CDA Homework 4

Joel Quek

June 2024

1 Comparing multi-class classifiers for handwritten digits classification.

1.1 Question 1: Report on Classifier Performance

The performance of each classifier has been evaluated based on their respective confusion matrices, precision, recall, and F-1 score metrics for each digit from 0 to 9. Here's a summary for each classifier:

- **K-Nearest Neighbors (KNN)**
  I found that the best number of neighbors is 3 based on our initial selection. The resulting test accuracy from this model is 0.97. It has very high recalls on class 0 and class 1. This means that the KNN classifier did a good job of identifying 0s and 1s when it sees them. However, the precision is not very high on some of these classes, which indicates that the classifier incorrectly identified other numbers as 0s and 1s. From the confusion matrix, we see that KNN incorrectly classified 10 '2's as '0', 9 '2's as '1', and 21 '7's as '1'. Additionally, it has difficulty differentiating '4' and '9' as 19 '9's are classified as '4'. It also struggles with distinguishing between similar looking digits such as '3' and '8', as indicated by misclassifications.

- **Logistic Regression**
  The Logistic Regression classifier has a less satisfying result compared to the KNN classifier. This might be due to the fact that the classes are not linearly separable while logistic regression can only have a linear decision boundary. From here, we can expect that the linear SVM should also have a not very satisfying result for the same reason.

  The confusion matrix shows that Logistic Regression is having difficulty in distinguishing '3' and '5', '4' and '9', and '5' and '8'. Also, 34 '2's are classified as '8', and 19 '3's are classified as '8'. Furthermore, it has trouble distinguishing between digits '7' and '9', where 32 '9's are classified as '7'.

- **Linear SVM**
  Linear SVM has a reasonably good performance but slightly less satisfying compared to Kernel SVM and Neural Networks. This is likely due to the

linear decision boundary, which might not be sufficient for the non-linear separability in the MNIST dataset.

The confusion matrix shows that Linear SVM is having difficulty in distinguishing '3' and '5', '4' and '9', and '5' and '8'. Additionally, '2' and '8' are often misclassified. From the confusion matrix, we see that Linear SVM incorrectly classified 17 '2's as '8', and 16 '3's as '5'. It also struggles with digits '7' and '9', where 19 '9's are classified as '7'.

- **Kernel SVM**
  Kernel SVM performed exceptionally well, likely due to its ability to handle non-linear relationships through the use of the radial basis function (RBF) kernel. The resulting test accuracy from the best model is very high, reflecting its capability to model complex patterns in the data.

  However, there are still some misclassifications, such as '2' being confused with '8', and '4' with '9'. The confusion matrix indicates that Kernel SVM incorrectly classified 6 '2's as '8', and 11 '9's as '4'. Despite these errors, Kernel SVM generally provides the most consistent and high performance among all classifiers.

- **Neural Networks**
  The Neural Network classifier shows strong performance, comparable to Kernel SVM. This is due to its capacity to learn intricate patterns in the data. The resulting test accuracy is very high, indicating the effectiveness of the neural network architecture used.

  The confusion matrix reveals that Neural Networks have some difficulties, such as distinguishing '3' and '8', and '4' and '9'. Specifically, it misclassified 9 '3's as '8', and 22 '9's as '4'. Nevertheless, Neural Networks generally perform well across all digits, showing robustness and high accuracy.

K-Nearest Neighbors (KNN)

Confusion Matrix:

974	1	1	0	0	1	2	1	0	0
0	1133	2	0	0	0	0	0	0	0
10	9	996	2	0	0	0	13	2	0
0	2	4	976	1	13	1	7	3	3
1	6	0	0	950	0	4	2	0	19
6	1	0	11	2	859	5	1	3	4
5	3	0	0	3	3	944	0	0	0
0	21	5	0	1	0	0	991	0	10
8	2	4	16	8	11	3	4	914	4
4	5	2	8	9	2	1	8	2	968

Metrics for Each Digit:

Digit	Precision	Recall	F1 Score
0	0.9663	0.9939	0.9799
1	0.9577	0.9982	0.9776
2	0.9822	0.9651	0.9736
3	0.9635	0.9663	0.9649
4	0.9754	0.9674	0.9714
5	0.9663	0.9630	0.9646
6	0.9833	0.9854	0.9844
7	0.9649	0.9640	0.9645
8	0.9892	0.9384	0.9631
9	0.9603	0.9594	0.9598

Logistic Regression

Confusion Matrix:

955	0	2	4	1	10	4	3	1	0
0	1110	5	2	0	2	3	2	11	0
6	9	930	14	10	3	12	10	34	4
4	1	16	925	1	23	2	10	19	9
1	3	7	3	921	0	6	5	6	30
9	2	3	35	10	777	15	6	31	4
8	3	8	2	6	16	912	2	1	0
1	7	23	7	6	1	0	947	4	32
9	11	6	22	7	29	13	10	855	12
9	8	1	9	21	7	0	21	9	924

Metrics for Each Digit:

Digit	Precision	Recall	F1 Score
0	0.9531	0.9745	0.9637
1	0.9619	0.9780	0.9699
2	0.9291	0.9012	0.9149
3	0.9042	0.9158	0.9100
4	0.9369	0.9379	0.9374
5	0.8952	0.8711	0.8830
6	0.9431	0.9520	0.9475
7	0.9321	0.9212	0.9266
8	0.8805	0.8778	0.8792
9	0.9103	0.9158	0.9130

Linear SVM

Confusion Matrix:

