

CSE 5819 Assignment #6

by: Aayushi Verma (uef24001)

This assignment is organized as follows:

1. ChatGPT - pg 2–5
2. Problems - pg 6–9
3. Coding - attached as a separate .ipynb file

1 Part 1: ChatGPT Self-Learning (20pts)

You can start from the following prompts, but you need to create subsequent questions that attempt to understand the basic concepts of machine learning.

1.1 What is the most used evaluation metric for classification?

The most commonly used evaluation metrics for classification include:

- **Accuracy:** The ratio of correctly predicted instances to the total instances.
- **Precision:** The ratio of true positive predictions to the sum of true and false positives.
- **Recall:** The ratio of true positive predictions to the sum of true positives and false negatives.
- **F1-score:** The harmonic mean of precision and recall, providing a balance between the two.
- **ROC-AUC:** The Area Under the Receiver Operating Characteristic Curve, measuring the trade-off between true positive rate and false positive rate.

1.2 What is the most used evaluation metric for regression?

The most commonly used evaluation metrics for regression include:

- **Mean Squared Error (MSE):** The average of the squared differences between predicted and actual values.
- **Root Mean Squared Error (RMSE):** The square root of the MSE, providing an interpretable error metric in the same units as the target variable.
- **Mean Absolute Error (MAE):** The average of the absolute differences between predicted and actual values.
- **R-squared (Coefficient of Determination):** A statistical measure indicating how well the regression predictions approximate the real data points.
- **Mean Absolute Percentage Error (MAPE):** The average percentage difference between predicted and actual values.

1.3 For supervised learning algorithms, what would be a reliable procedure of evaluating their performance?

A reliable procedure for evaluating the performance of supervised learning algorithms includes the following steps:

- **Train-Test Split:** Divide the dataset into two parts: a training set (used for model training) and a test set (used for final performance evaluation). A common split ratio is 80% training and 20% testing.
- **Cross-Validation:** Perform k -fold cross-validation (e.g., $k = 5$ or $k = 10$), where the dataset is split into k subsets. The model is trained on $k - 1$ subsets and tested on the remaining one. This process is repeated k times, and the results are averaged.
- **Evaluation Metrics:** Select appropriate evaluation metrics based on the type of task (e.g., accuracy, precision, recall, F1-score, ROC-AUC for classification, and MSE, MAE, R-squared for regression).
- **Hyperparameter Tuning:** Use techniques such as grid search or random search with cross-validation to fine-tune the model's hyperparameters for optimal performance.
- **Handling Class Imbalance:** If dealing with imbalanced datasets, use techniques like stratified sampling in cross-validation, resampling methods (oversampling/undersampling), or adjust performance metrics (e.g., weighted F1-score).
- **Test Set Evaluation:** After training and tuning the model, evaluate its performance on the previously unseen test set to get an unbiased estimate of its real-world performance.

1.4 Study bagging (bootstrapping aggregation)?

Bagging, or Bootstrap Aggregating, is an ensemble learning technique that improves the stability and accuracy of machine learning algorithms. The main ideas behind bagging are:

- **Bootstrapping:** Create multiple subsets of the training data by randomly sampling with replacement. Each subset has the same size as the original dataset but may include duplicate instances while omitting others.
- **Model Training:** Train a separate model on each of these bootstrapped subsets. The models can be of the same type (e.g., decision trees) or different types.
- **Aggregation:** Combine the predictions of all the models to produce a final output. For regression tasks, the average of the predictions is taken, while for classification tasks, a majority vote is used to determine the final class label.
- **Reduction of Variance:** By averaging the predictions of multiple models, bagging reduces overfitting and variance, leading to more robust and generalizable predictions.
- **Popular Algorithms:** One of the most well-known implementations of bagging is the Random Forest algorithm, which combines multiple decision trees trained on bootstrapped subsets of the data.

