Course: Machine Learning by Dr. Seyyed Salehi

Homework: HW1

Name: Mohammad Mohammadi

Student ID: 402208592

Question 1 - Linear Regression

We have a linear regression model with p parameters. with least squares method, we train the model with training data of (x1, y1), ..., (xN, yN) which are randomly chosen from the population. Now imagine B^ is the estimation of least squares. Imagine we have a series of test data (x-1 , y-1), ..., (x-N , y- N) which are chosen randomly from the same population. Now, with having :

$R_{te}(\beta) = \frac{1}{M}\sum_{i=1}^{M}(y_i - \beta^T x_i)^2, \quad R_{tr}(\beta) = \frac{1}{N}\sum_{i=1}^{N}(y_i - \beta^T x_i)^2$

show that we have :

$E[R_{tr}(\hat{\beta})] \leq E[R_{te}(\hat{\beta})]$

Answer 1

- $R_{tr}(\beta)$ is our training risk. It's equation is calculated upon training datapoints (xi, yi).
- Respectively, $R_{te}(\beta)$ is our testing risk. It's equation is calculated upon test datapoints (x¯i, y¯i).

Now using these definitions we have:

- By definition, β̂ is obtained by minimizing $R_{tr}(\beta)$ over the training data. Hence, for any β, including the true parameter vector, we have $R_{tr}(\hat{\beta}) \leq R_{tr}(\beta)$.
- Since β̂ is fitted to minimize $R_{tr}(\beta)$ on the training data, the expected training risk $E[R_{tr}(\hat{\beta})]$ is biased downwards because it does not account for the model's generalization to new, unseen data. In contrast, $E[R_{te}(\hat{\beta})]$ measures the expected risk over new samples from the population.
- The test data are not used to fit β̂, so $E[R_{te}(\hat{\beta})]$ includes the model's error across a broader set of samples from the same population. Because the test samples are independent of the training process and are chosen randomly, the overlap is expected to be inadequate, therefore, the expectation $E[R_{te}(\hat{\beta})]$ provides an unbiased estimate of the model's performance on new data.
- The process of fitting β̂ to the training data can lead to overfitting, especially if p (the number of parameters) is large relative to N (the number of training samples). Overfitting results in low training error but potentially high testing error due to the model capturing noise in the training data as if it were signal.
- The inequality $E[R_{tr}(\hat{\beta})] \leq E[R_{te}(\hat{\beta})]$ reflects the concept that, on average, a model's performance on the data it was trained on (training data) is at least as good as, if not better than, its performance on new, unseen data (test data). This is because the training process optimizes performance on the training set, potentially at the expense of generalizability to new data.

In conclusion of the facts above, the expected training error $E[R_{tr}(\hat{\beta})]$ is typically lower than the expected test error $E[R_{te}(\hat{\beta})]$ because the model parameters β̂ are optimized for the training data, which can lead to overfitting and a lack of generalizability to unseen data. The expected test error provides a more unbiased estimate of the model's performance on new data from the same population, which is obvious based on the facts mentioned.

Question 2 - Linear Regression

In Lasso Regression, optimal weight vector is:

$\omega^* = \arg\min_\omega J_\lambda(\omega)$

where:

$J_\lambda = 1/2\|y - X\omega\|_2^2 + \lambda\|\omega\|_1$

and we have:

$X \in \mathbb{R}^{n\times d}$

and we have done whitening on data. $(XX^T = I)$

(a) Show that whitening on data causes the features to become independent in a way that ω_i^* is derived solely from the ith feature. To Prove this, first start by showing that J can be written az below:

$J_\lambda(\omega) = g(y) + \sum_{i=1}^{d} f(X_{:,i}; y_i; \omega_i; \lambda)$

Where $X_{:,i}$ is the ith column of the matrix X.

(b) If $\omega_i \geq 0$, then find ω_i.

(c) If $\omega_i < 0$, then find ω_i.

(d) With respect to the last parts, under what circumstances ω_i becomes zero? How can we apply these circumstances?

(e) As you know, in regression ridge, normalization expression in cost function appears as $1\lambda/2||\omega||_2^2$. In this case when does the ω_i become zero? What's the difference between this and previous one?

Answer 2

Part a

By definition, whitening the data such that $XX^T = I$ ensures that the features are orthogonal and normalized. This implies that the covariance matrix of the features X is the identity matrix, meaning all features are independent of each other and have equal variance.

Decomposing $J_\lambda(\omega)$: We want to show that if the data is whitened, we can rewrite $J_\lambda(\omega) = \frac{1}{2}\|y - X\omega\|_2^2 + \lambda\|\omega\|_1$

as

$J_\lambda(\omega) = g(y) + \sum_{i=1}^{d} \left(\frac{1}{2}(y - X_{:,i}\omega_i)^2 + \lambda|\omega_i|\right)$.

In order to do so, let's start by simplifying the main equation, part by part:

1. **Squared Error Term:**

Let's first focus on the squared error term $\|y - X\omega\|_2^2$. Under the condition $XX^T = I$, we can expand this term as:

$(y - X\omega)^T(y - X\omega) = y^Ty - 2\omega^TX^Ty + \omega^TX^TX\omega$

Given that $XX^T = I$, it implies $X^TX = I$ for whitened data, simplifying to:

$y^Ty - 2\omega^TX^Ty + \omega^T\omega$

The term y^Ty is constant with respect to ω and can be considered as part of $g(y)$. The last term can be decomposed into individual contributions from each ω_i since $X^TX = I$ ensures independence among features.

2. **Regularization Term:**

The regularization term $\lambda\|\omega\|_1$ naturally decomposes into a sum over the individual components of ω, as $\|\omega\|_1$ is the sum of the absolute values of the components of ω.

Therefore, combining these insights, $J_\lambda(\omega)$ can be decomposed into:

$J_\lambda(\omega) = g(y) + \sum_{i=1}^{d} \left(\frac{1}{2}(y - X_{:,i}\omega_i)^2 + \lambda|\omega_i|\right)$

Where $g(y) = \frac{1}{2}y^Ty$ is a function solely of y, and each term in the sum $\frac{1}{2}(y - X_{:,i}\omega_i)^2 + \lambda|\omega_i|$ corresponds to $f(X_{:,i}; y_i; \omega_i; \lambda)$, which depends only on the ith feature $X_{:,i}$, its corresponding weight ω_i , and λ.