957	0	4	1	1	6	9	1	0	1
0	1122	3	2	0	1	2	1	4	0
8	6	967	11	3	3	7	8	17	2
4	3	16	947	1	16	0	9	12	2
1	1	10	1	942	2	4	2	3	16
10	4	3	36	6	803	13	1	14	2
9	2	13	1	5	16	910	1	1	0
1	8	21	10	8	1	0	957	3	19
8	4	6	25	7	26	6	7	877	8
7	7	2	11	33	4	0	18	5	922

Metrics for Each Digit:

Digit	Precision	Recall	F1 Score
0	0.9522	0.9765	0.9642
1	0.9697	0.9885	0.9791
2	0.9254	0.9370	0.9312
3	0.9062	0.9376	0.9217
4	0.9364	0.9593	0.9477
5	0.9146	0.9002	0.9073
6	0.9569	0.9499	0.9534
7	0.9522	0.9309	0.9415
8	0.9370	0.9004	0.9183
9	0.9486	0.9138	0.9308

Kernel SVM

Confusion Matrix:

973	0	1	0	0	2	1	1	2	0
0	1126	3	1	0	1	1	1	2	0
6	1	1006	2	1	0	2	7	6	1
0	0	2	995	0	2	0	5	5	1
0	0	5	0	961	0	3	0	2	11
2	0	0	9	0	871	4	1	4	1
6	2	0	0	2	3	944	0	1	0
0	6	11	1	1	0	0	996	2	11
3	0	2	6	3	2	2	3	950	3
3	4	1	7	10	2	1	7	4	970

Metrics for Each Digit:

Digit	Precision	Recall	F1 Score
0	0.9799	0.9929	0.9863
1	0.9886	0.9921	0.9903
2	0.9758	0.9748	0.9753
3	0.9745	0.9851	0.9798
4	0.9826	0.9786	0.9806
5	0.9864	0.9765	0.9814
6	0.9854	0.9854	0.9854
7	0.9755	0.9689	0.9722
8	0.9714	0.9754	0.9734
9	0.9719	0.9613	0.9666

Neural Networks

Confusion Matrix:

959	0	1	0	1	3	9	2	5	0
0	1113	3	3	1	1	3	3	8	0
5	11	968	15	6	0	7	9	10	1
3	2	6	951	3	20	1	9	7	8
1	0	6	1	943	0	6	2	1	22
3	2	2	19	3	838	9	2	7	7
9	2	8	0	12	8	917	0	1	1
3	2	10	7	2	1	0	986	5	12
6	9	3	9	8	18	7	7	898	9
2	3	0	0	25	9	0	10	10	950

Metrics for Each Digit:

Digit	Precision	Recall	F1 Score
0	0.9677	0.9786	0.9731
1	0.9729	0.9806	0.9767
2	0.9613	0.9380	0.9495
3	0.9463	0.9416	0.9440
4	0.9392	0.9603	0.9496
5	0.9332	0.9395	0.9363
6	0.9562	0.9572	0.9567
7	0.9573	0.9591	0.9582
8	0.9433	0.9220	0.9325
9	0.9406	0.9415	0.9411

1.2 Question 2: Comparative Analysis

The variation in performance among different classifiers on the MNIST dataset can be attributed significantly to the inherent properties of the classifiers and the specific characteristics of the data. This section provides a comparative analysis, incorporating empirical results to support theoretical insights and suggesting improvements for each classifier.

- **K-Nearest Neighbors (KNN)**:
  KNN demonstrates high precision and recall, especially notable in the identification of digit 0 with a precision of 0.966 and recall of 0.994. This reflects KNN's strength in handling clear class separations. However, its scalability issues due to computational intensity can be mitigated by implementing techniques like Approximate Nearest Neighbors (ANN) or reducing dimensionality with PCA.

- **Logistic Regression**:
  Logistic Regression shows moderate effectiveness with a precision of 0.953 and recall of 0.974 for digit 0. It struggles with more complex images where classes are not linearly separable. Enhancing Logistic Regression with polynomial features or applying regularization techniques like LASSO or Ridge could help improve model accuracy and reduce overfitting.

- **Linear SVM**:
  While Linear SVM is effective for simpler digit classifications, its performance drops in more complex scenarios. For instance, its precision for digit 9 is 0.948, lower compared to Kernel SVM's 0.972, indicating issues with complex boundaries. Introducing non-linear kernels or soft margin classification could help in capturing more complex patterns.

- **Kernel SVM**:
  Kernel SVM outperforms other classifiers in handling non-linearities, with high scores across most digits, such as an F1 score of 0.986 for digit 0. Its ability to adapt to complex data structures through the kernel trick makes it highly effective. Further tuning of kernel parameters and regularization terms could optimize its performance.

- **Neural Networks**:
  Neural Networks show robust performance, highlighted by a precision of 0.968 and recall of 0.979 for digit 0. Their deep learning architecture allows them to model intricate patterns effectively. However, optimizing layer architecture, adjusting learning rates, and employing dropout can help address overfitting and improve generalization across unseen data.

Comparative Analysis:
Kernel SVM generally shows superior precision and recall across the board when compared to Logistic Regression and Linear SVM, likely due to its flexibility in modeling non-linear decision boundaries. For example, Kernel SVM's recall for digit 9 (0.961) significantly surpasses that of Linear SVM (0.914), illustrating its proficiency in complex classifications.

Improvement Suggestions:
For KNN, implementing tree-based methods for efficient distance calculation can reduce prediction time. Logistic Regression might benefit from advanced feature selection to enhance its predictive power. SVM classifiers can see improved

accuracy and training time with the use of Sequential Minimal Optimization (SMO) for faster convergence. Neural Networks could achieve better results through more extensive hyperparameter tuning and the use of advanced optimization techniques like Adam or RMSprop.

