8-4-2020 1:00 PM

Classification-based method for estimating dynamic treatment regimes

Junwei Shen, *The University of Western Ontario*

Supervisor: He, Wenqing, *The University of Western Ontario*

A thesis submitted in partial fulfillment of the requirements for the Master of Science degree in Statistics and Actuarial Sciences

© Junwei Shen 2020

Abstract

Dynamic treatment regimes are sequential decision rules dictating how to individualize treatments to patients based on evolving treatments and covariate history. In this thesis, we investigate two methods of estimating dynamic treatment regimes. The first method extends outcome weighted learning from two-treatments to multi-treatments and allows for negative treatment outcome. We show that under two different sets of assumptions, the Fisher consistency can be maintained. The second method estimates treatment rules by a neural classification tree. A weighted squared loss function is defined to approximate the indicator function to maintain the smoothness. A method of tree reconstruction and pruning is proposed to increase the interpretability. Simulation studies and real application to data from Sequential Treatment Alternatives to Relieve Depression (STAR*D) clinical trial are conducted to illustrate the proposed methods.

Keywords: Classification methods, dynamic treatment regimes, neural classification tree, outcome weighted learning, personalized medicine, support vector machine

Lay Summary

Traditionally, treatments for patients are decided by clinical judgments based on clinician's experience or practice guidelines based on clinical evidence and expert opinions. Patients with the same disease often receive the same treatment. It is one-size-fits-all approach. However, patient heterogeneity makes it possible that the best treatment for one patient is suboptimal for another. Therefore, it is important to make an transition from the traditional one-size-fits-all approach to individualized treatment rule which takes personal characteristics into account and tailors treatments to patients. This thesis will present two methods of identifying individualized treatment rule, called multicategory outcome weighted learning and neural classification tree.

Acknowledgements

I would like to convey my profound gratitude to my supervisor, Dr. Wenqing He, for his scientific guidance, support and for sharing his expertise throughout my study at Western. This thesis would not have been possible without his insights and helpful comments.

I would like to express my thanks to Dr. Grace Yi. The data science meetings arranged by Dr. Yi and Dr. He deepened my interests in statistics and exposed me to the cutting-edge research problems.

I would like to thank my thesis examiners, Dr. Yun-Hee Choi, Dr. Cristian Bravo Roman, Dr. Jiandong Ren for taking the time to read my thesis and for the helpful comments.

Last but not least, I would like to thank my family and my friends at Western for their love and encouragement. The study and life at Western would not have been such happy without their constant support.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Dynamic treatment regimes

Personalized medicine is a medical paradigm where treatment is customized for each patient based on individual information. The motivation behind this paradigm is the fact that heterogeneity exists among different patients and when making medical decisions, the existing heterogeneity needs to be taken into account. For example, patients may respond differently to the same drug because of their personal difference. In this case, without considering personal information, the one-size-fits-all approach will result in inefficient or over treatment. Dynamic treatment regimes (DTR), also known as adaptive treatment strategies, generalize personalized medicine to time-varying treatment settings in which treatment is repeatedly tailored to a patient's dynamic state (Chakraborty and Murphy, 2014).

A simple example of dynamic treatment regimes is the treatment of alcohol dependence (Chakraborty and Moodie, 2013). Two stages are involved: initially, clinician prescribes either naltrexone (NTX) or cognitive behavioral therapy (CBT) to the patients. Patients are then classified as responder or non-responder based on the number of heavy-drinking days within the next two months after they take the initial treatment. If a patient experiences more than two heavy-drinking days during the following two months, the patient is labelled as a non-responder, otherwise a responder. At the second stage, the non-responders to NTX will be assigned either CBT or an augmentation of NTX with CBT and the non-responders to CBT will be assigned either NTX or an augmentation of CBT with NTX. All responders will receive telephone monitoring (TM) within the next six months.

Figure 1.1 gives a schematic of a possible DTR in the alcohol dependence example. This DTR consists of two decision rules: the first decision rule prescribes the initial treatment based on the baseline information H_0 and the second decision rule use intermediate outcome and updated information H_1 to assign the secondary treatment. Specifically, the DTR is: at the first stage, prescribe CBT if the baseline level of some variable exceeds the pre-specified threshold and otherwise prescribe NTX; at the second stage, if a patient is a responder to the initial treatment, prescribe TM as the secondary treatment; if a patient is a non-responder, switch to the other treatment or prescribe an augmentation based on whether the intermediate level of some variable exceeds the pre-specified threshold.

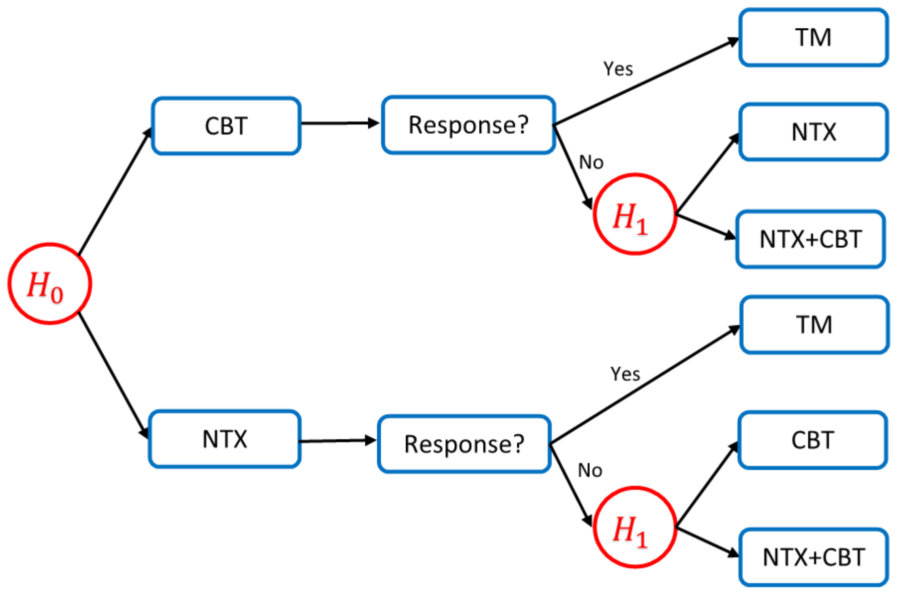


Figure 1.1: A schematic of DTR in the alcohol dependence example

Another example is Sequential Treatment Alternatives to Relieve Depression (STAR*D) clinical trial which will be used as numerical illustration in chapter 2 and 3. It was a multi-site, multi-step randomized clinical trial on 4041 patients with nonpsychotic major depressive disorder (Rush et al., 2004). The study compares treatment options for patients without satisfactory response with citalopram (CIT), a selective serotonin reuptake inhibitor antidepressant. The primary outcome is the clinician-rated Quick Inventory of Depressive Symptomatology (QIDS) score ranging from 0 to 27 in the sample. Higher values of QIDS score correspond to higher severity and thus represent a worse outcome. The study included four levels where each level consisted of a 12 week period of treatment. At the end of each level, patients whose 12-week clinician-rated QIDS score ≤ 5 or reduction in QIDS score $\geq 50\%$ will not move to further level. Chakraborty and Moodie (2013) gives a schematic of treatment assignment in the STAR*D study. At level 1, all patients received CIT. Patients who are eligible for level 2 treatment were randomized to one of the seven treatments including four switch options (venlafaxine[VEN], sertraline[SER], bupropion[BUP] and cognitive therapy[CT]) and three augment options (CT, BUP or buspirone[BUS] added to CIT). Patients without satisfactory response to CIT at level 1 and to CT at level 2 (either alone or in combination) could go to a supplementary level 2A where the patients were randomized to one of the two switch options (BUP, VEN). Patients who entered level 3 were randomized to receive one of the two switch options (mirtazapine[MRT], nortriptyline[NTP]) and two augment options (lithium[Li], thyroid hormone[THY]) while patients who entered level 4 were expected to receive one of the two switch options (tranylcypromine[TCP] or the combination of VEN + MRT).

The goal of constructing DTR is to improve treatment outcome as well as reduce medical resource waste by prescribing the treatment only when it is needed. An optimal DTR optimizes the expectation of a desired cumulative outcome over a population of interest (Laber et al., 2014). So an optimal DTR should maximize the expectation of treatment outcome over the population.

Currently, the methodologies in DTR mainly emerged from two different academic disciplines: reinforcement learning and causal inference. Methodologies originating from different fields use different terminologies. For example, the DTR and outcome in personalized medicine are respectively called policy and value in reinforcement learning. We will describe the termi-

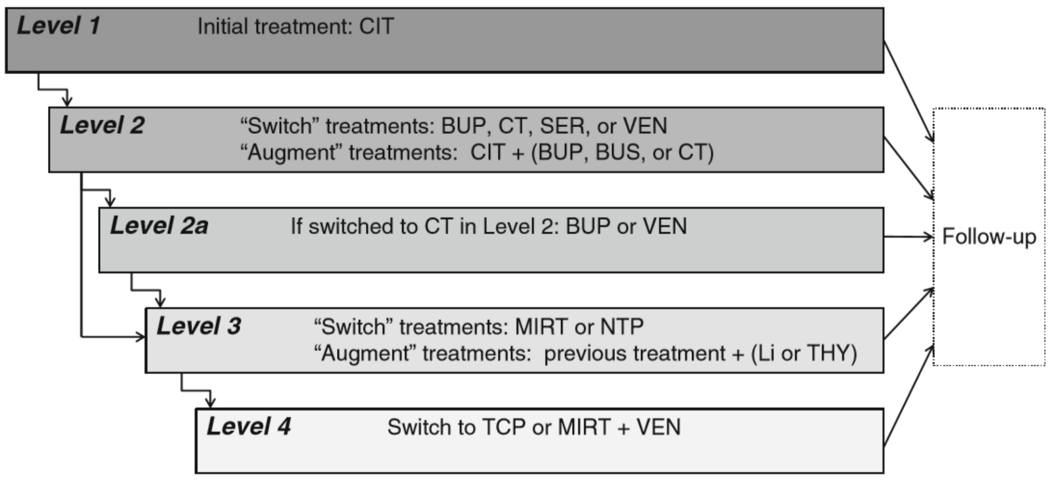

Level 1
Initial treatment: CIT
Level 2
"Switch" treatments: BUP, CT, SER, or VEN
"Augment" treatments: CIT + (BUP, BUS, or CT)
Level 2a
If switched to CT in Level 2: BUP or VEN
Level 3
"Switch" treatments: MIRT or NTP
"Augment" treatments: previous treatment + (Li or THY)
Level 4
Switch to TCP or MIRT + VEN
Follow-up

Figure 1.2: A schematic of treatment assignment in STAR*D (Chakraborty and Moodie, 2013)

nology in a coherent fashion and avoid the difference. Additionally, although different models use different techniques to obtain the optimal decision rule, a common framework is applicable to all models. In the remaining of this chapter, the potential outcomes framework and some common assumptions in DTR will be introduced first, then a conceptual overview of reinforcement learning and some existing popular models including both direct methods and indirect methods will be given.

1.2 Potential outcomes framework

In this section, the potential outcomes, also known as counterfactuals, and some necessary assumptions are briefly introduced.

Potential outcomes or counterfactuals is defined as a person's outcome had he followed a particular treatment regime, possibly different from the regime which he was actually observed to follow. The individual-level causal effect of a regime may then be viewed as the difference in outcomes if a person had followed that regime as compared to a placebo regime or a standard care protocol (Chakraborty and Moodie, 2013). For example, suppose we have two available treatments: a and a'. The individual-level causal effect should be the difference between outcomes under treatment a and a'. However, an individual will only take one treatment. Without loss of generality, assume the individual takes treatment a. Then the potential outcome Y_a under treatment a, is the observed outcome, and the potential outcome $Y_{a'}$ under treatment a', is unobservable. So individual-level causal effect actually cannot be observed. However, with some assumptions, the potential outcome can be connected to the observed outcome.

Before the statement of assumptions, some notations need to be introduced. The observable data trajectory for a participant in a T-stage treatment is denoted by $(X_1, A_1, X_2, \cdots, A_K, X_{T+1})$ where X_t is the covariate information at the beginning of stage t (before taking any treatment), A_t is the treatment at stage t. $\bar{X}_t = (X_1, \cdots, X_t)$ includes all covariate information up to stage t and $\bar{A}_t = (A_1, \cdots, A_t)$ denotes the treatment history up to stage t. Similarly, $\underline{X}_t = (X_t, \cdots, X_T)$ and $\underline{A}_t = (A_t, \cdots, A_T)$ respectively denote the covariate information and treatment assignment from stage t to the end of the treatment. $H_t = (\bar{X}_t, \bar{A}_{t-1})$ denotes all history information up to stage t. So a treatment regime d_t at stage t is a map from the space of history information to the space of treatments, $t = 1, 2, \cdots, T$. Additionally, all capital letters represent the random variables while the lowercase letters represent the realization of the corresponding random variables.

In general, three assumptions need to be made: consistency, no unmeasured confounders and positivity (Chakraborty and Moodie, 2013). The first two assumptions are required by the potential outcomes framework and the positivity assumption is required by the fact that the treatment or regime under consideration should be feasible.

Assumption 1 *Consistency: The potential outcomes under the observed treatment and the observed outcome agree.*

Assumption 2 *No unmeasured confounders: For any treatment sequence $\bar{a}_t$, and conditional on the history $H_t = (\bar{X}_t, \bar{A}_{t-1})$, treatment A_t is independent of future (potential) outcomes $X_{t+1}(\bar{a}_t)$, $X_{t+2}(\bar{a}_{t+1})$, $\cdots$,$X_T(\bar{a}_{T-1})$, $Y(\bar{a}_T)$, where $Y(\bar{a}_T)$ is the outcome under treatment sequence $\bar{a}_T$*

Assumption 3 *Positivity: Let $\pi_t(a_t|H_t)$ denote the conditional probability of receiving treat-*

ment a_t given H_t and let $f_t(H_t)$ denote the density function of H_t. Then for any t and for all histories h_t with $f(h_t) > 0$, $P[\pi_t(d_t(H_t)|H_t) > 0] = 1$.

The consistency assumption requires that the outcome for a given treatment is the same, regardless of the manner in which treatments are assigned. The no-unmeasured-confounder assumption allows us to view each stage as randomized trial if all relevant confounders are included. Positivity requires some subjects to have followed the regime $\bar{d}_T$, therefore the analysts are able to estimate the performance of the regime. (Chakraborty and Moodie, 2013)

1.3 Review of reinforcement learning

Reinforcement learning is characterized by a sequence of interactions between a learning agent and the environment it wants to learn about (Chakraborty and Moodie, 2013). The learning agent does not know what action should be taken but can only discover it by trying available actions. Beyond the agent and the environment, one can identify three features of a reinforcement learning system: policy, reward signal and value function (Sutton and Barto, 2018).

A policy defines the agent's behavior. It is a map from the space of states to the space of actions. Given a state, the policy will recommend an action for the agent to take. Reward signal is the goal in reinforcement learning . Each time after an agent takes some actions, the environment will update its state and send a reward to the agent. The agent's objective is to maximize the total rewards over a long run. While the reward signal indicates the immediate desirability, the value of a state with respect to a given policy, defined as the total amount of reward an agent can expect to accumulate over the future starting from the state, specifies the long-term desirability.

Elements of DTR include patients, treatment a_t, history information h_t, outcome y_t and treatment rule d. These elements of DTR respectively correspond to the agent, action, state, reward and policy in reinforcement learning. So the value of h_t under treatment rule d in DTR refers to the total expected future treatment outcome of a patient, starting with history information h_t, receiving treatment as the rule d suggests thereafter. More specifically,

$$V_t^d(h_t) = E_d\Big[\sum_{k=t}^{T} Y_k(H_k, A_k, X_{k+1})|H_t = h_t\Big], 1 \le t \le T$$

where Y_t is the outcome at stage t.

The optimal stage t value function for history h_t is

$$V_t^{opt}(h_t) = \max_{d\in\mathscr{D}} V_t^d(h_t)$$

The optimal value functions satisfy the Bellman equation (Bellman, 2010)

$$V_t^{opt}(h_t) = \max_{a_t\in\mathscr{A}_t} E\Big[Y_t(H_t, A_t, X_{t+1}) + V_{t+1}^{opt}(H_{t+1})|H_t = h_t, A_t = a_t\Big]$$

The marginal value of a policy d is the average value function under d averaged over all possible initial observations

$$V^d = E_{X_1}\Big[V^d(X_1)\Big] = E_d\Big[\sum_{k=1}^{T} Y_k(H_k, A_k, X_{k+1})\Big]$$

From now on, we will use terminologies treatment, outcome, history, treatment rule/regime instead of action, reward, state and policy, but we still use value function for measuring the performance of the DTR.

1.4 Review of indirect methods

Indirect approaches, as the name suggests, do not estimate the treatment regime directly. They instead first model the stage-specific conditional mean outcome and find the optimal treatment regime by maximizing the estimated conditional mean outcome. Popular indirect methods include Q-learning, A-learning, regret regression and G-estimation in structural nested mean model (Chakraborty and Moodie, 2013). These methods are originally developed for the observational data. We provide a detailed introduction of Q-learning and G-estimation. A-learning and regret regression fundamentally are extensions of the Q-learning.

1.4.1 Q-learning

Q-learning, which originates from reinforcement learning, characterizes DTR d by the Q-function defined as the total expected future outcome starting from a history h_t at stage t, taking treatment a_t and following the DTR d thereafter (Chakraborty and Moodie, 2013). Thus,

$$Q_t^d(h_t, a_t) = E\Big[Y_t(H_t, A_t, X_{t+1}) + V_{t+1}^d(H_{t+1})|H_t = h_t, A_t = a_t\Big]$$

The optimal Q-function at stage t is

$$Q_t^{opt}(h_t, a_t) = E\Big[Y_t(H_t, A_t, X_{t+1}) + V_{t+1}^{opt}(H_{t+1})|H_t = h_t, A_t = a_t\Big]$$

The difference between Q-function and value function lies in the fact that Q-function $Q_t^{opt}(h_t, a_t)$ measures the expected total outcomes associated with taking treatment a_t at stage t given the history h_t, and then following the optimal treatment regime thereafter, while the value function $V_t^{opt}(h_t)$ measures the outcome for patient with history h_t assuming that optimal treatment regime is followed in the future (Schulte et al., 2014). So Q-learning postulates model for Q-function and the optimal treatment at stage t is given by maximizing the estimated Q-function.

Illustration of Q-learning for two-stage case will be given first and the generalization to the T-stage is straightforward. For simplicity, it is assumed that the treatment is binary $A \in \{-1, 1\}$ and Q-function is modelled by a linear regression. More flexible models such as splines or neural network can also be applied to the Q-function (Chakraborty and Moodie, 2013).

In the two-stage case, the data is given by the trajectory $(X_1, A_1, X_2, A_2, X_3)$. So the histories $H_1 = X_1$ at the first stage and $H_2 = (X_1, A_1, X_2)$ at the second stage. Suppose Y_1 and Y_2 are respectively the outcome observed at the end of stage 1 and 2. In this case, $Y = Y_1 + Y_2$ is the total outcome. A two-stage DTR consists of two decision rules: $d_1(H_1)$ and $d_2(H_2)$ with each

$d_t(H_t) \in \{-1, 1\}$.

The optimal Q-function for two stages is defined as:

$$Q_2^{opt}(H_2, A_2) = E[Y_2|H_2, A_2]$$
$$Q_1^{opt}(H_1, A_1) = E[Y_1 + \max_{a_2} Q_2^{opt}(H_2, a_2)|H_1, A_1]$$

If the above two Q-functions were known, the optimal DTR (d_1^{opt}, d_2^{opt}) would be obtained by backwards induction in dynamic programming which first specifies the optimal treatment rule at the last stage and then moves from back to the front. That is,

$$d_t^{opt}(h_t) = \text{argmax}_{a_t} Q_t^{opt}(h_t, a_t), \quad t = 2, 1$$

Generally, the true Q-functions are unknown and because they are conditional expectations, a natural approach is to model them via regression models. For simplicity, linear regression is taken as an example.

Suppose the Q-function at stage t is modelled as

$$Q^{opt}(H_t, A_t; \beta_t, \phi_t) = \beta_t^T H_{t0} + (\phi_t^T H_{t1}) A_t$$

where $H_t = (H_{t0}, H_{t1})$. H_{t0} and H_{t1} denote the main effect of history and treatment effect of history respectively. So the Q-learning algorithm involves the following steps:

1. Stage 2 regression: $(\hat{\beta}_2, \hat{\phi}_2) = \text{argmin}_{\beta_2, \phi_2} \frac{1}{n} \sum_{i=1}^{n} (Y_{2,i} - Q_2^{opt}(H_{2,i}, A_{2,i}; \beta_2, \phi_2))^2$

2. Stage 2 optimal rule: $\hat{d}_2(h_2) = \text{argmax}_{a_2} Q_2(h_2, a_2; \hat{\beta}_2, \hat{\phi}_2)$

3. Stage 1 pseudo-outcome: $\hat{Y}_{1,i} = Y_{1,i} + \max_{a_2} Q_2^{opt}(h_{2,i}, a_2; \hat{\beta}_2, \hat{\phi}_2)$, $i = 1, \cdots, n$.

4. Stage 1 regression: $(\hat{\beta}_1, \hat{\phi}_1) = \text{argmin}_{\beta_1, \phi_1} \frac{1}{n} \sum_{i=1}^{n} (\hat{Y}_{1,i} - Q_1^{opt}(h_{1,i}, A_{1,i}; \beta_1, \phi_1))^2$

5. Stage 1 optimal rule: $\hat{d}_1(h_1) = \text{argmax}_{a_1} Q_1(h_1, a_1; \hat{\beta}_1, \hat{\phi}_1)$

Once the Q-functions have been estimated, the optimal decision rule at stage t is given by

$$\hat{d}_t^{opt}(h_t) = \text{argmax}_{a_t}\, Q_t^{opt}(h_t, a_t; \hat{\beta}_t, \hat{\phi}_t)$$

This process can be generalized to $T > 2$ stages in a similar way. Define $Q_{T+1}^{opt} \equiv 0$ and

$$Q_t^{opt}(H_t, A_t) = E[Y_t + \max_{a_{t+1}} Q_{t+1}^{opt}(H_{t+1}, a_{t+1}) | H_t, A_t], \quad t = 1, \cdots, T$$

Stage specific Q-function can be parameterized as

$$Q_t^{opt}(H_t, A_t; \beta_t, \phi_t) = \beta_t^T H_{t0} + (\phi_t^T H_{t1}) A_t, \quad t = 1, \cdots, T$$

For $t = T, T-1, \cdots, 1$, the regression parameters are estimated by backwards induction

$$(\hat{\beta}_t, \hat{\phi}_t) = \text{argmin}_{\beta_t, \phi_t} \frac{1}{n} \sum_{i=1}^{n} \left\{ Y_{ti} + \max_{a_{t+1}} Q_{t+1}^{opt}(H_{t+1}, a_{t+1}; \hat{\beta}_{t+1}, \hat{\phi}_{t+1}) - Q_t^{opt}(H_{ti}, A_{ti}; \beta_t, \phi_t) \right\}^2$$

Therefore, the estimated optimal DTR is $(\hat{d}_1^{opt}, \cdots, \hat{d}_T^{opt})$ where

$$\hat{d}_t^{opt}(h_t) = \text{argmax}_{a_t}\, Q_t^{opt}(h_t, a_t; \hat{\beta}_t, \hat{\phi}_t), \quad t = 1, \cdots, T$$

1.4.2 G-estimation in structural nested mean model

Q-learning directly models the conditional mean outcomes. When the model for the Q-function is misspecified, the resulting estimators for the true optimal regime can be inconsistent (Zhao et al., 2015). Structural nested mean model, unlike Q-learning, models contrasts of conditional mean outcomes and thus could be more robust to model misspecification (Chakraborty and Moodie, 2013; Robins, 2004).

An optimal blip-to-reference function $\gamma_t(h_t, a_t)$ at any stage t is defined as the expected difference in outcome when using a reference regime d_t^{ref} instead of a_t at stage t in persons with treatment and covariate history h_t who subsequently receive the optimal regime $\underline{d}_{t+1}^{opt}$

$$\gamma_t(h_t, a_t) = E\Big[Y(\bar{a}_t, \underline{d}_{t+1}^{opt}) - Y(\bar{a}_{t-1}, d_t^{ref}, \underline{d}_{t+1}^{opt})|H_t = h_t\Big]$$

where "optimal" refers to treatment subsequent to stage t and "blip" refers to the single-stage change in treatment at stage t.

Suppose $r_t(h_t, a_t)$ is specified up to a parameter vector ψ. The optimal regime is then given by

$$d_t^{opt}(h_t; \psi) = \text{argmax}_{a_t}\, \gamma_t(h_t, a_t; \psi)$$

for $t = 1, \cdots, T$. Once an estimator of ψ is constructed, the estimated optimal regime is obtained by maximizing the estimated optimal blip-to-reference function. G-estimation is proposed for estimating ψ in the optimal blip function. Define $G_t(\psi)$ as

$$\begin{aligned} G_t(\psi) &= Y + \sum_{k=t}^{T} \Big[\gamma_k(h_k, d_k^{opt}; \psi) - \gamma_k(h_k, a_k; \psi)\Big] \\ &= Y + \sum_{k=t}^{T} E\Big[Y(\bar{a}_{k-1}, \underline{d}_k^{opt}) - Y(\bar{a}_k, \underline{d}_{k+1}^{opt})|H_k = h_k\Big]. \end{aligned}$$

$G_t(\psi)$ is a person's outcome adjusted by the expected difference between the average outcome for patients who received a_t and patients who were given the optimal treatment at the start of stage t, where all patients had the same treatment and covariate history up to the start of stage $t-1$ and were subsequently treated optimally. It is proved that $G_t(\psi)$ equals the expectation of counterfactual outcome (Robins, 2004). Consider $S_t(A_t) = s_t(H_t, A_t)$ with parameter α as a vector-valued function of $\dim(\psi)$ chosen by the analyst to contain the variables thought to interact with treatment (Chakraborty and Moodie, 2013)

$$U(\psi, \alpha) = \sum_{t=1}^{T} G_t(\psi)\Big\{S_t(A_t) - E\Big[S_t(A_t)|H_t; \alpha\Big]\Big\},$$

then $U((\psi), \alpha)$ is an unbiased estimating function since $E\Big[U(\psi, \alpha)\Big] = 0$. A more efficient estimating function can be obtained by postulating appropriate model for $E\Big[G_t(\psi)|H_t\Big]$ (Chakraborty and Moodie, 2013). The refined estimating function is

$$U(\psi, \eta(\psi), \alpha) = \sum_{t=1}^{T} \Big(G_t(\psi) - E\Big[G_t(\psi)|H_t; \eta\Big]\Big)\Big\{S_t(A_t) - E\Big[S_t(A_t)|H_t; \alpha\Big]\Big\}$$

It is proved that the resulting estimator $\hat{\psi}$ is consistent if either $E\Big[G_t(\psi)|H_t; \eta\Big]$ or $p_t(A_t = 1|H_t; \alpha)$ is correctly modeled (Robins, 2004). This property is called doubly-robustness.