1.5 Why can the bagging procedure create models of lower variance?

The bagging procedure can create models of lower variance due to several key factors:

- **Ensemble of Models:** Bagging combines multiple models trained on different subsets of the data. Each model captures different patterns and noise in the data, which helps to smooth out individual model fluctuations.
- **Random Sampling with Replacement:** The bootstrapping process involves sampling with replacement, meaning some data points may be duplicated while others may be omitted. This results in different training sets for each model, leading to diverse models that are less likely to make the same errors.
- **Reduction of Overfitting:** Individual models, especially complex ones like decision trees, can overfit to the training data by capturing noise. By averaging the predictions of multiple models, bagging reduces the influence of any one model that may have overfitted, thereby decreasing overall model variance.
- **Majority Voting (Classification) or Averaging (Regression):** The aggregation step in bagging helps to average out the predictions, which reduces the impact of outliers and erratic predictions from individual models. This averaging effect leads to more stable and consistent final predictions.
- **Increased Robustness:** By leveraging the strengths of multiple models, bagging can improve generalization performance, making the ensemble more robust to variations in the data compared to single models.

1.6 What are the advantages of Support Vector Machine?

Support Vector Machines (SVM) offer several advantages, making them popular for various classification and regression tasks:

- **Effective in High-Dimensional Spaces:** SVMs are particularly effective in high-dimensional spaces, making them suitable for tasks involving large feature sets, such as text classification and image recognition.
- **Robust to Overfitting:** By maximizing the margin between classes, SVMs are less prone to overfitting, especially in high-dimensional datasets. The regularization parameter can further help control the trade-off between maximizing the margin and minimizing classification errors.
- **Versatile Kernel Trick:** SVMs can employ various kernel functions (e.g., linear, polynomial, radial basis function) to handle non-linear decision boundaries. This flexibility allows SVMs to adapt to complex data distributions.
- **Works Well with Smaller Datasets:** SVMs can perform well with smaller datasets, as they focus on the support vectors that define the decision boundary, rather than the entire dataset.
- **Clear Margin of Separation:** SVMs create a clear margin of separation between classes, which can lead to better generalization in the presence of noise in the data.

- **Binary and Multi-Class Classification:** Although inherently a binary classifier, SVMs can be extended to handle multi-class classification problems using strategies like one-vs-one or one-vs-all.
- **Robust to Outliers:** The decision boundary created by SVMs can be relatively unaffected by outliers, as the algorithm focuses on the support vectors that lie closest to the decision boundary.

1.7 What is Support Vector Regression?

Support Vector Regression (SVR) is an extension of Support Vector Machines (SVM) that is used for regression problems. SVR aims to find a function that approximates the relationship between the input features and the continuous target variable. The key concepts of SVR include:

- **Epsilon-Insensitive Loss Function:** SVR uses an epsilon-insensitive loss function, meaning that errors within a specified margin (epsilon) are not penalized. This allows the model to ignore small deviations between the predicted and actual values, providing a level of robustness to noise in the data.
- **Support Vectors:** Similar to SVM, SVR identifies support vectors that define the optimal regression function. Only the data points that lie outside the epsilon margin contribute to the model, focusing on the most relevant information for regression.
- **Margin of Tolerance:** The goal of SVR is to find a function that lies within the epsilon margin of the true values, ensuring that the predictions are as accurate as possible while maintaining a level of tolerance to errors.
- **Kernel Functions:** SVR can also utilize kernel functions, allowing it to handle non-linear relationships between the input features and the target variable. Common kernels include linear, polynomial, and radial basis function (RBF) kernels.
- **Regularization Parameter (C):** Similar to SVM, SVR has a regularization parameter C that controls the trade-off between achieving a low error on the training data and maintaining a smooth regression function. A smaller C results in a wider margin but allows more errors, while a larger C seeks to minimize errors at the cost of model complexity.
- **Applications:** SVR is widely used in various fields such as finance, bioinformatics, and engineering, where predicting continuous values is essential.