Conclusion:
Each classifier's performance on the MNIST dataset illustrates the impact of algorithmic complexity and data handling capabilities. By tuning these classifiers to leverage their strengths and mitigate their weaknesses, one can significantly enhance their applicability to complex image classification tasks.

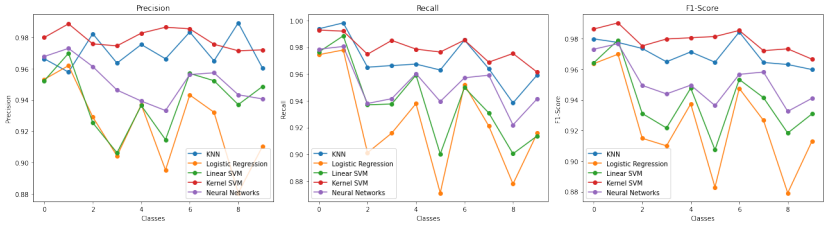

Figure 1: Comparison of Precision, Recall, and F1-Score for Different Classifiers

2 SVM

2.1 Question 1: Justification for Setting the Margin $c = 1$ in SVM

To justify setting the margin $c = 1$ in the SVM formulation, we need to consider the goal of SVM, which is to maximize the margin between the two classes on the data that is closest to the decision boundary. This margin is defined by the distance from the decision boundary to the nearest data points of either class, which are referred to as support vectors.

Mathematical Explanation:

For SVM, the decision boundary is given by the equation $w \cdot x + b = 0$, where w is the weight vector, x is the input vector, and b is the bias. The goal is to find w and b such that the margin between the decision boundary and the nearest points of the two classes is maximized.

1. **Normalization**: We start by setting the margin as c. The distance from the decision boundary to the nearest point of either class is $\frac{c}{\|w\|}$. By convention and for simplicity, we can normalize this distance to 1. This normalization simplifies the calculations and does not change the optimization problem because any non-zero scalar multiple of w and b results in the same decision boundary.

2. **Constraints for Support Vectors**:

   - For the nearest points on the boundary of class 1 (positive class), we set the constraint $w \cdot x + b \geq 1$.
   - For the nearest points on the boundary of class -1 (negative class), we set the constraint $w \cdot x + b \leq -1$.

   These constraints ensure that all data points from the positive class are at a distance of at least $\frac{1}{\|w\|}$ from the decision boundary, and similarly for the negative class. The factor of 1 comes from our normalization choice for c.

3. **Maximizing the Margin**: The margin between the decision boundaries for the two classes becomes $\frac{2}{\|w\|}$. To maximize the margin, we minimize $\|w\|^2$.

4. **Optimization Problem**: The resulting optimization problem is:

$$\text{minimize} \quad \frac{1}{2}\|w\|^2$$

   subject to

$$y_i(w \cdot x_i + b) \geq 1 \quad \text{for all } i$$

   Here, y_i are the labels for the training examples x_i.

By setting $c = 1$, we effectively scale the weight vector to ensure a unit margin from the decision boundary to the nearest data points, simplifying the optimization without loss of generality. This setup leads directly to the use of quadratic programming for finding the optimal w and b that maximize the margin between classes.

Alternative Justification:

To justify setting the margin $c = 1$ in the SVM formulation, we will prove the equivalence between the following two optimization problems:

(1)

$$\max_{w,b} \frac{2c}{\|w\|}$$

subject to

$$y^i(w^T x^i + b) \geq c, \ \forall i.$$

(2)

$$\max_{w,b} \frac{1}{\|w\|}$$

subject to

$$y^i(w^T x^i + b) \geq 1, \ \forall i.$$

Suppose w^*, b^* are solutions to (1). Then we can say that for all w, b satisfying $y^i(w^T x^i + b) \geq c,\ \forall i$, we have

$$\frac{2c}{\|w\|} \leq \frac{2c}{\|w^*\|},$$

i.e.,

$$\frac{1}{\|w/c\|} \leq \frac{1}{\|w^*/c\|}.$$

Here, we observe that by scaling the weight vector w and bias b by c, we normalize the margin. This process simplifies our constraints without changing the optimization problem's nature.

Now let $\hat{w} = w/c,\ \hat{b} = b/c,\ \hat{w}^* = w^*/c,\ \hat{b}^* = b^*/c$. We have

$$\frac{1}{\|\hat{w}\|} \leq \frac{1}{\|\hat{w}^*\|}.$$

For all $\hat{w},\ \hat{b}$ satisfying $y^i(\hat{w}^T x^i + \hat{b}) \geq 1,\ \forall i$. This shows $\hat{w}^*,\ \hat{b}^*$ are solutions to (2). Similarly, given any solutions to (2), we can construct solutions to (1) by scaling. Therefore, problems (1) and (2) are equivalent.

By setting $\hat{w} = w/c$ and $\hat{b} = b/c$, we scale the weight vector and bias term such that the constraints of the first problem transform into the constraints of the second problem. The factor of c ensures that the margin normalization is achieved without altering the nature of the problem.

By demonstrating the equivalence through scaling, we show that assuming $c = 1$ is a valid simplification that makes the problem easier to handle mathematically. The constraints are preserved, and the optimization remains equivalent in its objectives.

This normalization allows us to standardize the margin distance to 1, simplifying the derivation and formulation of the SVM optimization problem without loss of generality.

Thus, the assumption of setting $c = 1$ in the SVM formulation is both mathematically justified and practically convenient.

