. Expection-maximization algorithms √ in process
15. Expection-maximization algorithms & Factor analysis
16. Independent component analysis & RL √😊
17. MDP & value/policy iteration √😊
18. Continous state MDP & model simulation √😊
19. Reward model & linear dynamical system √😊
20. RL debugging & diagnostics

5. GDA & Naive Bayes

Generative Learning Algorithms

- Gaussian Discriminant Analysis (GDA)
- Generative and discriminative comparison
- Naive Bayes

Discriminative:

learn $p(y|x)$ or $h_\theta(x) = \begin{cases} 1 \\ 0 \end{cases}$ directly.

Generative learning algorithm:

learn $p(x|y)$ and $p(y)$ — class prior;   x — feature, y — class

Bayes rule:

$p(y=1|x) = \frac{p(x|y=1)p(y=1)}{p(x)}$

where $p(x) = p(x|y=1)p(y=1) + p(x|y=0)p(y=0)$

Gaussian Discriminant Analysis (GDA)

Suppose $x \in R^n$ (drop $x_0 = 1$ convention)

Assume $p(x|y)$ is Gaussian (features of each class follow multivariate Gaussian)

$x|y=0 \sim N(\mu, \Sigma); x|y=1 \sim N(\mu, \Sigma)$

GDA model: parameters $\mu_0, \mu_1 \in R^n, \Sigma \in R^{n \times n}, \phi \in R$

$p(x|y=0) = \frac{1}{(2\pi)^{n/2}|\Sigma|^{1/2}} \exp(-\frac{1}{2}(x-\mu_0)^T \Sigma^{-1}(x-\mu_0))$

$p(x|y=1) = \frac{1}{(2\pi)^{n/2}|\Sigma|^{1/2}} \exp(-\frac{1}{2}(x-\mu_1)^T \Sigma^{-1}(x-\mu_1))$

$p(y) = \phi^y(1-\phi)^{1-y}$ or $P(y=1) = \phi$

Training set: $\{(x^{(i)}, y^{(i)})\}_{i=1}^m$

Joint likelihood: $L(\phi, \mu_0, \mu_1, \Sigma) = \Pi_{i=1}^m p(x^{(i)}, y^{(i)}; \phi, \mu_0, \mu_1, \Sigma)$
$= \Pi_{i=1}^m p(x^{(i)}|y^{(i)})p(y^{(i)})$

In contrast, Discriminative: maximize

Conditional likelihood: $L(\theta) = \Pi_{i=1}^m p(y^{(i)}|x^{(i)}, \theta)$

Maximum likelihood estimation:

$\max_{\phi,\mu_0,\mu_1,\Sigma} L(\phi, \mu_0, \mu_1, \Sigma) \rightarrow$

$\phi = \frac{\Sigma_{i=1}^m y^{(i)}}{m} = \frac{\Sigma_{i=1}^m L\{y^{(i)}=1\}}{m}$ proportion of $y^{(i)} = 1$

$\mu_0 = \frac{\Sigma_{i=1}^m L\{y^{(i)}=0\}x^{(i)}}{\Sigma_{i=1}^m L\{y^{(i)}=0\}}$ look at all benign tumors and take averages

$\mu_1 = \frac{\Sigma_{i=1}^m L\{y^{(i)}=1\}x^{(i)}}{\Sigma_{i=1}^m L\{y^{(i)}=1\}}$

$\Sigma = \frac{1}{m}\Sigma_{i=1}^m (x^{(i)} - \mu_{y^{(i)}})(x^{(i)} - \mu_{y^{(i)}})^T$

Prediction: $\arg\max_y p(y|x) = \arg\max_y \frac{p(x|y)p(y)}{p(x)} = \arg\max_y p(x|y)p(y)$

Compared to Logistics Regression: has stronger assumption; if nearly Gaussian (most cases), performs better, else (e.g. poisson distribution) worse; more efficient

One GDA — Naive Bayes

10. Decision trees & ensemble methods

Decision trees: nonlinear; greedy, top-down, recursive partitioning

Looking for a split s_p: $s_p(j,t) = (\underbrace{\{x|x_j < t, x_j \in R_p\}}_{R_1}, \underbrace{\{x|x_j \geq t, x_j \in R_p\}}_{R_2})$

Define $L(R)$: loss on region R

Given C classes, define $\hat{p}_c$ to be the proportion of examples in R that are of class C