1.5 Review of direct methods

Direct methods, also known as policy search methods, directly estimate the marginal mean $E_d(Y)$ for all DTRs in a pre-specified class and then maximize the estimated marginal mean over all possible DTRs to obtain an estimated optimal DTR (Laber et al., 2014). Popular direct methods include inverse probability weighting and outcome weighted learning.

1.5.1 Inverse probability weighting

Inverse probability weighting method investigates the optimal treatment regimes in a pre-specified class of treatment regimes. It estimates the value function of each possible treatment regimes and choose the one with the maximum value.

Suppose d is an arbitrary regime under evaluation. When d is unobservable, the expectation of potential outcome can be estimated by changing probability measure under the assumption that P_d is absolutely continuous with respect to P_π, where P_d, P_π are the probability measure under regime d and exploration policy π. Absolute continuity indicates that any trajectory which can be observed under regime d has a positive probability of occurring under the exploration regime π. Then the value function can be written as

$$V^d = E_d Y = \int Y dP_d = \int Y\Big(\frac{dP_d}{dP_\pi}\Big) dP_\pi \tag{1.1}$$

where $\frac{dP_d}{dP_\pi}$ is the Radon-Nikodym derivative denoted by $w_{d,\pi}$ and $w_{d,\pi} = \prod_{t=1}^{T} \frac{I[A_t=d_t(H_t)]}{\pi_t(A_t|H_t)}$ with $\pi_t(A_t|H_t)$ being the conditional treatment probability. A natural estimate of V^d is its empirical value $\hat{V}^d$

$$\hat{V}^d = \mathbb{P}_n\Big[w_{d,\pi} Y\Big]$$

where $\mathbb{P}_n$ is the empirical average operator. By normalizing the weights, the inverse probability of treatment weighted (IPTW) estimator can be obtained as

$$\hat{V}^d_{IPTW} = \frac{\mathbb{P}_n[w_{d,\pi} Y]}{\mathbb{P}_n[w_{d,\pi}]}$$

For single stage, an augmented, doubly-robust estimator is the augmented inverse probability of treatment weighting (AIPTW), given by

$$\hat{V}^d_{AIPTW} = \mathbb{P}_n\Big\{\frac{I[A = d(H)Y}{\pi_c(H)} - \frac{I[A = d(H)] - \pi_c(H)}{\pi_c(H)} m(H)\Big\}$$

where

$$\pi_c(H) = \pi(H) I[d(H) = 1] + (1 - \pi(H)) I[d(H) = -1],$$
$$m(H) = \mu(1, H) I[d(H) = 1] + \mu(-1, H) I[d(H) = -1],$$

$\mu(A, H)$ is the estimated mean outcome for covariate H and treatment A and $\pi(H)$ is the estimated propensity score.

Once the values for all regimes in the pre-specified class of DTRs are estimated, the optimal DTR can be chosen as the one with the largest empirical value.

1.5.2 Outcome weighted learning

Outcome weighted learning (OWL) casts the original problem as a weighted classification problem. Different from the method introduced in section 1.5.1, OWL does not search the value of every possible treatment regime. It instead minimizes the weighted misclassification error rate for assigning patients to the observed treatment (Zhao et al., 2012).

A single stage treatment regime is employed as an illustration. In this case, the history H only includes prognostic value X. It is known in equation (1.1) that

$$d^{opt} = \text{argmax}_{d \in \mathscr{D}} E\Big[\frac{I(A = d(X))}{\pi(A, X)} Y\Big]$$

It is equivalent to

$$d^{opt} = \text{argmin}_{d \in \mathscr{D}} E\Big[\frac{I(A \neq d(X))}{\pi(A, X)} Y\Big],$$

which can be viewed as a weighted misclassification error and therefore, can be solved by classification techniques from machine learning (Zhao et al., 2012). It is known that minimizing the weighted misclassification error requires the weights to be nonnegative and thus the outcome should be nonnegative. Outcome weighted learning mainly uses support vector machine for solving the classification problem. So $d^{opt}(X) = \text{sign}(f(x))$ for some decision function f (Zhao et al., 2012). The optimal f^* can be obtained by minimizing

$$n^{-1} \sum_{i=1}^{n} \frac{Y_i}{\pi(A_i, X_i)} \phi(A_i f(X_i)) + \lambda_n ||f||^2$$

where $\phi(u) = (1 - u)^+$ is the hinge loss function, $x^+ = \max(x, 0)$ and $||f||$ is a norm for f.

Consider f as a linear function, $f(x) = \langle \beta, x \rangle + \beta_0$ where $\langle \cdot, \cdot \rangle$ denotes the inner product in

Euclidean space. As usual, the minimizing problem can be rewritten as

$$\begin{aligned} &\max_{\beta,\beta_0,\|\beta\|=1} C \\ \text{subject to} \quad & A_i(\langle\beta, X_i\rangle + \beta_0) \geq C(1-\xi_i) \\ & \xi_i \geq 0, \sum \frac{Y_i}{\pi_i}\xi_i < s \end{aligned}$$

by introducing slack variables ξ_i and $C > 0$ as the classifier margin. $\pi_i = \pi I(A_i = 1) + (1 - \pi)I(A_i = -1)$ and s is a constant depending on λ_n (Zhao et al., 2012). It is equivalent to

$$\begin{aligned} &\min \frac{1}{2}\|\beta\|^2 \\ \text{subject to} \quad & A_i(\langle\beta, X_i\rangle + \beta_0) \geq (1-\xi_i) \\ & \xi_i \geq 0, \sum \frac{Y_i}{\pi_i}\xi_i < s \end{aligned}$$

that is,

$$\begin{aligned} &\min \frac{1}{2}\|\beta\|^2 + \kappa \sum_{i=1}^{n} \frac{Y_i}{\pi_i}\xi_i \\ \text{subject to} \quad & A_i(\langle\beta, X_i\rangle + \beta_0) \geq (1-\xi_i), \xi_i \geq 0 \end{aligned}$$

The corresponding Lagrange function is

$$\frac{1}{2}\|\beta\|^2 + \kappa \sum_{i=1}^{n} \frac{Y_i}{\pi_i}\xi_i - \sum_{i=1}^{n} \alpha_i \Big\{ A_i(X_i^T\beta + \beta_0) - (1-\xi_i) \Big\} - \sum_{i=1}^{n} \mu_i\xi_i$$

with $\alpha_i \geq 0$, $\mu_i \geq 0$. After some simple mathematical operations, the dual problem can be written as

$$\max_{\alpha} \sum_{i=1}^{n} \alpha_i - \frac{1}{2} \sum_{i=1}^{n} \sum_{j=1}^{n} \alpha_i\alpha_j A_i A_j \langle X_i, X_j\rangle$$

$$\text{subject to} \quad 0 \leq \alpha_i \leq \kappa \frac{Y_i}{\pi_i}, \sum_{i=1}^{n} \alpha_i A_i = 0.$$

Finally, the estimator $\hat{\beta}$ is obtained by

$$\hat{\beta} = \sum_{\hat{\alpha}>0} \hat{\alpha} A_i X_i$$

and $\hat{\beta}_0$ is solved using the margin points subject to the Karush-Kuhn-Tucker conditions (Hastie et al., 2009).

Consider f as a nonlinear function in the reproducing kernel Hilbert space(RKHS) $\mathcal{H}_K$, $f(x) = \sum_{i=1}^{m} \alpha_i K(x, x_i)$, where K is a kernel function. It is known that the optimal decision function is given by

$$\sum_{i=1}^{n} \hat{\alpha}_i A_i K(x, x_i) + \hat{\beta}_0$$

where $(\hat{\alpha}_1, ..., \hat{\alpha}_n)$ is obtained by solving the dual problem

$$\max_{\alpha} \sum_{i=1}^{n} \alpha_i - \frac{1}{2} \sum_{i=1}^{n} \sum_{j=1}^{n} \alpha_i \alpha_j A_i A_j k(x_i, x_j)$$

$$\text{subject to} \quad 0 \leq \alpha_i \leq \kappa \frac{Y_i}{\pi_i}, \sum_{i=1}^{n} \alpha_i A_i = 0$$

1.6 Objectives and organization

Many methods in the literature focuses only on single stage and binary treatment. However, in reality it is common that patients and clinicians have more than two choices for treatment assignment. For chronic diseases such as depression or alcohol addiction, patients always receive long-time therapy involving more than one single decision point. In addition, in the field of DTR it is assumed that larger outcome is preferred. However, in some cases this prerequisite may not be satisfied. For example, in STAR*D, the original outcome is QIDS score of which larger values correspond to higher severity, thus smaller outcome is preferred. To make it consistent, researchers always take the negative of QIDS score as the new outcome. The negative outcome will cause problems when outcome weighted learning by uses hinge loss function to approximate the $0-1$ loss function and to solve the optimization problem by convex optimization techniques. The convexity will not be valid for negative outcomes causing the optimization procedure problematic.

So the main objective of this thesis is to explore the extension of the outcome weighted learning to more general settings such as multi-armed treatments and negative treatment outcome. The rest of the thesis is organized as follows. In Chapter 2, we propose an angle-based multicategory outcome weighted learning using multicategory support vector machine. The loss function is modified to allow for negative treatment outcome. To ensure the consistency, two further modifications are made: we either make assumptions on treatment effect or constrain the range of the decision function. Extension to multiple stages is also considered. In Chapter 3, we propose a method based on neural decision tree. The neural network is implemented to increase prediction accuracy while a reconstructed and pruned tree based on prediction result from neural network is used to maintain interpretability. The conclusion remarks and future work are described in Chapter 4. R code for the method proposed in Chapter 2 and Python code for the method proposed in Chapter 3 are attached in the Appendix.

Chapter 2

Multicategory Outcome Weighted Learning

2.1 Introduction

In this chapter, we investigate the optimal treatment rule in the case of multi-treatment with potential negative outcome based on outcome weighted learning. We also extend the multicategory outcome weighted learning to multiple stages.

We propose a multicategory outcome weighted learning method based on an angle-based multicategory support vector machine. A surrogate loss function is used when the weight is negative to maintain the convexity of the loss function so that the optimization can be solved through coordinate descent method. A direct modification without any constraint may not guarantee the Fisher consistency of the resulting classifier, so we propose two solutions: either make reasonable and feasible assumptions on treatment effect or bound the range of the decision function. The algorithm is outlined and the numerical studies are conducted to assess the performance of the proposed methods. A real data application is employed for illustration.

The rest of this chapter is organized as follows. Section 2.2 describes notations and introduces angle-based framework of multicategory classification. In section 2.3, the proposed multicategory outcome weighted learning is presented for treatment procedure with either single stage or multiple stages. The optimization algorithm is also described. Simulation study and the application to **STAR*D** data are carried out to assess the performance of the proposed model and to illustrate the use of the proposed method in section 2.4. The chapter is concluded in section 2.5. The proofs of the theorems are left in the Appendix in this chapter.

2.2 Notation and framework

Suppose there are T stage treatments for patients with K_t treatment options at stage t, $1 \leq t \leq T$. Let Y_t denote the observed treatment outcome at stage t and the overall outcome for the patient is defined as $Y = \sum_{t=1}^{T} Y_t$. Assume that larger values of outcome are preferred and each Y_t can be either positive or negative. The objective is to maximize the expected overall outcome $E_d(Y)$ under regime d. Let $A_t \in \mathcal{A}_t = \{1, 2, \cdots, K_t\}$ be the treatment assignment received by the patient at stage t, $X_t = (X_{t1}, \cdots, X_{tp})$ be the covariate information of the patient at stage t. $\bar{X}_t$ and $\bar{A}_t$ are used to denote the covariate information and treatment history up to stage t. The history at stage t is then $H_t = (\bar{X}_t, \bar{A}_{t-1})$ and let $\pi_t(A_t, H_t) = Pr(A = A_t | H = H_t)$ be the probability of receiving treatment A_t at stage t for a patient with history H_t. A dynamic treatment regime d is a vector $d = (d_1, \cdots, d_T)$ where d_t is the optimal treatment at stage t and $d_t^s = (d_t, \cdots, d_s)$, $\forall t < s$ are defined similarly. Besides, Y_t^i, Y^i,A_t^i, X_t^i refer to the observed value of the corresponding variables for patient i.

The outcome weighted learning (OWL) is employed to identify the optimal treatment regimes for patients where classification methods are utilized to formulate the assignment of treatments. When K-category treatments are considered, K-category classifiers are needed.

In the literature, many popular K-category classifiers use K classification functions and impose sum-to-zero constraint on the K classification functions to reduce the function space (Lee et al., 2004; Liu and Yuan, 2011). It is shown that constructing K functions with sum-to-zero constraint can be inefficient and an angle-based classification method for any binary large-margin loss function has been proposed to overcome this problem (Zhang and Liu, 2014).

The angle-based classification method can be described as follows. Define a specific simplex $\mathbf{W}$ using K vectors $W_1, \cdots, W_K$ in the $(K-1)$-dimensional space. $W_1, \cdots, W_K$ are defined as:

$$W_j = \begin{cases} (K-1)^{-1/2}\mathbf{1}_{K-1} & j = 1 \\ -(1+K^{1/2})/(K-1)^{3/2}\mathbf{1}_{K-1} + \{K/(K-1)\}^{1/2}\mathbf{e}_{j-1} & 2 \leq j \leq K \end{cases} \tag{2.1}$$

where $\mathbf{1}_{K-1}$ is a $(K-1)$-dimensional vector with all elements equal to 1 and $\mathbf{e}_j$ is a $(K-1)$-dimensional vector such that all elements is 0 except that the j-th element is 1.

Based on the definition, $\mathbf{W}$ consists of K unit directions in the $(K-1)$-dimensional space. The angles between any two directions W_j, $W_{j'}$ are equal. A vector in the $(K-1)$-dimensional space will have K angles with respect to those K directions. In the angle-based framework, a covariate vector X is mapped to a $(K-1)$-dimensional vector function $f(X) = (f_1(X), f_2(X), \cdots, f_{K-1}(X))$. The predicted class label j of X is determined by the class of which W_j has the smallest angle with $f(X)$. Since the norm of W_j, $j = 1, \cdots, K$ are equal, the vector W_j that has the smallest angle with $f(X)$ is the one which has the largest inner product with $f(X)$. So given any covariate vector X, predicting its class label is equivalent to finding $\text{argmax}_{1\leq j\leq k}\langle f(X), W_j\rangle$ and $f(X)$ automatically satisfies $\sum_{j=1}^{K}\langle f(X), W_j\rangle = 0$. It is believed that the angle-based classification method enjoys a better geometric interpretation of the least angle prediction rule, a lower computational cost as well as some good theoretical properties (Zhang and Liu, 2014). The benefit is that in stead of K classification functions with sum-to-zero constraint in most K-category classifiers, only $K-1$ functions are needed for angle-based classification methods and with the specific simplex $\mathbf{W}$ defined as in equation (2.1) the $K-1$ functions automatically satisfy $\sum_{j=1}^{K}\langle f(X), W_j\rangle = 0$ making the optimization procedure more efficient.

2.3 Method framework

2.3.1 Single stage

In the case of single stage, the notation K_t,Y_t,A_t, $\pi_t(A_t, H_t)$ is simplified as K, Y, A, $\pi(A, X)$.

Recall that in OWL,

$$d^{opt} = \text{argmin}_{d\in\mathscr{D}} E\Big[\frac{I(A \neq d(X))}{\pi(A, X)} Y\Big] \tag{2.2}$$

It can be viewed as weighted misclassification error. Due to the non-smoothness of the indicator function, different surrogate loss functions have been proposed in the literature (Zhao et al., 2012; Lou et al., 2018; Fu et al., 2019). For our method, we use the loss function in reinforced multicategory support vector machine (Liu and Yuan, 2011; Zhang et al., 2016) and its extension to angle-based framework is

$$V(f(X), A) = \gamma[(k-1) - \langle f(X), W_A\rangle]_+ + (1-\gamma)\sum_{a\neq A}[1 + \langle f(X), W_a\rangle]_+ \tag{2.3}$$

where A is the class label for the patient with covariate vector X and $0 \leq \gamma \leq 1$. $V(f(X), A)$ is a linear combination of two common loss functions in multicategory support vector machine and γ controls how these two loss functions are combined. When $\gamma = 0$, it reduces to the vector hinge loss function $\sum_{a\neq A}[1 + \langle f(X), W_a\rangle]_+$ while when $\gamma = 1$ it becomes naive hinge loss multiplied by $K - 1$, that is $[(K-1) - \langle f(X), W_A\rangle]_+$. The optimization problem becomes

$$\underset{f\in RKHS}{\text{argmin}} \Big\{\frac{1}{n}\sum_{i=1}^{n} \frac{Y_i}{\pi(A_i, X_i)} V(f(X_i), A_i)) + J(f)\Big\} \tag{2.4}$$

where RKHS denotes the Reproducing Kernel Hilbert Space and $J(f)$ is the penalty term for f. When the outcome is negative the convexity of the objective function cannot be maintained.

To overcome the problem, we rewrite the right hand side of equation (2.2) as

$$\begin{aligned}
&\operatorname{argmin}_{d\in\mathscr{D}} E\Big[\frac{|Y|I(Y\geq 0)}{\pi(A,X)}I(A\neq d(X)) - \frac{|Y|I(Y<0)}{\pi(A,X)}I(A\neq d(X))\Big] \\
&= \operatorname{argmin}_{d\in\mathscr{D}} E\Big[\frac{|Y|I(Y\geq 0)}{\pi(A,X)}I(A\neq d(X)) + \frac{|Y|I(Y<0)}{\pi(A,X)}I(A= d(X))\Big]
\end{aligned} \tag{2.5}$$

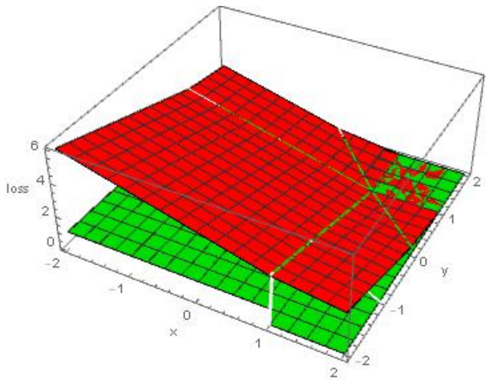
We modify the loss function similarly as in Chen et al. (2018).

$$V_Y(f(X),A) = \begin{cases} \gamma[(K-1)-\langle f(X),W_A\rangle]_+ + (1-\gamma)\sum_{a\neq A}[1+\langle f(X),W_a\rangle]_+ & Y\geq 0 \\ \gamma[(K-1)+\langle f(X),W_A\rangle]_+ + (1-\gamma)\sum_{a\neq A}[1-\langle f(X),W_a\rangle]_+ & Y<0 \end{cases} \tag{2.6}$$

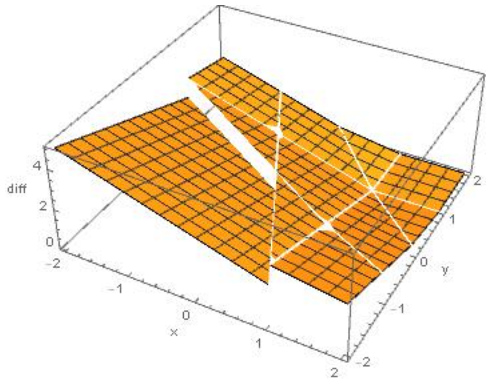
Thus, equation (2.4) can be modified as

$$\begin{aligned}
\operatorname*{argmin}_f\Big\{\frac{1}{n}\sum_{i=1}^n\Big\{&\frac{|Y_i|I(Y_i\geq 0)}{\pi(A_i,X_i)}\gamma[(k-1)-\langle f(X),W_A\rangle]_+ + (1-\gamma)\sum_{a\neq A}[1+\langle f(X),W_a\rangle]_+ \\
&+\frac{|Y_i|I(Y_i< 0)}{\pi(A_i,X_i)}\gamma[(k-1)+\langle f(X),W_A\rangle]_+ + (1-\gamma)\sum_{a\neq A}[1-\langle f(X),W_a\rangle]_+\Big\} + J(f)\Big\}
\end{aligned} \tag{2.7}$$

When the outcome is positive, $V_Y(f(X),A)$ reduces to $V(f(X),A)$ and when the outcome is negative, $V_Y(f(X),A)$ is a tight convex upper bound of $I(A=d(X))$. To compare $V_Y(f(X),A)$ when $Y<0$ with the indicator function $I(A=d(X))$, define the vector $\mathbf{g}=(\langle f(x),W_1\rangle,\cdots,\langle f(x),W_K\rangle)$. It is a vector function of x, but to simplify the notation we only use $\mathbf{g}$ when there is no confusion. The component of $\mathbf{g}$ is g_j, $j=1,\cdots,K$ satisfying $\sum_{j=1}^K g_i=0$. The indicator function $I(A=d(X))$ can then be written as $I(g_A>g_1,\cdots,g_A>g_{A-1},g_A>g_{A+1},\cdots,g_A>g_K)$. Figure 2.1 shows a picture of the effect of the modified loss function when $K=3$, $\gamma=0.5$. In this case, $\mathbf{g}$ is written as $\mathbf{g}=(x,y,z)$ and by symmetry we assume the true class label is 3. We should note that in figure 2.1a, for the interval $[1,+\infty)\times[1,+\infty)$ there is a mixture of both red and green color. It does not mean our modified loss function cannot bound the indicator function for this interval. It is due to the way that our plotting software Mathematica displays overlapped region. It is clearer in figure 2.1b, in the interval $[1,+\infty)\times[1,+\infty)$ the difference of the two functions remains 0.



(a)



(b)

Figure 2.1: Plots of the effect of the modified loss function. In panel (a), the green plane is the indicator function $I(A = d(X))$ and the red plane is the proposed modified loss function. Panel (b) is the plot of their difference.

2.3.1.1 Fisher consistency

Before presenting our results about consistency of the optimization problem (2.7), we introduce some assumptions and notations that is specific to this section.

Define conditional reward R_j to be $R_j(x) = E[Y|X = x, A = j]$. Its positive and negative parts are respectively defined as $R_j^+(x) = E[YI(Y \geq 0)|X = x, A = j]$ and $R_j^-(x) = E[YI(Y < 0)|X = x, A = j]$. Define the conditional risk function for decision function f as $r(f|X = x) = E\Big[\frac{|Y|}{\pi(A,X)} V_Y(f, A)|X = x\Big]$

Fisher consistency is an important property in classification literature. Instead of solving the original problem (2.2) we are solving the surrogate problem (2.7). Fisher consistency ensures that the solution to the surrogate problem (2.7) can lead to the solution to the original problem (2.2) given the whole population. A Fisher consistent classifier can achieve the best performance asymptotically. Without any modification, the classifier based on loss function (2.3) is Fisher consistent for $0 \leq \gamma \leq 0.5$ (Zhang et al., 2016). However, if the modification of the loss function is used as in equation (2.6) is used, it becomes more complicated with regard to the consistency. To ensure the consistency, in addition to the three assumptions stated in section 1.2, the following further assumptions need to be imposed

Assumption 4 *For a patient with covariate vector x, denote the best and worst treatments by i* *and j respectively. Then $R_i^+(x) > R_t^+(x) > R_j^+(x)$ and $R_i^-(x) > R_t^-(x) > R_j^-(x)$, for $\forall t \neq i, j$. Also, $R_s(x) = R_s^+(x) + R_s^-(x) > 0$, for $\forall 1 \leq s \leq K$.*

Assumption 5 *For any treatment s, $R_s^+(x) < \sum_{t \neq s} R_t^+(x)$ and $R_s^-(x) > \sum_{t \neq s} R_t^-(x)$.*

Assumptions 4 and 5 are reasonable in the following sense. First, for any treatment s, $R_s^+(x)$ and $|R_s^-(x)|$ respectively measure the beneficial and adverse effect of treatment s. The larger $R_s^+(x)$ and $|R_s^-(x)|$ are, the more beneficial or adverse effects the treatment s have on the patient. Assumption 4 requires that the best treatment should have a large probability of beneficial effect and a small probability of adverse effect while it is contrary for the worst treatment. Second, assumption 5 requires all treatments under consideration are comparable. If, for a treatment s, $R_s^+(x) > \sum_{t \neq s} R_t^+(x)$ or $R_s^-(x) < \sum_{t \neq s} R_t^-(x)$, it means the treatment s is a dominantly best or worst treatment which can be identified directly. With assumption 4 and 5, we can obtain the Fisher consistency in the next theorem.

Theorem 2.3.1 *If assumptions 4 and 5 are valid, the method of finding optimal treatment rule using classifier based on the loss function (2.6) is Fisher consistent for $\gamma \in [0, 0.5]$.*

If assumptions 4 and 5 do not hold, we can further modify the loss function to make it applicable to all cases. The result is given in theorem 2.3.2.

Theorem 2.3.2 *For any $\gamma \in [0, 1]$, if the constraint $\langle f, W_j \rangle \geq -\frac{1}{K-1}$ for any $j = 1, \cdots, K$ is valid, then the method of finding optimal treatment rule using the classifier based on the loss function (2.6) is still Fisher consistent.*

The proofs of theorem 2.3.1 and 2.3.2 are given in the Appendix in this chapter. To make it more explicit, We refer the loss function in theorem 2.3.2 as $V_Y^c(f(X), A)$ where c denotes the constraint $\langle f, W_j \rangle \geq -\frac{1}{K-1}$ with the loss function $V_Y(f(X), A)$. Since $V_Y^c(f(X), A)$ has an extra constraint compared with $V_Y(f(X), A)$, it is expected to be less efficient.

2.3.1.2 Computation details

In this section, we derive the dual problem of the optimization (2.7). We focus on both linear and nonlinear case with L_2 penalty, and present the results for $V_Y(f(X), A)$ and $V_Y^c(f(X), A)$ separately.