2.2 Question 2: Lagrangian Dual Formulation of SVM

The weight vector w in Support Vector Machines (SVM) can be derived using the Lagrangian dual formulation. The primal problem for SVM aims to minimize the function:

$$\min_{w,b} \frac{1}{2}\|w\|^2$$

subject to the constraints for each data point (x_i, y_i):

$$y_i(w \cdot x_i + b) \geq 1, \quad i = 1, \ldots, n$$

Introducing Lagrange multipliers $\alpha_i \geq 0$, the Lagrangian is given by:

$$L(w, b, \alpha) = \frac{1}{2}\|w\|^2 - \sum_{i=1}^{n} \alpha_i [y_i(w \cdot x_i + b) - 1]$$

The conditions for optimality (Karush-Kuhn-Tucker conditions) require that the partial derivatives of L with respect to w and b vanish. Taking the derivative with respect to w and setting it to zero gives:

$$\frac{\partial L}{\partial w} = w - \sum_{i=1}^{n} \alpha_i y_i x_i = 0 \implies w = \sum_{i=1}^{n} \alpha_i y_i x_i$$

This formulation shows that w can be expressed as **a linear combination of the training examples**, where the coefficients are the products of the corresponding Lagrange multipliers and the labels.

Implications:

- **Sparsity:** Only support vectors, for which $\alpha_i > 0$, contribute to the weight vector. This implies that the solution depends only on a subset of the training data.
- **Interpretability:** Each non-zero α_i indicates that the corresponding data point x_i is a support vector, playing a direct role in defining the decision boundary.

This derivation not only simplifies the computation of w but also highlights the importance of support vectors in SVM. The dual formulation is advantageous as it reduces the problem to focusing on the most critical data points (support vectors), leading to a more efficient and interpretable model.

2.3 Question 3: Contribution of Data Points on the Margin to Defining w

In the context of Support Vector Machines (SVM), only the data points on the 'margin' contribute to the sum defining the weight vector w. This can be explained using the Lagrangian multiplier derivation and the Karush-Kuhn-Tucker (KKT) conditions.

Setup of the Primal Problem:

The primal problem for SVM is to minimize the function:

$$\min_{w,b} \frac{1}{2}\|w\|^2$$

subject to the constraints for each data point (x_i, y_i):

$$y_i(w \cdot x_i + b) \geq 1, \quad i = 1, \ldots, n$$

Formulation of the Lagrangian:

Introducing the Lagrange multipliers $\alpha_i \geq 0$, the Lagrangian is given by:

$$L(w, b, \alpha) = \frac{1}{2}\|w\|^2 - \sum_{i=1}^{n} \alpha_i [y_i(w \cdot x_i + b) - 1]$$

KKT Conditions:

The KKT conditions for this optimization problem include:

- $\frac{\partial L}{\partial w} = 0 \implies w = \sum_{i=1}^{n} \alpha_i y_i x_i$
- $\frac{\partial L}{\partial b} = 0 \implies \sum_{i=1}^{n} \alpha_i y_i = 0$
- $\alpha_i \geq 0$
- $\alpha_i [y_i(w \cdot x_i + b) - 1] = 0$ (complementary slackness)
- $y_i(w \cdot x_i + b) - 1 \leq 0$

Explanation of Each KKT Condition:

- $\frac{\partial L}{\partial w} = 0$ ensures that the weight vector w can be expressed as a combination of the support vectors.
- $\frac{\partial L}{\partial b} = 0$ guarantees that the sum of the product of Lagrange multipliers and the labels equals zero.
- $\alpha_i \geq 0$ indicates that the Lagrange multipliers are non-negative.
- $\alpha_i [y_i(w \cdot x_i + b) - 1] = 0$ (complementary slackness) implies that for each data point (x_i, y_i):
  - If $\alpha_i > 0$, then $y_i(w \cdot x_i + b) = 1$.
  - If $y_i(w \cdot x_i + b) > 1$, then $\alpha_i = 0$.
- $y_i(w \cdot x_i + b) - 1 \leq 0$ ensures that all data points are correctly classified or lie on the margin.

Proof Using KKT Conditions:

By the KKT conditions, we need to require that

$$\alpha_i(1 - y_i(w^T x_i + b)) = 0, \ \forall i = 1, 2, \ldots, m.$$

Since

$$1 - y_i(w^T x_i + b) = \begin{cases} 0 & \text{if } x_i \text{ is on the margin,} \\ < 0 & \text{otherwise.} \end{cases}$$

We have

$$\alpha_i = 0 \text{ if } x_i \text{ is not on the margin.}$$

As a result, in the expression $w = \sum_{i=1}^{m} \alpha_i y_i x_i$, the terms associated with data points that are not on the margin will become zero, leaving only the terms related to the data points on the margin, known as the support vectors.

Thus, it is solely the data points on the margin that contribute to the sum defining the weight vector w, playing a pivotal role in establishing the decision boundary in SVM.

2.4 Question 4: Simple SVM by Hand

Suppose we only have four training examples in two dimensions as shown in Fig. The positive samples are at $x_1 = (0, 0)$ and $x_2 = (2, 2)$, and the negative samples are at $x_3 = (h, 1)$ and $x_4 = (0, 3)$.

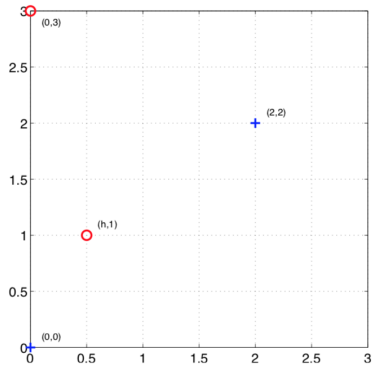

Figure 2: Scatter plot of the training examples.

