DATA MINING AND MACHINE LEARNING

ASSIGNMENT 2

1. [3 pts/ea] Logistic regression uses cross-entropy as the error function.

(i) Linear regression uses Sum Squared Error(SSR). Explain why Logistic regression uses cross-entropy.

Logistic regression uses cross-entropy as the error function primarily due to the nature of the output it predicts and the type of problem it addresses—binary classification.

1. Nature of the Output

Logistic Regression Output: Logistic regression predicts probabilities that a given input belongs to a particular class (e.g., class 1 vs. class 0). The output is constrained between 0 and 1 due to the sigmoid function applied to the linear combination of input features.

2. Cross-Entropy Loss

Cross-entropy measures the difference between two probability distributions: the true distribution (actual labels) and the predicted distribution (probabilities from logistic regression).

Formula: For binary classification, the cross-entropy loss can be defined as:

$$\boldsymbol{L(y, \hat{y}) = \frac{1}{N} L(y, \hat{y}) = -\frac{1}{N} \sum_{i=1}^{N} y_i \cdot \log(\hat{y}_i) + (1 - y_i) \log(1 - \hat{y}_i)}$$

Where:

y_i is the true label (0 or 1).

$\hat{y}_i$ is the predicted probability of the positive class.

3. Advantages of Cross-Entropy

Sensitivity to Misclassifications: Cross-entropy loss heavily penalizes predictions that are confident but wrong. For instance, if the model predicts a probability close to 1 for a negative class (0), the penalty is significant, encouraging the model to adjust more aggressively.

Probabilistic Interpretation: Since logistic regression outputs probabilities, using cross-entropy aligns with the probabilistic framework of the model. It quantifies how well the predicted probabilities match the actual labels.

4. Comparison with Sum Squared Error (SSE)

- Inappropriateness of SSE: While Sum Squared Error (SSE) is suitable for regression tasks where the output is continuous, it is not ideal for binary classification. SSE does not appropriately handle the probabilistic nature of the outputs and can lead to suboptimal convergence and predictions.
- Non-Convexity Issues: Using SSE can lead to non-convex loss landscapes, making optimization difficult. Cross-entropy, being derived from the likelihood of the Bernoulli distribution, ensures a convex loss function, which is easier to optimize.

Conclusion

Therefore, logistic regression uses cross-entropy as the error function because it effectively measures the performance of probabilistic predictions, provides strong penalties for incorrect confident predictions, and aligns with the underlying statistical model of binary classification. This leads to better training dynamics and improved model performance compared to using alternatives like Sum Squared Error.

(ii) Show the error function of logistic regression using Sum of Squared Error(SSR)

While logistic regression is typically associated with cross-entropy loss, it is possible to express the error function using the Sum of Squared Error (SSE). However, this approach is less common and may not yield optimal results for binary classification tasks. Therefore we can represent the error function using SSE as follows**:**

Error Function Using Sum of Squared Error (SSE)

Model Prediction: In logistic regression, the predicted probability $\hat{y}$ is given by the logistic function:

$$\hat{\boldsymbol{y}} = \boldsymbol{\sigma}(\boldsymbol{z}) = \frac{\mathbf{1}}{\mathbf{1}+\boldsymbol{e}^{-\boldsymbol{z}}} \textbf{ where } \boldsymbol{z} = \boldsymbol{\beta_0} + \boldsymbol{\beta_1 x_1} + \cdots + \boldsymbol{\beta_n x_n}$$

Here, $\beta_0, \beta_1,\ldots,\beta_n$ are the model parameters, and $x_1,\ldots,x_n$ are the input features

True Labels: Let y be the true label, where y can be either 0 or 1.

Sum of Squared Error: The SSE for logistic regression can be defined as:

$$\boldsymbol{L}(\boldsymbol{y}, \hat{\boldsymbol{y}}) = \frac{\mathbf{1}}{\boldsymbol{N}} \sum_{\boldsymbol{i}=\mathbf{1}}^{\boldsymbol{N}} (\boldsymbol{y_i} - \hat{\boldsymbol{y}}_{\boldsymbol{i}})^{\mathbf{2}}$$

Where:

- N is the number of samples.
- y_i is the true label for sample ii.
- $\hat{y}_i$ is the predicted probability for sample ii.