1.8 How many different formulations of SVM are there?

Support Vector Machines (SVM) have several formulations, each suited for different types of problems and data distributions. The main formulations include:

- **Hard Margin SVM:** This formulation assumes that the data is linearly separable without any noise. It aims to maximize the margin between the two classes while ensuring that all training samples are correctly classified. This approach is not robust to outliers.
- **Soft Margin SVM:** This formulation introduces a soft margin that allows for some misclassifications. It includes a regularization parameter C that controls the trade-off between maximizing the margin and minimizing classification errors. Soft Margin SVM is more robust to outliers and noise in the data.
- **SVM for Regression (Support Vector Regression, SVR):** This formulation adapts SVM for regression tasks. It aims to find a function that approximates the relationship between input features and continuous target variables, using an epsilon-insensitive loss function.
- **Multi-class SVM:** Although SVM is inherently a binary classifier, it can be extended to multi-class problems using techniques like one-vs-one (OvO) or one-vs-all (OvA). In OvO, a separate SVM is trained for every pair of classes, while in OvA, a separate SVM is trained for each class against all other classes.
- **Kernelized SVM:** This formulation leverages kernel functions to transform the input space, allowing SVM to find non-linear decision boundaries. Common kernels include linear, polynomial, radial basis function (RBF), and sigmoid.
- **Nu-SVM:** This is a variant of Soft Margin SVM that uses a parameter ν to control the number of support vectors and margin errors, providing additional flexibility in model training.

1.9 How to solve an SVM optimization problem?

Solving an SVM optimization problem involves finding the optimal hyperplane that maximizes the margin between different classes. The procedure can be broken down into the following steps:

- **Formulate the Optimization Problem:** For a binary classification problem with N training samples (\mathbf{x}_i, y_i) , where $y_i \in \{-1, 1\}$, the optimization problem can be formulated as:

$$\min_{\mathbf{w}, b} \quad \frac{1}{2} \|\mathbf{w}\|^2 \quad (1)$$

$$\text{subject to} \quad y_i(\mathbf{w} \cdot \mathbf{x}_i + b) \geq 1, \quad \forall i = 1, \dots, N \quad (2)$$

Here, \mathbf{w} is the weight vector, b is the bias term, and the constraints ensure that the samples are correctly classified with a margin of at least 1.

- **Introduce Slack Variables (Soft Margin SVM):** For non-linearly separable data, introduce slack variables ξ_i to allow for misclassifications:

$$\min_{\mathbf{w}, b, \xi} \quad \frac{1}{2} \|\mathbf{w}\|^2 + C \sum_{i=1}^N \xi_i \quad (3)$$

$$\text{subject to} \quad y_i(\mathbf{w} \cdot \mathbf{x}_i + b) \geq 1 - \xi_i, \quad \forall i = 1, \dots, N \quad (4)$$

$$\xi_i \geq 0 \quad (5)$$

The parameter C controls the trade-off between maximizing the margin and minimizing the classification errors.

- **Transform to the Dual Problem:** The primal problem can be transformed into a dual optimization problem using Lagrange multipliers α_i :

$$\max_{\alpha} \quad \sum_{i=1}^N \alpha_i - \frac{1}{2} \sum_{i=1}^N \sum_{j=1}^N \alpha_i \alpha_j y_i y_j \mathbf{x}_i \cdot \mathbf{x}_j \quad (6)$$

$$\text{subject to} \quad \sum_{i=1}^N \alpha_i y_i = 0 \quad (7)$$

$$0 \leq \alpha_i \leq C, \quad \forall i = 1, \dots, N \quad (8)$$

- **Solve the Dual Problem:** Use optimization techniques such as Sequential Minimal Optimization (SMO) or gradient descent methods to solve the dual problem for α_i .
- **Calculate the Weight Vector and Bias:** After obtaining the optimal α_i , the weight vector \mathbf{w} can be calculated as:

$$\mathbf{w} = \sum_{i=1}^N \alpha_i y_i \mathbf{x}_i$$

The bias term b can be computed using any support vector:

$$b = y_k - \mathbf{w} \cdot \mathbf{x}_k \quad \text{for any } k \text{ where } \alpha_k > 0$$

- **Make Predictions:** Finally, to make predictions for a new sample \mathbf{x} , use the decision function:

$$f(\mathbf{x}) = \mathbf{w} \cdot \mathbf{x} + b$$

The predicted class label is determined by the sign of $f(\mathbf{x})$.