This decomposition shows that due to the whitening process, the optimization problem for finding ω^* in Lasso Regression becomes separable by features. Each weight ω^* can be optimized independently based on its corresponding feature $X_{:,i}$ and the target y, reflecting the fact that whitening has rendered the features independent in the context of this optimization problem.

Part b

To find ω_i given $\omega_i \geq 0$, we start from the simplified objective function of Lasso Regression for a single weight ω_i, considering the data has been whitened and features have become independent. The objective function for the ith feature can be expressed as:

$f(\omega_i) = \frac{1}{2}(y - X_{:,i}\omega_i)^2 + \lambda|\omega_i|$

Given $\omega_i \geq 0$, the regularization term simplifies to $\lambda\omega_i$, and we aim to find the ω_i that minimizes this function.

First, we compute the derivative of $f(\omega_i)$ with respect to ω_i:

$\frac{df(\omega_i)}{d\omega_i} = -X_{:,i}^T(y - X_{:,i}\omega_i) + \lambda$

Obviously, to find the minimum of $f(\omega_i)$, we set its derivative equal to zero:

$-X_{:,i}^T(y - X_{:,i}\omega_i) + \lambda = 0$

Reordering the equation we have:

$\omega_i(X_{:,i}^TX_{:,i}) = X_{:,i}^Ty - \lambda$

Due to the whitening process (which means each feature vector $X_{:,i}$ is normalized), we have $X_{:,i}^TX_{:,i} = 1$, hence we can simplify the equation to:

$\omega_i = X_{:,i}^Ty - \lambda$

Now we should consider the condition $\omega_i \geq 0$, this leads us to the soft-thresholding operation used in Lasso, where ω_i is set to zero if applying the threshold would result in a negative value. The correct formula, taking into account the positivity constraint and the soft-thresholding operation, would be:

$$\omega_i = \max\left(0, X_{:,i}^T y - \lambda\right)$$

This equation directly solves for ω_i under the condition that $\omega_i \geq 0$, incorporating both the derivative of the objective function to finding the minimum of the objective function and the soft-thresholding operation required by the Lasso regularization term.

Part c

To find ω_i given $\omega_i < 0$, we need to revisit the Lasso Regression objective function with the regularization term reflecting this condition. Given the objective function for a single feature after whitening, we have:

$$f(\omega_i) = \frac{1}{2}(y - X_{:,i}\omega_i)^2 + \lambda|\omega_i|$$

Given $\omega_i < 0$, the absolute value in the regularization term becomes $-\omega_i$, (since $|\omega_i| = -\omega_i$ when $\omega_i < 0$). Thus, the objective function becomes:

$$f(\omega_i) = \frac{1}{2}(y - X_{:,i}\omega_i)^2 + \lambda(-\omega_i)$$

First, we compute the derivative of $f(\omega_i)$ with respect to ω_i and including the negative sign from the regularization term, we get::

$$\frac{df(\omega_i)}{d\omega_i} = -X_{:,i}^T(y - X_{:,i}\omega_i) - \lambda$$

Obviously, to find the minimum of $f(\omega_i)$, we set its derivative equal to zero:

$$-X_{:,i}^T(y - X_{:,i}\omega_i) - \lambda = 0$$

Reordering the equation we have:

$$\omega_i(X_{:,i}^T X_{:,i}) = X_{:,i}^T y + \lambda$$

Due to the whitening process (which means each feature vector $X_{:,i}$ is normalized), we have $X_{:,i}^T X_{:,i} = 1$, hence we can simplify the equation to:

$$\omega_i = X_{:,i}^T y + \lambda$$

Now we should consider the condition $\omega_i < 0$, we need to apply a condition similar to soft-thresholding but for negative ω_i. The correct formula, taking into account the negativity constraint and the soft-thresholding operation, would be:

$$\omega_i = \min\left(0, X_{:,i}^T y + \lambda\right)$$

This equation directly solves for ω_i under the condition that $\omega_i < 0$, applying the principle of soft-thresholding to ensure that the solution adheres to the $\omega_i < 0$ condition by adjusting for the negative regularization term.

Part d

In the context of Lasso Regression, the weight ω_i becomes zero under circumstances that are directly tied to the strength of the regularization parameter λ and the correlation between the ith feature $X_{:,i}$ and the target variable y. The Lasso method encourages sparsity in the model weights, which means it drives some weights to exactly zero, effectively selecting a subset of the most important features.

From the discussions on finding ω_i for $\omega_i \geq 0$ and $\omega_i < 0$, we derived that:

- For $\omega_i \geq 0, \omega_i = \max\left(0, X_{:,i}^T y - \lambda\right)$
- For $\omega_i < 0, \omega_i = \min\left(0, X_{:,i}^T y + \lambda\right)$

Circumstances under Which ω_i Becomes Zero:

ω_i becomes zero if the adjustment for the regularization (either adding or subtracting λ) brings the term within the soft-thresholding operation to a value that does not support maintaining a positive or negative weight. Specifically:

- If the correlation between $X_{:,i}$ and y (measured as $X_{:,i}^T y$) is less than λ, and we are considering the case for $\omega_i \geq 0$, then ω_i will be set to zero because the term $X_{:,i}^T y - \lambda$ will be non-positive.
- Similarly, if the absolute value of the correlation is less than λ when considering $\omega_i < 0$, then ω_i will also be zero because $X_{:,i}^T y + \lambda$ will be non-negative, but we're looking at cases where it would need to be negative to maintain a non-zero weight.

Applying These Circumstances:

To apply these circumstances in Lasso Regression for feature selection or regularization:

1. **Choose an Appropriate λ:** The choice of λ is crucial as it dictates the level of sparsity in the model. A larger λ encourages more weights to be set to zero, simplifying the model but potentially underfitting the data. Cross-validation is commonly used to select an optimal λ that balances model complexity and fit.
2. **Calculate Correlation:** For each feature $X_{:,i}$, we calculate its correlation with the target variable y (i.e., $X_{:,i}^T y$). This step is part of fitting the Lasso model.