Loss function 1: $V_Y(f(X), A)$

a. Linear case

For linear case, assume that $f_q(x) = x^T\beta_q, q = 1, \cdots, K-1$, where x is the covariate vector with constant 1 included and β_q is the coefficient parameter vector. The penalty term $J(f)$ is defined as $J(f) = \sum_{q=1}^{K-1}\beta_q^T\beta_q$. By introducing the slack variables $\xi_{ij}, i = 1, \cdots, n; j = 1, \cdots, K$, the optimization problem can be written as

$$\begin{aligned}
&\min_{\beta_q, \xi_{ij}} \frac{n\lambda}{2}\sum_{q=1}^{K-1}\beta_q^T\beta_q + \sum_{i:y_i\geq 0}\frac{y_i}{\pi(A_i, x_i)}[(1-\gamma)\sum_{j\neq A_i}\xi_{ij} + \gamma\xi_{i,A_i}] \\
&\qquad - \sum_{i:y_i<0}\frac{y_i}{\pi(A_i, x_i)}[(1-\gamma)\sum_{j\neq A_i}\xi_{ij} + \gamma\xi_{i,A_i}]
\end{aligned}$$

$$\begin{aligned}
\text{Subject to}\quad & \xi_{ij} \geq 0 \quad (i = 1, \cdots, n; j = 1, \cdots, K) \\
& \xi_{i,A_i} + \langle f(x_i), W_{A_i}\rangle - (K-1) \geq 0 \quad (i : y_i \geq 0) \\
& \xi_{ij} - \langle f(x_i), W_j\rangle - 1 \geq 0 \quad (i : y_i \geq 0; j \neq A_i) \\
& \xi_{i,A_i} - \langle f(x_i), W_{A_i}\rangle - (K-1) \geq 0 \quad (i : y_i < 0) \\
& \xi_{ij} + \langle f(x_i), W_j\rangle - 1 \geq 0 \quad (i : y_i < 0; j \neq A_i)
\end{aligned}$$

The Lagrangian function L can be defined as

$$\begin{aligned}
L =& \frac{n\lambda}{2}\sum_{q=1}^{K-1}\beta_q^T\beta_q + \sum_{i:y_i\geq 0}\frac{y_i}{\pi(A_i, x_i)}[(1-\gamma)\sum_{j\neq A_i}\xi_{ij} + \gamma\xi_{i,A_i}] \\
& - \sum_{i:y_i<0}\frac{y_i}{\pi(A_i, x_i)}[(1-\gamma)\sum_{j\neq A_i}\xi_{ij} + \gamma\xi_{i,A_i}] - \sum_{i=1}^{n}\sum_{j=1}^{K}\tau_{ij}\xi_{ij}
\end{aligned}$$

$$
\begin{aligned}
&- \sum_{i:y_i\geq 0} \alpha_{i,A_i}[\xi_{i,A_i} + \langle f(x_i), W_{A_i}\rangle - (K-1)] - \sum_{i:y_i\geq 0}\sum_{j\neq A_i} \alpha_{ij}[\xi_{ij} - \langle f(x_i), W_j\rangle - 1] \\
&- \sum_{i:y_i<0} \alpha_{i,A_i}[\xi_{i,A_i} - \langle f(x_i), W_{A_i}\rangle - (K-1)] - \sum_{i:y_i<0}\sum_{j\neq A_i} \alpha_{ij}[\xi_{ij} + \langle f(x_i), W_j\rangle - 1] \\
&= \frac{n\lambda}{2}\sum_{q=1}^{K-1}\beta_q^T\beta_q + (K-1)\sum_{i=1}^{n}\alpha_{i,A_i} + \sum_{i=1}^{n}\sum_{j\neq A_i}\alpha_{ij} + \sum_{i:y_i\geq 0}\sum_{j=1}^{K}[c_{ij} - \tau_{ij} - \alpha_{ij}]\xi_{ij} \\
&- \sum_{i:y_i<0}\sum_{j=1}^{K}[c_{ij} + \tau_{ij} + \alpha_{ij}]\xi_{ij} - \sum_{i:y_i\geq 0}\alpha_{i,A_i}\langle f(x_i), W_{A_i}\rangle + \sum_{i:y_i<0}\alpha_{i,A_i}\langle f(x_i), W_{A_i}\rangle \\
&+ \sum_{i:y_i\geq 0}\sum_{j\neq A_i}\alpha_{ij}\langle f(x_i), W_j\rangle - \sum_{i:y_i<0}\sum_{j\neq A_i}\alpha_{ij}\langle f(x_i), w_j\rangle
\end{aligned}
$$

where α_{ij}, τ_{ij}, $i = 1, \cdots, n; j = 1, \cdots, K$ are Lagrangian multipliers and $c_{ij} = \frac{y_i}{\pi(A_i,x_i)}[(1-\gamma)I(j \neq A_i) + \gamma I(j = A_i)]$. By solving $\frac{\partial L}{\partial \beta_q} = 0$ and $\frac{\partial L}{\partial \xi_{ij}} = 0$, we can obtain that

$$c_{ij} - \tau_{ij} - \alpha_{ij} = 0 \quad \text{for} \quad i : y_i \geq 0 \tag{2.8}$$

$$c_{ij} + \tau_{ij} + \alpha_{ij} = 0 \quad \text{for} \quad i : y_i < 0 \tag{2.9}$$

$$\beta_q = \frac{1}{n\lambda}\Big[\sum_{i:y_i\geq 0}\alpha_{i,A_i}W_{A_i,q}x_i - \sum_{i:y_i<0}\alpha_{i,A_i}W_{A_i,q}x_i - \sum_{i:y_i\geq 0}\sum_{j\neq A_i}\alpha_{ij}W_{jq}x_i + \sum_{i:y_i<0}\sum_{j\neq A_i}\alpha_{ij}W_{jq}x_i\Big] \tag{2.10}$$

where W_{jq} is the q-th component of W_j. Since maximizing L is equivalent to minimizing $-L$, by plugging equation (2.8) $\sim$ (2.10) in L we can obtain the dual problem

$$
\begin{aligned}
&\min_{\alpha_{ij}} \quad M \\
&\text{s.t.} \quad 0 \leq \alpha_{ij} \leq |c_{ij}|
\end{aligned}
$$

where

$$
\begin{aligned}
M =& \frac{1}{2n\lambda}\sum_{q=1}^{K-1}\Big[\sum_{i:y_i\geq 0}\alpha_{i,A_i}W_{A_i,q}x_i - \sum_{i:y_i<0}\alpha_{i,A_i}W_{A_i,q}x_i - \sum_{i:y_i\geq 0}\sum_{j\neq A_i}\alpha_{ij}W_{jq}x_i + \sum_{i:y_i<0}\sum_{j\neq A_i}\alpha_{ij}W_{jq}x_i\Big]^T \\
&\times\Big[\sum_{i:y_i\geq 0}\alpha_{i,A_i}W_{A_i,q}x_i - \sum_{i:y_i<0}\alpha_{i,A_i}W_{A_i,q}x_i - \sum_{i:y_i\geq 0}\sum_{j\neq A_i}\alpha_{ij}W_{jq}x_i + \sum_{i:y_i<0}\sum_{j\neq A_i}\alpha_{ij}W_{jq}x_i\Big]
\end{aligned}
$$

$$- (K-1)\sum_{i=1}^{n}\alpha_{i,A_i} - \sum_{i=1}^{n}\sum_{j\neq A_i}\alpha_{ij}$$

Then the optimization problem can be solved by coordinate descent algorithm outlined in Algorithm 1.

Algorithm 1: Estimating $f_q(x)$ by coordinate descent algorithm

```
Result: Estimated decision function $f_q(x)$, $q = 1, \cdots, K-1$
Initialization: define $\alpha = (\alpha_{ij})_{i=1,\cdots,n;j=1,\cdots,K}$ as an $n \times K$ matrix with the $(i, j)$ element
  equal to $\alpha_{ij}$. Initialize $\alpha^{(0)}$ as zero matrix and $m = 1$. $N$ is the maximum number of
  iterations and $tol$ is the preset tolerance ;
while $m < N$ do
    with $\alpha^{(m-1)}$ given, sequentially update $\alpha_{ij}^{(m-1)}$ to $\alpha_{ij}^{(m)}$. To get $\alpha_{ij}^{(m)}$, first fix $\alpha_{st}^{(m-1)}$,
      $(s,t) \neq (i,j)$, solve $\frac{\partial M}{\partial \alpha_{ij}} = 0$ to get solution $\hat{\alpha}_{ij}$ and the updated $\alpha_{ij}^{(m)}$ is determined
      as
        $$\alpha_{ij}^{(m)} = \begin{cases} 0 & \hat{\alpha}_{ij} \leq 0 \\ |c_{ij}| & \hat{\alpha}_{ij} \geq |c_{ij}| \\ \hat{\alpha}_{ij} & \text{otherwise} \end{cases}$$
    if $|\alpha^{(m)} - \alpha^{(m-1)}| < tol$ then
        stop the iteration;
    else
        $m = m + 1$;
    end
end
Plug $\hat{\alpha}$ in equation (2.10) to obtain the estimated decision function $f_q(x)$,
  $q = 1, \cdots, K-1$.
```

b. Nonlinear case

Define $k : \mathcal{X} \times \mathcal{X} \to \mathbb{R}$ as a kernel function which is continuous, symmetric and $\mathbf{K} = \big(k(x_i, x_j)\big)_{i=1,\cdots,n;j=1,\cdots,n}$ as the positive semidefinite gram matrix. The nonlinear decision boundary, can be assumed as $f_q(x) = \theta_{q0} + \sum_{i=1}^{n} \theta_{qi} k(x, x_i)$. The penalty term $J(f)$ is defined as $J(f) = \sum_{q=1}^{K-1} \theta_{q0}^2 + \sum_{q=1}^{K-1} \theta_q^T \mathbf{K} \theta_q$, where $\theta_q = (\theta_{q1}, \cdots, \theta_{qn})$. The optimization problem is then written as

$$\min_{\theta_{q0},\theta_q,\xi_{ij}} \frac{n\lambda}{2}\sum_{q=1}^{K-1}\theta_{q0}^2 + \frac{n\lambda}{2}\sum_{q=1}^{K-1}\theta_q^T\mathbf{K}\theta_q + \sum_{i:y_i\geq 0}\frac{y_i}{\pi(A_i,x_i)}[(1-\gamma)\sum_{j\neq A_i}\xi_{ij} + \gamma\xi_{i,A_i}]$$
$$- \sum_{i:y_i<0}\frac{y_i}{\pi(A_i,x_i)}[(1-\gamma)\sum_{j\neq A_i}\xi_{ij} + \gamma\xi_{i,A_i}]$$

$$\text{Subject to} \quad \xi_{ij} \geq 0 \quad (i = 1, \cdots, n; j = 1, \cdots, K)$$
$$\xi_{i,A_i} + \langle f(x_i), W_{A_i}\rangle - (K-1) \geq 0 \quad (i : y_i \geq 0)$$
$$\xi_{ij} - \langle f(x_i), W_j\rangle - 1 \geq 0 \quad (i : y_i \geq 0; j \neq A_i)$$
$$\xi_{i,A_i} - \langle f(x_i), W_{A_i}\rangle - (K-1) \geq 0 \quad (i : y_i < 0)$$
$$\xi_{ij} + \langle f(x_i), W_j\rangle - 1 \geq 0 \quad (i : y_i < 0; j \neq A_i)$$

The Lagrangian function L can be defined as

$$\begin{aligned}
L =& \frac{n\lambda}{2}\sum_{q=1}^{K-1}\theta_{q0}^2 + \frac{n\lambda}{2}\sum_{q=1}^{K-1}\theta_q^T\mathbf{K}\theta_q + \sum_{i:y_i\geq 0}\frac{y_i}{\pi(A_i,x_i)}[(1-\gamma)\sum_{j\neq A_i}\xi_{ij} + \gamma\xi_{i,A_i}] \\
&- \sum_{i:y_i<0}\frac{y_i}{\pi(A_i,x_i)}[(1-\gamma)\sum_{j\neq A_i}\xi_{ij} + \gamma\xi_{i,A_i}] - \sum_{i=1}^{n}\sum_{j=1}^{K}\tau_{ij}\xi_{ij} \\
&- \sum_{i:y_i\geq 0}\alpha_{i,A_i}[\xi_{i,A_i} + \langle f(x_i), W_{A_i}\rangle - (K-1)] - \sum_{i:y_i\geq 0}\sum_{j\neq A_i}\alpha_{ij}[\xi_{ij} - \langle f(x_i), W_j\rangle - 1] \\
&- \sum_{i:y_i<0}\alpha_{i,A_i}[\xi_{i,A_i} - \langle f(x_i), W_{A_i}\rangle - (K-1)] - \sum_{i:y_i<0}\sum_{j\neq A_i}\alpha_{ij}[\xi_{ij} + \langle f(x_i), W_j\rangle - 1] \\
=& \frac{n\lambda}{2}\sum_{q=1}^{K-1}\theta_{q0}^2 + \frac{n\lambda}{2}\sum_{q=1}^{K-1}\theta_q^T\mathbf{K}\theta_q + (K-1)\sum_{i=1}^{n}\alpha_{i,A_i} + \sum_{i=1}^{n}\sum_{j\neq A_i}\alpha_{ij} + \sum_{i:y_i\geq 0}\sum_{j=1}^{K}[c_{ij} - \tau_{ij} - \alpha_{ij}]\xi_{ij} \\
&- \sum_{i:y_i<0}\sum_{j=1}^{K}[c_{ij} + \tau_{ij} + \alpha_{ij}]\xi_{ij} - \sum_{i:y_i\geq 0}\alpha_{i,A_i}\langle f(x_i), W_{A_i}\rangle + \sum_{i:y_i<0}\alpha_{i,A_i}\langle f(x_i), W_{A_i}\rangle \\
&+ \sum_{i:y_i\geq 0}\sum_{j\neq A_i}\alpha_{ij}\langle f(x_i), W_j\rangle - \sum_{i:y_i<0}\sum_{j\neq A_i}\alpha_{ij}\langle f(x_i), w_j\rangle
\end{aligned}$$

where $\alpha_{ij}, \tau_{ij}, c_{ij}$ are the same as in linear case. Similarly, by assuming $\frac{\partial L}{\partial \xi_{ij}} = 0$, $\frac{\partial L}{\partial \theta_{q0}} = 0$ and $\frac{\partial L}{\partial \theta_q} = 0$, we obtain

$$c_{ij} - \tau_{ij} - \alpha_{ij} = 0 \quad \text{for} \quad i : y_i \geq 0 \tag{2.11}$$

$$c_{ij} + \tau_{ij} + \alpha_{ij} = 0 \quad \text{for} \quad i : y_i < 0 \tag{2.12}$$

$$\theta_{q0} = \frac{1}{n\lambda}\Big[\sum_{i:y_i\geq 0} \alpha_{i,A_i} W_{A_i,q} - \sum_{i:y_i<0} \alpha_{i,A_i} W_{A_i,q} - \sum_{i:y_i\geq 0}\sum_{j\neq A_i} \alpha_{ij} W_{jq} + \sum_{i:y_i<0}\sum_{j\neq A_i} \alpha_{ij} W_{jq}\Big] \tag{2.13}$$

$$\theta_q = \frac{1}{n\lambda}\mathbf{K}^{-1}\Big[\sum_{i:y_i\geq 0} \alpha_{i,A_i} W_{A_i,q}\mathbf{K_i} - \sum_{i:y_i<0} \alpha_{i,A_i} W_{A_i,q}\mathbf{K_i} - \sum_{i:y_i\geq 0}\sum_{j\neq A_i} \alpha_{ij} W_{jq}\mathbf{K_i} + \sum_{i:y_i<0}\sum_{j\neq A_i} \alpha_{ij} W_{jq}\mathbf{K_i}\Big] \tag{2.14}$$

where $\mathbf{K_i}$ is the ith column of the gram matrix $\mathbf{K}$ and W_{jq} is the same as in linear case. By plugging equation (2.11) ~ (2.14) in L, we obtain the dual problem

$$\min_{\alpha_{ij}} \quad M$$

$$\text{s.t.} \quad 0 \leq \alpha_{ij} \leq |c_{ij}|$$

where

$$\begin{aligned} M =& \frac{1}{2n\lambda}\sum_{q=1}^{K-1}\Big[\sum_{i:y_i\geq 0} \alpha_{i,A_i} W_{A_i,q}\mathbf{K_i} - \sum_{i:y_i<0} \alpha_{i,A_i} W_{A_i,q}\mathbf{K_i} - \sum_{i:y_i\geq 0}\sum_{j\neq A_i} \alpha_{ij} W_{jq}\mathbf{K_i} + \sum_{i:y_i<0}\sum_{j\neq A_i} \alpha_{ij} W_{jq}\mathbf{K_i}\Big]^T \mathbf{K}^{-1} \\ &\times \Big[\sum_{i:y_i\geq 0} \alpha_{i,A_i} W_{A_i,q}\mathbf{K_i} - \sum_{i:y_i<0} \alpha_{i,A_i} W_{A_i,q}\mathbf{K_i} - \sum_{i:y_i\geq 0}\sum_{j\neq A_i} \alpha_{ij} W_{jq}\mathbf{K_i} + \sum_{i:y_i<0}\sum_{j\neq A_i} \alpha_{ij} W_{jq}\mathbf{K_i}\Big] \\ &+ \frac{1}{2n\lambda}\sum_{q=1}^{K-1}\Big[\sum_{i:y_i\geq 0} \alpha_{i,A_i} W_{A_i,q} - \sum_{i:y_i<0} \alpha_{i,A_i} W_{A_i,q} - \sum_{i:y_i\geq 0}\sum_{j\neq A_i} \alpha_{ij} W_{jq} + \sum_{i:y_i<0}\sum_{j\neq A_i} \alpha_{ij} W_{jq}\Big]^2 \\ &- (K-1)\sum_{i=1}^{n} \alpha_{i,A_i} - \sum_{i=1}^{n}\sum_{j\neq A_i} \alpha_{ij} \end{aligned}$$

The estimating algorithm is the same as Algorithm 1.

Loss function 2: $V_Y^c(f(X), A)$

Unlike SVM whose solution only depends on support vectors, the classifier based on $V_Y^c(f(X), A)$ uses all training data to estimate the decision function. The condition $\langle f, W_j\rangle \geq -\frac{1}{K-1}$ can typically be approximated by modifying the loss function so that huge loss will be added when $\langle f, W_j\rangle < -\frac{1}{K-1}$ (Park and Liu, 2009). Thus we define a new loss function

$$l_u(f(X), A) = \begin{cases} V_Y(f(X), A) & \langle f(X), A\rangle \geq -\frac{1}{K-1} \\ u(-\frac{1}{K-1} - \langle f(X), A\rangle) & \langle f(X), A\rangle < -\frac{1}{K-1} \end{cases} \tag{2.15}$$

where $u \geq 0$ and $V_Y(f(X), A)$ is defined in equation (2.6). When $u \to +\infty$, $l_u(f(X), A) \to V_Y^c(f(X), A)$. So the optimization problem can be written as

$$\begin{aligned} \min \frac{n\lambda}{2} J(f) &+ \sum_{i:y_i \geq 0} \frac{y_i}{\pi(A_i, x_i)} \left[(1-\gamma)\sum_{j \neq A_i} \xi_{ij} + \gamma \xi_{i,A_i}\right] \\ &- \sum_{i:y_i<0} \frac{y_i}{\pi(A_i, x_i)} \left[(1-\gamma)\sum_{j \neq A_i} \xi_{ij} + \gamma \xi_{i,A_i}\right] \\ \text{Subject to} \quad & \xi_{ij} \geq 0 \quad (i = 1, \cdots, n; j = 1, \cdots, K) \\ & \xi_{i,A_i} + \langle f(x_i), W_{A_i}\rangle - (K-1) \geq 0 \quad (i : y_i \geq 0) \\ & \xi_{ij} - \langle f(x_i), W_j\rangle - 1 \geq 0 \quad (i : y_i \geq 0; j \neq A_i) \\ & \xi_{i,A_i} - \langle f(x_i), W_{A_i}\rangle - (K-1) \geq 0 \quad (i : y_i < 0) \\ & \xi_{ij} + \langle f(x_i), W_j\rangle - 1 \geq 0 \quad (i : y_i < 0; j \neq A_i) \\ & \xi_{ij} + u(\frac{1}{K-1} + \langle f(x_i, W_j)\rangle) \geq 0 \quad (i = 1, \cdots, n; j = 1, \cdots, K) \end{aligned}$$

The corresponding Lagrangian function L can be defined as

$$\begin{aligned} L = \frac{n\lambda}{2} J(f) &+ (K-1)\sum_{i=1}^{n} \alpha_{i,A_i} + \sum_{i=1}^{n}\sum_{j \neq A_i} \alpha_{ij} + \sum_{i:y_i \geq 0}\sum_{j=1}^{K} [c_{ij} - \tau_{ij} - \alpha_{ij} - v_{ij}]\xi_{ij} \\ &- \sum_{i:y_i<0}\sum_{j=1}^{K} [c_{ij} + \tau_{ij} + \alpha_{ij} + v_{ij}]\xi_{ij} - \sum_{i:y_i \geq 0} \alpha_{i,A_i}\langle f(x_i), W_{A_i}\rangle + \sum_{i:y_i<0} \alpha_{i,A_i}\langle f(x_i), W_{A_i}\rangle \end{aligned}$$

$$+\sum_{i:y_i\geq 0}\sum_{j\neq A_i}\alpha_{ij}\langle f(x_i),W_j\rangle-\sum_{i:y_i<0}\sum_{j\neq A_i}\alpha_{ij}\langle f(x_i),w_j\rangle-\sum_{i=1}^{n}\sum_{j=1}^{K}v_{ij}u\langle f(x_i),W_j\rangle-\sum_{i=1}^{n}\sum_{j=1}^{K}v_{ij}\frac{u}{K-1}$$

where c_{ij}, α_{ij} and τ_{ij} are defined the same as before and v_{ij} is the Lagrangian multiplier for the inequality constraint $\xi_{ij}+u(\frac{1}{K-1}+\langle f(x_i,W_j\rangle)\geq 0$. For linear decision rule, $J(f)=\sum_{q=1}^{K-1}\beta_q^T\beta_q$. For nonlinear decision rule, $J(f)=\sum_{q=1}^{K-1}\theta_{q0}^2+\sum_{q=1}^{K-1}\theta_q^T\mathbf{K}\theta_q$. After some similar mathematical operations as in the case of $V_Y(f(X),A)$, for both linear and nonlinear cases we have

$$\begin{cases} c_{ij}-\tau_{ij}-\alpha_{ij}-v_{ij}=0 & \text{for} \quad i:y_i\geq 0 \\ c_{ij}+\tau_{ij}+\alpha_{ij}+v_{ij}=0 & \text{for} \quad i:y_i<0 \end{cases} \tag{2.16}$$

For linear decision rule, β_q, $q=1,\cdots,K-1$ are functions of α_{ij}, v_{ij}, $i=1,\cdots,n$; $j=1,\cdots,K$ as

$$\begin{aligned}\beta_q=\frac{1}{n\lambda}\Big[&\sum_{i:y_i\geq 0}\alpha_{i,A_i}W_{A_i,q}x_i-\sum_{i:y_i<0}\alpha_{i,A_i}W_{A_i,q}x_i-\sum_{i:y_i\geq 0}\sum_{j\neq A_i}\alpha_{ij}W_{jq}x_i\\&+\sum_{i:y_i<0}\sum_{j\neq A_i}\alpha_{ij}W_{jq}x_i+\sum_{i=1}^{n}\sum_{j=1}^{K}v_{ij}uW_{jq}x_i\Big]\end{aligned} \tag{2.17}$$

For nonlinear decision rule, θ_{q0} and θ_q, $q=1,\cdots,K-1$ can be obtained by

$$\begin{aligned}\theta_{q0}=\frac{1}{n\lambda}\Big[&\sum_{i:y_i\geq 0}\alpha_{i,A_i}W_{A_i,q}-\sum_{i:y_i<0}\alpha_{i,A_i}W_{A_i,q}-\sum_{i:y_i\geq 0}\sum_{j\neq A_i}\alpha_{ij}W_{jq}\\&+\sum_{i:y_i<0}\sum_{j\neq A_i}\alpha_{ij}W_{jq}+\sum_{i=1}^{n}\sum_{j=1}^{K}v_{ij}uW_{jq}\Big]\end{aligned} \tag{2.18}$$

$$\begin{aligned}\theta_q=\frac{1}{n\lambda}\mathbf{K}^{-1}\Big[&\sum_{i:y_i\geq 0}\alpha_{i,A_i}W_{A_i,q}\mathbf{K_i}-\sum_{i:y_i<0}\alpha_{i,A_i}W_{A_i,q}\mathbf{K_i}-\sum_{i:y_i\geq 0}\sum_{j\neq A_i}\alpha_{ij}W_{jq}\mathbf{K_i}\\&+\sum_{i:y_i<0}\sum_{j\neq A_i}\alpha_{ij}W_{jq}\mathbf{K_i}+\sum_{i=1}^{n}\sum_{j=1}^{K}v_{ij}uW_{jq}\mathbf{K_i}\Big]\end{aligned} \tag{2.19}$$

So the dual problem for linear case is

$$\min_{\alpha_{ij}} \quad M$$

$$\text{s.t.} \quad 0 \leq \alpha_{ij} + v_{ij} \leq |c_{ij}|$$

where

$$M = \frac{1}{2n\lambda} \sum_{q=1}^{K-1} \Big[\sum_{i:y_i \geq 0} \alpha_{i,A_i} W_{A_i,q} x_i - \sum_{i:y_i<0} \alpha_{i,A_i} W_{A_i,q} x_i - \sum_{i:y_i \geq 0} \sum_{j \neq A_i} \alpha_{ij} W_{jq} x_i + \sum_{i:y_i<0} \sum_{j \neq A_i} \alpha_{ij} W_{jq} x_i$$
$$+ \sum_{i=1}^{n} \sum_{j=1}^{K} v_{ij} u W_{jq} x_i \Big]^T \times \Big[\sum_{i:y_i \geq 0} \alpha_{i,A_i} W_{A_i,q} x_i - \sum_{i:y_i<0} \alpha_{i,A_i} W_{A_i,q} x_i - \sum_{i:y_i \geq 0} \sum_{j \neq A_i} \alpha_{ij} W_{jq} x_i$$
$$+ \sum_{i:y_i<0} \sum_{j \neq A_i} \alpha_{ij} W_{jq} x_i + \sum_{i=1}^{n} \sum_{j=1}^{K} v_{ij} u W_{jq} x_i \Big] - (K-1) \sum_{i=1}^{n} \alpha_{i,A_i} - \sum_{i=1}^{n} \sum_{j \neq A_i} \alpha_{ij} + \sum_{i=1}^{n} \sum_{j=1}^{K} v_{ij} \frac{u}{K-1}$$

The dual problem for nonlinear case is

$$\min_{\alpha_{ij}, v_{ij}} \quad M$$
$$\text{s.t.} \quad 0 \leq \alpha_{ij} + v_{ij} \leq |c_{ij}|$$

where

$$M = \frac{1}{2n\lambda} \sum_{q=1}^{K-1} \Big[\sum_{i:y_i \geq 0} \alpha_{i,A_i} W_{A_i,q} \mathbf{K_i} - \sum_{i:y_i<0} \alpha_{i,A_i} W_{A_i,q} \mathbf{K_i} - \sum_{i:y_i \geq 0} \sum_{j \neq A_i} \alpha_{ij} W_{jq} \mathbf{K_i} + \sum_{i:y_i<0} \sum_{j \neq A_i} \alpha_{ij} W_{jq} \mathbf{K_i}$$
$$+ \sum_{i=1}^{n} \sum_{j=1}^{K} v_{ij} u W_{jq} \mathbf{K_i} \Big]^T \mathbf{K}^{-1} \times \Big[\sum_{i:y_i \geq 0} \alpha_{i,A_i} W_{A_i,q} \mathbf{K_i} - \sum_{i:y_i<0} \alpha_{i,A_i} W_{A_i,q} \mathbf{K_i} - \sum_{i:y_i \geq 0} \sum_{j \neq A_i} \alpha_{ij} W_{jq} \mathbf{K_i}$$
$$+ \sum_{i:y_i<0} \sum_{j \neq A_i} \alpha_{ij} W_{jq} \mathbf{K_i} + \sum_{i=1}^{n} \sum_{j=1}^{K} v_{ij} u W_{jq} \mathbf{K_i} \Big] + \frac{1}{2n\lambda} \sum_{q=1}^{K-1} \Big[\sum_{i:y_i \geq 0} \alpha_{i,A_i} W_{A_i,q} - \sum_{i:y_i<0} \alpha_{i,A_i} W_{A_i,q}$$
$$- \sum_{i:y_i \geq 0} \sum_{j \neq A_i} \alpha_{ij} W_{jq} + \sum_{i:y_i<0} \sum_{j \neq A_i} \alpha_{ij} W_{jq} + \sum_{i=1}^{n} \sum_{j=1}^{K} v_{ij} u W_{jq} \Big]^2 - (K-1) \sum_{i=1}^{n} \alpha_{i,A_i} - \sum_{i=1}^{n} \sum_{j \neq A_i} \alpha_{ij}$$
$$+ \sum_{i=1}^{n} \sum_{j=1}^{K} v_{ij} \frac{u}{K-1}$$

2.3.2 Multi-stage

In this section we consider $T > 1$. Q-learning uses backwards induction for $T > 1$. The principle is that the best treatment at the last stage is first estimated and then we move backwards to the previous stages. A brief example of two-stage case is given in section 1.4.1. For our proposed method, we use a similar technique as in Q-learning with a difference in the pseudo-outcome. Since Q-learning models the conditional mean outcome at each stage $t = 1, \cdots, T$, the pseudo-outcome is generated via the maximized Q-function in the next stage where the Q function is typically approximated by regression models. In our proposed method, we directly estimate the best treatment based on the pseudo-outcome obtained via the potential outcome as if the patients receive the estimated best treatment in all future stages using the doubly-robust estimator (Zhang et al., 2013).

