CSCI - 635: Intro To Machine Learning
Homework 2: Multiclass Classification

Anurag Kallurwar

November 13, 2023

---

1. Problem #1: Learning a Maximum Entropy Model

1.1 Problem #1a: Tackling the XOR Problem

1.1.1 Describe what hyper-parameters you selected, your process for choosing them, and observations made related to your model's ability to fit the data and its loss per epoch.

```
# Meta-Parameters
beta = 0.001 # regularization coefficient
alpha = 0.01 # step size coefficient
n_epoch = 1000 # number of epochs (full passes through the dataset)
eps = 0.00001 # controls convergence criterion
```

Figure 1: Hyper Parameters

In the training of the model, I learned the importance of gradient descent in model fitting. Minor adjustments in the parameters such as step size (α) and number of epochs (n_epoch) significantly impact the model's loss function. Through trial and error with different parameter values, we have the right balance between underfitting and overfitting which is crucial to obtain an accurate and generalized model. This process of fine-tuning the parameters and optimization is an important aspect of building a robust and generalized model. So, through trial and error, I aimed to minimize errors and improve accuracy. I maintained a good balance by keeping a significant value of regularization coefficient (β) to reduce the generalization error. The eps (ϵ) serves the dual purpose of halting the training upon convergence as well as checking gradient convergence by comparison with numerical calculations.

```
# Initializing parameters in theta
np.random.seed(13) # Seeding randomness of weight values
w = np.random.rand(X.shape[1], K)
# w = np.zeros((X.shape[1], K))
b = np.array([0])
theta = (b, w)
```

Figure 2: Randomly initialized theta Parameters

The θ (consisting of b and w) parameters were initialized with random values using a seed of 13. This was done because initializing them with zeros was resulting in zero derivatives, hindering the parameter updates during gradient descent.

1.1.2 Accuracy of the model

```
================================================================
Misclassification Errors
Error = 25.0%
Accuracy = 75.0%

================================================================
```

Figure 3: Accuracy of the model

This is the maximum accuracy achieved by the model. Given that a two-class multinoulli (multiclass) model essentially functions as a logistic regression model, it inherently exhibits a linear decision boundary. Because of that, it can accurately classify three inputs, but the fourth one will be misclassified, as shown in the plot below (Figure 4).

1.1.3 Plot

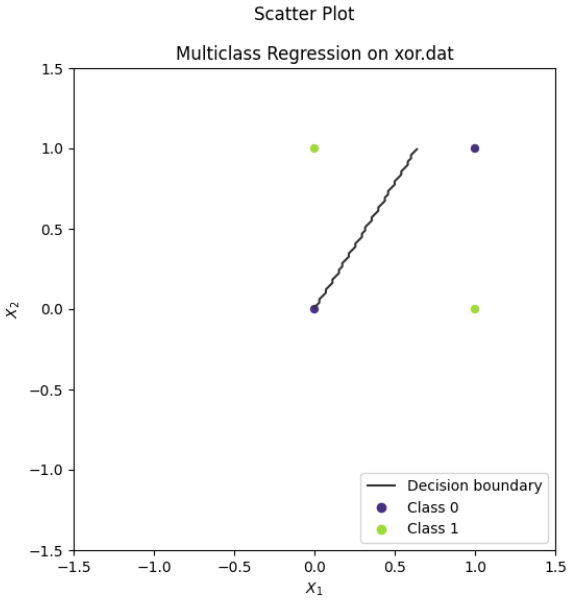

Figure 4: Multiclass classification model for XOR dataset

1.1.4 Gradient Convergence Check

```
================================================================
Gradient Convergence Check
(array([-3.05311332e-18]), array([[0.63358996, 0.37770191],
       [0.8100903 , 0.97296988]]))
(array(['CORRECT'], dtype='<U7'), array([['CORRECT', 'CORRECT'],
       ['CORRECT', 'CORRECT']], dtype='<U7'))

================================================================
```

Figure 5: Gradient Convergence Check

In the image above (see Figure 5), for each parameter in the θ parameters (as depicted in the first tuple), the program returned ”CORRECT” (as shown in the second tuple). This uses the eps (ϵ) (see Figure 1) for the numerical calculation, specifically the secant approximation and then the check is performed for every parameter.

1.2 Problem #1b: Softmax Regression & the Spiral Problem

```
# Meta-Parameters
beta = 0.001 # regularization coefficient
alpha = 0.01 # step size coefficient
n_epoch = 10000 # number of epochs (full passes through the dataset)
eps = 0.00001 # controls convergence criterion
```

Figure 6: Hyper Parameters

I used the same parameters as used for XOR problem, because the same parameters were able to achieve the best accuracy for the model.

