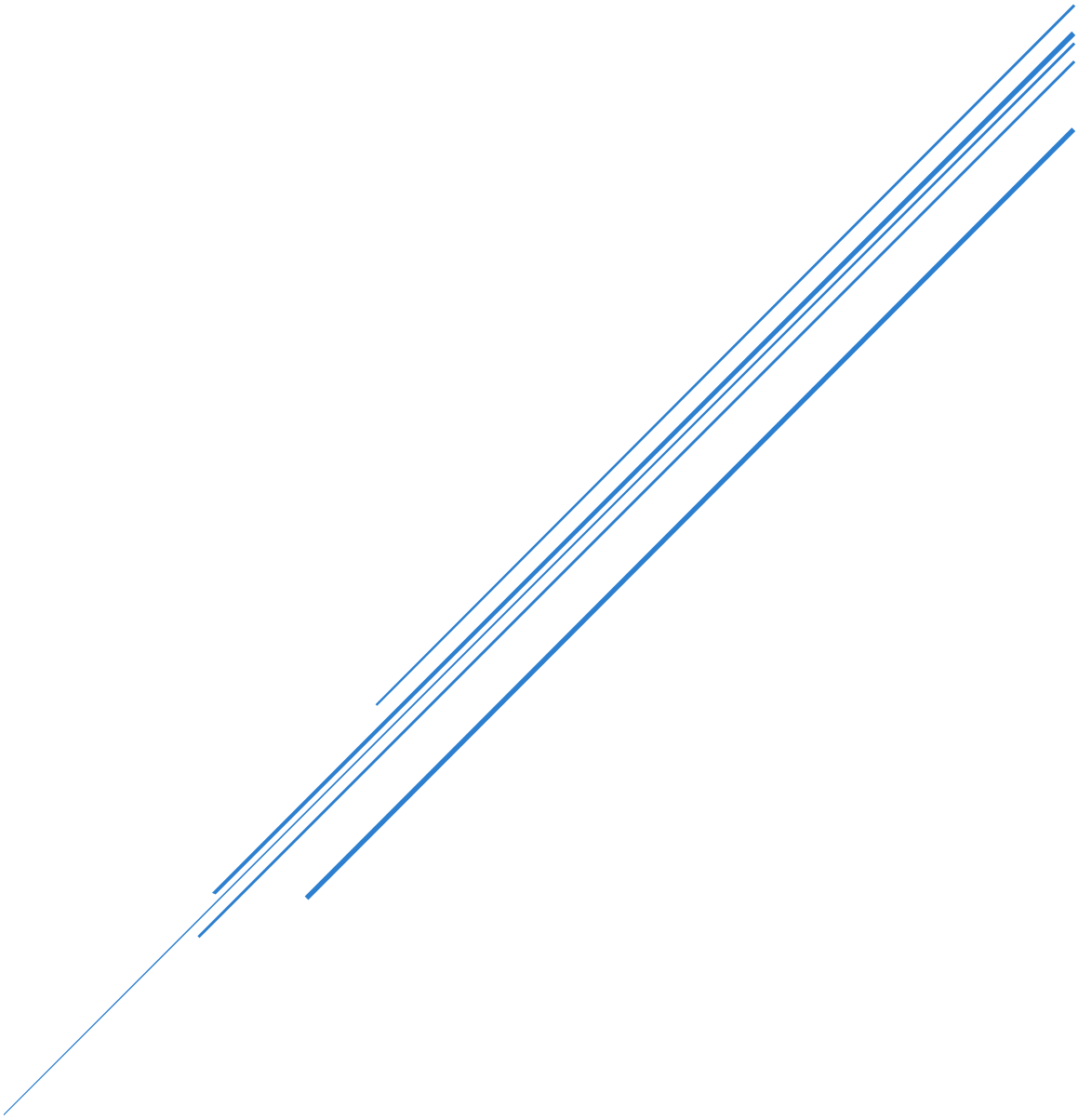
ML FINAL ASSIGNMENT REPORT

Final Assignment

Fahim Wayez
21-44499-1
Section: B



Q.1) Explain how locally weighted regression differs from linear regression, including their formulas. What is the advantage of locally weighted regression over linear regression?

Linear Regression: The formula for the linear regression is, $Y = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + \dots + \beta_n x_n$

where β represents coefficients and x represents independent variables. The model for this is basically, here fits a single, global straight line to capture the relationship between independent variables x and the dependent variable y across the entire dataset. Linear regression assumes if a linear relationship exists between the variables. The strength of this is basically these are simple to understand and implement, computationally efficient.