Substituting the Prediction: Substituting the logistic function into the SSE, we get:

$$\boldsymbol{L(y, \hat{y}) = \frac{1}{N} \sum_{i=1}^{N} (y_i - \frac{1}{1+e^{-z}})^2}$$

Interpretation: This function measures the squared difference between the actual label and the predicted probability. However, it can lead to issues:

- **Non-Convexity**: The loss surface may not be convex, making optimization more challenging.
- **Poor Penalty for Misclassifications**: Unlike cross-entropy, SSE does not heavily penalize confident but incorrect predictions.

Conclusion

Using Sum of Squared Error in logistic regression is theoretically possible, but it is not recommended due to the reasons mentioned above. Cross-entropy remains the preferred choice as it aligns better with the probabilistic nature of the model and provides a more effective optimization landscape.

2. [3 pts] Explain using the reason linear regression as a classifier is very sensitive to outliers.

Linear regression, when used as a classifier, is highly sensitive to outliers due to several key reasons:

Minimization of Sum of Squared Errors (SSE):

- **Error Calculation:** Linear regression aims to minimize the Sum of Squared Errors (SSE) between the predicted values and the actual values. The formula for SSE is:
- $\boldsymbol{L = \sum_{i=1}^{N} (y_i - \hat{y}_i)^2}$

Here, y_i is the true label, and $\hat{y}_i$ is the predicted value.

Impact of Squaring: Because the errors are squared, larger errors (which occur with outliers) have a disproportionately large effect on the total error. For example, an outlier with a large deviation from the predicted value will contribute significantly more to the SSE than smaller errors, skewing the model.

Influence on Model Parameters

- Parameter Estimation: The coefficients in linear regression are calculated using methods like Ordinary Least Squares (OLS), which directly depend on the error terms. When outliers are present, they can heavily influence the slope and intercept of the regression line.
- Shifting the Decision Boundary: If the outlier is far from the other data points, it can pull the regression line towards itself, resulting in a decision boundary that does not accurately represent the majority of the data. This can lead to misclassification of normal data points.

Assumption of Homoscedasticity

- Violation of Assumptions: Linear regression assumes that the error terms are homoscedastic (i.e., they have constant variance). Outliers can introduce heteroscedasticity, where the variance of errors increases with the level of the independent variable. This violation can lead to unreliable estimates and predictions.

Loss of Generalization

- Overfitting: The presence of outliers can lead to overfitting, where the model learns the noise in the training data rather than the underlying pattern. This can result in poor performance on unseen data, as the model becomes too tailored to the outliers.

Conclusion

Therefore, linear regression is sensitive to outliers because the squared error loss function magnifies the influence of large deviations, leading to biased parameter estimates and distorted decision boundaries. This sensitivity can result in significant misclassifications and reduced

model performance, making linear regression less robust in the presence of outliers compared to other classification methods.

4. [5 pts] Logistic regression is to maximize the following formula.

$$\prod_{i=1}^{N} \boldsymbol{p_1}(\boldsymbol{x_i}; \boldsymbol{w})^{y_i} \boldsymbol{p_0}(\boldsymbol{x_i}; \boldsymbol{w})^{1-y_i}$$

If we change above formula to below, does it still work? Explain exactly what is going to happen.

$$\prod_{i=1}^{N} \boldsymbol{p}(\boldsymbol{x_i}; \boldsymbol{w})^{y_i}$$

Original Formula (Maximized in Logistic Regression):

The original formula for logistic regression maximizes the likelihood:

$$\prod_{i=1}^{N} p_1(x_i; w)^{y_i} p_0(x_i; w)^{1-y_i}$$

This is the product of two probabilities, where:

- $p1(x_i, w)$ is the probability that $y_i=1$ given x_i and parameters w
- $p0(x_i, w)$ is the probability that $y_i=0$ given x_i and parameters w
- $y_i \in \{0,1\}$ are the true labels.

This is the classic form for a likelihood function in logistic regression, which accounts for both outcomes of the binary classification ($y_i=1$and $y_i=0$)

Modified Formula:

The modified version proposed in the question is:

$$\prod_{i=1}^{N} \boldsymbol{p}(\boldsymbol{x_i}; \boldsymbol{w})^{y_i}$$

In this formula:

- Only $p(x_i,w)$which appears to represent $p1(x_i,w)$ is being considered.
- The factor $(1-y_i)$ and the corresponding $p0(x_i,w))$ term are missing.