1.2.1 Accuracy of the model

```
=====================================================================
Misclassification Errors
Error = 51.333%
Accuracy = 48.667%

=====================================================================
```

Figure 7: Accuracy of the model

The accuracy achieved by the model is influenced by its nature as a three-class multinoulli (multiclass) model. In attempting to classify the data, the model imposes multiple linear boundaries based on the likelihood of each class. However, due to the inherent non-linear complexity of the spiral-shaped data, the model struggles to accurately classify it, as shown in the plot below. (see Figure 8).

1.2.2 Plot

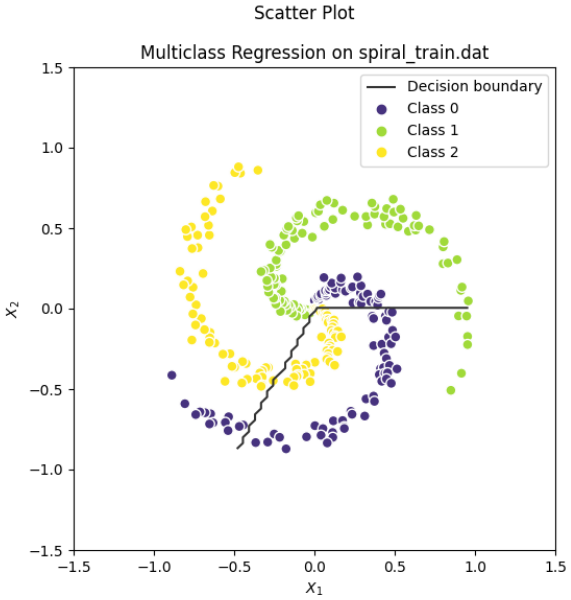

Figure 8: Multiclass classification model for spiral dataset

1.3 Problem #1c: The IRIS Maximum Entropy Model

```
# Meta-Parameters
beta = 0.001 # regularization coefficient
alpha = 0.01 # step size coefficient
n_epoch = 10000 # number of epochs (full passes through the dataset)
eps = 0.00001 # controls convergence criterion
m = 5 # Size of mini-batch
```

Figure 9: Hyper Parameters

Above are the parameters used for the model, here m represents the mini-batch size for the batch processing of training data, which is set to $\mathbf{5}$ for my model.

1.3.1 What ultimately happens as you train your model for more epochs? What phenomenon are you observing and why does this happen? What is a way to control for this?

If we train our model for more epochs, the training loss will naturally decrease, and accuracy will rise, eventually reaching gradient convergence and reach a stale state.

However, this prolonged training may trigger the occurrence of overfitting. As, the model will begin capturing irrelevant noise of the training data rather than generalizing, essentially in an attempt to memorize more and more data and increasing the variance error. This can be controlled by following:

(1) Early Halting/Stopping A halting mechanism can be implemented during training, interrupting the process when there is a very little (less than eps) change in the training loss, indicating a potential convergence or stabilization of the model. This convergence is significant in halting the training early and avoiding useless computations, sometime avoid overfitting.

(2) Regularization This will penalize the the model by introducing a penalty in the cost function. This results in an increase in the bias error which balances the generalization error, hence leading to a generalized model. We control the regularization with a parameter such as β in our case, so that the regularization term is never the dominant term in the cost function.

1.3.2 Accuracy of the model

```
==================================================================
Training Data
Error = 3.636%
Accuracy = 96.364%

==================================================================
Test Data
Error = 5.0%
Accuracy = 95.0%

==================================================================
```

Figure 10: Accuracy of the model

1.3.3 Training and Validation Losses

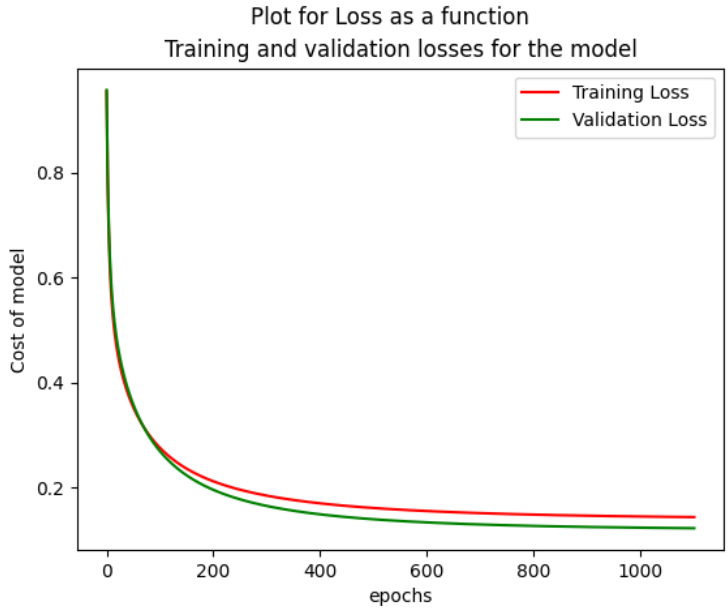

Figure 11: Training and Validation Losses for model

As you can see in the above plot that the validation loss is lower than the training loss as the number of epochs increase, it can be due to the issue of overfitting. This can be attributed to the model becoming complex, fitting the training data too closely and capturing irrelevant noise, which affects its generalization to unseen data.