Objective: $\max L(R_p) - (L(R_1) + L(R_2))$

Uses cross entropy loss: $L_{cross} = -\Sigma_c(\hat{p_c} \cdot \log_2 \hat{p_c})$

Regularization of decision trees:

1. min leaf size
2. max depth
3. max number of nodes
4. min decrease in loss
5. pruning with validation set

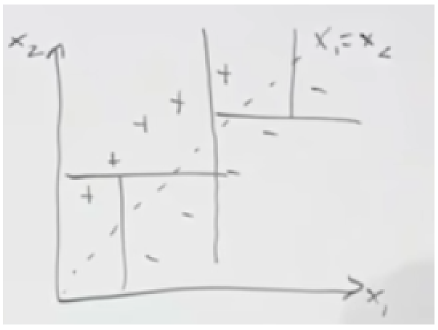
Runtime: n examples, f features, d depth (usually $d < log_2 n$)

Test time $O(d)$

Train time: $O(nfd)$ since each point is part of $O(d)$ nodes, cost of point at each node is $O(f)$

Strength: 1. easy to explain; 2. interpretable; 3. categorical variables; 4. fast

Weakness: 1. no additive structure



2. high variance, easy to overfit; 3. due to 1, 2 → low predictive accuracy

Ensembling

Take x_i's which are random variables (RV) that are independent identically distributed (i.i.d.),

$Var(x_i) = \sigma^2$, $Var(\bar{x}) = \frac{\sigma^2}{n}$

Drop independence assumption, so now x_i's are i.d., x_i's correlated by p,

$Var(\bar{X}) = p\sigma^2 + \frac{1-p}{n}\sigma^2$

Ways to ensemble:

1. different algorithms
2. different training sets
3. bagging (e.g. random forests)
4. boosting (e.g. Adaboost, xgboost)

Bagging - Bootstrap AGGregatING

True population P, Training set S~P

Assume P=S, Booststrap samples Z_1, ..., Z_M ~ S

Train model G_m on Z_m,

$G(m) = \frac{\Sigma_{m=1}^{M} G_m(x)}{M}$

Bias-Variance Analysis: $Var(\bar{X}) = p\sigma^2 + \frac{1-p}{n}\sigma^2$

Bootstrapping is driving down p, more M → less variance; bias slightly increases

DT are high variance low bias, ideal fit for bagging → random forests

Boosting

Decrease bias, addictive

Adaboost: Determine for classifier G_m a weight $\alpha_m = \log(\frac{1-err_m}{err_m})$, then $G(x) = \Sigma \alpha_m G_m$

14. Expection-maximization algorithms

— unsupervised learning

Density estimation

Mixture of Gaussian model

Suppose there's a latent (hidden/unobserved) random variable z, and $x^{(i)}, z^{(i)}$ are disjoint distribution,

where $z^{(i)} \sim \text{Multinomial}(\phi)$ or $z \in \{1, .., k\}, x^{(i)}|z^{(i)} = j \sim N(\mu_j, \Sigma_j)$

If we know the $z^{(i)}$'s, can use MLE:

$l(\phi, \mu, \Sigma) = \Sigma_{i=1}^{m} \log p(x^{(i)}, z^{(i)}; \phi, \mu, \Sigma)$

