STATISTICAL LEARNING: FINAL PROJECT

Team Members (Workload distribution as %):

Aditya Aiyer (33.$\bar{3}$ %)
Benjamin Minnick (33.$\bar{3}$ %)
Ghanesh Narasimhan (33.$\bar{3}$ %)

Due: December 18, 2019

Workload Distribution Details:
Each task was discussed collectively as a group
The main authors for the code of each task is as follows:
Task 1 Supervised Learning - Ghanesh Narasimhan
Task 2 Dynamic Programming - Benjamin Minnick
Task 2 TD with Q-Learning - Aditya Aiyer

TASK 1 : MACHINE LEARNING

PROBLEM DEFINITION

In this task, Transitional turbulent boundary layer (TBL) data from the JHU Turbulent database (JHTDB) is chosen and three different algorithms learnt in class is applied to the classification problem. In this report, the classification is performed using

1. Ridge Regression
2. Logistic Regression
3. Support Vector Classification (SVC)

Model selection is done using cross validation on each algorithm and the best model is reported based on mean squared error.

DATASET DESCRIPTION AND PROCESSING

TBL data in JHTDB [1] is based on simulations done by [6]. The simulation details and flow configuration can be found in [2]. The data consists of absolute values of 16 inputs,

Input no	Description	Label
1-3	Velocity components	$\|u\|, \|v\|, \|w\|$
4-5	Perturbation velocities	$\|u'\|, \|v'\|$
6-14	Velocity gradient	$\|\partial u/\partial x\|, \|\partial v/\partial x\|, \|\partial w/\partial x\|$ $\|\partial u/\partial y\|, \|\partial v/\partial y\|, \|\partial w/\partial y\|$ $\|\partial u/\partial z\|, \|\partial v/\partial z\|, \|\partial w/\partial z\|$
15-16	Coordinates	$\|x\|, \|y\|$

These inputs are used to classify whether the region of interest is either Turbulent (T) or Non-Turbulent (NT). Each input of the data is computed from a dataset of size $N_x, N_y, N_z = (48, 45, 30)$ at time $t = 0$ and $t = 1000$. The entire dataset at $t = 0$ is used as training data and $t = 1000$ as the test data. Hence, the test-train split of the training data is not required. However, for the cross validation (CV) procedure, the training data would be split into K folds (where K is the number of splits). CV is done using $K-1$ folds as training data and the K^{th} fold would be the test data.

The encoding of data as either T or NT is done using Self Organizing Map (SOM) technique [7], which is an unsupervised machine learning algorithm for classification problems. The number of clusters or classes (here , 2 classes - T or NT) is specified as the input to the SOM and the encoded data is the output. The SOM algorithm works well when the variance of each data has unit variance. Hence, the data is *normalized* by the standard deviation of the inputs.

The *normalized* 16-dimensional inputs with the T-NT encoding is saved in two separate HDF5 files for $t = 0$ and $t = 1000$. *One-Hot encoding* of the data and the mean is subtracted using *standardscaler* before applying the classification algorithms learnt in class.