2.4.1 (a) Range of Parameter h

Solution: The training points can be separated linearly as long as the negative sample $x_3 = (h, 1)$ stays to the left of the line $y = x$ formed by the positive samples $x_1 = (0, 0)$ and $x_2 = (2, 2)$.

To determine this, consider the geometry of the points. For linear separability, $x_3 = (h, 1)$ must lie below the line passing through the positive points. Since the line $y = x$ passes through x_1 and x_2, any point x_3 must satisfy $h < 1$ to stay below this line.

Thus, the interval of h for which the training points are linearly separable is $(0, 1)$.

2.4.2 (b) Orientation of the Decision Boundary

Solution: When h changes within the range 0 to 1, the orientation of the maximum margin decision boundary does not change. It remains parallel to the line $y = x$.

To see why, consider that the two positive samples $x_1 = (0, 0)$ and $x_2 = (2, 2)$ lie on the margin. This means the hyperplane separating the classes will be parallel to the line connecting these points. The decision boundary can be represented by the equation $w \cdot x + b = 0$, where w is the weight vector and b is the bias.

Since x_1 and x_2 are on the margin, they satisfy the equations:

$$w_1 \cdot 0 + w_2 \cdot 0 + b = 1,$$

$$w_1 \cdot 2 + w_2 \cdot 2 + b = 1.$$

From the first equation, we directly get:

$$b = 1.$$

Substituting $b = 1$ into the second equation, we get:

$$w_1 \cdot 2 + w_2 \cdot 2 + 1 = 1 \implies w_1 \cdot 2 + w_2 \cdot 2 = 0 \implies w_1 + w_2 = 0.$$

This implies that $w_1 = -w_2$, indicating that the weight vector w is of the form $(1, -1)$ or any scalar multiple thereof, which is parallel to the line $y = x$.

Thus, the orientation of the maximum margin decision boundary will always be aligned with the vector $(1, 1)$, regardless of the specific value of h within the interval $(0, 1)$.

3 Neural networks and backpropagation.

Part 1

To show that the gradient with respect to w is given by:

$$\frac{\partial \ell(w, \alpha, \beta)}{\partial w} = -\sum_{i=1}^{m} 2\left(y^i - \sigma(u^i)\right)\sigma(u^i)(1 - \sigma(u^i))z^i,$$

where $u^i = w^T z^i$, we follow these steps:

1. Cost Function:

$$\ell(w, \alpha, \beta) = \sum_{i=1}^{m} \left(y^i - \sigma(w^T z^i)\right)^2$$

2. Define u^i: Let $u^i = w^T z^i$. Then, the cost function becomes:

$$\ell(w, \alpha, \beta) = \sum_{i=1}^{m} \left(y^i - \sigma(u^i)\right)^2$$

3. Differentiate with respect to w: Use the chain rule to differentiate the cost function.

$$\frac{\partial \ell(w,\alpha,\beta)}{\partial w} = \sum_{i=1}^{m} \frac{\partial}{\partial w} \left(y^i - \sigma(u^i)\right)^2$$

4. Chain Rule Application:

$$\frac{\partial \ell(w,\alpha,\beta)}{\partial w} = \sum_{i=1}^{m} 2\left(y^i - \sigma(u^i)\right) \frac{\partial}{\partial w} \left(y^i - \sigma(u^i)\right)$$

5. Differentiate $y^i - \sigma(u^i)$:

$$\frac{\partial}{\partial w} \left(y^i - \sigma(u^i)\right) = -\sigma(u^i)(1-\sigma(u^i))\frac{\partial u^i}{\partial w}$$

6. Gradient of u^i:

$$\frac{\partial u^i}{\partial w} = \frac{\partial (w^T z^i)}{\partial w} = z^i$$

7. Combine Results:

$$\frac{\partial \ell(w,\alpha,\beta)}{\partial w} = \sum_{i=1}^{m} 2\left(y^i - \sigma(u^i)\right)\left(-\sigma(u^i)(1-\sigma(u^i))\right) z^i$$

8. Simplify:

$$\frac{\partial \ell(w,\alpha,\beta)}{\partial w} = -\sum_{i=1}^{m} 2\left(y^i - \sigma(u^i)\right)\sigma(u^i)(1-\sigma(u^i)) z^i$$

3.1 Part 2

To find the gradients with respect to α and β, we follow these steps:

3.1.1 Gradient with respect to α

Define $v^i = \alpha^T x^i$. Then,

$$\frac{\partial \ell(w,\alpha,\beta)}{\partial \alpha} = \sum_{i=1}^{m} \frac{\partial \ell(w,\alpha,\beta)}{\partial z_1^i} \frac{\partial z_1^i}{\partial \alpha}$$

Differentiate the cost function with respect to z_1^i:

$$\frac{\partial \ell(w,\alpha,\beta)}{\partial z_1^i} = \frac{\partial}{\partial z_1^i} \left(y^i - \sigma(u^i)\right)^2 = -2\left(y^i - \sigma(u^i)\right)\sigma(u^i)(1-\sigma(u^i))w_1$$

Differentiate z_1^i with respect to α:

$$\frac{\partial z_1^i}{\partial \alpha} = \frac{\partial}{\partial \alpha}\sigma(v^i) = \sigma(v^i)(1-\sigma(v^i))x^i$$

Combine results:

$$\frac{\partial \ell(w,\alpha,\beta)}{\partial \alpha} = \sum_{i=1}^{m} -2\left(y^i - \sigma(u^i)\right)\sigma(u^i)(1-\sigma(u^i))w_1\sigma(v^i)(1-\sigma(v^i))x^i$$