This modification breaks the standard logistic regression setup. Here's why:

- When $y_i=1$, $p(x_i,w)^{yi}$ will just represent the probability $p1(x_i,w)$ which is correct for the positive class.
- However, when $y_i=0$, this formulation does not account for the probability $p0(x_i,w)=1-p_1(x_i,w)$, which is the probability of the negative class. Instead, the modified formula just omits this information, effectively ignoring the negative class when $y_i=0$.

Consequences:

- In this new formula, when $y_i=0$, the contribution to the likelihood will be $p1(x_i,w)^0=1$, which means the outcome for those cases is not considered at all.
- This breaks the logic of logistic regression, which needs to balance both $p_1(xi,w)$ and $p_0(x_i,w)$ (i.e., the probabilities of both classes).

Conclusion:

The new formula does not work for logistic regression. By omitting the negative class probability, it no longer maximizes the full likelihood and loses the ability to properly handle cases where $y_i=0$. The original formulation ensures that both positive and negative outcomes are accounted for, making it the correct one.

5. [3 pts] Logistic regression with many discrete values uses one-vs-all approach. Explain why.

Logistic regression is typically used for binary classification tasks, where the goal is to predict one of two possible outcomes. However, when dealing with multi-class classification problems (where there are more than two discrete values), a common approach is the one-vs-all (OvA) strategy. This method is used because:

1. **Binary Nature of Logistic Regression**: Logistic regression models the probability of a binary outcome. To extend this to multiple classes, we can decompose the problem into several binary classification tasks.
2. **One Class at a Time**: In the one-vs-all approach, a separate logistic regression model is trained for each class. Each model predicts the probability of the input belonging to that specific class versus all other classes combined. For example, if there are three classes (A, B, and C), we would train three models:
    - Model 1: Class A vs. Not A (B and C)
    - Model 2: Class B vs. Not B (A and C)
    - Model 3: Class C vs. Not C (A and B)
3. **Simplicity and Interpretability**: Each model is simpler to understand and interpret since it focuses on distinguishing one class from the rest. This can make it easier to analyze the relationships between features and the target class.
4. **Flexibility**: The one-vs-all approach allows for flexibility in handling different class distributions and can be effective even if the classes are imbalanced.
5. **Final Prediction**: Once all models are trained, the final prediction for a new instance is made by selecting the class corresponding to the model that outputs the highest probability.

Therefore, the one-vs-all approach allows logistic regression to effectively handle multi-class classification by breaking it down into multiple binary classification problems, leveraging the strengths of logistic regression while accommodating the complexity of multiple discrete outcomes.

7.[3 pts] Given D and d defined in p. 7 in PCA slides, explain why minimizing difference between D and d is equivalent to maximizing the variance of the projected data.

Minimizing the difference between DD (the squared Euclidean distance between the original points) and dd (the squared Euclidean distance between the projected points) is equivalent to maximizing the variance of the projected data in the context of Principal Component Analysis (PCA), let's break down the concepts:

1. Squared Euclidean Distance D:

- $D=\|x1-x2\|^2$: This represents the distance between the original data points x1 and x2 in the original 2D space.

2. Squared Euclidean Distance d:

   - $d=\|F(x1)-F(x2)\|^2$: This represents the distance between the projections of these points onto a lower-dimensional space (a line in this case).

3. Error Function:

   - $E(x1,x2)=D-d$: This function measures the error or loss in distance when projecting from the original space to the lower-dimensional space.

Minimizing E

- By minimizing E(x1,x2), we are effectively trying to make d as close to D as possible. In other words, we want the distance between the projected points to be as similar as possible to the distance between the original points.

Relationship to Variance

1. Variance in PCA:

   - Variance measures the spread of the data points around their mean. In PCA, maximizing variance means finding the direction (or line) along which the data points are most spread out.

2. Projection and Variance:

   - When we project the original data onto a line, if the projected points (i.e., F(x1) and F(x2) maintain a large distance dd, then the variance of the projected data will also be large.
   - Conversely, if the projected distances d are small compared to D, it indicates that the projection has lost significant information about the spread of the data, resulting in lower variance.