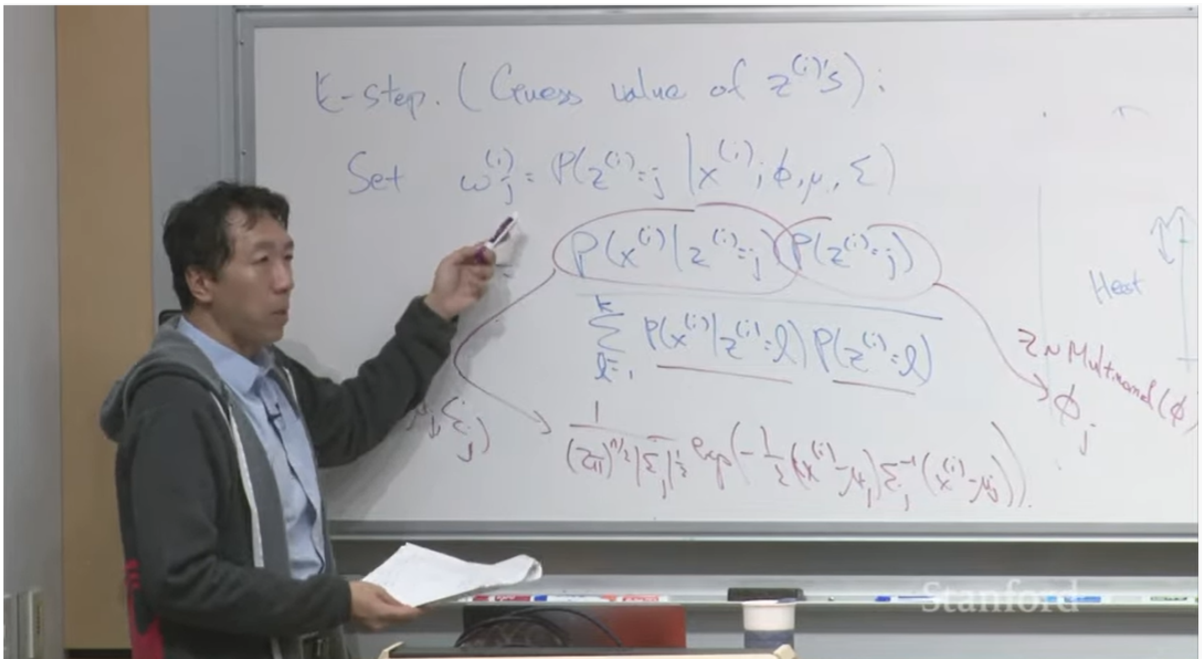
→ $\phi_j = \frac{\Sigma_{i=1}^{m} L\{y^{(i)}=j\}}{m}$ proportion of $z^{(i)} = j$

$\mu_j = \frac{\Sigma_{i=1}^{m} L\{z^{(i)}=j\} x^{(i)}}{\Sigma_{i=1}^{m} L\{z^{(i)}=j\}}$

EM (Expection Maximization) Algorithm

E-step (guess value of $z^{(i)}$'s)

Set $w_j^{(i)} = P(z^{(i)} = j | x^{(i)}; \phi, \mu, \Sigma) = \frac{P(x^{(i)}|z^{(i)}=j)P(z^{(i)}=j)}{\Sigma_{l=1}^{k} P(x^{(i)}|z^{(i)}=l)P(z^{(i)}=l)}$


E-step. (Guess value of $z^{(i)}$'s):
Set $w_j^{(i)} = P(z^{(i)} = j | x^{(i)}; \phi, \mu, \Sigma)$
$P(x^{(i)}|z^{(i)}=j) P(z^{(i)}=j)$
$\sum_{l=1}^{k} P(x^{(i)}|z^{(i)}=l) P(z^{(i)}=l)$
$\frac{1}{(2\pi)^{n/2}|\Sigma_j|^{1/2}} \exp\left(-\frac{1}{2}(x^{(i)}-\mu_j)\Sigma_j^{-1}(x^{(i)}-\mu_j)\right)$
$z \sim \text{Multinomial}(\phi)$
ϕ_j
Heat
Stanford

M-step

$\phi_j = \frac{1}{m}\Sigma_{i=1}^{m} w_j^{(i)}$ (replaced $L\{z^{(i)} = j\}$ with $w_j^{(i)} = E[L\{z^{(i)} = j\}]$)

$\mu_j = \frac{\Sigma_{i=1}^{m} w_j^{(i)} x^{(i)}}{\Sigma_{i=1}^{m} w_j^{(i)}}$

until 40:00

16.2 RL: MDP

Reward(state) R(s) = 1 for win; -1 for loss; 0 for tie

Challenge: credit assignment problem

Markov Decision Process (MDP)

$\{S, A, P_{sa}, \gamma, R\}$

S: set of states

A: set of actions

Psa: state transition probabilities. $\Sigma_{S'} P_{sa}(S') = 1$

γ — discount factor. $\gamma \in [0, 1)$

R — reward function

Choose action a_0

Get to $s_1 \sim P_{s_0 a_0}$

Choose action a_1

Get to $s_2 \sim P_{s_1 a_1}$

…

e.g. $\gamma = 0.99$

Total payoff: $R(s_0) + \gamma R(s_1) + \gamma^2 R(s_2) + ...$

Goal: choose actions over time to maximize the expected total payoff $E[R(s_0) + \gamma R(s_1) + \gamma^2 R(s_2) + ...]$

Ouput — Policy $\pi : S \to A$

When is state s, take action $\pi(s)$.