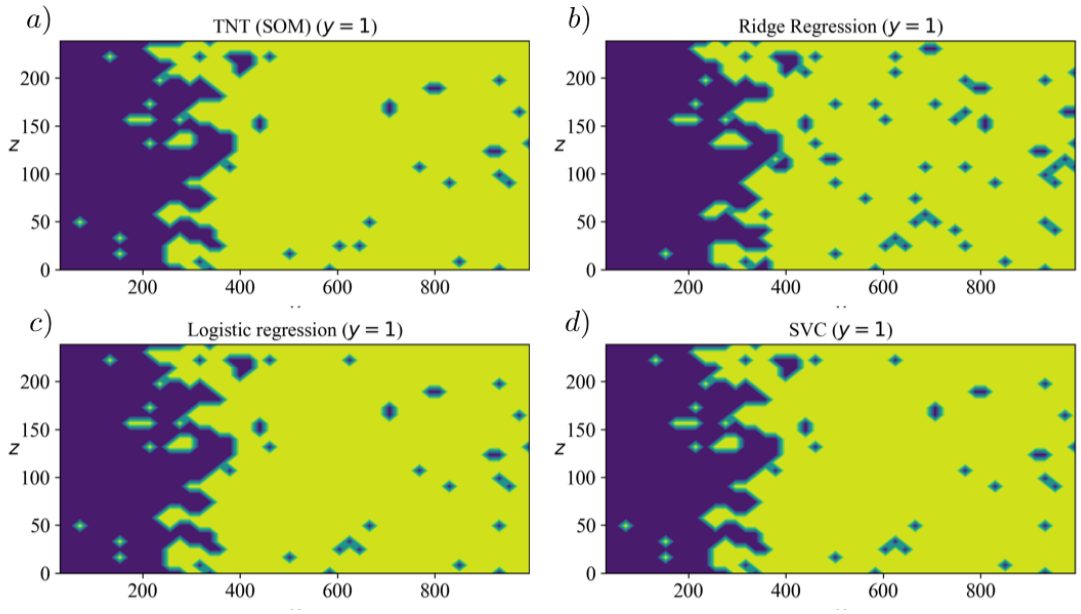

Figure 1: T-NT classification at $y = 1$ using a) Self Organizing Map, b) Ridge Regression, c) Logistic regression, d) Support Vector Machine. Blue - Non-Turbulent, Yellow - Turbulent.

CROSS VALIDATION

Following the data processing step, model selection is done for each classification algorithm using cross validation.

1. Ridge regression is done using the *RidgeClassifierCV* of $sklearn$. It performs Generalized Cross-Validation, which is a form of efficient Leave-One-Out cross-validation procedure.

2. Cross validation for Logistic Regression is done using *GridsearchCV*. CV selects the optimal C value which is the inverse of regularization strength.

3. Cross validation on Support Vector Machine is also done using *GridsearchCV*.

MODEL EVALUATION

Following the training step using $t = 0$ as the training data, the model is tested using the $t = 1000$ data. The test error is reported in the following table

Algorithm	Test error (Mean Squared Error)
Ridge Regression	0.03
Logistic Regression	0.002638
Support Vector Machine	0.002669

Based on the value of test error, both Logistic regression and SVC are equally good for the T-NT classification problem. Ridge regression has a high mean squared error when compared to the other 2 methods. Hence, Logistic regression and SVC are the best models with respect to this classification problem.

The contour plots of the T-NT map of the flow at $y = 1$ are shown in figure 1. The T-NT contours from SVC (Fig 1d) and Logistic regression (Fig 1c) are similar to the actual T-NT regions (Fig 1a). The T-NT contour from Ridge regression (Fig 1b) has notable differences with the actual classification values. These contour plots further validates the conclusion that SVC and Ridge regression are the best models to classify T-NT regions in a transitional turbulent boundary layer.

TASK 2 : CART-POLE PROBLEM

The cart-pole problem consists of cart with mass $M = 1.0$ kg attached to a track with length $L = 4.8$ m holding an inverted pendulum of length $l = 0.5$ m with mass $m = 0.1$ kg. The system is fully described by the position of the cart on the track, x, and the angle of the pendulum, θ, which can be computed from dynamical equations derived by Lagrangian equations of motion. It should be noted, our dynamical equation does not consider friction between the cart and track or between the pendulum and cart, as used in Barto et al. [5].

TASK 2.1 : DYNAMIC PROGRAMMING

In our dynamic programming approach to solve the cart-pole problem, we employ a value iteration approach. The model equations describing the cart-pole system are iterated through time using the forward Euler method with time-step size of $\Delta t = 0.02$ s. By these model equations, the state vector, $s = (\theta, \dot{\theta}, x, \dot{x})^T$, can be determined at any point in time given an initial state.

These continuous states are quantized into boxes to be used to evaluate and improve the policy through value iteration. Where, for each discretized state s, the Bellman equation is used,

$$V(s) \leftarrow \sum_a \pi(a|s) \sum_{s',r} p(s', r|s, a)[r + \gamma V(s')] \tag{1}$$

where the value function, $V(s)$, is re-evaluated using policy, π, reward, r, and discounting factor, γ. In the above expression s' denotes an end state being reached from state s. This is iterated until the value function converges for all states. In our work, convergence is measured using the infinity-norm, and converges within 7 time-steps.

We used the same boxes in the prompt to quantize the states. To evaluate the policy in these boxes, we arbitrarily choose a state within these boxes, often times the midpoint is taken. We reward the system for being in a state where the cart is not off the track and the pendulum is balanced with a value of 1. If the system state does not achieve such constraints, there is penalization by rewarding with a value of 0.