Conclusion

- Therefore, minimizing the difference $E(x1,x2)=D-d$ ensures that the projected points retain as much of their original spatial relationships as possible.
- This retention of relationships directly correlates with maximizing the variance of the projected data because the more similar d is to D, the more effectively we are capturing the spread of the original data in the lower-dimensional representation.

In summary, minimizing the difference between D and d in PCA aligns with maximizing the variance of the projected data, as it preserves the spatial relationships of the original data points.

9. [3 pts/ea] In Autoencoder, explain what happens in the following cases.

(i) Autoencoder network with linear activation functions

An autoencoder is a type of neural network used for unsupervised learning, primarily for dimensionality reduction or feature learning. It consists of two main parts: an encoder that compresses the input into a lower-dimensional representation and a decoder that reconstructs the original input from this representation.

1) Autoencoder Network with Linear Activation Functions

When an autoencoder uses linear activation functions throughout its layers, several key characteristics emerge:

Characteristics:

1. Linear Transformation:
    - Each layer performs a linear transformation of the input: $y=Wx+b$, where W is the weight matrix, x is the input, b is the bias, and y is the output.
    - The overall function of the autoencoder remains linear, regardless of the number of layers.
2. Limitation in Representation:
    - The entire network can be reduced to a single linear transformation because the composition of multiple linear functions is still a linear function.
    - This means the autoencoder cannot learn complex, non-linear relationships in the data. It essentially behaves like a linear regression model.
3. Reconstruction:

- When the input data is transformed through a linear autoencoder, the reconstruction will also be a linear combination of the input features.
- For example, if the input data lies on a non-linear manifold, the linear autoencoder will struggle to accurately reconstruct the input, often leading to poor performance.

4. Dimensionality Reduction:
    - While it can still perform dimensionality reduction, the reduced representation will not capture the underlying structure of the data effectively if the data is inherently non-linear.
5. Loss Function:
    - The loss function (commonly mean squared error) will still be applicable, but the optimization will not lead to a meaningful representation if the data requires non-linear transformations.

Therefore, an autoencoder with linear activation functions can only learn linear mappings, making it inadequate for capturing complex patterns in data. It can still perform tasks like dimensionality reduction, but its effectiveness is limited to scenarios where the relationships in the data are linear. For most real-world applications, non-linear activation functions (like ReLU, sigmoid, or tanh) are preferred to allow the model to learn richer representations.

(ii) The number of hidden nodes is greater than the number of input nodes

When the number of hidden nodes in an autoencoder is greater than the number of input nodes, several implications and characteristics arise:

Characteristics of an Autoencoder with More Hidden Nodes than Input Nodes

1. Increased Capacity:
    - The autoencoder has a larger capacity to learn complex representations due to the increased number of hidden nodes. This allows the model to capture more intricate patterns in the data.
2. Overfitting Risk:
    - With more hidden nodes than input nodes, there is a higher risk of overfitting. The model may learn to memorize the training data instead of generalizing well

to unseen data. This is especially true if the training dataset is small or not diverse.

3. Redundant Representations:
    - The additional hidden nodes may lead to redundant or unnecessary representations of the input data. The model might learn to encode the same information in multiple ways, which can complicate the learning process.
4. Potential for Non-Linear Transformations:
    - If non-linear activation functions are used, having more hidden nodes allows the autoencoder to learn non-linear transformations more effectively. This can enhance the model's ability to reconstruct complex data structures.
5. Dimensionality Expansion:
    - The hidden layer can serve as a form of dimensionality expansion rather than reduction. This means that instead of compressing the data, the autoencoder might create a higher-dimensional representation, which could be useful for certain tasks (e.g., feature extraction).
6. Training Dynamics:
    - The training process might require more epochs and careful tuning of hyperparameters (like learning rate and regularization) to achieve optimal performance. The increased complexity can make the training process more sensitive to these parameters.
7. Use Cases:
    - This configuration can be beneficial in scenarios where the goal is to capture detailed features from the input data, such as in image processing or when dealing with high-dimensional data.

In conclusion, having more hidden nodes than input nodes in an autoencoder can enhance its capacity to learn complex, non-linear representations. However, it also introduces challenges such as overfitting and redundancy. Careful consideration of the architecture, regularization techniques, and training strategies is essential to leverage the benefits while mitigating the drawbacks.