We define a Q-function at stage t, Q_t, as the reward obtained in future stages if the patient is assigned the estimated optimal treatment from stage t to the end stage T. Based on the definition, we have $Q_{T+1} = 0$ and for $t = 1, \cdots, T$, if a patient actually follows the estimated best treatment from stage t to the end, $Q_t = \sum_{s=t}^{T} Y_s$, otherwise it will be approximated by the doubly-robust estimator which will be described later. The pseudo-outcome at stage $t = 1, \cdots, T$, Y_t^{pse}, is then defined as $Y_t^{pse} = Y_t + Q_{t+1}$.

We can recast the estimation of potential outcome provided that patients follow the estimated best treatment from stage t to the end as a monotone coarsening problem, and it is shown that coarsening is at random (Zhang et al., 2013). For estimating Q_t, we start from stage t and all history information prior to stage t is viewed as the new baseline information. Define $N_{ts} = I(A_t = d_t, \cdots, A_s = d_s), t < s$ as an indicator function for whether or not the patient receives the recommended treatment from stage t to s. Then we define the coarsening discrete hazard $\lambda_{ts}(\bar{X}_s) = Pr(A_s \neq d_s(\bar{X}_s, \bar{A}_{s-1}) | \bar{X}_s, N_{t,s-1} = 1)$. It is the probability that the treatment received by patients ceases to be consistent with the dynamic treatment regimes d at stage s given that it is consistent from stage t to stage $s - 1$. The probability of the observed treatment being consistent with $\underline{d}_t$ at least up to stage s can be expressed as $M_{ts}(\bar{X}_s) = \prod_{p=t}^{s}\{1 - \lambda_{tp}(\bar{X}_p)\}$.

Then the doubly-robust estimator of Q_t (Zhang et al., 2013) is constructed as

$$Q_t = \frac{N_{tT} \sum_{s=t}^{T} Y_s}{M_{tT}(\bar{X}_T)} + \sum_{s=t}^{T} \frac{N_{t,s-1}(I[A_s \neq d_s(\bar{X}_s)] - \lambda_{ts}(\bar{X}_s))}{M_{ts}(\bar{X}_s)} L_{ts}(\bar{X}_s) \tag{2.20}$$

where $L_{ts}(\bar{X}_s)$ can be arbitrary function of $\bar{X}_s$ and the optimal choice with the smallest asymptotic variance is $E[Q_t|\bar{X}_s, N_{t,s-1} = 1]$ (Zhang et al., 2013).

From equation (2.20) we need to estimate $\lambda_{ts}(\bar{X}_s)$ and $L_{ts}(\bar{X}_s)$. For the estimation of $\lambda_{ts}(\bar{X}_s)$ we only need to specify the model for propensity score $\pi_s(\bar{x}_s, \bar{a}_{s-1}, a_s) = Pr(A_s = a_s|\bar{X}_s = x_s, \bar{A}_{s-1} = \bar{a}_{s-1})$. For randomized trial, $\pi_s(\bar{x}_s, \bar{a}_{s-1}, a_s)$ is determined. For observational study $\pi_s(\bar{x}_s, \bar{a}_{s-1}, a_s)$ needs to be modeled. A common choice to obtain the propensity score is the logistic regression or multinomial regression. Let $L_{ts}(\bar{X}_s) = 1 - \pi(\bar{X}_s, a_{t-1}, d_t^{s-1}(\bar{X}_{s-1}), d_s(\bar{X}_s))$ and take $L_{ts}(\bar{X}_s) = E[Q_t|\bar{X}_s, N_{t,s-1} = 1]$. We can define iteratively that $\mu_{tT}(\bar{x}_t, \bar{a}_t) = E[\sum_{p=t}^{T} Y_p|\bar{X}_T = \bar{x}_T, \bar{A}_T = \bar{a}_T]$ and $f_{tT}(\bar{x}_T, \bar{a}_{T-1}) = \mu_{tT}(\bar{x}_T, \bar{a}_{T-1}, d_T)$. For $s = T-1, \cdots, t$, define $\mu_{ts}(\bar{x}_s, \bar{a}_s) = E[f_{t,s+1}(\bar{x}_s, X_{s+1}, \bar{a}_s|\bar{X}_s = \bar{x}_s, \bar{A}_s = \bar{a}_s]$ and $f_{ts}(\bar{x}_s, \bar{a}_{s-1}) = \mu_{ts}(\bar{x}_s, \bar{a}_{s-1}, d_s)$. It is shown that $L_{ts}(\bar{X}_s) = \mu_{ts}(\bar{x}_s, d_t^s)$ (Zhang et al., 2013).

2.4 Numerical investigation

In this section, we describe both the simulation studies and real data application of the proposed method.

2.4.1 Simulation study

To assess the performance of the proposed methods, simulation studies were carried out for a variety of scenarios. We consider both linear and nonlinear decision rule with single stage and multi-stage. For linear decision rule, we restrict f to be a linear function of x and for nonlinear decision rule we use Gaussian kernel. We also evaluate the influence of reduced main effect, reduced interaction effect as well as increased number of treatments.

For each simulation setting, we first generate a tuning set with a sample size of 500 for training the tuning parameter which is λ in linear case and λ, τ in nonlinear case. We use a grid search to find the best tuning parameter. λ varies in $[0.1, 100]$ and τ in Gaussian kernel $k(x, y) = \exp\{-\|x - y\|_2^2/(2\tau^2)\}$ varies in $[0.1, 2]$. For the parameter u in $V_Y^c(f(X), A)$, we just use $u = 1000$ because when u becomes larger than 1000 the result will not change much. For each of our settings, we repeat the simulation 500 times. For each simulation run, we generate a data set with a sample size of 1500. We randomly choose 500 of them as training data and the remaining is used as testing data. For single stage, we use misclassification error rate and the empirical value function to assess the performance of the model. For multi-stage, we only use empirical value function to assess the model. The misclassification error rate in the single stage setting is defined as $\mathbb{P}_n[I(A^{opt} = d(X))]$ and the empirical value function is defined as $\mathbb{P}_n[\prod_{t=1}^{T} \frac{I(A_t = d_t(H_t))}{\pi_t(A_t|H_t)Y}]/\mathbb{P}_n[\prod_{t=1}^{T} \frac{I(A_t = d_t(H_t))}{\pi_t(A_t|H_t)}]$ where $\mathbb{P}_n$ is the empirical average operator. The misclassification error rate measures the possibility that the estimated dynamic treatment regime cannot detect the true optimal treatment. The empirical value function measures the outcome patients can obtain if they follow the estimated dynamic treatment regime. A smaller misclassification error rate or a larger empirical value function provide evidence that the estimated dynamic treatment regime is preferred.

We consider 9 scenarios (Liu and Yuan, 2011; Zhang et al., 2016):

1. A three-treatment case. The optimal treatment A^{opt} satisfies $Pr(A^{opt} = 1) = Pr(A^{opt} = 2) = Pr(A^{opt} = 3) = 1/3$. The covariate vector satisfies $X|A^{opt} = j \sim N(\mu_j, \sigma^2 I)$ where $\mu_1 = (1, 0, 0)^T$, $\mu_2 = (-0.5, \sqrt{3}/2, 0)^T$, $\mu_3 = (-0.5, -\sqrt{3}/2, 0)^T$, σ^2 is chosen such that the Bayes classification error is 5% and I is the identity matrix. The actual treatment is generated from multinomial distribution with

$$Pr(A^{obs} = 1|X) = \frac{1}{1 + \exp(-2 + X_1 + 2X_2 - X_3) + \exp(-1 - 2X_1 + 2X_2)}$$
$$Pr(A^{obs} = 2|X) = \frac{\exp(-2 + X_1 + 2X_2 - X_3)}{1 + \exp(-2 + X_1 + 2X_2 - X_3) + \exp(-1 - 2X_1 + 2X_2)}$$

$$Pr(A^{obs} = 3|X) = \frac{\exp(-1 - 2X_1 + 2X_2)}{1 + \exp(-2 + X_1 + 2X_2 - X_3) + \exp(-1 - 2X_1 + 2X_2)}$$

The outcome R follows $N(x^T\beta + 10I(A^{obs} = A^{opt}), 1)$ where $\beta = (0, 1, 1)^T$.

2. All the settings are the same as scenario 1 except that $\beta = 0.1 \times (0, 1, 1)^T$.

3. All the settings are the same as scenario 1 except that R follows $N(x^T\beta + 2I(A^{obs} = A^{opt}), 1)$

4. A four-treatment case. The optimal treatment A^{opt} satisfies $Pr(A^{opt} = 1) = Pr(A^{opt} = 2) = Pr(A^{opt} = 3) = Pr(A^{opt} = 4) = 1/4$. The covariate vector satisfies $X|A^{opt} = j \sim N(\mu_j, \sigma^2 I)$ where $\mu_j = (W_j, 0, 0)^T$, W_j is defined in section 2.2 when $K = 4$ and σ^2 is chosen such that the Bayes classification error is 5% and I is the identity matrix. The actual treatment is generated from multinomial distribution with

$$\begin{aligned} Pr(A^{obs} = 1|X) &= \frac{1}{S} \\ Pr(A^{obs} = 2|X) &= \frac{\exp(-2 + X_1 + 2X_2 - X_3)}{S} \\ Pr(A^{obs} = 3|X) &= \frac{\exp(-1 - 2X_1 + 2X_2)}{S} \\ Pr(A^{obs} = 4|X) &= \frac{\exp(-2X_1 + 2X_2 - 2X_4 - X_5)}{S} \end{aligned}$$

where $S = 1+\exp(-2+X_1+2X_2-X_3)+\exp(-1-2X_1+2X_2)+\exp(-2X_1+2X_2-2X_4-X_5)$ The outcome R follows $N(x^T\beta + 10I(A^{obs} = A^{opt}), 1)$ where $\beta = (0, 1, -1, 1, -1)^T$.

5. All the settings are the same as scenario 1 except that the covariate vector satisfies $X|A^{opt} = j \sim 0.5N(\mu_{ja}, \sigma^2 I) + 0.5N(\mu_{jb}, \sigma^2 I)$ where $\mu_{ja} = (\cos(j\pi/3), \sin(j\pi/3), 0)^T$, $\mu_{jb} = (\cos(j\pi/3 + \pi), \sin(j\pi/3 + \pi), 0)^T$, σ^2 is chosen such that the Bayes classification error is 5% and I is the identity matrix.

6. All the settings are the same as scenario 5 except that $\beta = 0.1 \times (0, 1, 1)^T$.

7. All the settings are the same as scenario 5 except that R follows $N(x^T\beta + 2I(A^{obs} = A^{opt}), 1)$

8. All the settings are the same as scenario 4 except that the covariate vector satisfies $X|A^{opt} = j \sim 0.5N(\mu_{ja}, \sigma^2 I) + 0.5N(\mu_{jb}, \sigma^2 I)$ where $\mu_{ja} = (\cos(j\pi/4), \sin(j\pi/4), \mathbf{0}_3^T)^T$, $\mu_{jb} = (\cos(j\pi/4 + \pi), \sin(j\pi/4 + \pi), \mathbf{0}_3^T)^T$, σ^2 is chosen such that the Bayes classification error is 5% and I is the identity matrix. The actual treatment is generated from multinomial distribution with

$$Pr(A^{obs} = 1|X) = \frac{1}{S}$$
$$Pr(A^{obs} = 2|X) = \frac{\exp(-2 + X_1 + 2X_2 - X_3 - 2X_4)}{S}$$
$$Pr(A^{obs} = 3|X) = \frac{\exp(-1 - 2X_1 + 2X_2 - 2X_5)}{S}$$
$$Pr(A^{obs} = 4|X) = \frac{\exp(X_1 - X_3 - X_4)}{S}$$

where $S = 1+\exp(-2+X_1+2X_2-X_3-2X_4)+\exp(-1-2X_1+2X_2-2X_5)+\exp(X_1-X_3-X_4)$

9. A two-stage case. The optimal treatment at stage t, $A_t^{opt}, t = 1, 2$ satisfies $Pr(A_t^{opt} = 1) = Pr(A_t^{opt} = 2) = Pr(A_t^{opt} = 3) = 1/3$. The covariate vector at stage t satisfies $X_t|A_t^{opt} = j \sim 0.5N(\mu_{ja}, \sigma^2 I) + 0.5N(\mu_{jb}, \sigma^2 I)$ where $\mu_{ja} = (\cos(j\pi/3), \sin(j\pi/3), \mathbf{0}_3^T)^T$, $\mu_{jb} = (\cos(j\pi/3 + \pi), \sin(j\pi/3 + \pi), \mathbf{0}_3^T)^T$, σ^2 is chosen such that the Bayes classification error is 5% and I is the identity matrix. The actual treatment at stage 1 is generated from multinomial distribution with

$$Pr(A_1^{obs} = 1|X_1) = \frac{1}{S}$$
$$Pr(A_1^{obs} = 2|X_1) = \frac{\exp(-1 - X_{11} + 2X_{12} - X_{13} - X_{14} - X_{15})}{S}$$
$$Pr(A_1^{obs} = 3|X_1) = \frac{\exp(-1 - 2X_{11} + 2X_{12} - 2X_{13} + 2X_{14})}{S}$$

where $S = 1+\exp(-1-X_{11}+2X_{12}-X_{13}-X_{14}-X_{15})+\exp(-1-2X_{11}+2X_{12}-2X_{13}+2X_{14})$

The actual treatment at stage 2 is generated from multinomial distribution with

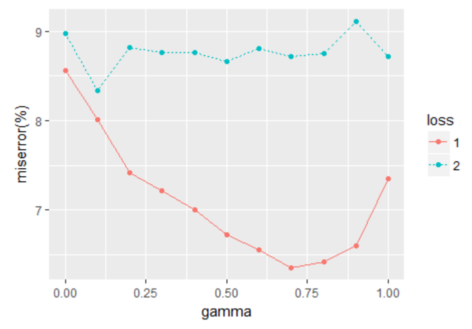
$$\begin{aligned}
Pr(A_2^{obs} = 1|X_2) &= \frac{1}{S} \\
Pr(A_2^{obs} = 2|X_2) &= \frac{\exp(-1 - X_{21} + X_{22} - X_{23} - 2X_{24})}{S} \\
Pr(A_2^{obs} = 3|X_2) &= \frac{\exp(-2X_{21} + X_{22} - 2X_{23} + X_{24})}{S}
\end{aligned}$$

where $S = 1 + \exp(-1 - X_{21} + X_{22} - X_{23} - 2X_{24}) + \exp(-2X_{21} + X_{22} - 2X_{23} + X_{24})$. The outcome R_1 follows $N(u_1, 1)$ where $u_1 = 10I(A_1^{opt} = A_1^{obs}) + X_{12} + X_{13} - X_{14}^2 + X_{11}X_{15}$. The outcome R_2 follows $N(u_2, 1)$ where $u_2 = 5I(A_2^{opt} = A_2^{obs}) + (X_{22}^2 + X_{24})^2 X_{21} + X_{23}X_{25}$.

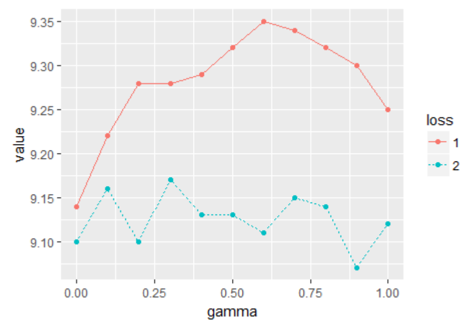
Scenarios 1 $\sim$ 4 have linear decision rule and scenarios 5 $\sim$ 8 have nonlinear decision rule. Scenarios 2 and 6 involve reduced main effect while scenarios 3 and 7 investigate the impact of the reduced interaction effect. Scenarios 4 and 8 consider a four-treatment case so that the effect of the number of treatment options can be observed. Finally, scenario 9 involves two stages as well as a more complex nonlinear main effect.

Figure 2.2 shows the simulation results for scenarios 1 and 5 using different values of γ. Loss 1 refers to the unconstrained loss function $V_Y(f(X), A)$, while loss 2 refers to $V_Y^c(f(X), A)$. It is shown in the figure that for $V_Y^c(f(X), A)$ the simulation results do not vary much for different γ, compared with $V_Y(f(X), A)$. For linear case, $V_Y(f(X), A)$ outperforms $V_Y^c(f(X), A)$ while for nonlinear case $V_Y^c(f(X), A)$ performs better. $\gamma = 0.5$ gives relatively stable results. $\gamma = 0.5$ may not perform the best in a single case but it always gives results close to the best one. Considering this observation and the fact that $V_Y(f(X), A)$ is Fisher consistent in $[0, 0.5]$, in other scenarios, we use $\gamma = 0.5$.

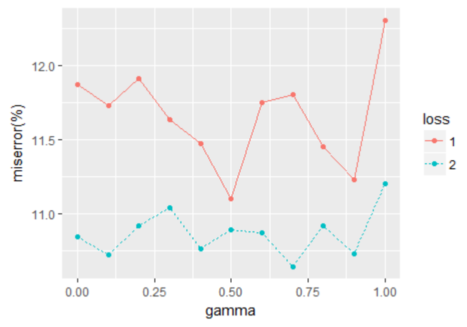
Table 2.1 shows the misclassification error rates obtained by validation data of size 1000, where OVR and OVO columns are results from One-Versus-Rest and One-Versus-One methods. The One-Versus-Rest approach constructs K classifiers each comparing one of the K classes to the remaining $K - 1$ classes and producing a real-valued confidence score for its prediction. A new observation is assigned to the class for which the corresponding classifier



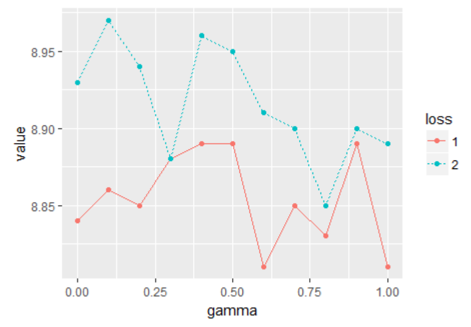
(a)



(b)



(c)



(d)

Figure 2.2: Simulation results for scenarios 1 and 5 for different γ. Loss 1 and 2 refer to $V_Y(f(X),A)$ and $V_Y^c(f(X),A)$ respectively. Figure 2.2a and figure 2.2b show the misclassification error rates and empirical value for linear decision boundary. Figure 2.2c and figure 2.2d give the same result for nonlinear decision boundary

produces the highest confidence score. The One-Versus-One approach constructs $\frac{K(K-1)}{2}$ classifiers each comparing a pair of classes. A new observation is assigned to the class to which it is most frequently assigned in these $\frac{K(K-1)}{2}$ pairwise classifications. From table 2.1, in terms of misclassification error rates, $V_Y(f(X),A)$ performs better than $V_Y^c(f(X),A)$ in the first two scenarios and they have comparable performance for scenarios $3 \sim 6$. However, for scenarios 7 and 8, misclassification error rates obtained from $V_Y^c(f(X),A)$ are apparently lower than those obtained from $V_Y(f(X),A)$. Also, estimates obtained from $V_Y^c(f(X),A)$ have comparable or smaller standard deviations than those from $V_Y(f(X),A)$ in all scenarios except scenario 2. These observations show that $V_Y^c(f(X),A)$ is better at dealing with complex situation and is more stable as expected given that the Fisher consistency of classifier based on $V_Y^c(f(X),A)$ does not require any assumptions about treatment outcome.

Table 2.1: Misclassification error rates approximated by validation data set of size 1000, averaged over 500 simulation runs; the numbers in parenthesis are standard deviations over 500 simulation runs

	Scenario	$V_Y(f(X),A)$	$V_Y^c(f(X),A)$	OVR	OVO
Linear	1	6.72% (3.1%)	8.66% (3.8%)	7.83% (3.7%)	9.31% (4.3%)
	2	6.05% (1.9%)	8.48% (3.6%)	6.72% (1.7%)	8.34% (3.1%)
	3	16.65% (11.5%)	17.2% (8.8%)	22.67% (12.5%)	17.23% (9.2%)
	4	9.84% (4.9%)	9.97% (3.7%)	8.80% (3.4%)	12.92% (3.7%)
Nonlinear	5	11.10% (4.8%)	10.89% (4.9%)	21.66% (6.6%)	24.68% (6.6%)
	6	11.32% (5.3%)	10.86% (3.7%)	22.44% (6.1%)	25.66% (6.3%)
	7	18.66% (7.6%)	14.28% (6.9%)	26.72% (7.9%)	28.11% (7.6%)
	8	22.02% (4.4%)	19.90% (4.4%)	32.24% (4.1%)	35.26% (4.2%)

Table 2.2 shows the empirical value function obtained by validation data of size 1000 which is a more comprehensive measure of the performance of the estimated treatment rule. We see that for linear case $V_Y(f(X),A)$ performs better than $V_Y^c(f(X),A)$ in terms of estimated values in first two scenarios and they have comparable performance for scenarios 3 and 4. However, for nonlinear cases, values from $V_Y^c(f(X),A)$ are higher than those from $V_Y(f(X),A)$ in all scenarios, which again shows that $V_Y^c(f(X),A)$ is superior to $V_Y(f(X),A)$ in the case of complex situations. Unlike misclassification error rate, estimator of value obtained from $V_Y^c(f(X),A)$ has larger variance. In the two-stage case, the approximated value from $V_Y^c(f(X),A)$ is larger than that from $V_Y(f(X),A)$ which suggests $V_Y^c(f(X),A)$ is better for multiple stages.

From the two tables we see that, both loss functions perform better in the scenarios with reduced main effect or larger interaction effect in either linear or nonlinear cases. It is reasonable because when the data set itself has more apparent boundary, the performance of a classifier is expected to be more accurate. Comparing scenarios 4, 8 to other scenarios, we can observe that when the number of treatment options increases, the estimation accuracy decreases. We also compare the performance of the two proposed loss functions with those of One-Versus-Rest (OVR) and One-Versus-One (OVO) methods. For linear case, the proposed loss functions , One-Versus-Rest and One-Versus-One gives comparable results. For nonlinear case or multiple stages, the two proposed loss functions perform apparently better than the One-Versus-Rest and the One-Versus-One methods in terms of both misclassification error and value. It suggests our proposed method performs better than the competitors with complex situation.

Table 2.2: Empirical value function approximated by validation data set of size 1000, averaged over 500 simulation runs; the numbers in parenthesis are standard deviations over 500 simulation runs

	Scenario	$V_Y(f(X), A)$	$V_Y^c(f(X), A)$	OVR	OVO
		Single stage: $T = 1$			
Linear	1	9.32 (0.541)	9.13 (0.650)	9.25 (0.582)	9.10 (0.602)
	2	9.41 (0.353)	9.16 (0.530)	9.33 (0.407)	9.19 (0.402)
	3	1.68 (0.344)	1.67 (0.387)	1.54 (0.383)	1.66 (0.326)
	4	9.02 (0.935)	8.99 (0.704)	9.10 (0.697)	8.67 (0.785)
Nonlinear	5	8.89 (0.589)	8.95 (0.604)	7.90 (0.720)	7.58 (0.741)
	6	8.84 (0.712)	8.92 (0.511)	7.78 (0.770)	7.46 (0.768)
	7	1.64 (0.286)	1.72 (0.266)	1.46 (0.247)	1.44 (0.229)
	8	7.79 (0.859)	7.96 (0.852)	6.79 (0.691)	6.48 (0.665)
		Two stage: $T = 2$			
	9	11.64 (0.858)	12.47 (0.933)	7.57 (0.305)	7.50 (0.446)

2.4.2 Application to STAR*D study

We applied the proposed method to the data from the Sequential Treatment Alternatives to Relieve Depression (STAR*D) clinical trial. A brief introduction of the study is described in section 1.1. Following Schulte et al. (2014) and Zhang et al. (2013), we only consider level 2 and 3 and simplify the treatment options at each level. We combine level 2 and level 2A as first stage and define level 3 as second stage. Following the work of Liu et al. (2018) and Zhang et al. (2013), at each stage, treatment(A_k), outcome(Y_k) and feature variables(H_k), $k = 1, 2$ are defined as follows:

A_1 : 1 if the patient takes an augment option and 2 if the patient takes a switch option at level 2 and 2A(stage 1)

A_2 : 1 if the patient takes an augment option and 2 if the patient takes a switch option at level 3(stage 2)

Y_1 : - QIDS score at the end of stage 1 if remission was achieved, $-\frac{1}{2}$ QIDS score at the end of stage 1 if the patient moved to stage 2

Y_2 : $-\frac{1}{2}$ QIDS score at the end of stage 2

H_1 : baseline QIDS score at the beginning of the trial, the slope of QIDS score based on QIDS score at baseline and stage 1, preference for treatment(1 for switch, -1 for augment, 0 for no preference)

H_2 : baseline QIDS score at stage 1, the slope of QIDS score based on QIDS score at stage 1 and stage 2, preference for treatment(1 for switch, -1 for augment, 0 for no perference)

There were 1246 patients entering first stage and 327 of them moved to second stage. In the analysis, the patients who did not enter stage 2 due to remission were assumed to receive optimal treatment at stage 2. To implement the proposed model, at each stage multinomial regression and linear regression were used to estimate treatment probability and pseudo-outcome respectively. Comparisons of different methods are based on 100 repetitions. For each repetition, the sample data is randomly split into training data and testing data. At the second stage, $\frac{2}{3}$ of the total 327 patients are chosen as training data. At the first stage, there are 919 patients who did not move to the second stage. About $\frac{2}{3}$ of these 919 patients (612 patients) along with the training observations at second stage are chosen as training data for first-stage model fitting. The testing value functions of the estimated DTRs, which is the weighted average of the outcome for all patients whose observed treatments are consistent with the estimated DTR in all stages, are computed over testing data.