Since the dynamics are deterministic, the probabilities written in the Bellman equation are known. The probability action a is taken given state s using our policy π, i.e. $\pi(a|s)$, is given as one with our greedy policy. The probability the dynamics end up at state s' and receive reward r given the current state s and action a is applied, i.e. $p(s', r|s, a)$ is also known since the model equations give the next known state.

After convergence, we report the value function and corresponding optimal policy for various fixed $\dot{\theta}$ and $\dot{x}$ as functions of θ and x in figure 2. We note that both the value function and policy are independent of x and $\dot{x}$. This is a result of the box allocations used and the states

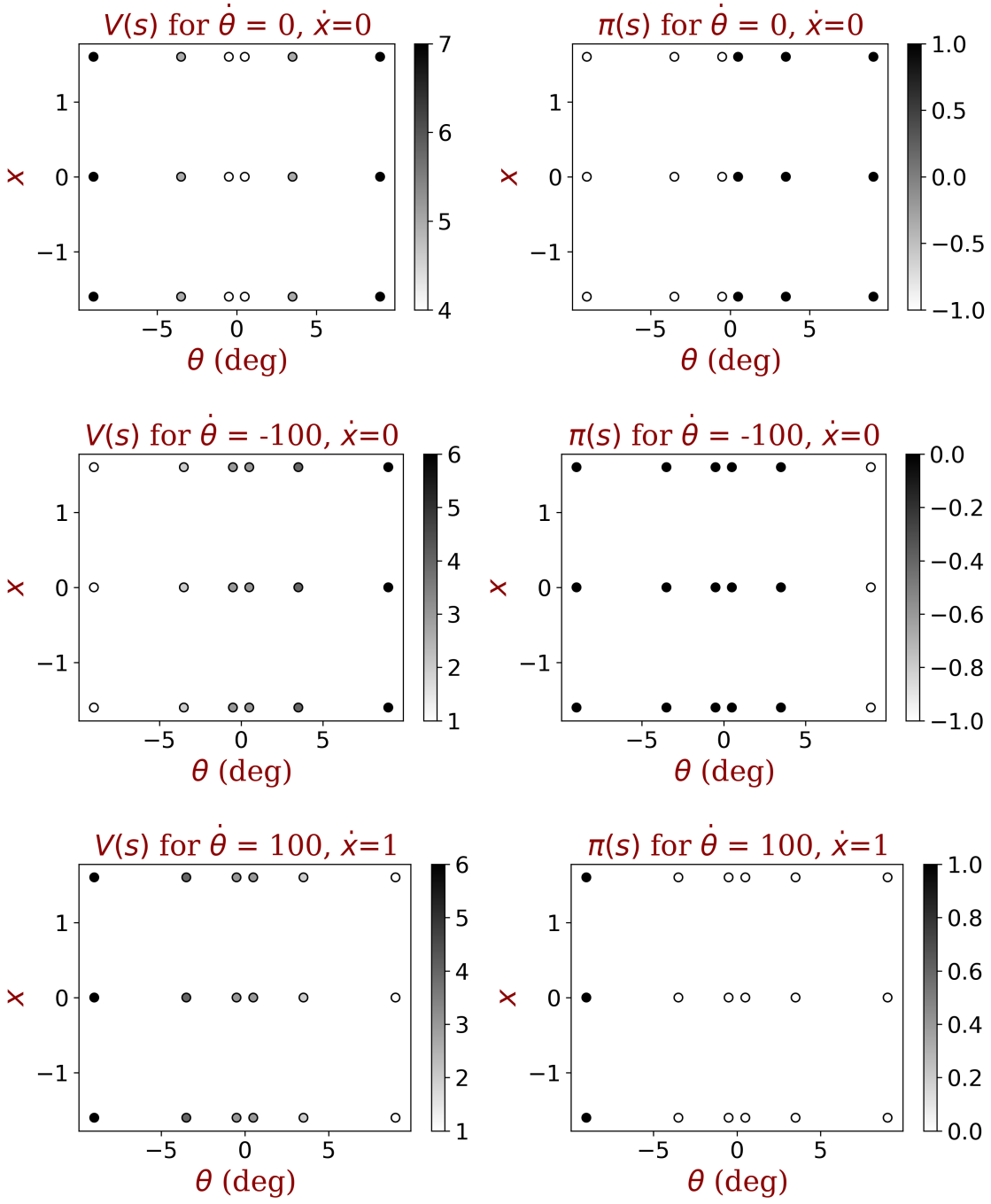

Figure 2: Value function $V(s)$ (left column) and policy $\pi(s)$ (right column) as functions of θ and x with fixed $\dot{\theta} = 0$, $\dot{x} = 0$ (top row), $\dot{\theta} = -100$, $\dot{x} = 0$ (middle row), and $\dot{\theta} = 100$, $\dot{x} = 1$ (bottom row). $\pi = -1$ corresponds to force left, $\pi = 1$ force right, and $\pi = 0$ corresponds to randomly choosing an action since both actions gave the same value.

arbitrarily taken in the boxes describing states near the edges of the track. If states closer to the edges of the track were used when evaluating and improving the policy, then the value function and optimal policy would show dependence on x.

We note when $\dot{\theta} = 0$, the policy forces the cart to the right when the pendulum has $\theta > 0$, and to the left when $\theta < 0$, in an attempt to balance the pendulum. The value function shows states with higher angle magnitudes is preferred for $\dot{\theta} = 0$. When $\dot{\theta} < 0$, a state with $\theta > 0$ has higher value and the policy will attempt to force to the left. For other θ, a random action is taken when $\dot{\theta} < 0$. The converse is true when $\dot{\theta} > 0$. Under this policy, a average cumulative reward of 6511.09 is recorded over 100 episodes, using a max number of iterations of 10,000.

TASK 2.2 : TEMPORAL DIFFERENCE WITH Q-LEARNING