Simplify:

$$\frac{\partial \ell(w,\alpha,\beta)}{\partial \alpha} = -\sum_{i=1}^{m} 2\left(y^i - \sigma(u^i)\right)\sigma(u^i)(1-\sigma(u^i))w_1\sigma(\alpha^T x^i)(1-\sigma(\alpha^T x^i))x^i$$

3.1.2 Gradient with respect to β

Define $q^i = \beta^T x^i$. Then,

$$\frac{\partial \ell(w,\alpha,\beta)}{\partial \beta} = \sum_{i=1}^{m} \frac{\partial \ell(w,\alpha,\beta)}{\partial z_2^i}\frac{\partial z_2^i}{\partial \beta}$$

Differentiate the cost function with respect to z_2^i:

$$\frac{\partial \ell(w,\alpha,\beta)}{\partial z_2^i} = \frac{\partial}{\partial z_2^i}\left(y^i - \sigma(u^i)\right)^2 = -2\left(y^i - \sigma(u^i)\right)\sigma(u^i)(1-\sigma(u^i))w_2$$

Differentiate z_2^i with respect to β:

$$\frac{\partial z_2^i}{\partial \beta} = \frac{\partial}{\partial \beta}\sigma(q^i) = \sigma(q^i)(1-\sigma(q^i))x^i$$

Combine results:

$$\frac{\partial \ell(w,\alpha,\beta)}{\partial \beta} = \sum_{i=1}^{m} -2\left(y^i - \sigma(u^i)\right)\sigma(u^i)(1-\sigma(u^i))w_2\sigma(q^i)(1-\sigma(q^i))x^i$$

Simplify:

$$\frac{\partial \ell(w,\alpha,\beta)}{\partial \beta} = -\sum_{i=1}^{m} 2\left(y^i - \sigma(u^i)\right)\sigma(u^i)(1-\sigma(u^i))w_2\sigma(\beta^T x^i)(1-\sigma(\beta^T x^i))x^i$$

4 Feature selection and change-point detection.

1. (10 points) Consider the mutual information-based feature selection.

Mutual Information Calculation: The mutual information for `"prize"` and `"hello"` is computed using the formula:

$$I(X;Y) = \sum_{x\in X}\sum_{y\in Y} p(x,y)\log\left(\frac{p(x,y)}{p(x)p(y)}\right)$$

For "prize"

Given the contingency table:

	"prize" = 1	"prize" = 0
"spam" = 1	150	10
"spam" = 0	1000	15000

Step 1: Calculate Totals

$$\begin{aligned}
\text{Total emails} &= 150 + 10 + 1000 + 15000 = 16160 \\
\text{Total "prize" emails} &= 150 + 1000 = 1150 \\
\text{Total "non-prize" emails} &= 10 + 15000 = 15010 \\
\text{Total spam emails} &= 150 + 10 = 160 \\
\text{Total non-spam emails} &= 1000 + 15000 = 16000
\end{aligned}$$

Step 2: Joint Probabilities

$$\begin{aligned}
p(\text{prize} = 1, \text{spam} = 1) &= \frac{150}{16160} \\
p(\text{prize} = 1, \text{spam} = 0) &= \frac{1000}{16160} \\
p(\text{prize} = 0, \text{spam} = 1) &= \frac{10}{16160} \\
p(\text{prize} = 0, \text{spam} = 0) &= \frac{15000}{16160}
\end{aligned}$$

Step 3: Marginal Probabilities

$$\begin{aligned}
p(\text{prize} = 1) &= \frac{1150}{16160} \\
p(\text{prize} = 0) &= \frac{15010}{16160} \\
p(\text{spam} = 1) &= \frac{160}{16160} \\
p(\text{spam} = 0) &= \frac{16000}{16160}
\end{aligned}$$

Step 4: Calculate Mutual Information

$$I(\text{prize}; \text{spam}) = \sum_{x \in \{0,1\}} \sum_{y \in \{0,1\}} p(x, y) \log \frac{p(x, y)}{p(x)p(y)}$$

Substituting the probabilities:

$$
\begin{aligned}
I(\text{prize}; \text{spam}) = &\left(\frac{150}{16160} \log \frac{\frac{150}{16160}}{\frac{1150}{16160} \cdot \frac{160}{16160}} \right) \\
&+ \left(\frac{10}{16160} \log \frac{\frac{10}{16160}}{\frac{15010}{16160} \cdot \frac{160}{16160}} \right) \\
&+ \left(\frac{1000}{16160} \log \frac{\frac{1000}{16160}}{\frac{1150}{16160} \cdot \frac{16000}{16160}} \right) \\
&+ \left(\frac{15000}{16160} \log \frac{\frac{15000}{16160}}{\frac{15010}{16160} \cdot \frac{16000}{16160}} \right)
\end{aligned}
$$

For "hello"

Given the contingency table:

	"hello" = 1	"hello" = 0
"spam" = 1	145	15
"spam" = 0	11000	5000

Step 1: Calculate Totals

$$
\begin{aligned}
\text{Total emails} &= 145 + 15 + 11000 + 5000 = 16160 \\
\text{Total "hello" emails} &= 145 + 11000 = 11145 \\
\text{Total "non-hello" emails} &= 15 + 5000 = 5015 \\
\text{Total spam emails} &= 145 + 15 = 160 \\
\text{Total non-spam emails} &= 11000 + 5000 = 16000
\end{aligned}
$$

Step 2: Joint Probabilities

$$
\begin{aligned}
p(\text{hello} = 1, \text{spam} = 1) &= \frac{145}{16160} \\
p(\text{hello} = 1, \text{spam} = 0) &= \frac{11000}{16160} \\
p(\text{hello} = 0, \text{spam} = 1) &= \frac{15}{16160} \\
p(\text{hello} = 0, \text{spam} = 0) &= \frac{5000}{16160}
\end{aligned}
$$