Figure 2.3 provides results of our proposed methods. The classifier based on $V_Y(f(X), A)$ (loss 1) with linear kernel outperforms others. The mean estimated value functions of DTR based on loss 1 with linear and gaussian kernel are -8.21 (sd = 0.82) and -9.34 (sd = 1.14) respectively, while loss 2 with linear and gaussian kernel gives DTRs of mean value -8.72 (sd = 1.01) and -9.28 (sd = 1.02).

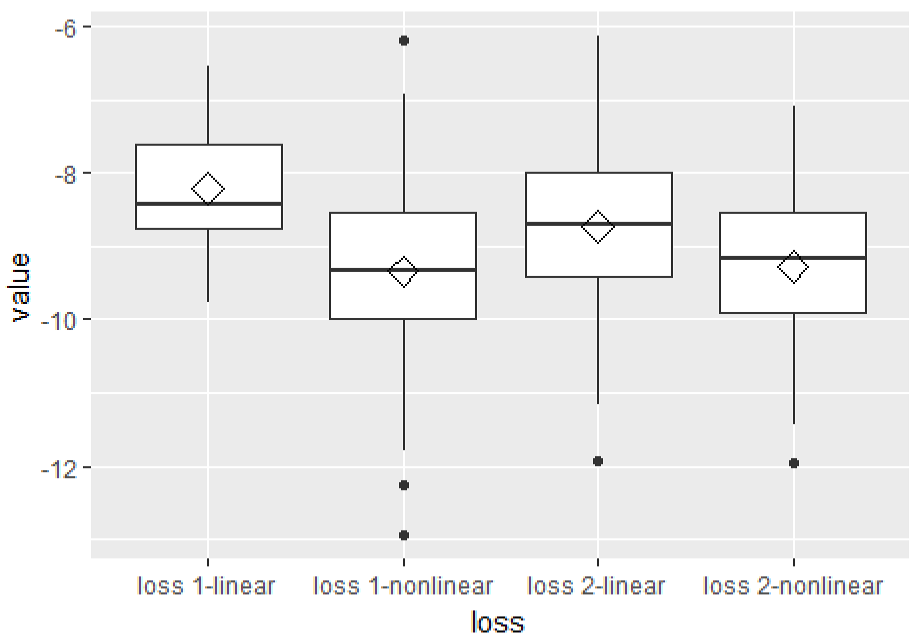


Figure 2.3: Estimated value function based on 100 repetitions of application for Sequential Treatment Alternatives to Relieve Depression data

2.5 Conclusion

In this chapter, we extend the standard outcome weighted learning to a multi-treatment, multi-stage setting with potential negative outcome. The method is based on an angle-based multi-category support vector machine and the loss function is modified when the outcome is negative. We provide two extra assumptions and show that when these assumptions are satisfied, the Fisher consistency of the modified loss function will still be valid. For cases where the assumptions do not hold, if the decision function is constrained, the classifier is Fisher consistent.

Our classification-based method is conceptually simple. It directly borrows the technique of multicategory support vector machine and modifies it to fit the DTR background. Simulation study and real data application are conducted to illustrate the proposed method. It shows that our method are better than the standard One-Versus-One and One-Versus-Rest methods. However, from the numerical investigation we also find the method does not perform satisfactorily when the interaction effect is very small or the number of treatment options is large. For these two cases, further improvement and modification are required.

2.6 Appendix

Proof of Theorem 2.3.1 To prove theorem 2.3.1, we first introduce the following lemma (Zhang and Liu, 2014).

Lemma 2.6.1 *Suppose we have arbitrary $f \in \mathbb{R}^{K-1}$. For any u, $v \in \{1, ..., K\}$ such that $u \neq v$, define $T_{u,v} = W_u - W_v$. For any scalar $z \in \mathbb{R}$, $\langle (f + zT_{u,v}), W_\omega \rangle = \langle f, W_\omega \rangle$, where $\omega \in \{1, ..., K\}$ and $\omega \neq u, v$. Furthermore, we have that $\langle (f+zT_{u,v}), W_u \rangle - \langle f, W_u \rangle = -\langle (f+zT_{v,u}), W_v \rangle + \langle f, W_v \rangle$.*

To simplify the notation, we use R_j^+, R_j^- instead of $R_j^+(x)$ and $R_j^-(x)$. Without loss of generality, assume that treatment 1 and 2 are respectively the best and the worst treatment. Suppose f^* is the minimizer of the conditional loss $r(f|x)$. The conditional loss $r(f|x)$ can be rewritten as

$$\begin{aligned} r(f|x) &= E\Big[\frac{|Y|}{\pi(A, x)} V_Y(f, A)|x\Big] \\ &= \sum_{i=1}^{K} R_i^+ V_1(f, i) - \sum_{i=1}^{K} R_i^- V_2(f, i) \end{aligned} \tag{2.21}$$

where $V_1(f, i) = \gamma[(K-1) - \langle f(X), W_i \rangle]_+ + (1-\gamma) \sum_{a \neq i} [1 + \langle f(X), W_a \rangle]_+$ and $V_2(f, i) = \gamma[(K-1) + \langle f(X), W_i \rangle]_+ + (1-\gamma) \sum_{a \neq i} [1 - \langle f(X), W_a \rangle]_+$.

First, we need to show $\langle f^*(x), W_1 \rangle \geq \langle f^*(x), W_j \rangle$, for $j \neq 1$. If there exists a treatment j such that $\langle f^*, W_1 \rangle < \langle f^*, W_j \rangle$. By lemma 2.6.1, we can construct f^{**} such that $\langle f^{**}, W_1 \rangle = \langle f^*, W_j \rangle$, $\langle f^{**}, W_j \rangle = \langle f^*, W_1 \rangle$ and $\langle f^{**}, W_t \rangle = \langle f^*, W_t \rangle, \forall t \neq 1, j$. One can verify that

$$\begin{array}{lll} V_1(f^*, W_t) = V_1(f^{**}, W_t) & V_2(f^*, W_t) = V_2(f^{**}, W_t) & \forall t \neq 1, j \\ V_s(f^*, W_1) = V_s(f^{**}, W_j) & V_s(f^*, W_j) = V_s(f^{**}, W_1) & s = 1, 2 \end{array}$$

Thus, $r(f^*|x) - r(f^{**}|x) = (R_1^+ - R_j^+)[V_1(f^*, 1) - V_1(f^*, j)] + (R_1^- - R_j^-)[V_2(f^*, j) - V_2(f^*, 1)]$. Since $\langle f^*, W_j \rangle > \langle f^*, W_1 \rangle$, we have $V_1(f^*, 1) - V_1(f^*, j) > 0$ and $V_2(f^*, j) - V_2(f^*, 1) > 0$. Based on assumptions 4, $r(f^*|x) - r(f^{**}|x) > 0$ which contradicts to the definition of f^*.

Similarly, we can show that $\langle f^*, W_2 \rangle \leq \langle f^*, W_j \rangle$, for $j \neq 2$.

Second, we need to show $\langle f*, W_1\rangle \leq K-1$. If $\langle f^*, W_1\rangle > K-1$, because $\sum_{i=1}^{K}\langle f^*, W_i\rangle = 0$, we can find j such that $\langle f^*, W_j\rangle < -1$. There exists f^{**} such that $\langle f^{**}, W_t\rangle = \langle f^*, W_t\rangle$, $\forall t \neq 1, j$ and $\langle f^{**}, W_1\rangle = \langle f^*, W_1\rangle - \epsilon > K-1$, $\langle f^{**}, W_j\rangle = \langle f^*, W_j\rangle + \epsilon < -1$ where ϵ is a small positive value. Then one can verify $r(f^*|x) - r(f^{**}|x) = P - N$ where $P = (1-\gamma)\epsilon\sum_{i=1}^{K} R_i^+ - R_1^+(1-\gamma)\epsilon + R_j^+\gamma\epsilon$ and

$$N = \begin{cases} (1-\gamma)\epsilon\sum_{i=1}^{K} R_i^- + R_1^-\gamma\epsilon - R_j^-(1-\gamma)\epsilon & \langle f^*, W_j\rangle \leq -(K-1) \\ (1-\gamma)\epsilon\sum_{i=1}^{K} R_i^- + R_1^-\gamma\epsilon - R_j^-\epsilon & \langle f^*, W_j\rangle > -(K-1) \end{cases}$$

Under the assumption 5, when $\gamma \in [0, 0.5]$, $P > 0$ and $N < 0$. Therefore, $r(f^*|x) - r(f^{**}|x) = P - N < 0$ which contradicts to the definition of f^*.

Similarly, we can show that $\langle f^*, W_2\rangle \geq -(K-1)$.

Next, we need to show $\text{argmax}_j\langle f^*, W_j\rangle$ is unique. Since $\langle f^*, W_2\rangle \geq -(K-1)$ and $\langle f^*(x), W_1\rangle \geq \langle f^*(x), W_j\rangle$ for $\forall j \neq 1$, we have $\langle f^*, W_1\rangle \geq 1$. Then we have two cases:

a. $\langle f^*, W_1\rangle > 1$

If there exists j such that $\langle f^*, W_1\rangle = \langle f^*, W_j\rangle > 1$, we can construct f^{**} such that $\langle f^{**}, W_t\rangle = \langle f^*, W_t\rangle$, $\forall t \neq 1, j$ and $\langle f^{**}, W_1\rangle = \langle f^*, W_1\rangle + \epsilon > 1$, $\langle f^{**}, W_j\rangle = \langle f^*, W_j\rangle - \epsilon > 1$ where ϵ is a small positive number. One can verify that $r(f^*|x) - r(f^{**}|x) = P - N$, where $P = (R_1^+ - R_j^+)\epsilon > 0$ and $N = (R_j^- - R_1^-)\epsilon < 0$. So we have $r(f^*|x) > r(f^{**}|x)$ which contradicts to the definition of f^*.

b. $\langle f^*, W_1\rangle = 1$

Since $\langle f^*, W_2\rangle \geq -(K-1)$, if $\langle f^*, W_1\rangle = 1$, we have $\langle f^*, W_t\rangle = 1, \forall t \neq 2$ and $\langle f^*, W_2\rangle = -(K-1)$. There exists $j \neq 1, 2$ and f^{**} such that $\langle f^{**}, W_t\rangle = \langle f^*, W_t\rangle$, $\forall t \neq 2, j$ and $\langle f^{**}, W_2\rangle = \langle f^*, W_2\rangle + \epsilon \in (-(K-1), -1]$, $\langle f^{**}, W_j\rangle = \langle f^*, W_j\rangle - \epsilon \in [-1, 1)$ where ϵ is a small positive number. One can verify that $r(f^*|x) - r(f^{**}|x) = [(1-\gamma)\sum_{i=1}^{K} R_i^+ + R_2^+\gamma - R_j^+ - R_j^- + R_2^-]\epsilon > 0$ under the assumptions 4, 5 and $\gamma \in [0, 0.5]$. So it contradicts to the definition of f^*.

Therefore, we have proved $\langle f^*, W_1\rangle > \langle f^*, W_t\rangle, \forall t \neq 1$ and the Fisher consistency is obtained.

▮

Proof of Theorem 2.3.2 Under the constraint $\langle f^*, W_t\rangle \geq -\frac{1}{K-1}, \forall t \in \{1, ..., K\}$ and the condition $\sum_{i=1}^{K}\langle f^*, W_i\rangle = 0$, we have $-\frac{1}{K-1} \leq \langle f^*, W_t\rangle \leq 1, \forall t$. Define $m_j^+ = \sum_{i\neq j} R_i^+$, $m_j^- = \sum_{i\neq j} R_i^-$, $m_j = m_j^+ + m_j^-$ and $j_0 = \mathrm{argmin}_j\, m_j$. We can show

$$\begin{aligned}
r(f|x) &= \sum_{i=1}^{K} R_i^+\Big[(1-\gamma)\sum_{j\neq i}(1+\langle f, W_j\rangle)_+ + \gamma(K-1-\langle f, W_i\rangle)_+\Big] \\
&\quad - \sum_{i=1}^{K} R_i^-\Big[(1-\gamma)\sum_{j\neq i}(1-\langle f, W_j\rangle)_+ + \gamma(K-1+\langle f, W_i\rangle)_+\Big] \\
&= \sum_{i=1}^{K} R_i^+ \sum_{j\neq i}(1+\langle f, W_j\rangle) - \sum_{i=1}^{K} R_i^- \sum_{j\neq i}(1-\langle f, W_j\rangle) \\
&= \sum_{j=1}^{K}(1+\langle f, W_j\rangle)\sum_{i\neq j} R_i^+ - \sum_{j=1}^{K}(1-\langle f, W_j\rangle)\sum_{i\neq j} R_j^- \\
&= \sum_{j=1}^{K}\langle f, W_j\rangle m_j + \sum_{j=1}^{K}(m_j^+ - m_j^-) \\
&= \sum_{j\neq j_0}\langle f, W_j\rangle(m_j - m_{j_0}) + \sum_{j=1}^{K}(m_j^+ - m_j^-) \\
&\geq -\sum_{j\neq j_0}\frac{1}{K-1}(m_j - m_{j_0}) + \sum_{j=1}^{K}(m_j^+ - m_j^-)
\end{aligned}$$

So f^*, the minimizer of the conditional loss $r(f|x)$ satisfies

$$\langle f^*, W_j\rangle = \begin{cases} 1 & j = j_0 \\ -\dfrac{1}{K-1} & \text{otherwise} \end{cases}$$

Thus, $\mathrm{argmax}_j\langle f^*, W_j\rangle = \mathrm{argmin}_j\, m_j$ and the Fisher consistency is obtained. ▮

Chapter 3

Dynamic Treatment Regimes based on Neural Classification Tree

3.1 Introduction

In the previous chapter, we extended outcome weighted learning to multicategory setting as well as negative outcome where we approximated the $0-1$ loss function by the hinge loss function in reinforeced multicategory support vector machine. In this chapter, we make use of weighted squared loss to approximate the weighted misclassification error and apply a classification tree model to find the optimal treatment regimes. Due to the smoothness of the weighted squared loss function, neural network can be used for the training of the tree to improve prediction accuracy. Unlike traditional decision tree which determines splitting variables and cutting points in a greedy manner, the neural classification tree estimates all cutting points simultaneously. The traditional classification tree is then reconstructed based on the estimated result from training data to increase the interpretability.

The rest of the section organizes as follows. Section 3.2 briefly reviews the neural network and the decision tree. Section 3.3 describes the neural network architecture of the proposed model and a variable selection method used to reduce tree depth. A tree reconstruction and pruning process is proposed in section 3.4 to construct a traditional classification tree. We present all model details for single stage treatment. The extension to multiple stage treatment is the same as that in section 2.3.2. The simulation study and application to **STAR*D** data are carried out to assess the performance of the model in section 3.5. The chapter is concluded in section 3.6.

3.2 Literature review

3.2.1 Neural network

The idea of neural network is to extract linear combination of the inputs as derived features and then model the target as a nonlinear function of these features (Hastie et al., 2009). A neural network is typically organized in layers including input layer, hidden layers and output layer. In each layer there are one or more nodes which connect different layers via activation functions. Typical activation functions include sigmoid, tanh, ReLU and softmax functions (Hastie et al., 2009). Figure 3.1 provides an example of a standard neural network. It includes one input layer, one hidden layer and one output layer. The numbers of nodes in these layers are 4, 3 and 3 respectively. Given all trainable parameters fixed, the four input features plus a bias term B_1 are weighted summed. The summation flows into the hidden layer by applying an activation function. The result in hidden layer plus another bias term B_2 is then transformed to the output in the output layer with an objective function being employed for measuring the prediction accuracy. The trainable parameters are then updated by minimizing the objective loss function via the gradient descent. This forward and backpropagation procedure is repeated until either convergence or the maximum number of iterations is achieved.

The neural network is flexible for both regression and classification in the sense that it can capture nonlinear and complex relationship between inputs and outputs since it is actually a mathematical model with approximation functions. It has shown excellent performance in many areas such as image recognition and natural language processing (Hastie et al., 2009).

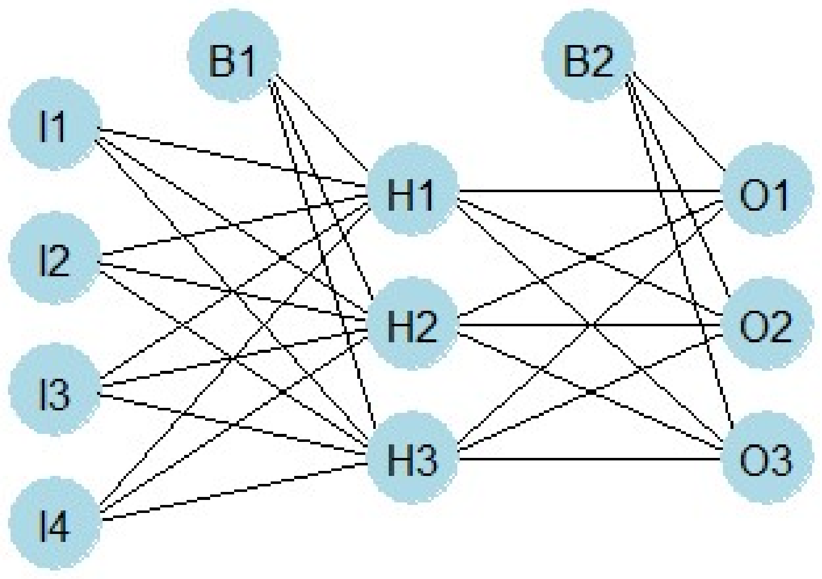


Figure 3.1: An example of standard neural network; *I*, *H* and *O* denote nodes in input, hidden and output laye. *B* denotes the bias term

.

However, neural networks are black-box models meaning that we are not able to interpret how each feature influences the final output.

3.2.2 Classification tree

The idea of classification tree is to paritition the feature space into a set of rectangles and then fit a simple model in each node (Hastie et al., 2009). It is simple, interpretable and also powerful for tabular data.

Specifically, suppose the feature space is parititioned to M regions $R_1, \cdots, R_M$, the observation with covariate vector x will be classified as

$$f(x) = \sum_{m=1}^{M} c_m I(x \in R_m)$$

where I is the indicator function , c_m is the class label prediction in region R_m with N_m observations, and is determined by the majority vote

$$c_m = \text{argmax}_k \, \hat{p}_{km}$$

where $\hat{p}_{km}$ is the proportion of class k observations in region m

$$\hat{p}_{km} = \frac{1}{N_m} \sum_{x_i \in R_m} I(y_i = k) \tag{3.1}$$

The best partition is searched by a greedy algorithm. Take the top split as an example and for simplicity, assume the covariate vector $X = (X_1, \cdots, X_p)$ is continuous. Two nodes formed by variable j and cutting point s is

$$R_1(j, s) = \{X | X_j \leq s\} \quad \text{and} \quad R_2(j, s) = \{X | X_j > s\}$$

The impurity measure Q_1, Q_2 can be computed for each node. Popular choices of impurity measure for classification tree include

Misclassification error:	$\frac{1}{N_m} \sum_{i \in R_m} I(y_i \neq c_m)$
Gini index:	$\sum_{k \neq k'} \hat{p}_{km} \hat{p}_{k'm}$
Cross-entropy:	$-\sum_{k=1}^{K} \hat{p}_{km} \log \hat{p}_{km}$

where $\hat{p}_{km}$ is defined in equation (3.1). The best splitting variable and points are then found by minimizing $Q_1 + Q_2$ over all possible pairs (j, s). For the new daughter node, this process will be recursively repeated until the prespecified stopping criterion is met, for example, the maximum number of nodes is achieved or the number of observations in each node is smaller than the prespecified threshold.

3.3 Neural network architecture for the DTR

Without loss of generality, suppose the input vector $X = (x_1, \cdots, x_m) \in R^m$ is continuous, output vector $s = (s_1, \cdots, s_K) \in [0,1]^K$ satisfying $\sum_{j=1}^{K} s_j = 1$. m and K are the number of features and treatments respectively. In addition to the input and output layer in the standard neural network, a neural decision tree includes a binning layer for split decisions and a layer of Kronecker product for representing terminal node (Yang et al., 2018).

The basic idea is that for each feature x_j, the binning layer outputs an almost one-hot vector indicating the bins to which x_j belongs via a one-layer neural network. For example, consider three bins with two cutting points -1, 1 for the j-th feature. For observations $x_j = -2$, $x_j = 0$ and $x_j = 2$ the outputs of the one-layer neural network are expected to approximate the exact one-hot vectors $(1,0,0)$, $(0,1,0)$ and $(0,0,1)$ respectively. In total there are m such vectors. The Kronecker product of these m vectors indexes the leaf node where the covariate vector X belongs to (Yang et al., 2018).

To better illustrate the idea of using a binning layer, we take variable x_j as an example. Suppose there are q cutting points $p_1 < p_2 < \cdots < p_q$, where q is a tuning parameter selected by cross validation or manually pre-set and $p_t, t = 1, \cdots, q$ are trainable parameters.

Define $\omega = (1, 2, \cdots, q+1)$ a constant vector and $b = (0, -p_1, -p_2 - p_3, \cdots, -p_1 - p_2 - \cdots - p_q)$ the bias vector. A one-layer neural network can be constructed for variable x_j (Yang et al., 2018)

$$f_{\omega,b,\tau}(x_j) = \text{softmax}((\omega x_j + b)/\tau) \tag{3.2}$$

It outputs a $(q+1)$-dimensional vector. The i-th component of the softmax function $\Big(\text{softmax}(z)\Big)_i = \frac{e^{z_i}}{\sum_{j=1}^{q+1} e^{z_j}}$ with $z = (z_1, \cdots, z_{q+1})$. τ controls how close the output approximates the exact one-hot vector. The smaller the τ is, the closer the result from equation (3.2) is to the real one-hot vector.

To illustrate the rationale behind (3.2) as well as the specification of ω and b, assume the maximizer of (3.2) for a specific x_j is k. For simplicity, assume $k \neq 1, q+1$. The situation for

$k = 1$ or $q + 1$ is very similar. We then have

$$\begin{cases} kx_j - (p_1 + \cdots + p_{k-1}) > (k-1)x_j - (p_1 + \cdots + p_{k-2}) \\ kx_j - (p_1 + \cdots + p_{k-1}) > (k+1)x_j - (p_1 + \cdots + p_k) \end{cases} \tag{3.3}$$

So $x_j \in (p_k, p_{k+1})$. It shows that the output based on (3.2) with the specified ω and b indicates the interval to which a specific x_j belongs. When x_j is a categorical variable, the one-hot encoder can be viewed as binning output.

Since there are m features, in binning layer m one-layer neural networks will be constructed. Each one has the same structure as (3.2) for continuous variable or uses one-hot encoding for a categorical variable. Theoretically, the number of cutting points q can vary for different features, but for simplicity we assume there is a common q for all continuous standardized features. So the one-layer neural networks for continuous variables in binning layer only differ in the choice of $p_1, \cdots, p_q$.

The m vectors obtained from binning layer are then taken Kronecker product. Since each of the m vectors indicates the bins to which a feature belongs, the Kronecker product actually partitions the feature space and indicates the region where an observation with covariate vector X belongs to. The layer of Kronecker product is then fully-connected to the output layer of K nodes with softmax as activation function where K is the number of available treatments.

Another important element in neural network is its objective function. Recall that OWL minimizes weighted classification error rate $E\Big[\frac{Y}{\pi(A,X)} I(A \neq a)\Big]$ where Y is the outcome, A is the observed treatment, a is the estimated optimal treatment and $\pi(A, X) = Pr(A|X)$ is the treatment assignment probability. In this chapter, we further assume $E(Y|A, X) \geq 0$.

Define the weighted squared loss as

$$L(s) = E_{A,X}\Big[W_{A,X} \sum_{j=1}^{K} (s_j - v_j)^2\Big] \tag{3.4}$$

where $W_{A,X} = \frac{E(Y|A,X)}{\pi(A,X)} \geq 0$ is the weight, $v = (v_1, \cdots, v_K)$ is a one-hot vector for multiclassification. If patient i receives treatment j, then $v_t^{(i)} = 0$ for $t \neq j$ and $v_j^{(i)} = 1$. $s(X) = (s_1(X), \cdots, s_K(X))$ is the decision vector function satisfying $\sum_{i=1}^K s_i(X) = 1$.

We have the following theorem

Theorem 3.3.1 *If s^* minimizes the weighted squared loss (3.4) with the constraint $\sum_{j=1}^K s_j = 1$, $d(x) = \operatorname{argmax}_{j \leq K} s_j(x)$ gives the optimal treatment.*

Proof The weighted squared loss $L(s)$ given the observation $X = x$ is

$$\begin{aligned} L(s|x) &= E_{A|X=x}\Big[W_{A,X} \sum_{j=1}^K (s_j - d_j)^2 | x\Big] \\ &= \sum_{A=1}^K E(Y|A,x) \sum_{j=1}^K (s_j - d_{jA})^2 \end{aligned} \tag{3.5}$$

where $d_{jA} = I(j = A)$. So it suffices to show that if a measurable function $s^* = (s_1^*, \cdots, s_K^*)$ minimizes $L(s|x)$ in (3.5) for every x, then we must have $d(x) = \operatorname{argmax}_{j \leq K} s_j(x)$ gives the optimal treatment.