3. **Apply Soft-Thresholding:** For each feature, we apply the soft-thresholding rule as discussed. This involves comparing the absolute value of the correlation $|X_{:,i}^T y|$ to λ and setting ω_i to zero if it falls below the threshold.

By following these steps, Lasso Regression automatically performs feature selection by removing features whose correlation with the target variable is not strong enough to overcome the penalty imposed by the regularization term λ. This property makes Lasso an attractive option for both regularization and feature selection in linear models.

Part e

In Ridge Regression, the regularization term in the cost function is given by $1\lambda/2||\omega||_2^2$, where $||\omega||_2^2$ is the squared L2 norm of the weight vector ω, and λ is the regularization parameter. This term penalizes large weights by adding to the cost function a value that increases with the square of the magnitude of the weights. The key difference from Lasso Regression, which uses an L1 penalty $\lambda||\omega||_1$, is in how these regularization terms influence the weights.

When does ω_i become zero in Ridge Regression?

In Ridge Regression, ω_i does not become exactly zero due to the nature of the L2 penalty. The L2 norm penalizes the square of the weights, and as such, while it encourages the weights to be small to minimize the overall cost function, it does not set them to zero. The gradient descent steps used to minimize the cost function in Ridge Regression will decrease the magnitude of the weights, but the continuous nature of the L2 penalty means there's no specific threshold at which the weight directly goes to zero as there is with the L1 penalty in Lasso Regression.

Difference between Ridge and Lasso Regression regarding ω_i

- **Sparsity:** Lasso Regression can produce sparse models, where some weights are exactly zero. This property makes Lasso useful for feature selection, as it can completely remove some features from the model. Ridge Regression, on the other hand, tends to shrink the weights evenly but does not set them to zero. All features remain part of the model, but their influence is moderated.
- **Penalty and Regularization Effect:** The L1 penalty in Lasso Regression makes the cost function non-differentiable at $\omega_i = 0$, which leads to sparsity. The L2 penalty in Ridge Regression is differentiable everywhere, and its effect is to ensure that the weights do not grow too large, protecting against overfitting by distributing the penalty across all weights more uniformly.
- **Usage:** Lasso Regression is preferred when we suspect that many features are irrelevant or when we want a model that is simple and interpretable due to having only a subset of all the possible predictors. Ridge Regression is preferred when we believe most features have a similar influence on the output or when we have multicollinearity among the features, as the L2 penalty helps handle multicollinearity better by distributing weights across correlated features.

Refs

1. **Linear Regression, Lasso (L1 regularization), and Ridge (L2 regularization):**
   - Elements of Statistical Learning by Trevor Hastie, Robert Tibshirani, and Jerome Friedman. (Contents on web refferenced to this book)
   - An Introduction to Statistical Learning by Gareth James, Daniela Witten, Trevor Hastie, and Robert Tibshirani. (Contents on web refferenced to this book)
   - https://www.mygreatlearning.com/blog/understanding-of-lasso-regression/
   - https://builtin.com/data-science/l2-regularization
2. **Optimization and Regularization Techniques:**
   - Convex Optimization by Stephen Boyd and Lieven Vandenberghe. (Contents on web refferenced to this book)
3. **Machine Learning and Data Science Blogs:**
   - Towards Data Science on Medium: Some articles in the serie on Lasso and Ridge regression.
   - Distill.pub

Question 3 - Linear Classification

The update rule for Perceptron weight vector is:

$$\text{if } x^{(t)} \text{ is misclassified then: } \omega^{(t+1)} = \omega^{(t)} + \eta x^{(t)} y^{(t)}$$

(a) Show that in Perceptron Classifier, weight vector ω can be writen as a linear combination of data $x^{(i)}$. Also find the coefficients a_i in the linear combination $\omega = \sum_{i=1}^N \alpha_i x^{(i)}$.

(b) In what case is the mentioned rule for updating weight vector ω equal to the rule below for updating a_i:

$$\text{if } \sum_{i=1}^N \alpha y^{(i)} y^{(j)} x^{(j)T} < 0 \text{ then } \alpha_i = \alpha_i + 1$$

Answer

Part a

To demonstrate that the weight vector ω in a Perceptron Classifier can be written as a linear combination of the data $x^{(i)}$, and to find the coefficients a_i in the linear combination$\omega = \sum_{i=1}^{N} \alpha_i x^{(i)}$, we need to delve into the update rule of the Perceptron and its implications over multiple iterations.

The Perceptron update rule is given as:

$\text{if } x^{(t)} \text{ is misclassified then: } \omega^{(t+1)} = \omega^{(t)} + \eta x^{(t)} y^{(t)}$

This rule suggests that the weight vector is updated by adding a scaled version of the misclassified input vector $x^{(t)}$, where the scaling factor is $\eta y^{(t)}$. $y^{(t)}$ is the true label of $x^{(t)}$, and η is the learning rate.

Constructing Linear Combination of ω

Initially, let's assume the weight vector ω is initialized to zero ($\omega^{(0)} = 0$). Each time a data point $x^{(t)} is misclassified, ω is updated according to the rule above. After T updates, the weight vector ω can be represented as the sum of all updates:

$$\omega = \sum_{t=1}^{T} \eta x^{(t)} y^{(t)}$$

Here, the sum is over all misclassified points over the training process, not over all data points in the dataset. However, if we consider $a_i = 0$ for correctly classified instances (i.e., instances that do not lead to an update), we can rewrite the expression to include all N instances in the dataset:

$$\omega = \sum_{i=1}^{N} \alpha_i x^{(i)}$$

Finding Coefficients a_i

The coefficient a_i corresponds to $\eta y^{(i)}$ for each time $x^{(i)}$ was misclassified and thus contributed to updating ω. If a particular $x^{(i)}$ was never misclassified, $a_i = 0$. If $x^{(i)}$ was misclassified multiple times, a_i accumulates the sum of $\eta y^{(i)}$ for each misclassification.

Given that η is a constant learning rate, a_i for each $x^{(i)}$ that contributed to an update can be represented as:

$$\alpha_i = \eta \cdot \text{Count}_{\text{misclassified}}(x^{(i)}) \cdot y^{(i)}$$

Where $\text{Count}_{\text{misclassified}}(x^{(i)})$ denotes the number of times $x^{(i)}$ was misclassified and led to an update. This is a simplified representation, assuming η is constant throughout the training. If η changes, a_i would accumulate the specific $\eta y^{(i)}$ values used at each update.