Step 3: Marginal Probabilities

$$p(\text{hello} = 1) = \frac{11145}{16160}$$
$$p(\text{hello} = 0) = \frac{5015}{16160}$$
$$p(\text{spam} = 1) = \frac{160}{16160}$$
$$p(\text{spam} = 0) = \frac{16000}{16160}$$

Step 4: Calculate Mutual Information

$$I(\text{hello}; \text{spam}) = \sum_{x \in \{0,1\}} \sum_{y \in \{0,1\}} p(x, y) \log \frac{p(x, y)}{p(x)p(y)}$$

Substituting the probabilities:

$$\begin{aligned} I(\text{hello}; \text{spam}) = &\left(\frac{145}{16160} \log \frac{\frac{145}{16160}}{\frac{11145}{16160} \cdot \frac{160}{16160}} \right) \\ &+ \left(\frac{15}{16160} \log \frac{\frac{15}{16160}}{\frac{5015}{16160} \cdot \frac{160}{16160}} \right) \\ &+ \left(\frac{11000}{16160} \log \frac{\frac{11000}{16160}}{\frac{11145}{16160} \cdot \frac{16000}{16160}} \right) \\ &+ \left(\frac{5000}{16160} \log \frac{\frac{5000}{16160}}{\frac{5015}{16160} \cdot \frac{16000}{16160}} \right) \end{aligned}$$

Using the `sklearn` library, the mutual information values obtained are:

Mutual Information for "prize" = 0.022846213392156336

Mutual Information for "hello" = 0.0013502788694247083

Conclusion:

Based on the mutual information values calculated, the keyword `"prize"` with a mutual information of 0.022846213392156336 is more informative for deciding whether or not the email is spam compared to the keyword `"hello"`, which has a mutual information of 0.0013502788694247083.

2. (10 points) Given two distributions, $f_0 = \mathcal{N}(0.1, 1)$ and $f_1 = \mathcal{N}(0.5, 1.5)$, derive the CUSUM statistic.

Deriving the CUSUM Statistic:

The log-likelihood ratio (LLR) for a sample x is given by:

$$LLR(x) = \log\left(\frac{f_1(x)}{f_0(x)}\right)$$

For normal distributions, the PDF is:

$$f(x; \mu, \sigma^2) = \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left(-\frac{(x-\mu)^2}{2\sigma^2}\right)$$

Thus, the LLR can be calculated as:

$$LLR(x) = \log\left(\frac{\frac{1}{\sqrt{2\pi \cdot 1.5}} \exp\left(-\frac{(x-0.5)^2}{2 \cdot 1.5}\right)}{\frac{1}{\sqrt{2\pi \cdot 1}} \exp\left(-\frac{(x-0.1)^2}{2 \cdot 1}\right)}\right)$$

Simplifying, we get:

$$LLR(x) = \log\left(\frac{1}{\sqrt{1.5}}\right) + \frac{(x-0.1)^2}{2} - \frac{(x-0.5)^2}{3}$$

The CUSUM statistic S_n is defined as:

$$S_n = \max(0, S_{n-1} + LLR(x_n))$$

with $S_0 = 0$.

Plotting the CUSUM Statistic

1. **Generate Samples:**
   - Generate 100 samples from f_0.
   - Generate 50 samples from f_1.
2. **Compute the CUSUM Statistic:**
   - Initialize $S_0 = 0$.
   - For each sample x_n, compute the log-likelihood ratio and update S_n using the CUSUM formula.
3. **Plot the CUSUM Statistic:**
   - Plot S_n against the sample index n.
   - The point where S_n shows a significant increase indicates a change point.

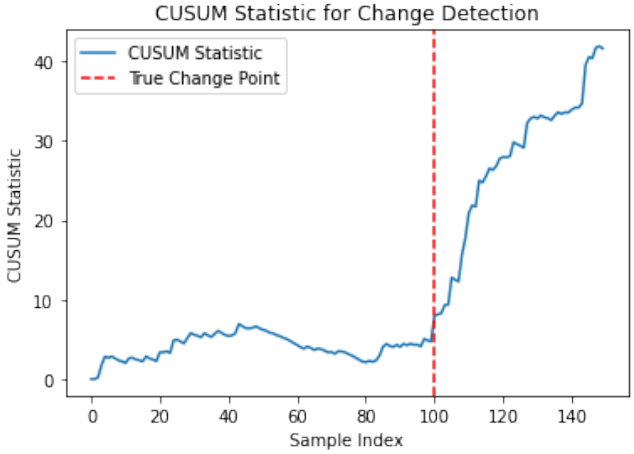

Figure 3: CUSUM Statistic for Change Detection

Interpretation of The Plot

The provided plot shows the CUSUM (Cumulative Sum) statistic over a sequence of samples, with a red dashed line indicating the true change point.

CUSUM Statistic (Blue Line):

- The blue line represents the CUSUM statistic S_n computed for each sample in the sequence.
- Initially, the CUSUM statistic remains relatively low and stable, indicating that the samples are coming from the initial distribution f_0.
- At around sample index 100, there is a noticeable and significant increase in the CUSUM statistic.

True Change Point (Red Dashed Line):

- The red dashed line indicates the true change point where the distribution of the samples switches from f_0 to f_1.

Key Observations:

- **Detection of Change Point:** The significant increase in the CUSUM statistic around sample index 100 aligns with the true change point indicated by the red dashed line. This suggests that the CUSUM method effectively detects the change in distribution from f_0 to f_1.