Temporal difference learning is an unsupervised technique in which the learning agent learns to predict the expected value of a variable occurring at the end of a sequence of states. Reinforcement learning (RL) extends this technique by allowing the learned state-values to guide actions which subsequently change the environment state.

Q-learning is an off policy reinforcement learning algorithm that seeks to find the best action to take given the current state. It's considered off-policy because the q-learning function learns from actions that are outside the current policy, like taking random actions, and therefore a policy is not needed. Performing Q-learning involves creating a table for all possible action state pairs with updates performed according to the Bellman equation,

$$Q(s,a) \leftarrow Q(s,a) + \alpha \left(r + \gamma \max_{a' \in A} Q(s',a') \right), \tag{2}$$

where α is the learning rate. Selecting an action based on equation (2) is known as exploiting since we use the information we have available to us to make a decision. An important feature of Q-learning algorithms is also to explore, i.e take an action randomly. Acting randomly allows the agent to explore new states that may not be sampled during the exploitation process. A balance between exploration/exploitation is obtained using an exploration factor ϵ. The way this is implemented algorithmically is,

Algorithm 1: Exploration-Exploitation Q-learning

```
if random() < explore rate then
    Select random action (Exploration)
else
    Select best action ( Exploitation )
```

The parameters that now need to be determined in our problem are the learning rate (α) and the exploration rate (ϵ). We use a a decaying exploration and learning rate following the work of [3]. The environment (physics) of the problem is defined using OpenAI gym [4]. In Q-Learning, we discretize the continuous dimensions to a number of bins to discretize the state space. In order to reduce the dimensionality of the search space we modify the bounds of $\dot{x}$ and $\dot{\theta}$. Through our simulations we found that the number of bins needed for $\dot{x}$ could be small. We therefore discretize the system using $3 \times 1 \times 6 \times 3$ bins for $(x, \dot{x}, \theta, \dot{\theta})$.

We show the evolution of the position of the cart (x) and the angle of the pole (θ) in figure 3 for some successful episodes. We can see that the the model tries to keep theta (left panel) in