Part b

It's removed!

Just a correction:

The correct mathematical expression should consider the inner product $x^{(i)T} x^{(j)}$, thus correcting the rule to:

$$\text{if } \sum_{i=1}^{N} \alpha_i y^{(i)} y^{(j)} x^{(i)T} x^{(j)} < 0 \text{ then } \alpha_j = \alpha_j + 1$$

Question 4 - Linear Classification

Take a Multinomial Naive Bayes model for the problem of binary classification on textual data. The number of words in our dictionary (Total number of model features) is d. For a textual sample X, values $c_1, c_2, \ldots, c_d$ are the features vector. In other words, each c_i shows the number of times that the ith word has been seen in our expression. Parameters of this model are as below: (y is the output of the model, or the sample class)

$$P_y = P(y = 1)$$
$$P_{i|y=1} = P(\text{word } i \text{ appears in a specific document position} | y = 1)$$
$$P_{i|y=0} = P(\text{word } i \text{ appears in a specific document position} | y = 0)$$

(a) Write an expression for the conditional probability $P(y = 1|x)$ for the textual sample x based on the model parameters.

(b) Show that decision boundry of the trained model is linear.

(c) Show that the conditional probability written in part (a) is a logistic function:

$$P(y = 1|x) = \frac{1}{1 + e^{-(\theta^T x + \theta_0)}}$$

Answer

Part a

To compute the conditional probability $P(y=1|x)$ for the textual sample x using a Multinomial Naive Bayes model, we will apply Bayes' theorem, leveraging the naive assumption that features (word occurrences in this case) are conditionally independent given the class y. This allows us to simplify the computation of the joint probability of features given the class.

For binary classification with y∈{0,1}, and specifically for $P(y=1|x)$, Bayes' theorem can be written as:

$$P(y=1|x)=\frac{P(x|y=1)P(y=1)}{P(x)}$$

Since P(x) is the same for both classes, when we're comparing P(y=1|x) to P(y=0|x), it's sufficient to compare their numerators for classification purposes. However, we focus on expressing P(y=1|x) fully.

Expanding P(x|y=1) under the naive Bayes assumption and given the multinomial distribution of words, we get:

$$P(x|y=1)=\prod_{i=1}^{d}P(\text{word } i|y=1)^{c_i}$$

Where c_i is the count of occurrences of word i in the document, and d is the size of the vocabulary.

So, P(y=1|x) can be written as:

$$P(y=1|x)=\frac{\prod_{i=1}^{d}P(\text{word } i|y=1)^{c_i}P(y=1)}{P(x)}$$

To compute P(x), which is the probability of observing the document regardless of class, we can sum over all possible classes:

$$P(x)=P(x|y=1)P(y=1)+P(x|y=0)P(y=0)$$

Given the components we have:

$$P(x|y=1)P(y=1)=\left(\prod_{i=1}^{d}P(\text{word } i|y=1)^{c_i}\right)P(y=1)$$

$$P(x|y=0)P(y=0)=\left(\prod_{i=1}^{d}P(\text{word } i|y=0)^{c_i}\right)P(y=0)$$

Therefore, the complete expression for P(y=1|x) becomes:

$$P(y=1|x)=\frac{\left(\prod_{i=1}^{d}P(\text{word } i|y=1)^{c_i}\right)P(y=1)}{\left(\prod_{i=1}^{d}P(\text{word } i|y=1)^{c_i}\right)P(y=1)+\left(\prod_{i=1}^{d}P(\text{word } i|y=0)^{c_i}\right)P(y=0)}$$

Part b

To show that the decision boundary of the trained Multinomial Naive Bayes model for binary classification on textual data is linear, we need to analyze the decision rule based on comparing the posterior probabilities P(y=1|x) and P(y=0|x). Specifically, a classifier predicts y=1 if P(y=1|x)>P(y=0|x), and y=0 otherwise. This decision is equivalent to comparing the logarithm of the ratios of these probabilities due to the monotonic property of the logarithm function, which preserves the inequality direction.

Starting from the expressions for P(y=1|x) and P(y=0|x), we can use the log probability for the decision rule to avoid numerical underflow and to simplify the expression:

$$\log\left(\frac{P(y=1|x)}{P(y=0|x)}\right)>0$$

Using the formula for (y=1|x) and P(y=0|x), this can be expanded to:

$$\log\left(\frac{\left(\prod_{i=1}^{d}P(\text{word } i|y=1)^{c_i}\right)P(y=1)}{\left(\prod_{i=1}^{d}P(\text{word } i|y=0)^{c_i}\right)P(y=0)}\right)>0$$

Simplifying logarithms, we get:

$$\sum_{i=1}^{d}c_i\log\left(\frac{P(\text{word } i|y=1)}{P(\text{word } i|y=0)}\right)+\log\left(\frac{P(y=1)}{P(y=0)}\right)>0$$

This equation represents a linear decision rule in the space of c_i (the counts of words in the document). The reason it is linear is because the decision criterion is based on a weighted sum of the features (c_i), where the weights are the log ratios of the conditional probabilities of each word given the classes. The term $log(\frac{P(y=0)}{P(y=1)})$ acts as a bias term.

Thus, the decision boundary between y=1 and y=0 is defined by the set of points for which:

$$\sum_{i=1}^{d} c_i \log\left(\frac{P(\text{word } i|y=1)}{P(\text{word } i|y=0)}\right) + \log\left(\frac{P(y=1)}{P(y=0)}\right) = 0$$

This is the equation of a hyperplane in the d-dimensional feature space, which confirms that the decision boundary of the Multinomial Naive Bayes model is linear.

For the sake of better comprehension we can see that in 2-dimensional feature space if we let:

$w_1 = \log\left(\frac{P(\text{word } 1|y=1)}{P(\text{word } 1|y=0)}\right)$

$w_2 = \log\left(\frac{P(\text{word } 2|y=1)}{P(\text{word } 2|y=0)}\right)$

$b = \log\left(\frac{P(y=1)}{P(y=0)}\right)$

The equation simplifies to:

$$w_1 c_1 + w_2 c_2 + b = 0$$

Which is the equation of a line in 2D space, where c1 and c2 are the axes coordinates, w1 and w2 are the coefficients that determine the direction and steepness of the line, and b is the intercept with the c1-c2 plane.