Define

$$\begin{aligned} G(s, \lambda, x) &= L(s|x) - \lambda(\sum_{j=1}^K s_j - 1) \\ &= \sum_{A=1}^K E(Y|A,x) \sum_{j \neq A} s_j^2 + \sum_{A=1}^K E(Y|A,x)(s_A - 1)^2 - \lambda(\sum_{j=1}^K s_j - 1) \\ &= \sum_{A=1}^K E(Y|A,x) \sum_{j=1}^K s_j^2 - 2\sum_{A=1}^K E(Y|A,x) s_A - \lambda(\sum_{j=1}^K s_j - 1) + \sum_{A=1}^K E(Y|A,x) \end{aligned} \tag{3.6}$$

So for $j = 1, \cdots, K$, let

$$\frac{\partial G}{\partial s_j} = 2\sum_{A=1}^K E(Y|A,x) s_j - 2E(Y|j,x) - \lambda = 0 \tag{3.7}$$

Summing up the K equations, we obtain

$$\begin{aligned} \lambda &= 0 \\ s_j &= \frac{E(Y|j,x)}{\sum_{A=1}^{K} E(Y|A,x)} \quad \text{for} \quad j = 1, \cdots, K \end{aligned} \tag{3.8}$$

Because of the convexity of the weighted squared error loss, equation (3.8) is the minimizer. So $\operatorname{argmax}_{j \le K} s_j(x) = \operatorname{argmax}_{j \le K} E(Y|j,x)$. ∎

One strength of neural classification tree is that unlike traditional decision tree who searches split variables and points in greedy algorithm, it estimates all cutting points simultaneously by directly minimizing the overall loss. The optimization as well as implementation is easily carried out in TensorFlow. However, it still has some shortcomings. First, when the number of features or cutting points is large, the tree will also be very large because the use of Kronecker product may lead to useless splits or inclusion of noisy variables. Second, due to the error resulted from approximating one-hot vector by softmax function in binning layer, neural classification tree may give different prediction results for observations in the same region. In addition, it cannot give the sequential tree structure as the traditional tree does since only final nodes are known, and this will hurt part of the interpretability. For the second problem, we propose a traditional tree construction and pruning method for growing the tree sequentially after training neural classification tree in next section. To solve the first problem, a variable selection needs to be implemented before training the neural network.

Variable selection in DTR is quite different from that in other fields in the sense that only prescriptive variables which influence the optimal treatment assignment are of interest. Predictive variables which plays a role in the outcome may be useless in determining treatment. Various techniques have been proposed for the variable selection in DTR (Gunter et al., 2011; Lu et al., 2013; Qian and Murphy, 2011; Zhang and Zhang, 2018; Fan et al., 2016). We make use of the sequential advantage selection in our approach. Compared to other methods, sequential advantage selection can handle a large number of covariates even if the sample size is small and it excludes marginally important but jointly unimportant variables or vice versa (Fan et al., 2016).

Let $V = \{j_1, \cdots, j_k\}$ be an arbitrary model with $x_{j_1}, \cdots, x_{j_k}$ as the selected covariates and $F = \{1, \cdots, p\}$ be the full model. X_i is the covariate vector for i-th patient and $X_{i(M)} = \{X_{ij} : j \in V\}$ is the covariate for i-th patient corresponding to model V. The sequential advantage of a variable X_j with $k-1$ variables $X_{j_1}, \cdots, X_{j_{(k-1)}}$ already in the model is defined as:

$$S_j^{(k)} = \frac{1}{n}\sum_{i=1}^{n}\Big\{\max_a \hat{E}(Y|X_{V_j^{(k)}} = x_{iV_j^{(k)}}, A = a) - \hat{E}(Y|X_{V_j^{(k)}} = x_{iV_j^{(k)}}, A = a_{opt}^{(k-1)}(x_{iV^{(k-1)}}))\Big\} \tag{3.9}$$

where $V^{(k-1)} = \{j_1, \cdots, j_{k-1}\}$, $V_j^{(k)} = V^{(k-1)} \cup \{j\}$, and $a_{opt}^{(k-1)}(x_{iV^{(k-1)}})$ is the optimal treatment regime based on $k-1$ variables $X_{j_1}, \cdots, X_{j_{k-1}}$

In summary, the basic idea of the procedure is that starting from null model, at each time k we select variables which maximizes $S_j^{(k)}$ in equation (3.9). The selecting process is repeated until the maximum number of selected features is met or the ratio of the maximum advantage at the current step and the sum of maximum advantages prior and up to the current step is below the pre-set threshold. In this thesis, random forest is used for modelling outcome Y in equation (3.9).

3.4 Tree reconstruction and pruning

Without loss of generality, assume the selected features are $\{X_1, \cdots, X_m\}$, the cutting points for variable X_j obtained from neural classification tree is $\{p_{j1}, \cdots, p_{jq}\}$ and the possible nodes are represented by the pair (X_j, p_{ji}) for $j = 1, \cdots, m$ and $i = 1, \cdots, q$. We first describe how to find the top parental node and each daughter node can be determined similarly.

Denote V_0 as the overall misclassification rate. It is the empirical misclassification rate from the neural classification tree. For each pair (X_j, p_{ji}),$j = 1, \cdots, m$ and $i = 1, \cdots q$, the observations can be divided into two subsets: observations with $X_j \geq p_{ji}$ and observations with $X_j < p_{ji}$. Then the misclassification rates V_a and V_b on the two subsets are computed. Since the inequality

$$\frac{t_1 + t_2}{n_1 + n_2} < \frac{t_1}{n_1} + \frac{t_2}{n_2}$$

always holds when $0 < t_1 < n_1$ and $0 < t_2 < n_2$, we have $V_0 < V_a + V_b$. By searching over all pairs (X_j, p_{ji}), $j = 1, \cdots, m$, $i = 1, \cdots, q$, the pair minimizing $V_a + V_b$ is selected as the top parental node. For example, assume the pair is (X_1, p_{11}). Then at the top parental node, the splitting variable is X_1 and corresponding cutting point is p_{11}. After the top parental node is specified, the observations are divided into two subsets. A daughter node for each subset will be determined by repeating the same process in the current subset except that the nodes in previous branches will not be considered any more.

The tree reconstruction process will stop if either of the two following criterions is met: the predicted treatments of all observations in the current node are identical or all pairs have been used for splitting. In the reconstructed tree, the number of different predicted treatments in each terminal node may be more than 1. In this case, we adjust the prediction by taking the majority vote, or by random assignment if the numbers of predicted treatments are the same.

The tree is then pruned from bottom. Suppose sibling nodes O_1 and O_2 have one common parental node O and the predicted treatments under O_1, O_2 after adjustment by majority vote or random assignment is P and Q respectively. If $V_{O_1} + V_{O_2} < \min_{P,Q}\{V_{OP}, V_{OQ}\} + \eta$, the two nodes O_1 and O_2 will be merged where V_{OP} and V_{OQ} are the misclassification rates if all observations under node O are assigned treatment P and Q respectively. η is a prespecified parameter. After merging, the new predicted treatment under node O is $\mathrm{argmin}_{i \in \{P,Q\}} V_{Oi}$.

The tree reconstruction and pruning processes are different from those for the traditional classification tree. In traditional classification trees, the sum of misclassification errors (or other impurity measures) of the two daughter nodes is always smaller than the misclassification rate (or other impurity measures) of the parental node and the predicted class is determined when the splitting node is given. In our method, we estimate all cutting points simultaneously using neural network. The prediction result is then used for tree reconstruction and adjusted after the tree is reconstructed. The adjusted prediction is then used for tree pruning so that we finally grow a tree of which the structure is the same as the traidional tree but the cutting points are trained simultaneously. In stead of the prediction from the black-box neural network, we can also get prediction from our reconstructed and pruned tree so that the interpretability is maintained.

3.5 Numerical investigation

In this section, we describe both the simulation studies and real data application of the proposed method.

3.5.1 Simulation study

To evaluate the performance of the proposed neural classification tree, simulation studies were carried out for a variety of scenarios. We consider either the treatment probability is known or estimated from data. For the first case, we randomly assign treatments so that the treatment categories are balanced and treatment probabilities are determined. For the second case, we construct multinomial regression to estimate the treatment probability. Both tree type and non-tree type data with either single stage or multi-stage treatments are considered.

For each of the settings, we repeat the simulation 500 times. For each simulation run, we generate a data set with a sample size of 3500. We randomly choose half of them as training data and the remaining is used as testing data. For the single stage, we use misclassification error rate and the empirical value function for assessment. For multi-stage, we only use empirical value function to assess the model. The misclassification error rate in the single stage setting is defined as $\mathbb{P}_n[I(A^{opt} = d(X))]$ and the empirical value function is defined as $\mathbb{P}_n[\prod_{t=1}^{T} \frac{I(A_t=d_t(H_t))}{\pi_t(A_t|H_t)Y}]/\mathbb{P}_n[\prod_{t=1}^{T} \frac{I(A_t=d_t(H_t))}{\pi_t(A_t|H_t)}]$ where $\mathbb{P}_n$ is the empirical average operator.

We consider 11 scenarios:

1. A three-treatment case. The number of variables is set to be 1000. The covariate vector X is generated from multivariate normal distribution with mean $\mathbf{0}$ and identity variance.

The actual treatment is generated from multinomial distribution with

$$Pr(A^{obs} = 1|X) = \frac{1}{1 + \exp(-2 + X_1 + 2X_2 - X_3) + \exp(-1 - 2X_1 + 2X_2)}$$
$$Pr(A^{obs} = 2|X) = \frac{\exp(-2 + X_1 + 2X_2 - X_3)}{1 + \exp(-2 + X_1 + 2X_2 - X_3) + \exp(-1 - 2X_1 + 2X_2)}$$
$$Pr(A^{obs} = 3|X) = \frac{\exp(-1 - 2X_1 + 2X_2)}{1 + \exp(-2 + X_1 + 2X_2 - X_3) + \exp(-1 - 2X_1 + 2X_2)}$$

The true optimal treatment satisfies:

$$A^{opt} = \begin{cases} 1 & X_1 \leq 0 \quad \text{and} \quad X_2 \leq 0.5 \\ 2 & X_1 > 0 \quad \text{and} \quad X_3 \leq 0.5 \\ 3 & \text{otherwise} \end{cases}$$

The outcome R follows $N(2 + X_4 + X_5 + 10I(A^{obs} = A^{opt}), 1)$.

2. All the settings are the same as scenario 1 except that the observed treatment $A^{obs} \sim \text{Uniform}\{1, 2, 3\}$.

3. All the settings are the same as scenario 1 except that

$$A^{opt} = \begin{cases} 1 & -0.3 \geq X_1 \leq 0.5 \\ 2 & X_1 < -0.3 \quad \text{and} \quad X_2 \geq 1; X_1 > 0.5 \quad \text{and} \quad X_2 \leq 2 \\ 3 & \text{otherwise} \end{cases}$$

4. All the settings are the same as scenario 3 except that the observed treatment $A^{obs} \sim \text{Uniform}\{1, 2, 3\}$

5. All the settings are the same as scenario 1 except that

$$A^{opt} = \begin{cases} 1 & X_1 < -0.5 \\ 2 & X_1 \geq -0.5 \quad \text{and} \quad X_1 + X_2 < 0.5 \\ 3 & \text{otherwise} \end{cases}$$

6. All the settings are the same as scenario 5 except that the observed treatment $A^{obs} \sim$ Uniform$\{1, 2, 3\}$.

7. All the settings are the same as scenario 1 except that

$$A^{opt} = \begin{cases} 1 & X_1 X_2 < -0.2 \\ 2 & X_1 X_2 \geq -0.2 \quad \text{and} \quad X_3 < 0 \\ 3 & \text{otherwise} \end{cases}$$

8. All the settings are the same as scenario 7 except that the observed treatment $A^{obs} \sim$ Uniform$\{1, 2, 3\}$.

9. A four-treatment case. The number of variables is 1000. The covariate vector X is generated from multivariate normal distribution with mean $\mathbf{0}$ and identity variance. The actual treatment is generated from multinomial distribution with

$$\begin{aligned} Pr(A^{obs} = 1|X) &= \frac{1}{S} \\ Pr(A^{obs} = 2|X) &= \frac{\exp(X_1 + 3X_2 - X_3 + X_5)}{S} \\ Pr(A^{obs} = 3|X) &= \frac{\exp(-2X_1 + 2X_2)}{S} \\ Pr(A^{obs} = 4|X) &= \frac{\exp(-2X_1 + 2X_2 - 4X_3 - 2X_4 - X_5)}{S} \end{aligned}$$

where $S = 1 + \exp(X_1 + 3X_2 - X_3 + X_5) + \exp(-2X_1 + 2X_2) + \exp(-2X_1 + 2X_2 - 4X_3 - 2X_4 - X_5)$

The true optimal treatment satisfies:

$$A^{opt} = \begin{cases} 1 & X_1 < -0.65 \\ 2 & X_2 > -0.4 \quad \text{and} \quad X_3 < 0 \\ 3 & X_2 > -0.4 \quad \text{and} \quad X_3 \geq 0 \\ 4 & \text{otherwise} \end{cases}$$

The outcome R follows $N(2 + X_4 + X_5 + 10I(A^{obs} = A^{opt}), 1)$.

10. All the settings are the same as scenario 9 except that $A^{obs} \sim \text{Uniform}\{1, 2, 3, 4\}$.

11. A two-stage case. The covariate vector $\mathbf{X_1}$, $\mathbf{X_2}$ follows multivariate normal distribution with mean $\mathbf{0}$ and identity variance. The actual treatment at stage 1 is generated from multinomial distribution with

$$Pr(A_1^{obs} = 1|X_1) = \frac{1}{S}$$
$$Pr(A_1^{obs} = 2|X_1) = \frac{\exp(-1 - X_{11} + 2X_{12} - X_{13} - X_{14} - X_{15})}{S}$$
$$Pr(A_1^{obs} = 3|X_1) = \frac{\exp(-1 - 2X_{11} + 2X_{12} - 2X_{13} + 2X_{14})}{S}$$

where $S = 1 + \exp(-1 - X_{11} + 2X_{12} - X_{13} - X_{14} - X_{15}) + \exp(-1 - 2X_{11} + 2X_{12} - 2X_{13} + 2X_{14})$ The actual treatment at stage 2 is generated from multinomial distribution with

$$Pr(A_2^{obs} = 1|X_2) = \frac{1}{S}$$
$$Pr(A_2^{obs} = 2|X_2) = \frac{\exp(-1 - X_{21} + X_{22} - X_{23} - 2X_{24})}{S}$$
$$Pr(A_2^{obs} = 3|X_2) = \frac{\exp(-2X_{21} + X_{22} - 2X_{23} + X_{24})}{S}$$

where $S = 1 + \exp(-1 - X_{21} + X_{22} - X_{23} - 2X_{24}) + \exp(-2X_{21} + X_{22} - 2X_{23} + X_{24})$. The true optimal treatment at first stage A_1^{opt} satisfies

$$A_1^{opt} = \begin{cases} 1 & X_{11} \leq 0 \quad \text{and} \quad X_{12} \leq 0.5 \\ 2 & X_{11} > 0 \quad \text{and} \quad X_{13} \leq 0.5 \\ 3 & \text{otherwise} \end{cases}$$

The true optimal treatment at second stage A_2^{opt} satisfies

$$A_2^{opt} = \begin{cases} 1 & X_{21} \leq -0.1 \quad \text{and} \quad X_{22} \leq 0.7 \\ 2 & X_{21} > -0.1 \quad \text{and} \quad X_{23} \leq 0.4 \\ 3 & \text{otherwise} \end{cases}$$

The outcome R_1 follows $N(u_1, 1)$ where $u_1 = 10I(A_1^{opt} = A_1^{obs}) + 2 + X_{12} + X_{14} - X_{13} - X_{15}$. The outcome R_2 follows $N(u_2, 1)$ where $u_2 = 5I(A_2^{opt} = A_2^{obs}) + 2 + (X_{22}^2 + X_{24})^2 X_{21} + X_{23}X_{25}$.

Scenarios 1 $\sim$ 4 have standard tree type boundary. Scenarios 1 and 2 involve single split for each feature while in scenarios 3 and 4 the number of split points for features is 2. Scenarios 5 $\sim$ 8 have non-standard tree type boundary. The split rules in scenarios 5 and 6 involve linear combination of features while scenarios 7 and 8 have nonlinear split rule. Scenarios 9 and 10 consider a four-treatment case so that the effect of the number of treatment options can be observed. Finally, scenario 11 involves two stages as well as a more complex nonlinear main effect.

Table 3.1 and 3.2 show the misclassification error rates and empirical value function obtained by validation data of size 1750. The column NCT shows the result obtained from the neural classification tree while the column NCT-correct gives the result obtained from the reconstructed and pruned tree. The columns OVR and OVO give the simulation results from One-Versus-Rest and One-Versus-One methods based on support vector machine. For scenarios 1 $\sim$ 4, 9 $\sim$ 10, in terms of both misclassification error rates and empirical value function, NCT and NCT-correct show comparable performance while for scenarios 5 $\sim$ 8 NCT performs

Table 3.1: Misclassification error rates approximated by validation data set of size 1750, averaged over 500 simulation runs; the numbers in parenthesis are standard deviations over 500 simulation runs

Scenario	NCT	NCT-correct	OVR	OVO
1	8.67% (3.7%)	8.92% (4.8%)	17.85% (2.5%)	18.66% (2.4%)
2	2.82% (1.8%)	2.22% (2.2%)	9.22% (1.2%)	9.59% (1.2%)
3	20.35% (3.3%)	20.04% (5.9%)	22.67% (2.5%)	17.23% (2.4%)
4	5.39% (1.9%)	5.02% (4.4%)	12.26% (1.5%)	11.11% (1.3%)
5	18.18% (5.9%)	23.76% (9.0%)	33.05% (3.7%)	35.37% (3.7%)
6	14.97% (2.5%)	17.78% (3.9%)	8.76% (1.2%)	8.65% (1.2%)
7	17.13% (4.5%)	24.28% (5.8%)	25.05% (2.9%)	26.43% (2.7%)
8	7.90% (2.7%)	17.06% (3.0%)	10.19% (1.2%)	10.52% (1.2%)
9	16.52% (8.0%)	18.03% (9.5%)	38.10% (3.8%)	39.59% (3.3%)
10	4.30% (4.0%)	3.98% (4.5%)	12.53% (1.7%)	12.80% (1.6%)

Table 3.2: Empirical value function approximated by validation data set of size 1750, averaged over 500 simulation runs; the numbers in parenthesis are standard deviations over 500 simulation runs

Scenario	NCT	NCT-correct	OVR	OVO
		Single stage: $T = 1$		
1	11.14 (0.933)	11.14 (0.621)	10.21 (0.457)	10.13 (0.463)
2	11.72 (0.212)	11.78 (0.256)	11.06 (0.177)	11.03 (0.183)
3	10.75 (0.588)	10.87 (0.968)	9.98 (0.441)	10.01 (0.407)
4	11.46 (0.211)	11.48 (0.462)	10.77 (0.194)	10.89 (0.183)
5	10.20 (0.865)	9.72 (1.362)	8.68 (0.544)	8.45 (0.521)
6	10.50 (0.294)	10.21 (0.448)	11.12 (0.162)	11.13 (0.167)
7	10.35 (0.642)	9.71 (1.140)	9.48 (0.546)	9.33 (0.552)
8	11.21 (0.294)	10.37 (0.384)	10.98 (0.181)	10.96 (0.183)
9	10.31 (1.279)	10.21 (1.593)	8.27 (0.892)	8.11 (0.869)
10	11.57 (0.424)	11.59 (0.486)	10.75 (0.237)	10.73 (0.225)
		Two stage: $T = 2$		
11	17.43 (1.827)	17.21 (1.909)	10.89 (1.496)	10.96 (1.634)

better than NCT-correct. It suggests that the corrected tree is valid if only one variable is involved at each split point, which is the case for the standard decision tree. If the split rule involves linear or nonlinear combination of different variables, NCT performs better. For scenario 11, the estimated value function computed from NCT and NCT-correct are very close. It suggests the correction method works well for dynamic case. Comparing the results in scenarios 1 and 2 with results in scenarios 3 and 4, when the true number of cutting points for variables increases, there is an apparent increase in misclassification error rate. However, from table 3.2 the corresponding decrease in estimated value function is relatively small. It indicates that the proposed methods are still valid when there are various cutting points for variables, since for DTR the estimated value function is a more comprehensive measure than the misclassification error. Comparing the results in scenarios $1, 3, 5, 7, 9$ and results in scenarios $2, 4, 6, 8, 10$, all methods perform better when the treatment probability is known. The results obtained from One-Versus-Rest (OVR), One-Versus-One (OVO) are also given in table 3.1 and table 3.2. It shows that the proposed methods give better results than OVR and OVO in all scenarios except in scenario 6. For the two-stage case, this comparison is more obvious.

3.5.2 Application to STAR*D study

A brief introduction of the data set is described in section 1.1. Variables are described in section 2.4.1. The only difference is that neural classification tree requires the outcome to be positive. Thus, keeping other elements defined the same as in Chapter 2, we define the treatment outcome at each stage as:

Y_1 : 27 - QIDS score at the end of stage 1 if remission was achieved, $13.5 - \frac{1}{2}$ QIDS score at the end of stage 1 if the patient moved to stage 2

Y_2 : $13.5 - \frac{1}{2}$ QIDS score at the end of stage 2

Figure 3.2 gives comparison of our proposed methods. Three different number of cutting points 1, 2 and 3 are set for each feature. Comparison between results from NCT and its corrected tree is also given. From the figure, it shows all settings gives similar results. When the number of cutting points is 1, the mean estimated value function based on NCT and NCT-correct are 18.30 (sd = 0.57) and 18.17 (sd = 0.66), respectively. When the number of cutting points is 2, the mean estimated value function based on NCT and NCT-correct are 18.23 (sd = 0.57) and 18.20 (sd = 0.71), respectively. When the number of cutting points is 3, the mean estimated value function based on NCT and NCT-correct are 18.24 (sd = 0.49) and 18.24

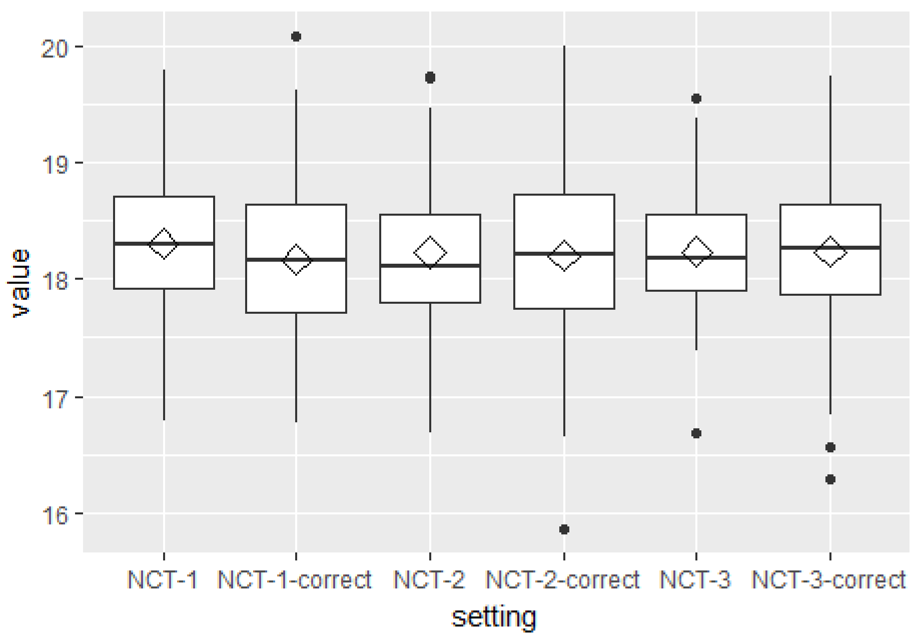


Figure 3.2: Estimated value function based on 100 repetitions of application for Sequential Treatment Alternatives to Relieve Depression data

(sd = 0.66), respectively. The results suggest that only 1 cutting point needs to be considered for each feature and the reconstructed and pruned tree is valid for prediction for this data set.

3.6 Conclusion

In this chapter, we proposed a method of estimating dynamic treatment regimes based on neural classification tree. The advantage of the proposed method is that the neural network estimates split variables and cutting points simultaneously rather than in greedy manner. In addition, the true complex relationship between optimal treatment and treatment reward is not assumed and therefore can be approximated flexibly. However, the prediction of observations in the same node may vary using the neural classification tree. The order of the split variables or points is unknown. We further propose a method of reconstructing and pruning the tree based on the output from neural classification tree to overcome these disadvantages.

Simulation study and real data application are conducted to illustrate our method. Overall, the proposed methods work well. However, from the numerical investigation we find the corrected NCT does not perform very satisfactorily when the true split rules involve linear or nonlinear combinations of variables. For these two cases, further improvement and modification are required.

Chapter 4

Conclusion

This thesis was motivated by the Sequential Treatment Alternatives to Relieve Depression (STAR*D) study. The study included four levels and at each level patients were randomized to various treatment options with both clinical and treatment information considered. The goal of this study was to compare the effectiveness of different multi-level treatment options for patients with major depressive disorder based on the collected information.

In this study, the primary outcome is the clinician-rated Quick Inventory of Depressive Symptomatology (QIDS) score of which higher values correspond to higher severity. In order to make it consistent with the prerequisite that larger outcome is preferred in the field of DTR, researchers always replace the the original outcome with its negative which results in negative outcome.

Outcome weighted learning, as the foundation of classification-based methods for estimating dynamic treatment regimes, focuses on binary treatment, single stage as well as non-negative outcome. Motivated by the STAR*D study, the objective of this thesis is to explore extension of outcome weighted learning to multi-armed treatment, multiple stages as well as negative outcome.

In this thesis, two methods were discussed to achieve this objective. The first method is based on multicategory support vector machine. An angle-based loss function which linearly combines naive hinge loss and vector hinge loss with a tuning parameter $\gamma \in [0, 1]$. The loss function is then modified to allow for negative treatment outcome with two different sets of assumptions. One is about the treatment outcome ranking and the other one is about the lower bound of the inner products $\langle f(X), W_j \rangle$, $j = 1, \cdots, K$ where K is the number of treatment options and W_j is defined as in equation (2.1). We theoretically prove that under the assumptions of treatment outcome ranking, the Fisher consistency of the classifier can be maintained for $\gamma \in [0, 0.5]$. Under the assumption that $\langle f(X), W_j \rangle \geq -\frac{1}{K-1}$, $j = 1, \cdots, K$, the Fisher consistency of the classifier can be maintained for any $\gamma \in [0, 1]$. The second proposed method combines neural network and decision tree. While neural network is known for flexibility with approximation functions, decision tree is highly interpretable. To ensure the smoothness of the objective function in neural network, we proposed to use a weighted squared loss function for the classification problem and proved that the output can give the optimal treatment regime. The prediction is hard to interpret for neural network since it is a black-box method and the neural network classification tree does not give a tree structure as the standard decision tree. To solve these problems, we further proposed a method of tree reconstruction. We adjust the prediction based on the reconstructed tree and then prune the tree.