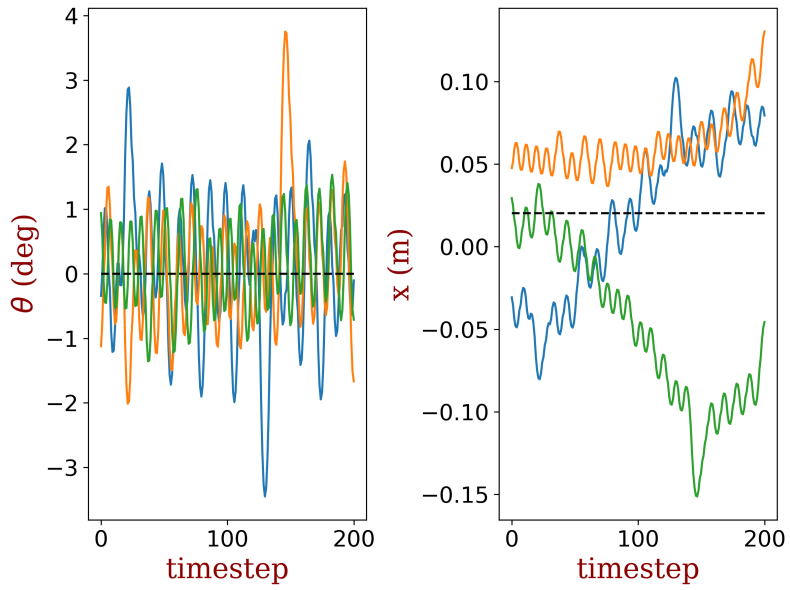

Figure 3: The left panel depicts the angle the pole makes with the vertical as a function of timestep. The right pane shows the position of the cart. We see that the pole is balanced for 200 timesteps for all the episodes depicted.

a tight range, where the average angle is zero (denoted by the black dashed line). The x values are also small, which means that it is favourable for the cart to be near the origin for a success. We plot the reward and the cumulative reward per timestep in figure 4. We select 10 episodes at random and average the rewards. A reward per timestep, denotes the probability of success at that step. From the figure we can see that for all the discount factors the initial timesteps are a success. There is no clear trend as a function of the discount factor. The cumulative rewards would be a straight line if we had all successes. We see it sloping downwards as some episodes sampled had some failures. All three simulations converged in under 270 epsiodes with 200 timesteps used per episode. The convergence criterion used was that the model should give 100 successes in a row, where a success is a cumulative reward of 195 or greater. We note that if we increase the number of bins a larger number of episodes is required for convergence.

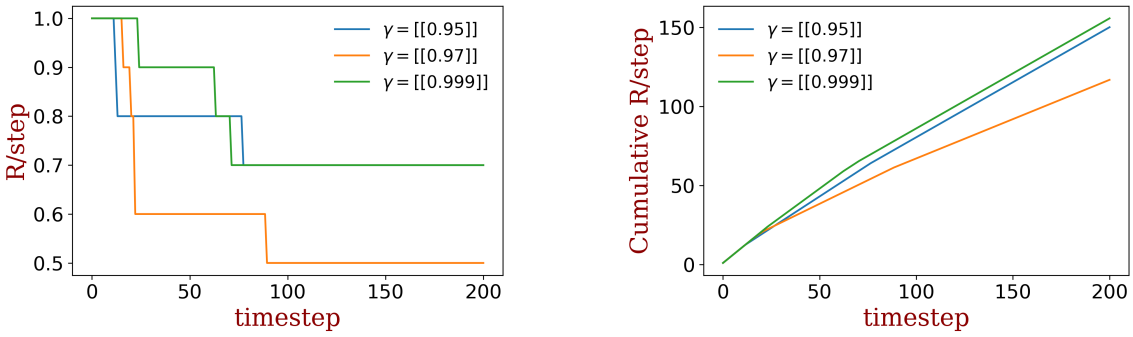

Figure 4: Left panel shows the reward per timestep averaged over 10 episodes. The right panel plots the cumulative reward per step as a function of timestep averaged over 10 episodes.

REFERENCES

[1] http://turbulence.pha.jhu.edu.

[2] https://doi.org/10.7281/T17S7KX8.

[3] https://medium.com/@tuzzer/cart-pole-balancing-with-q-learning-b54c6068d947.

[4] https://gym.openai.com/envs/CartPole-v0/.

[5] A. G. Barto, R. S. Sutton, and C. W. Anderson. Neuronlike adaptive elements that can solve difficult learning control problems. *IEEE Transactions on Systems, Man, and Cybernetics*, SMC-13:834–846, 1983.

[6] J. Lee and T. Zaki. Detection algorithm for turbulent interfaces and large-scale structures in intermittent flows. *Comput. Fluids*, 175:142, 2018.

[7] Z. Wu, C. Meneveau, and T. Zaki. Application of a self-organizing map to identify the turbulent-boundary-layer interface in a transitional flow. *Phys. Rev. Fluids*, 4:023902, 2019.