Part c

To show that the conditional probability P(y=1|x) from part (a) can be expressed as a logistic function, we take the expression we derived in part (a):

$$P(y=1|x) = \frac{\left(\prod_{i=1}^{d} P(\text{word } i|y=1)^{c_i}\right) P(y=1)}{\left(\prod_{i=1}^{d} P(\text{word } i|y=1)^{c_i}\right) P(y=1) + \left(\prod_{i=1}^{d} P(\text{word } i|y=0)^{c_i}\right) P(y=0)}$$

We showed that the decision rule can be written in terms of a logistic function when comparing P(y=1|x) to P(y=0|x). To align it with the logistic function form, we recall the log-odds representation:

$$\log\left(\frac{P(y=1|x)}{P(y=0|x)}\right) = \sum_{i=1}^{d} c_i \log\left(\frac{P(\text{word } i|y=1)}{P(\text{word } i|y=0)}\right) + \log\left(\frac{P(y=1)}{P(y=0)}\right)$$

Let's define $\theta_i = \log\left(\frac{P(\text{word } i|y=1)}{P(\text{word } i|y=0)}\right) \quad \text{for} \quad i=1 \quad \text{to} \quad d$ and $\theta_0 = \log\left(\frac{P(y=1)}{P(y=0)}\right)$. Then, we can rewrite the log-odds as:

$$\log\left(\frac{P(y=1|x)}{P(y=0|x)}\right) = \sum_{i=1}^{d} \theta_i c_i + \theta_0 = \theta^T x + \theta_0$$

Where $\theta^T x$ denotes the dot product of the parameter vector θ and the feature vector x (with x including $c_1, \ldots, c_d$).

The logistic function is defined as:

$$P(y=1|x) = \frac{1}{1+e^{-z)}}$$

Where $z = \theta^T x + \theta_0$. Substituting z with our expression for the log-odds gives:

$$P(y=1|x) = \frac{1}{1+e^{-(\theta^T x + \theta_0)}}$$

Question 5 - Linear Classification

Assume a classification problem with 3 classes in 2 dimenssions with distributions as below:

$$P(x|\omega_1) = \mathcal{N}(0, I)$$

$$P(x|\omega_2) = \mathcal{N}(\begin{bmatrix}1 & 1\end{bmatrix}^T, I)$$

$$P(x|\omega_3) = 0.5 \times \mathcal{N}(\begin{bmatrix}0.5 & 0.5\end{bmatrix}^T, I) + 0.5 \times \mathcal{N}(\begin{bmatrix}-0.5 & 0.5\end{bmatrix}^T, I)$$

$$P(\omega_1) = P(\omega_2) = p(\omega_3)$$

(a) With posterior probability calculation, calssify point $x = \begin{bmatrix}0.3 & 0.3\end{bmatrix}^T$ for the least probability of error mode.

(b) Assume for a specific point, we don't have the first characteristic (which means $x = \begin{bmatrix}* & 0.3\end{bmatrix}^T$). classify this point.

Answer

Part a

To classify the point x=[0.3 0.3] with the least probability of error, we compute the posterior probability for each class and then choose the class with the highest posterior probability. The posterior probability for class ω_i, given x, is given by Bayes' theorem as below:

$$P(\omega_i|x) = \frac{P(x|\omega_i)P(\omega_i)}{P(x)}$$

As we have $P(\omega_1) = P(\omega_2) = p(\omega_3)$, assuming these probabilities are equal and sum to 1, we can simplify the computation by focusing on the numerator $P(x \mid \omega_i)P(\omega_i)$, since the denominators will be the same across all classes and P(x) is a normalizing constant that does not affect the comparison.

From the question, we have distributions for each class as below:

1. for ω_1:

$$P(x|\omega_1) = \mathcal{N}(0, I)$$

2. for ω_2:

$$P(x|\omega_2) = \mathcal{N}(\begin{bmatrix}1 & 1\end{bmatrix}^T, I)$$

3. for ω_3:

$$P(x|\omega_3) = 0.5 \times \mathcal{N}(\begin{bmatrix}0.5 & 0.5\end{bmatrix}^T, I) + 0.5 \times \mathcal{N}(\begin{bmatrix}-0.5 & 0.5\end{bmatrix}^T, I)$$

Our normal distribution $\mathcal{N}(\mu, \Sigma)$ has a probability density function (PDF) as below:

$$P(x) = \frac{1}{(2\pi)^{d/2}|\Sigma|^{1/2}} e^{-\frac{1}{2}(x-\mu)^T\Sigma^{-1}(x-\mu)}$$

In our case, $\Sigma = I$, the identity matrix, so $|\Sigma|^{\frac{1}{2}} = 1$ and $\Sigma^-1 = I$. Thus, for a 2-dimensional identity covariance matrix, the PDF simplifies to:

$$P(x) = \frac{1}{2\pi} e^{-\frac{1}{2}(x-\mu)^T(x-\mu)}$$

Let's calculate the likelihood $P(x \mid \omega_i)$ for each class for the point x = [0.3 0.3], and identify the class with the highest posterior probability to classify point x.

We use MultivariateNormal class from scipy library of python to do so as below:

```python
import numpy as np
from scipy.stats import multivariate_normal

# Define the point x
x = np.array([0.3, 0.3])

# Mean vectors for each class
mu_1 = np.array([0, 0])
mu_2 = np.array([1, 1])
mu_3_1 = np.array([0.5, 0.5])
mu_3_2 = np.array([-0.5, 0.5])

# Covariance matrix (identity for all classes)
cov = np.array([[1, 0], [0, 1]])

# Calculate likelihoods P(x|ω_i)
likelihood_1 = multivariate_normal.pdf(x, mean=mu_1, cov=cov)
likelihood_2 = multivariate_normal.pdf(x, mean=mu_2, cov=cov)
likelihood_3 = 0.5 * multivariate_normal.pdf(x, mean=mu_3_1, cov=cov) + 0.5 * multivariate_normal.pdf(x, mean=mu_3_2, cov


print("For w1, P(x|w1)=", likelihood_1)
print("For w2, P(x|w2)=", likelihood_2)
print("For w3, P(x|w3)=", likelihood_3)

```

```
For w1, P(x|w1)= 0.1454566657817508
For w2, P(x|w2)= 0.09750251890301381
For w3, P(x|w3)= 0.1330980768626826
```

Given that $P(\omega_1) = P(\omega_2) = P(\omega_3)$ and are constants that cancel out in the comparison, the class with the highest posterior probability for point x is ω_1, as it has the highest likelihood.