Locally Weighted Regression: This is very similar to linear regression, but with weights "w_i" is assigned to each data point based on its proximity distance to the new data to the new data point x^* for which we are making a prediction. The formula is,

$$Y^* = \Sigma(w\_i * (\beta + \beta_1 x\_i_1 + \beta_2 x\_i_2 + \dots + \beta_n x\_i_n) / \Sigma(w\_i)$$

where y^* is the predicted value for the new data point (x^*), w_i is the weight assigned to the i-th data point based on its distance to x^* and β is basically the coefficients estimated using the weighted data points. Locally weighted regression fits a local linear model around each new data point using weighted

data points from the neighborhood. Data points closer to the new points have higher weights, influencing the prediction more. This allows this regression method to capture non-linear relationships better, although it does not require a strict linear relationship across the entire dataset. But this model can handle non-linear relationships, potentially leading to more accurate predictions in such scenarios.

Advantages of Locally Weighted Regression:

The key advantages of this model over linear regression is its ability to capture non-linear relationships between variables. While linear regression assumes a straight-line fit, LWR can adapt to local trends in the data, leading to potentially more accurate predictions for non-linear patterns. But it has some limitations as well. Like it is computationally more expensive, it requires storing the entire training data for predictions, and the effectiveness of LWR depends on the chosen weight function that determines how weights are assigned based on distance.

Q. 2) Given you want to apply a model to predict whether a patient has malignant or benign tumor, where model output $y = 1$ means malignant and $y = 0$ means benign. Explain how the binary logistic regression model is used to train patient data and then predict tumor of a new patient. Include formulas and learning algorithm used in your answer.

Binary logistic regression is a powerful tool for modeling the probability of an event (malignant tumor in this case) based on a set of independent variables (patient data). In this scenario, firstly to train the model, we need to have a dataset containing patient information like age, family history, test results, etc. and a binary label indicating tumor type ($y = 1$ for malignant, $y = 0$ for benign).

To core of logistic regression is the linear equation, but unlike linear regression, we use a S-shaped sigmoid function to transform the linear output into a probability between 0 and 1, which is denoted by $\sigma(z)$.

Equation: The equation for the logistic regression model looks like this:

$$\log(\text{odds}(y=1)) = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + \cdots + \beta_n x_n$$

where β_0 is the intercept term, β_i are the coefficients for each independent variable (patient data point) x_i and $\text{odds}(y=1)$ represents the odds of a patient having a malignant tumor.

Learning Algorithm: The goal is to find the values of β that best fit the training data. This is achieved using a maximum likelihood estimation algorithm. The algorithm iteratively adjust β values to minimize the difference between predicted probabilities and actual labels (0 or 1) in the training data.

Like for this particular scenario, to predict tumor for a new patient, we need to have a new patient with their corresponding data points x_1 to x_n. We could then use the trained model equation with the new patient's data points and the estimated coefficient β to calculate the log-odds of having a malignant tumor. Afterwards we could apply the sigmoid functions σ to the log-odds value to get the predicted probability between 0 and 1, that the new patient has a malignant tumor.

Now, there could be two interpretations:

- A probability closer to 1 suggests a higher likelihood of malignancy.
- A probability closer to 0 suggests a higher likelihood of a benign tumor.

After that the challenge is to choose a threshold. It is a crucial step to set a threshold probability to classify the tumor. For example, we might choose a threshold of 0.7. If the predicted probability for a new patient is greater than 0.7, we classify the tumor as malignant. the and vice versa for values below 0.7.

Q.3) a) Given the output, $y_{(n)}$, of 3 training items of softmax regression are represented by the following one-hot vectors where $y \in \{1, 2, 3\}$: $y_1 = [1\ 0\ 0]$, $y_2 = [0\ 1\ 0]$ and $y_3 = [0\ 0\ 1]$. Write the expanded form of the softmax cost function $J(w)$ for these 3 items, and the softmax output function $f(x; w)$.

b) What is the relationship between softmax and binary logistic regression?