The two methods have different advantages. The multicategory outcome weighted learning directly borrows techniques of multicategory support vector machine. So it is conceptually simple. The neural classification tree considers both accuracy and interpretability. The true relationship between treatment and outcome is not assumed and hence can be modelled flexibly and the optimization procedure can be easily implemented in TensorFlow.

There are several possible directions for future research. As seen in our simulation study, when the number of treatment increases the prediction accuracy will decrease. Currently, we only consider a very small number of treatments. Improvement needs to be made in the presence of large number of treatments. Furthermore, Laber and Zhao (2015) considers purity measure for continuous treatment in their tree-based method. They use a kernel smoother to approximate the indicator function. A distribution over treatment for each covariate is defined to approximate the treatment rule. However, in the field of DTRs, the extension to continuous treatment has not been well studied. Modifying our model for large number of treatments or even continuous treatment is of interest. Additionally, for our second model, although we use sequential advantage selection for variable selection, other feature selection techniques to choose variables can also be applied since the variable selection is independent of the proposed model. A variable importance measure may be defined to better rank the variables.

Other explanation techniques in machine learning literature such as LIME and TreeSHAP can also be employed for the DTR problems to enhance interpretability. Local interpretable model-agnostic explanations (LIME) constructs interpretable representation for actual features and identifies an interpretable surrogate model over the interpretable representation which can be a good approximation to the original predictions locally. The surrogate model can be any interpretable model such as decision tree (Ribeiro et al., 2016). SHapley Additive exPlanation (SHAP) method assigns each feature an importance value for a particular prediction and explains the prediction based on the contribution of each feature (Lundberg and Lee, 2017). TreeSHAP, as a variant of SHAP, is designed specifically for tree ensemble methods with reduced computation complexity (Lundberg et al., 2018). We may further explore those methods in DTR setting and compare with our proposed methods.

Bibliography

Richard Bellman. *Dynamic programming*. Princeton:Princeton University Press, 2010.

Bibhas Chakraborty and Erica E.M. Moodie. *Statistical methods for dynamic treatment regimes: reinforcement learning, causal inference, and personalized medicine*. Springer, 2013.

Bibhas Chakraborty and Susan A. Murphy. Dynamic treatment regimes. *Annual Review of Statistics and Its Application*, 1(1):447–464, 2014.

Jingxiang Chen, Haoda Fu, Xuanyao He, Michael R. Kosorok, and Yufeng Liu. Estimating individualized treatment rules for ordinal treatments. *Biometrics*, 74(3):924–933, 2018.

Ailin Fan, Wenbin Lu, and Rui Song. Sequential advantage selection for optimal treatment regimes. *The Annals of Applied Statistics*, 10(1):32–53, 2016.

Sheng Fu, Qinying He, Sanguo Zhang, and Yufeng Liu. Robust outcome weighted learning for optimal individualized treatment rules. *Journal of Biopharmaceutical Statistics*, 29(3): 606–624, 2019.

Lacey Gunter, Ji Zhu, and Susan A. Murphy. Variable selection for qualitative interactions. *Statistical Methodology*, 8(1):42–55, 2011.

Trevor Hastie, Robert Tibshirani, and Jerome Friedman. *The elements of statistical learning*. New York: Springer-Verlag, 2009.

Eric B. Laber and Ying-Qi Zhao. Tree-based methods for individualized treatment regimes. *Biometrika*, 102(3):501–514, 2015.

Eric B. Laber, Daniel J. Lizotte, Min Qian, William E. Pelham, and Susan A. Murphy. Dynamic treatment regimes: technical challenges and applications. *Electronic Journal of Statistics*, 8 (1):1225–1272, 2014.

Yoonkyung Lee, Yi Lin, and Grace Wahba. Multicategory support vector machines: theory and application to the classification of microarray data and satellite radiance data. *journal of the American Statistical Association*, 99(465):67–81, 2004.

Ying Liu, Yuanjia Wang, Michael R. Kosorok, Yingqi Zhao, and Donglin Zeng. Augmented outcome-weighted learning for estimating optimal dynamic treatment regimes. *Statistics in Medicine*, 37(26):3776–3788, 2018.

Yufeng Liu and Ming Yuan. Reinforced multicategory support vector machines. *Journal of Computational and Graphical Statistics*, 20(4):901–919, 2011.

Zhilan Lou, Jun Shao, and Menggang Yu. Optimal treatment assignment to maximize expected outcome with multiple treatments. *Biometrics*, 74(2):506–516, 2018.

Wenbin Lu, Hao Helen Zhang, and Donglin Zeng. Variable selection for optimal treatment decision. *Statistical methods in medical research*, 22(5):493–504, 2013.

Scott Lundberg and Su-In Lee. A unified approach to interpreting model predictions, 2017.

Scott M. Lundberg, Gabriel G. Erion, and Su-In Lee. Consistent individualized feature attribution for tree ensembles, 2018.

Seo Young Park and Yufeng Liu. From the support vector machine to the bounded constraint machine. *Statistics and Its Interface*, 2(3):285–298, 2009.

Min Qian and Susan A. Murphy. Performance guarantees for individualized treatment rules. *Annals of Statistics*, 39(2):1180–1210, 2011.

Marco Tulio Ribeiro, Sameer Singh, and Carlos Guestrin. “Why Should I Trust You?”: Explaining the Predictions of Any Classifier. *arXiv e-prints*, art. arXiv:1602.04938, February 2016.

James M. Robins. Optimal structural nested models for optimal sequential decisions. *Proceedings of the Second Seattle Symposium in Biostatistics*, pages 189–326, 2004.

A.John Rush, Maurizio Fava, Stephen R. Wisniewski, Philip W. Lavori, Madhukar H. Trivedi, Harold A. Sackeim, Michael E. Thase, Andrew A. Nierenberg, Frederic M. Quitkin, T. Michael Kashner, David J. Kupfer, Jerrold F. Rosenbaum, Jonathan Alpert, Jonathan W. Stewart, Patrick J. McGrath, Melanie M. Biggs, Kathy Shores-Wilson, Barry D. Lebowitz, Louise Ritz, George Niederehe, and for the STAR*D Investigators Group 1. Sequenced treatment alternatives to relieve depression(star*d): Rationale and design. *Controlled Clinical Trials*, 25(1):119–142, 2004.

Phillip J. Schulte, Anastasios A. Tsiatis, Eric B. Laber, and Marie. Q- and a-learning methods for estimating optimal dynamic treatment regimes. *Statistical Science*, 29(4):640–661, 2014.

Richard S. Sutton and Andrew G. Barto. *Reinforcement learning: an introduction*. A Bradford Book, 2018.

Yongxin Yang, Irene Garcia Morillo, and Timothy M. Hospedales. Deep neural decision trees. *arXiv preprint arXiv:1806.06988*, 2018.

Baqun Zhang and Min Zhang. Variable selection for estimating the optimal treatment regimes in the presence of a large number of covariate. *The Annals of Applied Statistics*, 12(4): 2335–2358, 2018.

Baqun Zhang, Anastasios A. Tsiatis, Eric B. Laber, and Marie Davidian. Robust estimation of optimal dynamic treatment regimes for sequential treatment decisions. *Biometrika*, 100(3): 681–694, 2013.

Chong Zhang and Yufeng Liu. Multicategory angle-based large-margin classification. *Biometrika*, 101(3):625–640, 2014.

Chong Zhang, Yufeng Liu, Junhui Wang, and Hongtu Zhu. Reinforced angle-based multicategory support vector machines. *Journal of Computational and Graphical Statistics*, 25(3): 806–825, 2016.

Ying-Qi Zhao, Donglin Zeng, A. John Rush, and Michael R. Kosorok. Estimating individualized treatment rules using outcome weighted learning. *Journal of the American Statistical Association*, 107(499):1106–1118, 2012.

Ying-Qi Zhao, Donglin Zeng, Eric B. Laber, and Michael R. Kosorok. New statistical learning methods for estimating optimal dynamic treatment regimes. *Journal of the American Statistical Association*, 110(510):583–598, 2015.

Appendix A

R Functions for First Model

```r
library(nnet)
library(MASS)

W.gen = function(k)
{
  X = matrix(0,k,k-1)
  X[1,]=rep((k-1)^(-1/2),k-1)
  for (index in 2:k)
  {
    X[index,]=rep( -(1+sqrt(k))/((k-1)^(1.5)), k-1)
    X[index,index-1]=X[index,index-1]+sqrt(k/(k-1))
  }
  return(X)
}
#W.gen generates the representation matrix for treatment

A.matrix.gen=function(k,a.train )

{
  nobs=length(a.train)
  A.matrix = matrix(0,nobs,k-1)
  X=W.gen(k)
  for (index in 1:nobs)
  {
    A.matrix[index,] = X[a.train[index],]
  }
  return(A.matrix)
}
#a.train: treatment assignment for training data, should be a vector
#A.matrix.gen generate the representation matrix for treatment assignment
    for specific dataset


X.matrix.gen=function(covariate)
{
  A=matrix(0,nrow(covariate),ncol(covariate)+1)
  A[,1]=1
  A[,-1]=as.matrix(covariate)
  return(A)
}
#X.matrix.gen generates design matrix with intercept 1
#the i-th row is covariate information for the i-th observation
#(i,j) element is the j-th variable for observation i

propensity_model=function(covariate,a.train)
{
  Data=data.frame(covariate,y=a.train)
  model=multinom(y~X1+X2+X3,data=Data)
  #model=multinom(y~X1+X2+X3+X4+X5,data=Data)
  return(model)
 }

propensity_score=function(covariate,a.train,model,i)
{
  data=data.frame(t(covariate[i,]))
  prob=predict(model,newdata=data,type='probs')
  score=as.numeric(prob[a.train[i]])
  return(score)
}
#estimate propensity score


alpha_upper=function(i,j,gamma,covariate,reward,a.train,model)
{
  omega=reward[i]/propensity_score(covariate,a.train,model,i)
  if(j==a.train[i])
    c=gamma*omega
  else
    c=(1-gamma)*omega
  return(abs(c))
}
#compute the upper bound of each alpha_ij

######prediction function for linear case
pred=function(k,newdata,beta)
{
  nobs=nrow(newdata)
  X=X.matrix.gen(newdata)
  response=rep(0,nobs)
  W=W.gen(k)
  for(i in 1:nobs)
  {
    f=beta%*%X[i,]
    angle=W%*%f
    response[i]=which.max(angle)
  }
  return(response)
```

```r
}

#######prediction function for nonlinear case
pred=function(k,covariate,newdata,theta,kpara)
{
  covariate=as.matrix(covariate)
  newdata=as.matrix(newdata)
  nobs=nrow(newdata)
  response=rep(0,nobs)
  W=W.gen(k)
  nobs_train=nrow(covariate)
  kernel_vector=rep(0,nobs_train+1)
  for(i in 1:nobs)
  {
    kernel_vector=rep(0,nobs_train+1)
    kernel_vector[1]=1
    for(j in 1:nobs_train)
      kernel_vector[j+1]=kernel(newdata[i,],covariate[j,],kpara)
    f=theta%*%kernel_vector
    angle=W%*%f
    response[i]=which.max(angle)
  }
  return(response)
}

kernel<-function(x,y,kpara)
{
  difference=x-y
  norm=t(difference)%*%difference
  expo=-norm/(2*kpara^2)
  return(exp(expo))
}