Therefore, with the least probability of error, the point x=[0.3 0.3] should be classified into class ω_1.

Part b

As distributions are Gaussian and we're missing only one of two dimensions, we focusing on the available dimension.

Since the covariance matrix for each class is the identity matrix I and the distributions are independent across dimensions, we can consider only the second dimension for classifying this point. Essentially, we evaluate $P(x_2|\omega_i)$ for each class i, where $x_2 = 0.3$ is the second characteristic of x.

The distributions along the second dimension change as below:

1. For ω_1, it's N(0,1) since the mean vector is $[0, 0]^T$ and the covariance matrix is I.
2. For ω_2, it's N(1,1) since the mean vector is $[1, 1]^T$.
3. For ω_2, the distribution is a mixture: 0.5 x N(0.5, 1) + 0.5 x N(0.5, 1), with the means for the second characteristic being 0.5 in both components.

To compute the probability of x =0.3 given each class ω_i using these distributions, we use MultivariateNormal class from scipy library of python as part (a) with respective changes:

```
import numpy as np
from scipy.stats import multivariate_normal

# For the second dimension, we're only looking at the value 0.3
x_2 = 0.3

# Means for the second dimension in each distribution
mu_1_2 = 0
mu_2_2 = 1
mu_3_1_2 = 0.5
mu_3_2_2 = 0.5

# Calculate likelihoods P(x_2|ω_i) for the second characteristic
likelihood_1_2 = multivariate_normal.pdf(x_2, mean=mu_1_2, cov=1)
likelihood_2_2 = multivariate_normal.pdf(x_2, mean=mu_2_2, cov=1)
likelihood_3_2 = 0.5 * multivariate_normal.pdf(x_2, mean=mu_3_1_2, cov=1) + \
                 0.5 * multivariate_normal.pdf(x_2, mean=mu_3_2_2, cov=1)

likelihood_1_2, likelihood_2_2, likelihood_3_2

print("For w1, P(x|w1)=", likelihood_1_2)
print("For w2, P(x|w2)=", likelihood_2_2)
print("For w3, P(x|w3)=", likelihood_3_2)

```

```
For w1, P(x|w1)= 0.3813878154605241
For w2, P(x|w2)= 0.3122539333667613
For w3, P(x|w3)= 0.3910426939754559
```

Given these likelihoods and assuming equal priors for each class, the class with the highest probability for the point x=[* 0.3], as it has the highest likelihood value. Therefore, with the least probability of error, this point should be classified into class ω_3 .

Question 6 - Probabilistic perspective on regression

In linear regression problem, we intend to give different weights to different training samples. To be exact we want to $J(\theta)$, which is defined as below:

$$J(\theta) = \frac{1}{2}\sum_{i=1}^{m} \omega(i)(\theta^T x^{(i)} - y^{(i)})^2$$

(a) Show that matrix W exists such that we have:

$$J(\theta) = (X\theta - y)^T W (X\theta - y)$$

(b) Find the θ that minimizes $J(\theta)$ by calculating $\nabla_\theta J(\theta)$ and setting it to zero. (Notice: In the case where all weights are equal, we know that $\theta^* = (X^T X)^{-1} X^T y$. Your answer for this part should be a closed form that is a function o WX and y.)

(c) Assume we have dataset $\{(x^{(i)}, y^{(i)}) : i = 1, \dots, m\}$ of m independent samples. We intend to model $y^{(i)}$ in such a way that they are derived from conditional distributions with different levels of variance. Specifically assume we have:

$$p(y^{(i)}|x^{(i)}; \theta) = \frac{1}{\sqrt{2\pi}\sigma^{(i)}} \exp\left(-\frac{(y^{(i)} - \theta^T x^{(i)})^2}{2(\sigma^{(i)})^2}\right)$$

In other words this is a Gaussian distribution function with average of $\theta^T x^{(i)}$ and variance of $(\sigma^{(i)})^2$. $\sigma^{(i)}$ is constant and has a specific value. Show that finding maximum likelihood estimation (MLE) for θ, equalls to solving a weighted linear regression problem. Specifically find values of $\omega^{(i)}$ with respect to $\sigma^{(i)}$.

Answer

Part a

For the final equation we should construct the W and X matrices.

- X is a matrix of size m×n (where m is the number of training samples and n is the number of features), with each row representing a training example $x^{(i)T}$.
- W is a diagonal matrix of size m×m with ω(i) as the diagonal elements, representing the weights for each training example. This means something like W=diag(ω(1),ω(2),...,ω(m)).

Given these definitions and from the question's equation, Xθ yields a vector of size m×1, where each element $(X\theta)_i = \theta^T x^{(i)}$ represents the predicted output for the ith training example.

The term Xθ−y then gives a vector of prediction errors (differences between predicted and actual values) for each training example.

Multiplying this error vector by W weights the errors according to their importance. Specifically, W(Xθ−y) performs element-wise multiplication of the error vector by the weights.

Finally, $(X\theta - y)^T W(X\theta - y)$ computes the weighted sum of squared errors. The left multiplication by $(X\theta - y)^T$ effectively sums up these weighted squared errors, because for any vector v, and a diagonal matrix W with weights on the diagonal, $v^T W v$ gives $\sum v_i^2 \omega(i)$, which matches the given definition of J(θ).

Hence, we have shown that the matrix W=diag(ω(1),ω(2),...,ω(m)) allows us to write J(θ) in the form:

$$J(\theta) = (X\theta - y)^T W(X\theta - y)$$

Here the conversion is complete, but there is a constant (1/2) coefficient, which we can take into effect in matrix W values for example.