Ans. (a): **Softmax output function $f(x; w)$:** The softmax function, denoted by $f(z\_i; w)$, takes a vector of weighted sums (z_i) as input and transforms it into a probability distribution across k classes ($y \in \{1, 2, 3\}$ in this case). The formula is,

$$f(z\_i; w) = \exp(z\_i) / \Sigma(\exp(z\_j)) \text{ for all } j \text{ classes } (1 \text{ to } k)$$

where, z_i is the weighted sum for class i (result of linear transformation with weights w), $\exp(z\_i)$ applies the exponential function to z_i, and $\Sigma(\exp(z\_j))$ is the summation of exponentials of all weighted sums (z_j) across all k classes.

Expanded softmax cost function $J(w)$: The softmax cost function $J(w)$ measures the difference between the predicted probabilities $f(z\_i; w)$ from the model and the actual one-hot encoded labels (y_i) for all training items. Here's the expanded form of 3 items:

$$\begin{aligned} J(w) = -(&y_{10} * \log(f(z_1; w)_0) + y_{11} * \log(f(z_1; w)_1) + y_{12} * \log(f(z_1; w)_2) + \\ &y_{20} * \log(f(z_2; w)_0) + y_{21} * \log(f(z_2; w)_1) + y_{22} * \log(f(z_2; w)_2) + \\ &y_{30} * \log(f(z_3; w)_0) + y_{31} * \log(f(z_3; w)_1) + y_{32} * \log(f(z_3; w)_2)) \end{aligned}$$

where, y_i is the one-hot encoded ~~table~~ label vector for training item i.

- $y_1 = [1\ 0\ 0]$ (all elements are 0 except the one for class 1)
- $y_2 = [0\ 1\ 0]$ (all elements are 0 except the one for class 2)
- $y_3 = [0\ 0\ 1]$ (all elements are 0 except the one for class 3)

and $f(z\_i, W)$ is the softmax output vector for training item i with probabilities for each class.

Ans. b) Softmax is generalization of logistic regression for multi-class classification problems ($K > 2$). Logistic Regression is used for binary classification ($y \in \{0, 1\}$). It uses the sigmoid function to transform the weighted sum into a probability between 0 and 1. And, softmax regression is used for multi-class classification ($y \in \{1, 2, \cdots, K\}$). It applies the softmax function to a vector of weighted sums, ensuring the output probabilities sum to 1 and represents the class distribution. In essence, logistic regression is a special case of softmax regression when there are only two classes.

Q.4) What is the penalty term of ridge/L2 regularization and how does it reduce overfitting?

In Ridge Regression (L2 regularization), the penalty term is the sum of the squared coefficients, β of the model. It is added to the original cost function, usually mean squared error, to discourage the model from having very large coefficients. Here's a breakdown of how it reduces overfitting:

The penalty term in L2 regularization is mathematically represented as, $\lambda * \Sigma(\beta\_i^2)$ where, λ is a hyper-parameter that controls the strength of the regularization. A higher λ value leads to a stronger penalty on large coefficients. β_i represents each coefficients in the model, and Σ denotes the summation across all coefficients.

Impact on overfitting: Overfitting occurs when a model becomes too ~~complicated~~ complex and memorizes the training data noise instead of learning the underlying patterns. Large coefficient values can contribute to overfitting as they can lead to the model giving excessive weight to specific features.

L2 regularization addresses this by penalizing large coefficients. It helps by reducing the magnitude of coefficients, meaning that, by adding the squared coefficients to the cost function, the model is discouraged from having very large β values. This pushes the coefficients towards smaller values, making the model less sensitive to specific features and promoting a more generalizable solution. Also, it prevents overreliance on single features.

meaning when coefficients are smaller, no single feature dominates the model's predictions. This encourages the model to consider all features more evenly, leading to a more robust fit that performs better on unseen data. Finally, it promotes smoother decision ~~de~~ boundaries.

In simpler terms, we could imagine a model separating data points belonging to different classes with a decision boundary. L2 regularization by shrinking coefficients can lead to smoother decision boundaries, less prone to capturing irrelevant data noise.

Overall, the penalty term in L2 regularization acts as a control mechanism to prevent the model from becoming overly complex and fitting the noise in the data. This helps to reduce overfitting and improve the model's generalization ability to unseen data.

Q.5) a) Write the pseudocode/steps of applying Policy iteration to solve an MDP, including the equation.

b) What is the advantage of using an exploration-based policy like ε-greedy, to solve an MDP?

Ans. a) Policy iteration is an algorithm used to find an optimal policy for a Markov Decision Process (MDP). Here's the pseudocode along with the equations and the advantage of using an exploration-based policy like ε-greedy.