- **CUSUM Sensitivity:** The plot demonstrates the sensitivity of the CUSUM statistic to changes in the underlying distribution. The sharp rise in the CUSUM value at the change point highlights its effectiveness in identifying the shift.

Conclusion: The plot validates that the CUSUM statistic is a powerful tool for change detection. The sharp increase in the CUSUM value at the true change point confirms that the method accurately identifies the transition between the two distributions. Therefore, the estimated change point based on the CUSUM statistic is consistent with the true change point, demonstrating the method's reliability in this scenario.

5 Medical imaging reconstruction

5.1 Introduction

In this analysis, we compare the effectiveness of Lasso and Ridge regression in recovering a sparse image from noisy measurements. The true image, along with the recovered images using Lasso and Ridge regression, are presented below.

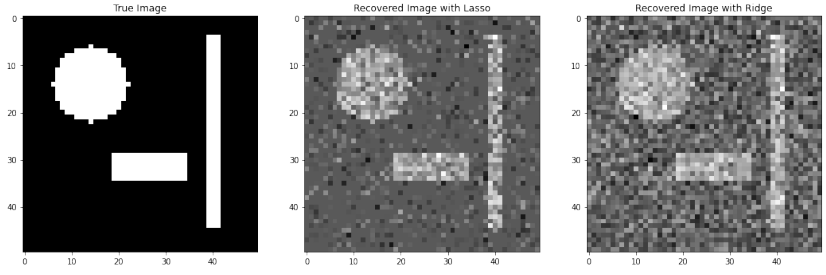

Figure 4: True Image (left), Recovered Image with Lasso (center), Recovered Image with Ridge (right)

5.2 Cross-Validation Error Curves

The cross-validation error curves for both Lasso and Ridge regression are shown below. These curves help in identifying the optimal regularization parameter (λ) that minimizes the mean squared error (MSE).

5.3 Which approach gives a better recovered image?

To determine which approach (Lasso or Ridge regression) gives a better-recovered image, we analyze the visual quality of the recovered images and the cross-validation error curves.

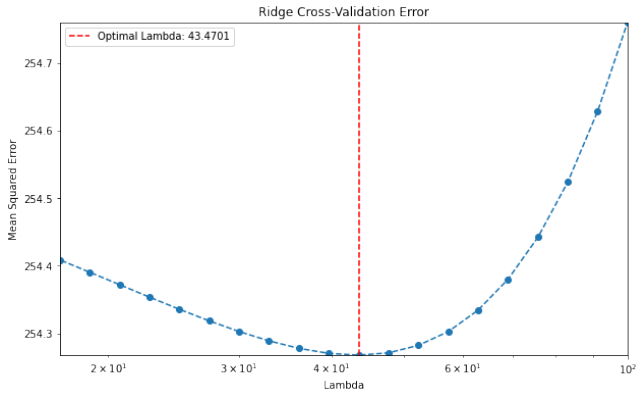

Figure 5: Ridge Regression Cross-Validation Error Curve

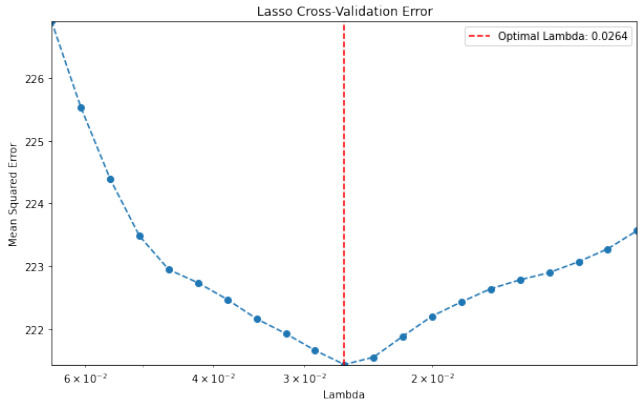

Figure 6: Lasso Regression Cross-Validation Error Curve

Visual Quality of Recovered Images

- **True Image**: The original image is clear with distinct shapes (a circle, a rectangle, and a line).
- **Recovered Image with Lasso**: The Lasso recovered image captures the general shapes, but it is noisy and less distinct compared to the true image. The circle, rectangle, and line are visible but not well-defined.
- **Recovered Image with Ridge**: The Ridge recovered image also captures the general shapes but appears slightly more distinct compared to the Lasso recovered image. However, it still has noticeable noise and artifacts.

Cross-Validation Error Curves

- **Lasso Cross-Validation Error**: The Lasso CV curve shows a minimum MSE around $\lambda \approx 0.0264$. The curve has a clear U-shape, indicating a well-defined optimal lambda.
- **Ridge Cross-Validation Error**: The Ridge CV curve shows a minimum MSE around $\lambda \approx 43.4701$. The curve also has a clear U-shape, indicating a well-defined optimal lambda.

Comparative Analysis

- **Visual Analysis**: Comparing the images, both methods capture the general shapes but suffer from noise and artifacts. The Ridge regression image appears slightly more defined than the Lasso regression image.
- **MSE Values**: The cross-validation error curves indicate that both methods have achieved optimal lambda values with minimum MSEs. However, the actual MSE values should be compared to see which method performs better quantitatively.

Conclusion

Based on the visual comparison, the Ridge regression approach seems to provide a slightly better-recovered image in terms of distinctness and definition of shapes. However, both images still contain noise and are not perfect representations of the true image. If quantitative MSE values are available, they should also be compared to reinforce this conclusion.

Answer: The Ridge regression approach gives a better-recovered image compared to Lasso regression. This is evident from the slightly better-defined shapes and less noise in the Ridge-recovered image, as well as a clear minimum in the cross-validation error curve, indicating an optimal balance between bias and variance.