Pratap Dey
Email: ipratapdey@gmail.com
Mobile - (+91) 8159994838
https://www.linkedin.com/in/pratap-dey/

Contlo
Assignment
Date: 16th December, 2023

Q/A Assignment:

Solution 1
In the given scenario where a certain feature, let's call it 'n', is copied to create a new feature denoted as '$n+1$'. If we then retrain a logistic regression model, the weights assigned to the new features, w_{new_n} and $w_{\text{new}_{n+1}}$, are likely to be adjusted so that their sum closely matches the original weight assigned to 'n', which we'll call w_n.
This adjustment occurs because the logistic regression model distributes the importance originally attributed to feature 'n' between both the original and duplicated features, ensuring a balanced allocation of significance.

Solution 2
True statement will be :(option b) "E is better than A with over 95% confidence, B is worse than A with over 95% confidence. You need to run the test for longer to tell where C and D compare to A with 95% confidence."

We have a 95% confidence level indicating that template E surpasses template A, while template B falls short in comparison to template A. However, a more extended test is required to establish a 95% confidence comparison between templates C and D with template A.

Upon examination, the z-scores for templates B, C, and D are all below 1.96, indicating that we cannot assert with 95% confidence that they differ significantly from template A. Conversely, the z-score for template E exceeds 1.96, providing us with 95% confidence that it outperforms template A.

Solution 3
In logistic regression with sparse feature vectors, the computational cost for a single gradient descent iteration can be outlined as follows, given m training examples, n features, and k average non-zero entries in each sparse vector (where $k \ll n$):

1. **Forward Propagation:** - Cost per example: $O(k)$ - Total cost for all examples: $O(m \cdot k)$
2. **Compute Gradient:** - Cost per example: $O(k)$ - Total cost for all examples: $O(m \cdot k)$
3. **Update Weights/ Backward Propagation:** - Cost: $O(k)$

Total computational cost for one iteration:

$$O(m \cdot k) \text{ (prediction)} + O(m \cdot k) \text{ (gradient)} + O(k) \text{ (weight update)}$$

Approximating the dominant term:

$$O(m \cdot k)$$

Solution 4
These approaches carry implications for the nature of the training data for $V2$:

a. Opting for stories near the decision boundary can serve as a strategy to enhance the classifier's performance in instances where uncertainty is high. This approach may yield a model more adept at handling ambiguous cases, thereby improving overall accuracy.

b. Randomly selecting labeled stories offers a broader and potentially more diverse training dataset.

While this approach does not specifically target the model's weaknesses, it can contribute to an overall performance boost by encompassing a wide array of examples.

c. Selecting stories where $V1$'s predictions are both incorrect and farthest from the decision boundary aims to rectify $V1$'s most confidently made errors. This targeted approach has the potential to significantly enhance $V2$'s accuracy by focusing on correcting cases where $V1$ went wrong.

In terms of enhancing the pure accuracy of classifier $V2$, one might anticipate the following ranking:

1. Method c: By concentrating on stories where $V1$ made the most confident errors, $V2$ can learn from these mistakes, leading to potentially substantial accuracy improvements, assuming these errors are indicative of common mistakes that require correction.

2. Method a: Training on stories close to the decision boundary aids $V2$ in handling ambiguous cases, which are often challenging for classifiers. Improvements in this area can be highly beneficial; however, this method may not directly address confidently wrong predictions.

3. Method b: While this method can yield a general improvement, it does not specifically target $V1$'s weaknesses and may include many 'easy' examples that $V1$ already handles effectively.

Solution 5

(a) Maximum Likelihood Estimate (MLE): The MLE for p is obtained by maximizing the likelihood function. The likelihood function is the probability of observing the given sequence of heads and tails, given the parameter p.

$$\text{Likelihood}(\theta) = \binom{n}{k} p^k (1-p)^{n-k}$$

Taking the logarithm (log-likelihood) and maximizing with respect to p, we get:

$$\frac{d}{dp}(\log(\text{Likelihood})) = \frac{k}{p} - \frac{n-k}{1-p}$$

Setting the derivative to zero:

$$\frac{k}{\hat{p}_{\text{MLE}}} - \frac{n-k}{1-\hat{p}_{\text{MLE}}} = 0$$

Solving for $\hat{p}_{\text{MLE}}$, we find:

$$\hat{p}_{\text{MLE}} = \frac{k}{n}$$

(b) Bayesian Estimate:

Assume a uniform prior distribution $P(p) = 1$ for $0 \leq p \leq 1$, and $P(p) = 0$ otherwise.

For the binomial distribution the likelihood will be: $P(k|p) = \binom{n}{k} p^k (1-p)^{n-k}$, where k is the number of heads observed in n tosses.

Then applying Bayes' theorem to obtain the posterior distribution:

$$P(p|k,n) \propto P(k|p)P(p)$$

This results in a beta distribution: $P(p|k,n) \propto p^k (1-p)^{n-k}$.

So, The Bayesian estimate is the mean of the posterior distribution:

$$\text{Bayesian Estimate} = \frac{k+1}{n+2}$$

(c) Maximum a Posteriori (MAP) Estimate:

The MAP estimate is the mode of the posterior distribution:

$$\text{MAP Estimate} = \frac{k}{n}$$

Coding Assignment: Implementation and Optimization of GPT-2 Model:

The implementation and optimization of the GPT-2 model can be found in the following GitHub repository:
`https://github.com/pratapdey/GPT-2-small/blob/main/Implementation%20and%20Optimization%20of%20GPT-2%20Model.ipynb`

Alternatively, the full assignment is in the following repository:
`https://github.com/pratapdey/GPT-2-small`