2 Problems

2.1 Question One

The method called “bagging” was not discussed in class lecture. “Bagging” is a short name of Bootstrap Aggregating. We would like you to self-study this concept via ChatGPT or any other online materials you trust. Bagging is not a model evaluation method but a model training method.

(a) Please describe the procedure of Bagging.

Bagging, short for Bootstrap Aggregating, is an ensemble learning technique designed to improve the stability and accuracy of machine learning algorithms. The procedure of bagging can be described in the following steps:

- **Bootstrap Sampling:** Create multiple subsets of the original training dataset by randomly sampling with replacement. Each subset, known as a bootstrap sample, may contain duplicate instances and will typically be of the same size as the original dataset.
- **Train Base Learners:** For each bootstrap sample, train a separate base learner (e.g., decision tree, neural network). Each model is trained independently, and the randomness introduced by bootstrap sampling ensures that the models capture different patterns in the data.
- **Aggregate Predictions:** Once all base learners are trained, aggregate their predictions to make a final prediction. For classification tasks, the most common aggregation method is majority voting, where the class predicted by the majority of the models is chosen. For regression tasks, the predictions are typically averaged.
- **Final Model Output:** The output of the bagging procedure is a single ensemble model that combines the strengths of all the individual base learners, often leading to improved performance and reduced variance compared to any single model.
- **Evaluation and Tuning:** After constructing the bagging ensemble, evaluate its performance on a validation dataset. Hyperparameters, such as the number of base learners or the type of base learner, can be tuned to optimize performance further.

(b) Why Bagging generally reduce model variance?

Bagging (Bootstrap Aggregating) generally reduces model variance due to the following reasons:

- **Independence of Base Learners:** By training multiple models on different bootstrap samples of the data, bagging introduces diversity among the base learners. Each model captures different aspects of the data, which helps to smooth out the predictions when aggregated.
- **Averaging Effect:** In regression tasks, the predictions from the individual models are averaged, while in classification tasks, majority voting is used. This aggregation process tends to cancel out the errors made by individual models. If one model makes a high variance error due to a particular subset of data, other models are less likely to make the same mistake, leading to a more stable overall prediction.
- **Reduction of Overfitting:** Models with high variance often overfit the training data, capturing noise and specific patterns that do not generalize well to new data. Bagging mitigates this effect by combining the outputs of multiple models, each trained on slightly different data, which can lead to a more generalized model that performs better on unseen data.
- **Error Reduction through Diversity:** When base learners are trained on different samples, they are likely to make different errors. The aggregation of these diverse predictions helps to reduce the overall error, resulting in lower variance in the ensemble model compared to individual models.
- **Robustness to Outliers:** Bagging is more robust to outliers since the influence of any single data point is diminished across the ensemble. As each model sees a different subset of the data, the impact of outliers on the final prediction is reduced.

2.2 Support Vector Machine

(a) [10 pts] The following is the primal formulation of L2 SVM (with the squared slack variables), which is a variant of the standard SVM that we discussed in our lecture.

$$\begin{aligned}
\min_{w,b,\xi} \quad & \frac{1}{2} \mathbf{w}^T \mathbf{w} + \frac{C}{2} \sum_{i=1}^N \xi_i^2 \\
\text{s.t.} \quad & y_i(\mathbf{w}^T \mathbf{x}_i + b) \geq 1 - \xi_i, \quad i \in \{1, \dots, N\}, \\
& \xi_i \geq 0, \quad i \in \{1, \dots, N\}
\end{aligned} \tag{9}$$

If we remove the last constraints ($\xi_i \geq 0$), we might get a simpler optimization problem:

$$\begin{aligned}
\min_{w,b,\xi} \quad & \frac{1}{2} \mathbf{w}^T \mathbf{w} + \frac{C}{2} \sum_{i=1}^N \xi_i^2 \\
\text{s.t.} \quad & y_i(\mathbf{w}^T \mathbf{x}_i + b) \geq 1 - \xi_i, \quad i \in \{1, \dots, N\}.
\end{aligned} \tag{10}$$

Please provide the Lagrangian of the above simplified formulation.