Part b

To find the value of θ that minimizes J(θ), we need to calculate the gradient of J(θ) with respect to θ, set it to zero, and solve it for θ. The matrix form expression for J(θ) is:

$$J(\theta) = (X\theta - y)^T W(X\theta - y)$$

The gradient of J(θ) with respect to θ:

$$\nabla_\theta J(\theta) = \nabla_\theta[(X\theta - y)^T W(X\theta - y)] = \nabla_\theta[\theta^T X^T W X\theta - \theta^T X^T W y - y^T W X\theta + y^T W y]$$

- $y^T W y$ is a constant with respect to θ, its gradient is zero.
- $\theta^T X^T W y$ is a scalar, so $\theta^T X^T W y = y^T W X\theta$

So we have:

$$= \nabla_\theta[\theta^T X^T W X\theta - 2y^T W X\theta]$$
$$= 2X^T W X\theta - 2X^T W y = 0$$

And we have:

$$2X^T W X\theta = 2X^T W y$$

Where θ is:

$$\theta = (X^T W X)^{-1} X^T W y$$

And this θ is a closed-form expression.

Part c

To show that finding the Maximum Likelihood Estimation (MLE) for θ in the given setting is equivalent to solving a weighted linear regression problem, we will start with the assumption that $y^{(i)}$ follows a Gaussian distribution with mean $\theta^T x^{(i)}$ and variance $(\sigma^{(i)})^2$, as given by:

$$p(y^{(i)}|x^{(i)};\theta) = \frac{1}{\sqrt{2\pi}\sigma^{(i)}}\exp\left(-\frac{(y^{(i)} - \theta^T x^{(i)})^2}{2(\sigma^{(i)})^2}\right)$$

Given a dataset of m independent samples, the likelihood of observing the dataset given θ is the product of the probabilities for each individual observation:

$$L(\theta) = \prod_{i=1}^{m} \frac{1}{\sqrt{2\pi}\sigma^{(i)}}\exp\left(-\frac{(y^{(i)} - \theta^T x^{(i)})^2}{2(\sigma^{(i)})^2}\right)$$

Taking the natural logarithm of the likelihood (log-likelihood) to simplify the product into a sum, we get:

$$\log L(\theta) = \sum_{i=1}^{m}\left[-\log(\sqrt{2\pi}\sigma^{(i)}) - \frac{(y^{(i)} - \theta^T x^{(i)})^2}{2(\sigma^{(i)})^2}\right]$$

Simplifying further and dropping constants that do not depend on θ, we have:

$$\log L(\theta) = -\sum_{i=1}^{m}\left[\frac{(y^{(i)} - \theta^T x^{(i)})^2}{2(\sigma^{(i)})^2}\right]$$

To maximize the log-likelihood, we equivalently minimize the negative log-likelihood. This yields a cost function J(θ) that we seek to minimize:

$$J(\theta) = \sum_{i=1}^{m}\left[\frac{(y^{(i)} - \theta^T x^{(i)})^2}{2(\sigma^{(i)})^2}\right]$$

Comparing this expression to the weighted linear regression cost function:

$$J(\theta) = \frac{1}{2}\sum_{i=1}^{m} \omega(i)(\theta^T x^{(i)} - y^{(i)})^2$$

We can see that the two are equivalent if we set:

$$\omega(i) = \frac{1}{(\sigma^{(i)})^2}$$

Thus, the weights $\omega_{(i)}$ in the weighted linear regression problem are inversely proportional to the variances of the conditional distributions of the output variable $y^{(i)}$. Hence, solving for the MLE of θ under the assumption of Gaussian errors with different variances for each observation is equivalent to solving a weighted linear regression problem with weights $\omega(i) = \frac{1}{(\sigma^{(i)})^2}$.

Question 7 - Probabilistic perspective on classification

Assume a Naive Bayes classification problem with 3 classes and 2 features. One of the features come from Bernoulli's distribution and the other one comes from Gaussian distribution. Features are showed with $X = [X_1, X_2]^T$ and class is showed with Y.

Initial distribution is as below:

$$P[Y=0] = 0.5, \quad P[Y=1] = 0.25, \quad P[Y=2] = 0.25$$

Features distribution is as below:

$$p_{X_1|Y}(x_1|Y=c) = \text{Ber}(x_1;\theta_c),$$
$$p_{X_2|Y}(x_2|Y=c) = \text{Normal}(x_2;\mu_c,\sigma_c^2)$$

Also Assume that:

$$\sigma_c = \begin{cases} 1 & \text{if } c=0 \\ 1 & \text{if } c=1 \\ 1 & \text{if } c=2 \end{cases},$$

$$\mu_c = \begin{cases} -1 & \text{if } c=0 \\ 0 & \text{if } c=1 \\ 1 & \text{if } c=2 \end{cases},$$

$$\theta_c = \begin{cases} 0.5 & \text{if } c=0 \\ 0.5 & \text{if } c=1 \\ 0.5 & \text{if } c=2 \end{cases}$$

(a) Calculate $p_{Y|X_1,X_2}(y|x_1=0, x_2=0)$. (The answer should be a vector in R^3 space that sum of its elements is 1.)

(b) Calculate $p_{Y|X_1}(y|x_1=0)$.

(c) Calculate $p_{Y|X_2}(y|x_2=0)$.

(d) Analyze the pattern you found in the answers of the previous parts.

Answer

Part a

To calculate P for each class c∈{0,1,2}, we'll use Bayes' theorem in the context of Naive Bayes classification. The posterior probability of class c given features $X_1 = x_1$ and $X_2 = x_2$ is given by:

$$p_{Y|X_1,X_2}(y=c|x_1,x_2) = \frac{p_{X_1|Y}(x_1|Y=c)\cdot p_{X_2|Y}(x_2|Y=c)\cdot P[Y=c]}{p_{X_1,X_2}(x_1,x_2)}$$

For $X_1 = x_1$ and $X_2 = x_2$, we substitute these values into the formula. Also, $p_{X_1,X_2}(x_1,x_2)$ acts as a normalizing constant to ensure the probabilities sum to 1, which can be calculated as the sum of numerators for all classes.