17. RL: MDP & Value function & Value/Policy iteration & Learning Psa

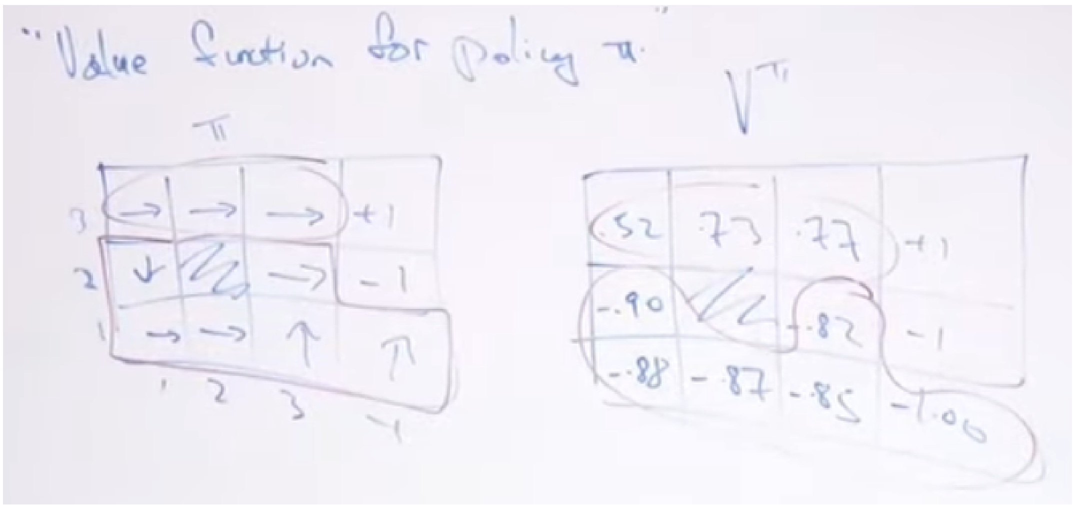
Define: V^π, V^*, π^*

For a policy π, $V^\pi : S \mapsto R$ is s.t.

$V^\pi(s)$ is the expected total payoff for starting in state s and excuting π:

$V^\pi(s) = E[R(s_0) + \gamma R(s_1) + \gamma^2 R(s_2) + ...|\pi, s_0 = s]$

— “Value function for policy π”



Bellman’s equation:

$V^\pi(s) = R(s) + \gamma \cdot \Sigma_{s'} P_{s\pi(s)}(s')V^\pi(s')$

In state s, take action $a = \pi(s)$, $s' \sim P_{sa}$ where $= \pi(s)$.

Given π, get a linear system of equation in terms of $V^\pi(s)$.

V^* is the optimal value function,

$V^*(s) = \max_\pi V^\pi(s)$.

Bellman’s equation (another form):

$V^*(s) = R(s) + \gamma \max_a \Sigma_{s'} P_{sa}(s') V^*(s')$

π^* is the optimal value function,

$\pi^*(s) = \arg\max_a \gamma \cdot \Sigma_{s'} P_{sa}(s') V^*(s')$.

So: $V^*(s) = V^{\pi^*}(s) \geq V^{\pi}(s)$ for any π, s.

Strategy:

1. Find V^*.

Value iteration

Initialize $V(s) := 0$ for every s.