Given the simplified formulation, we can write the Lagrangian as:

$$\mathcal{L}(w, b, \xi, \alpha) = \frac{1}{2} \mathbf{w}^T \mathbf{w} + \frac{C}{2} \sum_{i=1}^N \xi_i^2 - \sum_{i=1}^N \alpha_i (y_i(\mathbf{w}^T \mathbf{x}_i + b) - (1 - \xi_i)) \tag{11}$$

where $\alpha_i \geq 0$ are the Lagrange multipliers.

(b) [15 pts] Please find the partial derivative of the Lagrangian in (a) with respect to w , b , and ξ_i .

We need to take the derivative of the Lagrangian (Eq. 11) with respect to each variable: \mathbf{w} , b , and ξ_i :

$$\begin{aligned}
\frac{\partial \mathcal{L}}{\partial \mathbf{w}} &= \frac{\partial}{\partial \mathbf{w}} \left(\frac{1}{2} \mathbf{w}^T \mathbf{w} + \frac{C}{2} \sum_{i=1}^N \xi_i^2 - \sum_{i=1}^N \alpha_i (y_i(\mathbf{w}^T \mathbf{x}_i + b) - (1 - \xi_i)) \right) \\
&= \mathbf{w} - \sum_{i=1}^N \alpha_i y_i \mathbf{x}_i
\end{aligned} \tag{12}$$

$$\begin{aligned}
\frac{\partial \mathcal{L}}{\partial b} &= \frac{\partial}{\partial b} \left(\frac{1}{2} \mathbf{w}^T \mathbf{w} + \frac{C}{2} \sum_{i=1}^N \xi_i^2 - \sum_{i=1}^N \alpha_i (y_i(\mathbf{w}^T \mathbf{x}_i + b) - (1 - \xi_i)) \right) \\
&= - \sum_{i=1}^N \alpha_i y_i
\end{aligned} \tag{13}$$

$$\begin{aligned}
\frac{\partial \mathcal{L}}{\partial \xi_i} &= \frac{\partial}{\partial \xi_i} \left(\frac{1}{2} \mathbf{w}^T \mathbf{w} + \frac{C}{2} \sum_{i=1}^N \xi_i^2 - \sum_{i=1}^N \alpha_i (y_i(\mathbf{w}^T \mathbf{x}_i + b) - (1 - \xi_i)) \right) \\
&= C \xi_i - \alpha_i
\end{aligned} \tag{14}$$

2.3 Cluster Analysis

[25 pts] Consider the following set of one-dimensional data points: $\{0.1, 0.25, 0.45, 0.55, 0.8, 0.9\}$. All the points are located in the range between $[0, 1]$.

(a) [15 pts] Suppose we apply K-Means clustering to obtain three clusters, A, B, and C. If the initial centroids are located at $\{0, 0.4, 1\}$, respectively, show the cluster assignments and locations of the updated centroids after the first three iterations by filling out the following table. (Hint: although you can directly calculate all distances, if you draw a real axis and put down the points, it might be easier to see the cluster assignment.)

Table 1 shows the final results from this exercise. My working is below.

For the first iteration, we start with our initial cluster centroids $C_0 = \{0, 0.4, 1\}$, and calculate the Euclidean distance between each point to each centroid. This is shown in Table 2.

Iter	Cluster Assignment of Data Points						Centroid Location		
	0.1	0.25	0.45	0.55	0.8	0.9	A	B	C
1	A	B	B	B	C	C	0.1	0.116	0.15
2	A	A	B	B	C	C	0.175	0.5	0.85
3	A	A	B	B	C	C	0.175	0.5	0.85

Table 1: Final results of this question.

	0.1	0.25	0.45	0.55	0.8	0.9
$A_0 = 0$	0.1	0.25	0.45	0.55	0.8	0.9
$B_0 = 0.4$	0.3	0.15	0.05	0.15	0.4	0.5
$C_0 = 1$	0.9	0.75	0.55	0.45	0.2	0.1

Table 2: Euclidean distances between points and centroid C_0 locations. The determined cluster for each point, based on shortest distance, is shown in blue.