As for the distributions we have:

- For X_1 (Bernoulli):$p_{X_1|Y}(x_1|Y=c) = \theta_c^{x_1}(1-\theta_c)^{1-x_1}$
- For X_2 (Gaussian with $\sigma_c = 1$):$p_{X_2|Y}(x_2|Y=c) = \frac{1}{\sqrt{2\pi}\sigma_c}\exp\left(-\frac{(x_2-\mu_c)^2}{2\sigma_c^2}\right)$

Substituting $\theta_c = 0.5, \sigma_c = 1$, and μ_c for each class, we can write a code snipet to calculate the posterior probabilities as below:

```python
from scipy.stats import norm, bernoulli
import numpy as np

# Given values
P_Y = np.array([0.5, 0.25, 0.25])  # Prior probabilities
theta_c = 0.5  # Same for all classes
sigma_c = 1  # Same for all classes
mu_c = np.array([-1, 0, 1])  # Means for each class
x_1 = 0
x_2 = 0

# Calculate Bernoulli and Normal probabilities for each class
bernoulli_probs = bernoulli.pmf(x_1, theta_c)
normal_probs = norm.pdf(x_2, mu_c, sigma_c)

# Calculate unnormalized posteriors
unnormalized_posteriors = bernoulli_probs * normal_probs * P_Y

# Normalize to get probabilities
posterior_probs = unnormalized_posteriors / unnormalized_posteriors.sum()


print("The posterior probability for Y=0: ", posterior_probs[0])
print("The posterior probability for Y=1: ", posterior_probs[1])
print("The posterior probability for Y=2: ", posterior_probs[2])
print("Sum of values: ", posterior_probs[0] + posterior_probs[1] + posterior_probs[2])

```

```
The posterior probability for Y=0:  0.4302258370717044
The posterior probability for Y=1:  0.35466124439244345
The posterior probability for Y=2:  0.2151129185358522
Sum of values:  1.0
```

Part b and c

To calculate $p_{Y|X_1}(y|x_1 = 0)$ and $p_{Y|X_2}(y|x_2 = 0)$, we can apply Naive Bayes approach with focusing on one feature at a time:

For (b)

Given the Bernoulli distribution of X_1, we use the given θ_c for each class and $x_1 = 0$. The prior probabilities $P[Y = c]$are also provided.

The posterior probability for class c having $X_1 = x_1$ is proportional to:

$$p_{Y|X_1}(y = c|x_1) \propto p_{X_1|Y}(x_1|Y = c) \cdot P[Y = c]$$

As $x_1 = 0$, we can use $p_{X_1|Y}(0|Y = c) = 1 - \theta_c$ for all classes (with $\theta_c = 0.5$).

For (c)

For X_2, given the normal distribution of X_2 with μ_c for each class and $\sigma_c = 1$, we can calculate the likelihood $p_{X_2|Y}(x_2 = 0|Y = c) \text{ for } x_2 = 0.$

The posterior probability for class c given $X_2 = x_2$ is proportional to:

$$p_{Y|X_2}(y = c|x_2) \propto p_{X_2|Y}(x_2|Y = c) \cdot P[Y = c]$$

Calculations

To perform the calculations for both parts, we normalize the results to sum to 1.

```python
from scipy.stats import norm, bernoulli
import numpy as np

# For p_{Y|X_1}(y|x_1 = 0)
bernoulli_likelihoods_x1_0 = bernoulli.pmf(0, theta_c)  # Same for all classes
posterior_probs_x1_0 = P_Y * bernoulli_likelihoods_x1_0
posterior_probs_x1_0 /= posterior_probs_x1_0.sum()

# For p_{Y|X_2}(y|x_2 = 0)
normal_likelihoods_x2_0 = norm.pdf(0, mu_c, sigma_c)
posterior_probs_x2_0 = P_Y * normal_likelihoods_x2_0
posterior_probs_x2_0 /= posterior_probs_x2_0.sum()

posterior_probs_x1_0, posterior_probs_x2_0

print("Part (b), The posterior probability for Y=0: ", posterior_probs_x1_0[0])
print("Part (b), The posterior probability for Y=1: ", posterior_probs_x1_0[1])
print("Part (b), The posterior probability for Y=2: ", posterior_probs_x1_0[2])
print("Part (b), Sum of values: ", posterior_probs_x1_0[0] + posterior_probs_x1_0[1] + posterior_probs_x1_0[2])

print("Part (c), The posterior probability for Y=0: ", posterior_probs_x2_0[0])
print("Part (c), The posterior probability for Y=1: ", posterior_probs_x2_0[1])
print("Part (c), The posterior probability for Y=2: ", posterior_probs_x2_0[2])
print("Part (c), Sum of values: ", posterior_probs_x2_0[0] + posterior_probs_x2_0[1] + posterior_probs_x2_0[2])

```

```
Part (b), The posterior probability for Y=0:  0.5
Part (b), The posterior probability for Y=1:  0.25
Part (b), The posterior probability for Y=2:  0.25
Part (b), Sum of values:  1.0
Part (c), The posterior probability for Y=0:  0.4302258370717044
Part (c), The posterior probability for Y=1:  0.3546612443924434
Part (c), The posterior probability for Y=2:  0.2151129185358522
Part (c), Sum of values:  1.0
```

Analytical results:

Part (b): These are the same as the prior probabilities P[Y=c], due to the uniform distribution of X_1 across all classes.

Part (c): These probabilities reflect the influence of the Gaussian-distributed feature X_2 on class likelihoods, adjusted by the different means μ_c for each class.

Part d

The calculation for $p_{Y|X_1}(y|x_1 = 0)$ results in posterior probabilities that are identical to the prior probabilities of each class. This is because the feature X_1 is distributed uniformly across all classes ($\theta_c = 0.5$ for all c), making it non-informative in this specific case ($x_1 = 0$).

On the other hand, $p_{Y|X_2}(y|x_2 = 0)$ shows variation in the posterior probabilities based on the Gaussian distribution's means (μ_c) for each class. The closer the mean of a class's Gaussian distribution to the observed value $x_2 = 0$, the higher the likelihood and hence the posterior probability of that class, given the observation. This illustrates the influence of the feature's distribution parameters on class probabilities.