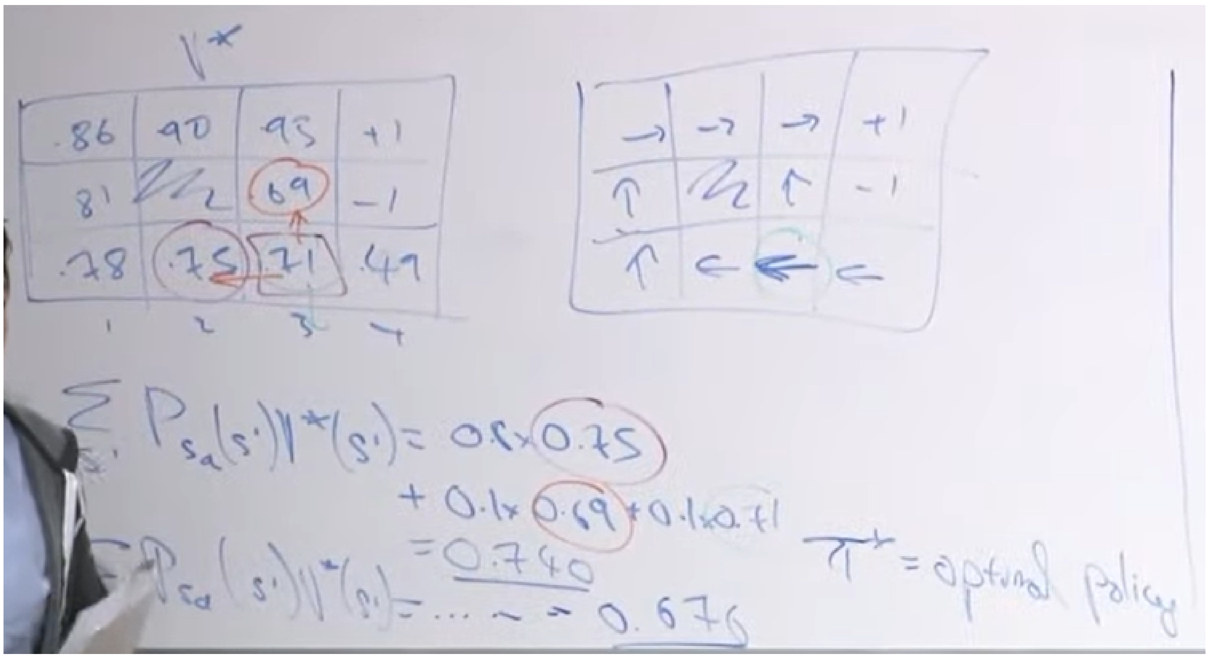
For every s update:

$V(s) := R(s) + \gamma \max_a \Sigma_{s'} P_{sa}(s') V^*(s')$ # synchronous update

Bellman backup operator: $V = B(V)$

V converges to V^* exponentially.

2. Use argmax equation to find π^*.



V*
.86
.90
.93
+1
.81
.69
-1
.78
.75
.71
.49
1
2
3
4
+1
-1
Σ P_sa(s')V*(s') = 0.8×0.75 + 0.1×0.69 + 0.1×0.71 = 0.740
Σ P_sa(s')V*(s') = ... ~ ~ 0.676
π* = optimal policy

Policy iteration

Initialize π randomly.

Repeat:

Set $V := V^{\pi}$. (solve Bellman's equation to get V^{π})

Set $\pi(s) := \arg\max_a \cdot \Sigma_{s'} P_{sa}(s') V^*(s')$.

V^{π} — linear system of equations

Value iteration converges faster but will not get the exact optimal value (like gradient descent).

Policy iteration gets the optimal value. Slow when the number of states is large.

What if don't know P_{sa}? Need to estimate from data.

$P_{sa}(s')$ = # times took action "a" in state s and get to s' / # times took action "a" in state s

or 1 / |S| if above is 0 / 0

Putting it together:

Repeat:

Take actions wrt (with repect to) policy π to get experience in MDP

Update estimates of P_{sa} (and possibly R)

Solve Bellman's equation using value iteration to get V

Update $\pi(s) := \arg\max_a \Sigma_{s'} P_{sa}(s') V^(s')$

Exploration vs. Exploitation tradeoff

ϵ-greedy ($\epsilon = 0.1$): 0.9 chance wrt π, 0.1 chance randomly

18. RL: Continous state MDP & model simulation

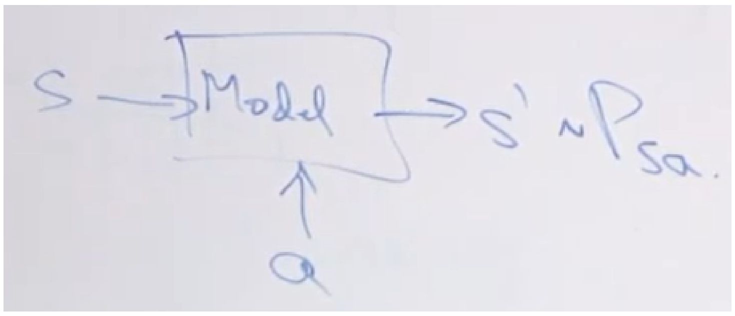
Discretization

Problems:

- Naive representation for V^* (and π^*)
- Curse of dimensionality
  - $S = R^n$, and discretize each dimension into k values, get k^n discrete states

$V^*(s) \approx \theta^T \phi(s)$

Model (simulator) of MDP:



How to get model?