2. Problem #2: Learning a Naïve Bayes Classifier

In your writeup, report the parameters for your model as computed in part a, the classification error you achieved in part b, and (most importantly) explain the process that you undertook in part c) - both how you modified your code to handle this problem, as well as how you chose features to use and/or remove and what classification error your simpler classifier obtained.

2.1 Maximum Likelihood Parameters

$$
\begin{aligned}
P('is\ spam' = True) &= 0.172 \\
P('is\ spam' = False) &= 0.828 \\
P('in\ html' = True \mid 'is\ spam' = True) &= 0.756 \\
P('in\ html' = True \mid 'is\ spam' = False) &= 0.587 \\
P('in\ html' = False \mid 'is\ spam' = True) &= 0.244 \\
P('in\ html' = False \mid 'is\ spam' = False) &= 0.413 \\
P('\ has\ emoji' = True \mid 'is\ spam' = True) &= 0.198 \\
P('\ has\ emoji' = True \mid 'is\ spam' = False) &= 0.147 \\
P('\ has\ emoji' = False \mid 'is\ spam' = True) &= 0.802 \\
P('\ has\ emoji' = False \mid 'is\ spam' = False) &= 0.853 \\
P('\ sent\ to\ list' = True \mid 'is\ spam' = True) &= 0.07 \\
P('\ sent\ to\ list' = True \mid 'is\ spam' = False) &= 0.312 \\
P('\ sent\ to\ list' = False \mid 'is\ spam' = True) &= 0.93 \\
P('\ sent\ to\ list' = False \mid 'is\ spam' = False) &= 0.688 \\
P('\ from\ .com' = True \mid 'is\ spam' = True) &= 0.744 \\
P('\ from\ .com' = True \mid 'is\ spam' = False) &= 0.275 \\
P('\ from\ .com' = False \mid 'is\ spam' = True) &= 0.256 \\
P('\ from\ .com' = False \mid 'is\ spam' = False) &= 0.725 \\
P('\ has\ my\ name' = True \mid 'is\ spam' = True) &= 0.349 \\
P('\ has\ my\ name' = True \mid 'is\ spam' = False) &= 0.601 \\
P('\ has\ my\ name' = False \mid 'is\ spam' = True) &= 0.651 \\
P('\ has\ my\ name' = False \mid 'is\ spam' = False) &= 0.399 \\
P('\ has\ sig' = True \mid 'is\ spam' = True) &= 0.663 \\
P('\ has\ sig' = True \mid 'is\ spam' = False) &= 0.324 \\
P('\ has\ sig' = False \mid 'is\ spam' = True) &= 0.337 \\
P('\ has\ sig' = False \mid 'is\ spam' = False) &= 0.676 \\
P('\ \#\ sentences' \mid 'is\ spam' = True) &= (3.977, 1.929) \\
P('\ \#\ sentences' \mid 'is\ spam' = False) &= (6.191, 2.53) \\
P('\ \#\ words' \mid 'is\ spam' = True) &= (68.837, 8.908) \\
P('\ \#\ words' \mid 'is\ spam' = False) &= (70.771, 30.212)
\end{aligned}
$$

2.2 Classification Error

```
==================================================================
Misclassification Errors
Error = 12.0%
Accuracy = 88.0%

==================================================================
```

Figure 12: Accuracy of the model when considering all the features

2.3 Best Feature Subset

```
==================================================================
Naives Bayes's Classifier based on selected features
[' has emoji', ' from .com', ' has my name', ' # sentences', ' # words', ' is spam']

==================================================================
```

Figure 13: Feature subset selected

The above image displays the features within the selected feature subset, determined to be the most effective.

2.3.1 Process to find the best subset and predict

In the process, I compared the maximum likelihood parameters (see section 2.1) to identify features influencing the separation the classes. Then I conducted trial and error by combining these influential features. The continuous features were more straightforward, because the observable overlapping factor between the Gaussian probability density functions which helps in checking the effectiveness of the feature. The prediction function, as shown below (see Figure 14) is designed to accommodate any feature subset and effectively predict for unseen data. This adaptability comes from the fact that the maximum likelihood parameters doesn't change even when certain features are not present in the unseen data, so the same set of parameters can be utilized for predictions across different feature subsets.

```python
def predict(data_b, features, maximum_likelihood_parameters):
    """
    Predict for dataset b based on given subset of features
    :param data_b: Input dataset
    :param features: features to be used
    :return: None
    """
    # Keeping only the given features
    data_b = data_b[features]
```

Figure 14: The predict function

2.3.2 Classification Error

```
================================================================
Misclassification Errors
Error = 10.0%
Accuracy = 90.0%
```

Figure 15: Accuracy of the model when considering the feature subset