We then determine the new centroid locations, based on the mean of the points within each cluster (Eq. 15), to obtain the new centroid locations.

$$\begin{aligned}
 A_1 &= \frac{0.1}{1} = 0.1 \\
 B_1 &= \frac{0.25 + 0.45 + 0.55}{3} = 0.416 \\
 C_1 &= \frac{0.8 + 0.9}{2} = 0.85
 \end{aligned} \tag{15}$$

For the second iteration, we start with our cluster centroids $C_1 = \{0.1, 0.416, 0.85\}$, and calculate the Euclidean distance between each point to each centroid. This is shown in Table 3.

	0.1	0.25	0.45	0.55	0.8	0.9
$A_1 = 0.1$	0	0.15	0.35	0.45	0.7	0.8
$B_1 = 0.416$	0.316	0.166	0.034	0.134	0.384	0.484
$C_1 = 0.85$	0.75	0.6	0.4	0.3	0.05	0.05

Table 3: Euclidean distances between points and centroid C_1 locations. The determined cluster for each point, based on shortest distance, is shown in blue.

We then determine the new centroid locations, based on the mean of the points within each cluster (Eq. 16), to obtain the new centroid locations.

$$\begin{aligned}
 A_2 &= \frac{0.1 + 0.25}{1} = 0.175 \\
 B_2 &= \frac{0.45 + 0.55}{2} = 0.5 \\
 C_2 &= \frac{0.8 + 0.9}{2} = 0.85
 \end{aligned} \tag{16}$$

Finally, for the third iteration, we start with our cluster centroids $C_2 = \{0.175, 0.5, 0.85\}$, and calculate the Euclidean distance between each point to each centroid. This is shown in Table 4.

We then determine the final centroid locations, based on the mean of the points within each cluster (Eq. 17), to obtain the new centroid locations.

$$\begin{aligned}
 A_3 &= \frac{0.1 + 0.25}{1} = 0.175 \\
 B_3 &= \frac{0.45 + 0.55}{2} = 0.5 \\
 C_3 &= \frac{0.8 + 0.9}{2} = 0.85
 \end{aligned} \tag{17}$$

(b) [5 pts] Find the sum-of-squared errors (SSE) of the clustering after the third iteration. To calculate the SSE, we need to compute the squared distances of each data point from its respective cluster centroid.

For each cluster, the SSE after the third iteration is given by:

	0.1	0.25	0.45	0.55	0.8	0.9
$A_2 = 0.175$	0.075	0.075	0.275	0.375	0.625	0.725
$B_2 = 0.5$	0.4	0.25	0.05	0.05	0.3	0.4
$C_2 = 0.85$	0.75	0.6	0.4	0.3	0.05	0.05

Table 4: Euclidean distances between points and centroid C_2 locations. The determined cluster for each point, based on shortest distance, is shown in blue.

$$\begin{aligned}
SSE_A &= (0.1 - 0.175)^2 + (0.25 - 0.175)^2 = 0.01125 \\
SSE_B &= (0.45 - 0.5)^2 + (0.55 - 0.5)^2 = 0.005 \\
SSE_C &= (0.8 - 0.85)^2 + (0.9 - 0.85)^2 = 0.005
\end{aligned} \tag{18}$$

Then the total SSE is given by:

$$\begin{aligned}
SSE &= SSE_A + SSE_B + SSE_C \\
&= 0.01125 + 0.005 + 0.005 \\
&= 0.02125
\end{aligned} \tag{19}$$

(c) [5 pts] For the dataset given in part (1), is it possible to obtain empty clusters? Why?

It is possible to obtain empty clusters, either because of poor random initialization, or because of the selected distance metric. For example, if the initial centroid locations were too far from the first few points, then it is possible the distance would have been too great in the first few iterations to be assigned to cluster A. Similarly, if we used some other distance metric, e.g. cosine distance, then that could have affected how each point is assigned to each data cluster.