- Physics simulator
- Learn model from data

$s_0^{(1)} \quad \underset{\longrightarrow}{a_0^{(1)}} \quad s_1^{(1)} \quad \underset{\longrightarrow}{a_1^{(1)}} \quad ... \quad s_T^{(1)}$

…

$s_0^{(M)} \quad \underset{\longrightarrow}{a_0^{(M)}} \quad ...$

Apply supervised learning to estimate s_{t+1} (s') as function of s_t, a_t (s, a)

E.g. linear regression version: $s_{t+1} = As_t + Ba_t$ (deterministic)

or $s_{t+1} = As_t + Ba_t + \epsilon_t$ (stochastic)

$\min_{A,B} \Sigma_{i=1}^m \Sigma_{t=0}^T ||s_{t+1}^{(i)} - As_t^{(i)} + Ba_t^{(i)}||^2$

Value iteration:

$V(s) := R(s) + \gamma \max_a \Sigma_{s'} P_{sa}(s') V^*(s')$

$V(s) := R(s) + \gamma \max_a E_{s' \sim P_{sa}}[V(s')] = \max_a E_{s' \sim P_{sa}}[R(s) + \gamma V(s')]$

Fitted Value iteration:

```
Sample $\{s^{(1)}, s^{(2)}, ..., s^{(m)}\} \subseteq S$ randomly
Initialize $\theta := 0$
Repeat {
    For i = 1, …, m {.
        For each action $a \in A$ {
            Sample $s'_1, ..., s'_k \sim P_{s^{(i)}a}$
            Set $q(a) = \frac{1}{k} \Sigma_{j=1}^k [R(s^{(i)}) + \gamma V(s'_j)]$
        }
        Set $y^{(i)} = \max_a q(a)$
    }
    $\theta := \arg\min_\theta \frac{1}{2} \Sigma_{i=1}^m (\theta^T \phi(s^{(i)}) - y^{(i)})^2$
```

}

Fitted VI gives approximation to V^*.

Implicitly define π^*.

19. RL: State-action Rewards & Finite horizon MDP & Linear Dynamical System

State-action Rewards: $R : S \times A \mapsto \mathbb{R}$

$R(s_0, a_0) + \gamma R(s_1, a_1) + \gamma^2 R(s_2, a_2) + ...$

Finite horizon MDP: $\{S, A, P_{sa}, T, R\}$

$R(s_0, a_0) + R(s_1, a_1) + ...R(s_T, a_T)$

Optimal action may depend on time (left) — non-stationary policy $\pi_t^*(s)$

Non-stationary state transitions $P_{s_t a_t}^{(t)}$

Non-stationary reward $R^{(t)}(s, a)$

Linear Quadratic Regulation (LQR)

$\{S, A, P_{sa}, T, R\}$

Assumption 1: $s_{t+1} \approx As_t + Ba_t + w_t$(noise) linear

Assumption 2: $R(s, a) = -(s^T Us + a^T Va)$ quadratic cost function

where $U, V \geq 0$ (positive semidefinite) $\leftrightarrow s^T Us, a^T Va \geq 0$

(Want $S \approx 0$, Choose $U = I, V = I$, then $R(s, a) = -(||s||^2 + ||a||^2)$)

Total payoff: $R(s_0, a_0) + R(s_1, a_1) + ...R(s_T, a_T)$

Dynamic programming algorithm for LQR:

$V_T^*(s_T) = \max_{a_T} R(s_T, a_T) = \max_{a_T} -s_T^T U s_T + a_T^T V a_T) = \max_{a_T} -s_T^T U s_T$

$\pi_T^*(s_T) = \vec{0}$

Suppose $V_{t+1}^*(s_{t+1}) = s_{t+1}^T s_{t+1} + s_{t+1}$

$V_t^*(s_t) = \max_{a_t} R(s_t, a_t) + E_{s_{t+1} \sim P_{s_t a_t}} [V_{t+1}^*(s_{t+1})]$

Resulting formula:

$V_t^*(s_t) = s_t^T \Phi_t s_t + \Psi_t$

where $\Phi_t = A(\Phi_{t+1} - \Phi_{t+1} B(B^T \Phi_{t+1} B - V)^{-1} B \Phi_{t+1}) A_t - U$, $\Phi_T = -U$

$\Psi_t = -tr \Sigma_w \Phi_{t+1} + \Psi_{t+1}$, $\Psi_t = 0$