Pseudocode:

1. Initialize an arbitary policy $\pi(s,a)$ for all states(s) and actions(a)
2. Loop:
   2.1. ** Policy Evaluation: **
      - For each state(s):
         - Initialize $V_{\pi}(s)$ (value of state s under policy π) to some value (e.g; 0)
         - Loop until covergence:
            - For each state(s):
               - $V_{\pi}(s) \leftarrow \Sigma_a \pi(s,a) * [R(s,a) + \gamma * \Sigma_{s'} P(s'|s,a) * V_{\pi}(s')]$

   - (γ: discount factor, $R(s,a)$: reward for taking action a in state s; $P(s'|s,a)$: probability of transitioning to state s' after taking action a in state s)

   2.2. ** Policy Improvement: **
      - For each state(s):
         - Choose the action a that maximizes the Q-value (expected future reward):
            - $Q_{\pi}(s,a) \leftarrow \Sigma_{s'} P(s'|s,a) * [R(s,a) + \gamma * V_{\pi}(s')]$
         - Update policy $\pi(s,a)$ to favor the action with the highest Q-value in state s.

   2.3. Check for termination:
      - If the policy π hasn't changed since the last iteration, terminate the loop.

3. Return the optimal policy π^*.

Equations: Policy evaluation is the step which estimates the value ($V_{\pi}(s)$) of each state under the current policy (π). It uses an iterative approach, considering the reward received for taking an action ($R(s,a)$) and the discounted expected value of the next state ($\gamma * \Sigma_{s'} P(s'|s,a) * V_{\pi}(s')$). And policy improvement is the algorithm that improves the policy by finding the action with the highest Q-value (expected future reward) in each state. The Q-value considers the immdiate reward and the discounted

value of the next state under the current policy.

Ans. b) While policy Iteration can converge to an optimal policy with a deterministic policy, meaning always choosing the best action, it might not explore the entire state space effectively during initial iterations. This could lead to suboptimal solutions if the optimal policy requires visiting less-likely states initially.

ε-greedy is an exploration-based policy that addresses this issue. It works with the probability $(1-\varepsilon)$, the agent chooses the action with the highest Q-value(exploration) With a small probability (ε), it explores by choosing a random action or exploration, which basically helps the agent discover potentially valuable states that might not be visited under a purely deterministic policy. By balancing exploration and exploitation, ε-greedy allows the agent to learn about the entire state space and converge to a better solution.

In summary, Policy iteration provides a structured approach to find an optimal policy for MDPs. While it can converge without exploration, using an exploration-based policy like ε-greedy ensures the agent explores the entire state space effectively, leading to a potentially better solution.

Q.6) a) What makes Q-learning an off-policy algorithm?

b) What is the difference between on-policy and off-policy algorithm?

Ans. a) In Q-learning, the learning process involves two key aspects. The first one is Action selection policy (π). This policy determines which action to take in a given state during training. It can be an exploration-based policy like ε-greedy (exploit it most of the time, explore a small percentage of the time) or another policy altogether. And the second one is Q-value function update, which uses the Bellman equation to estimate the Q-value (expected future reward) of taking a specific action (a) in a given state(s). Here's the simplified form:

$$Q(s,a) \leftarrow Q(s,a) + \alpha[R(s,a) + \gamma * \max_{a'} Q(s',a')]$$

where α is the learning rate, $R(s,a)$ is the reward for taking action a in state s, γ is the discount factor, and $\max_{a'} Q(s',a')$ is the maximum Q-value for next state(s') considering all possible actions(a').

Crucially, the Q-value update only uses the maximum Q-value for the next state (s'), regardless of the policy that might lead to that state. This allows Q-learning to learn the value of different actions across various scenarios, even if the data used for learning wasn't generated by the current policy being evaluated (π).

Ans. b) The key difference between on-policy and off-policy algorithms lies in how they utilize data for learning. on-policy algorithms learns from data generated by the current policy being evaluated which is basically the policy used for making decisions during training. These algorithms aim to improve the current policy itself. Examples could be SARSA or State-Action-Reward-State-Action. On the other hand, off policy algorithms learns from data generated by any policy, not necessarily the current policy being evaluated. This data could come from past experiences with different policies, random exploration, or even a combination. These algorithms focus on learning a general value function (Q-value function in Q-learning) that can be used to evaluate and potentially improve various policies. Examples could be Q-learning, Deep Q-Networks (DQN).