```

```
Gram<-function(covariate,kpara)
{
  covariate=as.matrix(covariate)
  nobs=nrow(covariate)
  gram=matrix(0,nrow=nobs,ncol=nobs)
  for(i in 1:nobs)
    for(j in 1:nobs)
      gram[i,j]=kernel(covariate[i,],covariate[j,],kpara)
  return(gram)
}

######## loss function without constraint:linear case#######
alpha_ij_update_2_c=function(k,i,j,lambda,gamma,covariate,atrain,initial,
    reward, score)
{
  nobs=nrow(covariate)
  ncov=ncol(covariate)
  f=0
  X=X.matrix.gen(covariate)
  W=W.gen(k)
  Amatrix=A.matrix.gen(k,atrain)
  xinner=t(X[i,])%*%X[i,]
  sum_w=t(W[j,])%*%W[j,]
  denom=sum_w*xinner
  alpha_initial=initial
  A=0
  for(q in 1:(k-1))
  {
    B1=rep(0,ncov+1)
    B2=rep(0,ncov+1)
    B3=rep(0,ncov+1)
    B4=rep(0,ncov+1)
    for(index_i in 1:nobs)
    {
      if(reward[index_i]>=0)
      {
        B1=B1+alpha_initial[index_i,atrain[index_i]]*Amatrix[index_i,q]*X[
    index_i,]
        for(index_j in 1:k)
          if(index_j!=atrain[index_i])
            B3=B3+alpha_initial[index_i,index_j]*W[index_j,q]*X[index_i,]
      }
      else
      {
        B2=B2+alpha_initial[index_i,atrain[index_i]]*Amatrix[index_i,q]*X[
    index_i,]
        for(index_j in 1:k)
          if(index_j!=atrain[index_i])
            B4=B4+alpha_initial[index_i,index_j]*W[index_j,q]*X[index_i,]
      }
    }
    if(reward[i]>=0)
    {
      if(j==atrain[i])
        B1=B1-alpha_initial[i,atrain[i]]*Amatrix[i,q]*X[i,]
      else
        B3=B3-alpha_initial[i,j]*W[j,q]*X[i,]
    }
    else
    {
      if(j==atrain[i])
        B2=B2-alpha_initial[i,atrain[i]]*Amatrix[i,q]*X[i,]
      else
        B4=B4-alpha_initial[i,j]*W[j,q]*X[i,]
    }
    B=B1-B2-B3+B4
    f=0
    for(index in 1:(ncov+1))
      f=f+B[index]*X[i,index]
    A=A+f*W[j,q]

  }
  if(reward[i]>=0)
  {
    if(j==atrain[i])
      A=-A+nobs*lambda*(k-1)
    else
      A=A+nobs*lambda
  }
  else
  {
    if(j==atrain[i])
      A=A+nobs*lambda*(k-1)
    else
      A=-A+nobs*lambda
  }
  new_alpha_ij=A/denom
  omega=reward[i]/score[i]
  c=0
  if(j==atrain[i])
    c=gamma*omega
  else
    c=(1-gamma)*omega
  c=abs(c)
  upper=c
  if(new_alpha_ij>upper)
    new_alpha_ij=upper
  else if(new_alpha_ij<0)
    new_alpha_ij=0
  alpha_initial[i,j]=new_alpha_ij
  return(alpha_initial)
}

alpha_optim_c=function(k,lambda,gamma,covariate,a.train,alpha_initial,
    reward,score,maxiter=1000)
{
  nobs=nrow(covariate)
  t=0
  #score=sapply(c(1:nobs),propensity_score,covariate=covariate,a.train=a.
    train,model=model)
  pre_alpha=alpha_initial
  iter_alpha=alpha_initial
  while(t<=maxiter)
  {
    t=t+1
    for(i in 1:nobs)
      for(j in 1:k)
      {
        new_alpha=alpha_ij_update_2_c(k,i,j,lambda,gamma,covariate,a.train,
    iter_alpha,reward,score)
        iter_alpha=new_alpha
      }
    difference=sum(abs(pre_alpha-new_alpha))
    if(difference<0.001*nobs*k)
    {
      alpha=new_alpha
      break
    }
    else
    {
      pre_alpha=new_alpha
      alpha=new_alpha
    }
    print(t)
  }
  return(alpha)
}

beta_q_alpha=function(k,q,covariate,lambda,alpha,reward,a.train)
{
  W=W.gen(k)
  X=X.matrix.gen(covariate)
  A.matrix=A.matrix.gen(k,a.train)
  nobs=nrow(covariate)
  A=0
  B=0
  C=0
  D=0
  index_pos=which(reward>=0)
  index_neg=which(reward<0)
  for(i in index_pos)
    for(j in 1:k)
    {
      if(j==a.train[i])
    A=A+alpha[i,j]*A.matrix[i,q]*X[i,]
      else C=C+alpha[i,j]*W[j,q]*X[i,]
    }
  for(i in index_neg)
    for(j in 1:k)
    {
    if(j==a.train[i])
      B=B+alpha[i,j]*A.matrix[i,q]*X[i,]
    else
      D=D+alpha[i,j]*W[j,q]*X[i,]
    }
  beta_q=(A-B-C+D)/(lambda*nobs)
  return(beta_q)
}
#beta_q_alpha generates beta_q if the alpha has been found by the dual
    problem
#beta_q is the coefficient vector in f_q(x) q is between 1 and k-1, length=
    ncol(covariate)+1
#alpha stores the estimation of all alpha_ij, should be a matrix nrow=nobs,
     ncol=k
#lambda is the tuning parameter,scalar
#reward is the outcome, vector

beta_alpha=function(k,covariate,lambda,alpha,reward,a.train)
{
  beta=matrix(0,nrow=k-1,ncol=ncol(covariate)+1)
  for(q in 1:k-1)
    beta[q,]=beta_q_alpha(k,q,covariate,lambda,alpha,reward,a.train)
  return(beta)
}

########### loss function without constraint:nonlinear case##########
alpha_ij_update_c=function(k,i,j, lambda, gamma,covariate, atrain,initial,
    reward, score, kpara,gram)
{
  nobs=nrow(covariate)
  W=W.gen(k)
  alpha_initial=initial
  A=0
  for(q in 1:(k-1))
  {
    B1=0
    B2=0
    B3=0
```

```
B4=0
C1=0
C2=0
C3=0
C4=0
for(index_i in 1:nobs)
{
  if(reward[index_i]>=0)
  {
    B1=B1+alpha_initial[index_i,atrain[index_i]]*W[atrain[index_i],q]*gram[index_i,i]
    C1=C1+alpha_initial[index_i,atrain[index_i]]*W[atrain[index_i],q]
    for(index_j in 1:k)
      if(index_j!=atrain[index_i])
      {
        B3=B3+alpha_initial[index_i,index_j]*W[index_j,q]*gram[index_i,i]
        C3=C3+alpha_initial[index_i,index_j]*W[index_j,q]
      }
  }
  else
  {
    B2=B2+alpha_initial[index_i,atrain[index_i]]*W[atrain[index_i],q]*gram[index_i,i]
    C2=C2+alpha_initial[index_i,atrain[index_i]]*W[atrain[index_i],q]
    for(index_j in 1:k)
      if(index_j!=atrain[index_i])
      {
        B4=B4+alpha_initial[index_i,index_j]*W[index_j,q]*gram[index_i,i]
        C4=C4+alpha_initial[index_i,index_j]*W[index_j,q]
      }
  }
```

```
    }
    if(reward[i]>=0)
    {
      if(j==atrain[i])
      {
        B1=B1-alpha_initial[i,atrain[i]]*W[atrain[i],q]*gram[i,i]
        C1=C1-alpha_initial[i,atrain[i]]*W[atrain[i],q]
      }
      else
      {
        B3=B3-alpha_initial[i,j]*W[j,q]*gram[i,i]
        C3=C3-alpha_initial[i,j]*W[j,q]
      }
    }
    else
    {
      if(j==atrain[i])
      {
        B2=B2-alpha_initial[i,atrain[i]]*W[atrain[i],q]*gram[i,i]
        C2=C2-alpha_initial[i,atrain[i]]*W[atrain[i],q]
      }
      else
      {
        B4=B4-alpha_initial[i,j]*W[j,q]*gram[i,i]
        C4=C4-alpha_initial[i,j]*W[j,q]
      }
    }
    B=B1-B2-B3+B4
    C=C1-C2-C3+C4
    A=A+(B+C)*W[j,q]
  }
  if(reward[i]>=0)
  {
    if(j==atrain[i])
      A=-A+nobs*lambda*(k-1)
    else
      A=A+nobs*lambda
  }
  else
  {
    if(j==atrain[i])
      A=A+nobs*lambda*(k-1)
    else
      A=-A+nobs*lambda
  }
  new_alpha_ij=A/2
  omega=reward[i]/score[i]
  c=0
  if(j==atrain[i])
    c=gamma*omega
  else
    c=(1-gamma)*omega
  c=abs(c)
  upper=c
  if(new_alpha_ij>upper)
    new_alpha_ij=upper
  else if(new_alpha_ij<0)
    new_alpha_ij=0
  alpha_initial[i,j]=new_alpha_ij
  return (alpha_initial)
}


alpha_optim_c=function(k,lambda,gamma,covariate,a.train,alpha_initial,
    reward,score,kpara,maxiter=1000)
{
```

```r
  nobs=nrow(covariate)
  t=0
  gram=Gram(covariate,kpara)
  #score=sapply(c(1:nobs),propensity_score,covariate=covariate,a.train=a.
    train,model=model)
  pre_alpha=alpha_initial
  iter_alpha=alpha_initial
  while(t<=maxiter)
  {
    t=t+1
    for(i in 1:nobs)
      for(j in 1:k)
      {
        new_alpha=alpha_ij_update_c(k,i,j,lambda,gamma,covariate,a.train,
    iter_alpha,reward,score,kpara,gram)
        iter_alpha=new_alpha
      }
    diff=sum(abs(pre_alpha-new_alpha))
    if(diff<0.0001*nobs*k)
    {
      alpha=new_alpha
      break
    }
    else
    {
      pre_alpha=new_alpha
      alpha=new_alpha
    }
    print(t)
  }
  return(alpha)
}
```

```
theta_q_0_alpha=function(k,q,covariate,lambda,alpha,reward,a.train)
{
  W=W.gen(k)
  nobs=nrow(covariate)
  A=0
  B=0
  C=0
  D=0
  index_pos=which(reward>=0)
  index_neg=which(reward<0)
  for (i in index_pos) {
    for(j in 1:k)
    {
      if(j==a.train[i])
        A=A+alpha[i,j]*W[j,q]
      else
        C=C+alpha[i,j]*W[j,q]
    }
  }
  for(i in index_neg)
    for(j in 1:k)
    {
      if(j==a.train[i])
        B=B+alpha[i,j]*W[j,q]
      else
        D=D+alpha[i,j]*W[j,q]
    }
  theta_q_0=(A-B-C+D)/(lambda*nobs)
  return(theta_q_0)
}

theta_q_alpha=function(k,q,covariate,lambda,alpha,reward,a.train,kpara)
{
```

```
W=W.gen(k)
gram=Gram(covariate,kpara)
nobs=nrow(covariate)
A=0
B=0
C=0
D=0
index_pos=which(reward>=0)
index_neg=which(reward<0)
for(i in index_pos)
{
  KKi=rep(0,nobs)
  KKi[i]=1
  for(j in 1:k)
  {
    if(j==a.train[i])
      A=A+alpha[i,j]*W[j,q]*KKi
    else
      C=C+alpha[i,j]*W[j,q]*KKi
  }
}
for(i in index_neg)
{
  KKi=rep(0,nobs)
  KKi[i]=1
  for(j in 1:k)
  {
    if(j==a.train[i])
      B=B+alpha[i,j]*W[j,q]*KKi
    else
      D=D+alpha[i,j]*W[j,q]*KKi
  }
}
```

```
  theta_q=(A-B-C+D)/(lambda*nobs)
  return(theta_q)
}

theta_alpha=function(k,covariate,lambda,alpha,reward,a.train,kpara)
{
  theta=matrix(0,nrow=k-1,ncol=nrow(covariate)+1)
  for(q in 1:(k-1))
    theta[q,]=c(theta_q_0_alpha(k,q,covariate,lambda,alpha,reward,a.train),
   theta_q_alpha(k,q,covariate,lambda,alpha,reward,a.train,kpara))
  return(theta)
}

##############loss function with constraint: linear case#############
alpha_ij_update_c=function(k,i,j,lambda,gamma,covariate, atrain,initial_
    alpha,initial_lower,reward, v,score)
{
  nobs=nobs(covariate)
  ncov=ncol(covariate)
  f=0
  X=X.matrix.gen(covariate)
  W=W.gen(k)
  Amatrix=A.matrix.gen(k,atrain)
  xi=X[i,]
  xinner=t(X[i,]%*%X[i,]
  sum_w=t(W[j,])%*%W[j,]
  denom=sum_w*xinner
  alpha_initial=initial_alpha
  lower_initial=initial_lower
  A=0
  for(q in 1:(k-1))
  {
    B1=rep(0,ncov+1)
```

```
B2=rep(0,ncov+1)
B3=rep(0,ncov+1)
B4=rep(0,ncov+1)
for(index_i in 1:nobs)
{
  if(reward[index_i]>=0)
  {
    B1=B1+alpha_initial[index_i,atrain[index_i]]*Amatrix[index_i,q]*X[
index_i,]
    for(index_j in 1:k)
      if(index_j!=atrain[index_i])
        B3=B3+alpha_initial[index_i,index_j]*W[index_j,q]*X[index_i,]
  }
  else
  {
    B2=B2+alpha_initial[index_i,atrain[index_i]]*Amatrix[index_i,q]*X[
index_i,]
    for(index_j in 1:k)
      if(index_j!=atrain[index_i])
        B4=B4+alpha_initial[index_i,index_j]*W[index_j,q]*X[index_i,]
  }
}
if(reward[i]>=0)
{
  if(j==atrain[i])
    B1=B1-alpha_initial[i,atrain[i]]*Amatrix[i,q]*X[i,]
  else
    B3=B3-alpha_initial[i,j]*W[j,q]*X[i,]
}
else
{
  if(j==atrain[i])
    B2=B2-alpha_initial[i,atrain[i]]*Amatrix[i,q]*X[i,]
```

```
      else
        B4=B4-alpha_initial[i,j]*W[j,q]*X[i,]
    }
    B=B1-B2-B3+B4
    L=rep(0,ncov+1)
    for(index_i in 1:nobs)
      for(index_j in 1:k)
      {
        L=L+lower_initial[index_i,index_j]*v*W[index_j,q]*X[index_i,]
      }
    f=0
    for(index in 1:(ncov+1))
      f=f+(B+L)[index]*X[i,index]
    A=A+f*W[j,q]
  }
  if(reward[i]>=0)
  {
    if(j==atrain[i])
      A=-A+nobs*lambda*(k-1)
    else
      A=A+nobs*lambda
  }
  else
  {
    if(j==atrain[i])
      A=A+nobs*lambda*(k-1)
    else
      A=-A+nobs*lambda
  }
  new_alpha_ij=A/denom;
  omega=reward[i-1]/score[i-1]
  c=0
  if(j==atrain[i])
```

```
    c=gamma*omega
  else
    c=(1-gamma)*omega
  c=abs(c)
  upper=c-lower_initial[i,j]
  if(new_alpha_ij>upper)
    new_alpha_ij=upper
  else if(new_alpha_ij<0)
    new_alpha_ij=0
  alpha_initial[i,j]=new_alpha_ij
  return (alpha_initial)
}

lower_ij_update_c=function(k, i,j, lambda, gamma, covariate, atrain,
    initial_alpha, initial_lower, reward,v, score)
{
  nobs=nrow(covariate)
  ncov=ncol(covariate)
  X=X.matrix.gen(covariate)
  W=W.gen(k)
  xi=X[i,]
  xinner=t(xi)%*%xi
  sum_w=t(W[j,])%*%W[j,]
  denom=sum_w*xinner*v*v
  alpha_initial=initial_alpha
  lower_initial=initial_lower
  A=0
  for(q in 1:(k-1))
  {
    B=rep(0,ncov+1)
    C=rep(0,ncov+1)
    D=rep(0,ncov+1)
    E=rep(0,ncov+1)
    L=rep(0,ncov+1)
    for(index_i in 1:nobs)
    {
      for(index_j in 1:k)
      {
        if(index_j==atrain[index_i])
        {
          if(reward[index_i]>=0)
            B=B+alpha_initial[index_i,index_j]*W[index_j,q]*X[index_i,]
          else
            C=C+alpha_initial[index_i,index_j]*W[index_j,q]*X[index_i,]
        }
        else
        {
          if(reward[index_i]>=0)
            D=D+alpha_initial[index_i,index_j]*W[index_j,q]*X[index_i,]
          else
            E=E+alpha_initial[index_i,index_j]*W[index_j,q]*X[index_i,]
        }
        if(index_i!=i || index_j!=j)
          L=L+lower_initial[index_i,index_j]*v*W[index_j,q]*X[index_i,]
      }
    }
    med=B-C-D+E+L
    sum=0
    for( index_j in 1:(ncov+1))
      sum=sum+med[index_j-1]*X[i,index_j]
    A=A+sum*W[j,q]*v
  }
  A=-A+nobs*lambda*v/(k-1)
  new_lower_ij=A/denom
  omega=reward[i-1]/score[i-1]
  c=0
  if(j==atrain[i])
    c=gamma*omega
  else
    c=(1-gamma)*omega
  c=abs(c)
   upper=c-alpha_initial(i-1,j-1)
  if(new_lower_ij<0)
    new_lower_ij=0
  else if(new_lower_ij>upper)
    new_lower_ij=upper
  lower_initial[i,j]=new_lower_ij
  return (lower_initial)
}

para_optim_c=function(k,lambda,gamma, covariate,a.train,alpha_initial,lower
    _initial,reward,v,score,maxiter=1000)
{
  nobs=nrow(covariate)
  t=0
 # score=sapply(c(1:nobs),propensity_score,covariate=covariate,a.train=a.
    train,model=model)
  pre_alpha=alpha_initial
  iter_alpha=alpha_initial
  pre_lower=lower_initial
  iter_lower=lower_initial
  while(t<=maxiter)
  {
    t=t+1
    for(i in 1:nobs)
      for(j in 1:k)
      {
        new_alpha=alpha_ij_update_c(k,i,j,lambda,gamma,covariate,a.train,
    iter_alpha,iter_lower,reward,v,score)
```

```
        iter_alpha=new_alpha
        new_lower=lower_ij_update_c(k,i,j,lambda,gamma,covariate,a.train,
   iter_alpha,iter_lower,reward,v,score)
        iter_lower=new_lower
      }
    difference1=sum(abs(pre_alpha-new_alpha))
    difference3=sum(abs(pre_lower-new_lower))
    sum_difference=difference1+difference3
    if(sum_difference<0.0001*2*nobs*k)
    {
      alpha=new_alpha
      lower=new_lower
      break
    }
    else
    {
      pre_alpha=new_alpha
      alpha=new_alpha
      pre_lower=new_lower
      lower=new_lower
    }

  }
  return(list(alpha,lower))
}

beta_q_para=function(k,q,covariate,lambda,alpha,lower,reward,v,a.train)
{
  W=W.gen(k)
  X=X.matrix.gen(covariate)
  A.matrix=A.matrix.gen(k,a.train)
  nobs=nrow(covariate)
  A=0
  B=0
  C=0
  D=0
  L=0
  index_pos=which(reward>=0)
  index_neg=which(reward<0)
  for(i in index_pos)
    for(j in 1:k)
    {
      if(j==a.train[i])
        A=A+alpha[i,j]*A.matrix[i,q]*X[i,]
      else C=C+alpha[i,j]*W[j,q]*X[i,]
    }
  for(i in index_neg)
    for(j in 1:k)
    {
      if(j==a.train[i])
        B=B+alpha[i,j]*A.matrix[i,q]*X[i,]
      else
        D=D+alpha[i,j]*W[j,q]*X[i,]
    }
  for(i in 1:nobs)
    for(j in 1:k)
    {
      L=L+lower[i,j]*v*W[j,q]*X[i,]
    }

  beta_q=(A-B-C+D+L)/(lambda*nobs)
  return(beta_q)
}

beta_alpha=function(k,covariate,lambda,alpha,lower,reward,v,a.train)
{
  beta=matrix(0,nrow=k-1,ncol=ncol(covariate)+1)
  for(q in 1:k-1)
    beta[q,]=beta_q_para(k,q,covariate,lambda,alpha,lower,reward,v,a.train)
  return(beta)
}

#############loss function with constraint:nonlinear case##########
alpha_ij_update_c=function(k,i,j,lambda,gamma,covariate,atrain, initial_
    alpha,initial_lower,reward,v, score,kpara,gram)
{
  nobs=nrow(covariate)
  W=W.gen(k)
  alpha_initial=initial_alpha
  lower_initial=initial_lower
  A=0
  for( q in 1:(k-1))
  {
    B1=0
    B2=0
    B3=0
    B4=0
    C1=0
    C2=0
    C3=0
    C4=0
    EB=0
    EC=0
    for(index_i in 1:nobs)
    {
      if(reward[index_i]>=0)
      {
        B1=B1+alpha_initial[index_i,atrain[index_i]]*W[atrain[index_i],q]*
    gram[index_i,i]
```

```
   C1=C1+alpha_initial[index_i,atrain[index_i]]*W[atrain[index_i],q]
   for(index_j in 1:k)
   {
    EB=EB+lower_initial[index_i,index_j]*v*W[index_j,q]*gram[index_i,
i]
    EC=EC+lower_initial[index_i,index_j]*v*W[index_j,q]
    if(index_j!=atrain[index_i])
    {
     B3=B3+alpha_initial[index_i,index_j]*W[index_j,q]*gram[index_i,
i]
     C3=C3+alpha_initial[index_i,index_j]*W[index_j,q]
    }
   }
  }
  else
  {
   B2=B2+alpha_initial[index_i,atrain[index_i]]*W[atrain[index_i],q]*
gram[index_i,i]
   C2=C2+alpha_initial[index_i,atrain[index_i]]*W[atrain[index_i],q]
   for(index_j in 1:k)
   {
    EB=EB+lower_initial[index_i,index_j]*v*W[index_j,q]*gram[index_i,
i]
    EC=EC+lower_initial[index_i,index_j]*v*W[index_j,q]
    if(index_j!=atrain[index_i])
    {
     B4=B4+alpha_initial[index_i,index_j]*W[index_j,q]*gram[index_i,
i]
     C4=C4+alpha_initial[index_i,index_j]*W[index_j,q]
    }
   }
  }
 }
```

```
    if(reward[i]>=0)
    {
      if(j==atrain[i])
      {
        B1=B1-alpha_initial[i,atrain[i]]*W[atrain[i],q]*gram[i,i]
        C1=C1-alpha_initial[i,atrain[i]]*W[atrain[i],q]
      }
      else
      {
        B3=B3-alpha_initial[i,j]*W[j,q]*gram[i,i]
        C3=C3-alpha_initial[i,j]*W[j,q]
      }
    }
    else
    {
      if(j==atrain[i])
      {
        B2=B2-alpha_initial[i,atrain[i]]*W[atrain[i],q]*gram[i,i]
        C2=C2-alpha_initial[i,atrain[i]]*W[atrain[i],q]
      }
      else
      {
        B4=B4-alpha_initial[i,j]*W[j,q]*gram[i,i]
        C4=C4-alpha_initial[i,j]*W[j,q]
      }
    }
    B=B1-B2-B3+B4+EB
    C=C1-C2-C3+C4+EC
    A=A+(B+C)*W[j,q]
  }
  if(reward[i]>=0)
  {
    if(j==atrain[i])
```

```
      A=-A+nobs*lambda*(k-1)
    else
      A=A+nobs*lambda
  }
  else
  {
    if(j==atrain[i])
      A=A+nobs*lambda*(k-1)
    else
      A=-A+nobs*lambda
  }
  new_alpha_ij=A/2
  omega=reward[i]/score[i]
  c=0
  if(j==atrain[i])
    c=gamma*omega
  else
    c=(1-gamma)*omega
  c=abs(c)
  upper=c-lower_initial[i,j]
  if(new_alpha_ij>upper)
    new_alpha_ij=upper
  else if(new_alpha_ij<0)
    new_alpha_ij=0
  alpha_initial[i,j]=new_alpha_ij
  return (alpha_initial)
}

lower_ij_update_c=function(k,i,j,lambda,gamma,covariate,atrain,initial_
    alpha,initial_lower,reward,v,score,kpara,gram)
{
  nobs=nrow(covariate)
  W=W.gen(k)
```

```
alpha_initial=initial_alpha
lower_initial=initial_lower
A=0
for(q in 1:(k-1))
{
  B1=0
  B2=0
  B3=0
  B4=0
  C1=0
  C2=0
  C3=0
  C4=0
  EB=0
  EC=0
  for(index_i in 1:nobs)
  {
    if(reward[index_i]>=0)
    {
      B1=B1+alpha_initial[index_i,atrain[index_i]]*W[atrain[index_i],q]*
 gram[index_i,i]
      C1=C1+alpha_initial[index_i,atrain[index_i]]*W[atrain[index_i],q]
      for(index_j in 1:k)
      {
        EB=EB+lower_initial[index_i,index_j]*v*W[index_j,q]*gram[index_i,
 i]
        EC=EC+lower_initial[index_i,index_j]*v*W[index_j,q]
        if(index_j!=atrain[index_i])
        {
          B3=B3+alpha_initial[index_i,index_j]*W[index_j,q]*gram[index_i,
 i]
          C3=C3+alpha_initial[index_i,index_j]*W[index_j,q]
        }
      }
    }
    else
    {
      B2=B2+alpha_initial[index_i,atrain[index_i]]*W[atrain[index_i],q]*gram[index_i,i]
      C2=C2+alpha_initial[index_i,atrain[index_i]]*W[atrain[index_i],q]
      for(index_j in 1:k)
      {
        EB=EB+lower_initial[index_i,index_j]*v*W[index_j,q]*gram[index_i,i]
        EC=EC+lower_initial[index_i,index_j]*v*W[index_j,q]
        if(index_j!=atrain[index_i])
        {
          B4=B4+alpha_initial[index_i,index_j]*W[index_j,q]*gram[index_i,i]
          C4=C4+alpha_initial[index_i,index_j]*W[index_j,q]
        }
      }
    }
  }
  EB=EB-lower_initial[i,j]*v*W[j,q]*gram[i,i]
  EC=EC-lower_initial[i,j]*v*W[j,q]
  B=B1-B2-B3+B4+EB
  C=C1-C2-C3+C4+EC
  A=A+(B+C)*W[j,q]*v
}
A=-A-v*nobs*lambda/(k-1)
new_lower_ij=A/(2*v*v)
omega=reward[i]/score[i]
c=0
if(j==atrain[i])
  c=gamma*omega
else
  c=(1-gamma)*omega
c=abs(c)
upper=c-alpha_initial[i,j]
if(new_lower_ij>upper)
  new_lower_ij=upper
else if(new_lower_ij<0)
  new_lower_ij=0
lower_initial[i,j]=new_lower_ij
return (lower_initial)
}

para_optim_c=function(k,lambda,gamma,kpara,covariate,a.train,alpha_initial,
  lower_initial,reward,v,score,maxiter=1000)
{
nobs=nrow(covariate)
t=0
gram=Gram(covariate,kpara)
#score=sapply(c(1:nobs),propensity_score,covariate=covariate,a.train=a.
  train,model=model)
pre_alpha=alpha_initial
iter_alpha=alpha_initial
pre_lower=lower_initial
iter_lower=lower_initial
while(t<=maxiter)
{
  t=t+1
  for(i in 1:nobs)
    for(j in 1:k)
    {
      new_alpha=alpha_ij_update_c(k,i,j,lambda,gamma,covariate,a.train,
  iter_alpha,iter_lower,reward,v,score,kpara,gram)
      iter_alpha=new_alpha
```

```
          new_lower=lower_ij_update_c(k,i,j,lambda,gamma,covariate,a.train,
    iter_alpha,iter_lower,reward,v,score,kpara,gram)
          iter_lower=new_lower
        }
      difference1=sum(abs(pre_alpha-new_alpha))
      difference3=sum(abs(pre_lower-new_lower))
      sum_difference=difference1+difference3
      if(sum_difference<0.0001*2*nobs*k)
      {
        alpha=new_alpha
        lower=new_lower
        break
      }
      else
      {
        pre_alpha=new_alpha
        alpha=new_alpha
        pre_lower=new_lower
        lower=new_lower
      }
    }
    return(list(alpha,lower))
}

theta_q_0_para=function(k,q,covariate,lambda,alpha,lower,reward,a.train,v)
{
  W=W.gen(k)
  nobs=nrow(covariate)
  A=0
  B=0
  C=0
  D=0
  E=0
    index_pos=which(reward>=0)
    index_neg=which(reward<0)
    for (i in index_pos) {
      for(j in 1:k)
      {
        if(j==a.train[i])
          A=A+alpha[i,j]*W[j,q]
        else
          C=C+alpha[i,j]*W[j,q]
      }
    }
    for(i in index_neg)
      for(j in 1:k)
      {
        if(j==a.train[i])
          B=B+alpha[i,j]*W[j,q]
        else
          D=D+alpha[i,j]*W[j,q]
      }
    for(i in 1:nobs)
      for(j in 1:k)
        E=E+lower[i,j]*v*W[j,q]
    theta_q_0=(A-B-C+D+E)/(lambda*nobs)
    return(theta_q_0)
}

theta_q_para=function(k,q,covariate,lambda,alpha,lower,reward,a.train,v,
    kpara)
{
    W=W.gen(k)
    #gram=Gram(covariate,kpara)
    nobs=nrow(covariate)
    A=0
```

```
B=0
C=0
D=0
E=0
index_pos=which(reward>=0)
index_neg=which(reward<0)
for(i in index_pos)
{
  KKi=rep(0,nobs)
  KKi[i]=1
  for(j in 1:k)
  {
    E=E+lower[i,j]*v*W[j,q]*KKi
    if(j==a.train[i])
      A=A+alpha[i,j]*W[j,q]*KKi
    else
      C=C+alpha[i,j]*W[j,q]*KKi
  }
}
for(i in index_neg)
{
  KKi=rep(0,nobs)
  KKi[i]=1
  for(j in 1:k)
  {
    E=E+lower[i,j]*W[j,q]*v*KKi
    if(j==a.train[i])
      B=B+alpha[i,j]*W[j,q]*KKi
    else
      D=D+alpha[i,j]*W[j,q]*KKi
  }
}
theta_q=(A-B-C+D+E)/(lambda*nobs)
```

```
  return(theta_q)
}

theta_para=function(k,covariate,lambda,alpha,lower,reward,a.train,v,kpara)
{
  covariate=as.matrix(covariate)
  theta=matrix(0,nrow=k-1,ncol=nrow(covariate)+1)
  for(q in 1:(k-1))
    theta[q,]=c(theta_q_0_para(k,q,covariate,lambda,alpha,lower,reward,a.
   train,v),theta_q_para(k,q,covariate,lambda,alpha,lower,reward,a.train,v
   ,kpara))
  return(theta)
}
```

Appendix B

Python Functions for Second Model

```
import pandas as pd
import tensorflow as tf
import numpy as np
import math
from sklearn import linear_model
from sklearn import metrics
from sklearn.ensemble import RandomForestRegressor
import statsmodels.api as sm
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import LabelBinarizer
from functools import reduce
from sklearn.linear_model import LinearRegression
import random
from random import sample

def tf_kron_prod(a, b):
    res = tf.einsum('ij,ik->ijk', a, b)
    res = tf.reshape(res,[res.shape[0],tf.reduce_prod(res.shape[1:])])
    return res

def tf_bin(x, cut_points, temperature=0.1):
    D = cut_points.get_shape().as_list()[0]
    W = tf.reshape(tf.linspace(1.0, D + 1.0, D + 1), [1, -1])
    cut_points = tf.contrib.framework.sort(cut_points)
    b = tf.cumsum(tf.concat([tf.constant(0.0, shape=[1]), -cut_points], 0))
    h = tf.matmul(x, W) + b
    res = tf.nn.softmax(h / temperature)
    return res

def nn_decision_tree(x, cut_points_list, leaf_score, leaf_bias, temperature
    =0.1):
    leaf = reduce(tf_kron_prod,map(lambda z: tf_bin(x[:, z[0]:z[0] + 1], z
    [1], temperature), enumerate(cut_points_list)))
    h=tf.matmul(leaf, leaf_score)+leaf_bias
    return tf.nn.softmax(h)

def tree_construction(k,d,ncut,index,cut_points_list,train_predict,
    train_A_opt,train_covariate):
    cutpoints = np.zeros(shape=(d, ncut))
    for i in range(d):  # record all trained cutting points
        cutpoints[i] = cut_points_list[i].eval()  # the order is consistent
     with index[k:]
    comptree = []  # recording node
    index_comptree = []  # recording observations in each node
    nlayer = 0
    end = 0
    comptree_layer = []
    # reconstruct tree
    while (end == 0):
        if nlayer == 0:
            v = np.mean(train_predict != train_A_opt)
            vbase = 2
            for i in range(d):
                for j in range(ncut):
```

```python
                index1 = train_covariate[:, index[i + k] - k] < cutpoints[i, j]
                index2 = train_covariate[:, index[i + k] - k] >= cutpoints[i, j]
                v1 = np.mean(train_predict[index1] != train_A_opt[index1])
                v2 = np.mean(train_predict[index2] != train_A_opt[index2])
                if v1 + v2 < vbase:
                    index_left = index1
                    index_right = index2
                    vbase = v1 + v2
                    index_i = index[i + k] - k
                    index_j = cutpoints[i, j]
        index_comptree.append([np.arange(ntrain)[index_left], np.arange(ntrain)[index_right]])
        comptree.append([index_i, index_j])
    else:
        index_comptree_layer = []
        comptree_layer = []
        for node in range(pow(2, nlayer)):
            if comptree[nlayer - 1][2 * int(node / 2) + 1] == False:
                comptree_layer.extend([False, False])
                index_comptree_layer.extend([index_comptree[nlayer - 1][node], index_comptree[nlayer - 1][node]])
                continue
            vbase = 2
            sample_index = index_comptree[nlayer - 1][node]
            v = np.mean(train_predict[sample_index] != train_A_opt[sample_index])
            if len(set(train_predict[sample_index])) == 1:
                comptree_layer.extend([False, False])
                index_comptree_layer.extend([sample_index, sample_index
])
                continue
            indicator = []
            t = node
            layer_index = nlayer
            while (t >= 0):
                if t == 0 and layer_index == 0:
                    break
                layer_index = layer_index - 1
            indicator.append(comptree[layer_index][int(t / 2) * 2:(int(t / 2) * 2 + 2)])
                t = int(t / 2)
            if (np.array(indicator).shape[0] >= d * ncut):
                comptree_layer.extend([False, False])
                index_comptree_layer.extend([sample_index, sample_index])
                continue
            sample_covariate = train_covariate[sample_index, :]
            sample_predict = train_predict[sample_index]
            sample_true = train_A_opt[sample_index]
            sample_weight = train_weight[sample_index]
            for i in range(d):
                for j in range(ncut):
                    if [index[i + k] - k, cutpoints[i, j]] in indicator:
                        continue
                    index1 = sample_covariate[:, index[i + k] - k] < cutpoints[i, j]
                    index2 = sample_covariate[:, index[i + k] - k] >= cutpoints[i, j]
                    if np.sum(index1) == 0 or np.sum(index2) == 0:
                        indicator.append([index[i + k] - k, cutpoints[i, j]])
```

```python
                            continue
                        v1 = np.mean(sample_predict[index1] != sample_true[index1])
                        v2 = np.mean(sample_predict[index2] != sample_true[index2])
                        if v1 + v2 < vbase:
                            index_left = sample_index[index1]
                            index_right = sample_index[index2]
                            vbase = v1 + v2
                            index_i = index[i + k] - k
                            index_j = cutpoints[i, j]
                if (np.array(indicator).shape[0] >= d * ncut):
                    comptree_layer.extend([False, False])
                    index_comptree_layer.extend([sample_index, sample_index])
                    continue
                else:
                    index_comptree_layer.extend([index_left, index_right])
                    comptree_layer.extend([index_i, index_j])
            index_comptree.append(index_comptree_layer)
            comptree.append(comptree_layer)
        if np.sum(comptree_layer) == 0 and nlayer != 0:
            end = 1
        else:
            nlayer = nlayer + 1
    return [nlayer, comptree,index_comptree]

def pred_adjust(k,nlayer,comptree,index_comptree,train_predict):
    ntrain=train_predict.size
    train_predict_adjust = np.zeros(ntrain)
    for i in range(pow(2, nlayer)):
        s = np.zeros(k)
        obs_index = index_comptree[nlayer - 1][i]
        for j in range(k):
            s[j] = np.sum(train_predict[obs_index] == (j + 1))
        train_predict_adjust[obs_index] = np.argmax(s) + 1
    return train_predict_adjust

def tree_pruning(nlayer,comtree,index_comptree,train_predict_adjust,eta=0.05):
    layer_index = nlayer - 1
    while (layer_index >= 0):
        for i in range(pow(2, layer_index)):
            if comptree[layer_index + 1][4 * i + 1] != False or comptree[layer_index + 1][4 * i + 3] != False:
                continue
            trt_left = list(set(train_predict_adjust[index_comptree[layer_index][2 * i]]))
            trt_right = list(set(train_predict_adjust[index_comptree[layer_index][2 * i + 1]]))
            if trt_left == trt_right:
                comptree[layer_index][2 * i] = False
                comptree[layer_index][2 * i + 1] = False
                new = list(index_comptree[layer_index][2 * i]) + list(index_comptree[layer_index][2 * i + 1])
                index_comptree[layer_index][2 * i] = np.array(new)
                index_comptree[layer_index][2 * i + 1] = np.array(new)
                continue
            shape1 = index_comptree[layer_index][2 * i].shape[0]
            shape2 = index_comptree[layer_index][2 * i + 1].shape[0]
            left = np.repeat(trt_left[0], shape1 + shape2)
            right = np.repeat(trt_right[0], shape1 + shape2)
            new = list(index_comptree[layer_index][2 * i]) + list(index_comptree[layer_index][2 * i + 1])
            new = np.array(new)
            # miserror1=np.mean(train_predict_adjust[new]!=train_A_opt[new
   ])
            # miserror_left=np.mean(left!=train_A_opt[new])
            # miserror_right=np.mean(right!=train_A_opt[new])
            miserror1 = np.average(train_predict_adjust[new] != train_A_obs
   [new],
                                   weights=np.reshape(train_weight[new],
   newshape=new.shape[0]))
            miserror_left = np.average(left != train_A_obs[new],
                                       weights=np.reshape(train_weight[new
   ], newshape=new.shape[0]))
            miserror_right = np.average(right != train_A_obs[new],
                                        weights=np.reshape(train_weight[new
   ], newshape=new.shape[0]))
            if miserror_left < miserror_right and miserror_left < miserror1
    + eta:
                comptree[layer_index][2 * i] = False
                comptree[layer_index][2 * i + 1] = False
                index_comptree[layer_index][2 * i] = new
                index_comptree[layer_index][2 * i + 1] = new
                train_predict_adjust[new] = left
                continue
            elif miserror_right < miserror_left and miserror_right <
   miserror1 + eta:
                comptree[layer_index][2 * i] = False
                comptree[layer_index][2 * i + 1] = False
                index_comptree[layer_index][2 * i] = new
                index_comptree[layer_index][2 * i + 1] = new
                train_predict_adjust[new] = right
                continue
        layer_index = layer_index - 1
    return [comptree, index_comptree,train_predict_adjust]

def prediction(comptree_pred, index_comptree_pred, train_result, covariate)
```

```
    :
        layer_index_pred = 0
        node_index = 0
        while (comptree_pred[layer_index_pred][2 * node_index + 1] != False
    ):
            feature_index = comptree_pred[layer_index_pred][2 * node_index]
            cut_index = comptree_pred[layer_index_pred][2 * node_index + 1]

            if covariate[feature_index] < cut_index:
                node_index = 2 * node_index
            else:
                node_index = 2 * node_index + 1

            layer_index_pred = layer_index_pred + 1

        test_result = list(set(train_result[index_comptree_pred[
    layer_index_pred - 1][node_index]]))[0]
        return test_result

d=index.shape[0]-k #the number of selected features
ncut=1#the number of cutting points per feature
num_cut=np.repeat(ncut,d)
num_leaf = np.prod(np.array(num_cut) + 1)
num_class=k

# network architecture
sess = tf.InteractiveSession()

x_ph = tf.placeholder(tf.float32, [ntrain, d])
y_ph = tf.placeholder(tf.float32, [ntrain, num_class])
w_ph = tf.placeholder(tf.float32, [ntrain,1])

cut_points_list = [tf.Variable(tf.random_uniform([i])) for i in num_cut]
```

```
leaf_score = tf.Variable(tf.random_uniform([num_leaf, num_class]))
leaf_bias = tf.Variable(tf.random_uniform([1,num_class]))

y_pred = nn_decision_tree(x_ph, cut_points_list, leaf_score, leaf_bias,
    temperature=0.1)
loss=tf.losses.mean_squared_error(y_ph,y_pred,weights=w_ph)
opt=tf.train.AdadeltaOptimizer(0.2)
train_step = opt.minimize(loss)
```

Curriculum Vitae

Name:	Junwei Shen
Post-Secondary Education and Degrees:	The University of Western Ontario London, Ontario, Canada 2018 - 2020 Thesis Based - MSc in Statistics Sichuan University Chengdu, Sichuan, China 2014 - 2018 BSc in Mathematics and Applied Mathematics
Related Work Experience:	Teaching Assistant The University of Western Ontario 2017 - Present