**HO CHI MINH UNIVERSITY OF TECHNOLOGY**

**OFFICE FOR INTERNATIONAL STUDY PROGRAMS**

🙞∙∙∙☼∙∙∙🙜



**FACULTY OF MECHANICAL ENGINEERING**

**APPLICATIONS OF ARTIFICIAL INTELLIEGENCE PROJECT**

PROBLEM SET 1

CLASS CC01

STUDENT: NGUYỄN ĐỨC KHẢI-2152661

INSTRUCTOR: PHD PHÙNG THANH HUY

**Hồ Chí Minh City – 2023**

**Problem 1. Linear regression**

Let denote the size of observations is

We have the following original linear regression model

Let represent the mean value of (denoted as ) over all observations as:

Then

The term is a constant since it involves the sum of products of constants and means of the independent variables. Let's denote this sum as C:

Now we can replace this term in our equation:

Notice that is also a constant. Since the equation must hold for all i, let's consider the average across all for the left-hand side of the equation:

Now let's apply the same logic to the constant terms on the right-hand side:

On the other hand:

So

Therefore

In conclusion, the original linear regression model can be represented as:

**Problem 2. PCA**

**1. Calculate mean value of the whole dataset regardless of the classes**

Class 1 has 11 values and Class 2 has 14 values, we can calculate the overall mean (u) by taking the weighted sum of the means:

**2. Calculate the eigen value and eigen vectors of E**

To find the eigenvalues of E, we need to solve the characteristic equation:

Where is the identity matrix and represents the eigen values

These are the eigen values

* With

To find the null space, solve the matrix equation

Take , then

Therefore, we have eigen vector

* With

To find the null space, solve the matrix equation

Take , then

Therefore, we have eigen vector

**3. Project the dataset to the principle component 1 () and write the criterion function**

Based on the previous answer, we calculated two eigenvalues:

The corresponding eigenvector for the largest eigenvalue (which represents ) was determined by:

To project the data points onto , we calculate the dot product of each data point with the eigenvector . If we represent our dataset as a matrix where each row is a data point, the projection of the dataset onto is given by:

The criterion function, which measures the quality of the projection, is typically the variance of the projected data. The variance along is equal to the corresponding eigenvalue , which, in this case, is equal .

If we want to express the criterion function in terms of the original data matrix and the eigenvector , we can calculate the variance of the projected data as follows:

Where:

- is the criterion function for

- is the projection of the ( i )-th data point onto

- is the mean of the projected data points.

- is the number of data points.

However, since is the direction of maximum variance, is maximized and is equal to the largest eigenvalue of the covariance matrix . Therefore, in this context, the criterion function is simply:

**4. Represent PC1 on the figure and project the data to (calculation is not necessary**)

A graph of numbers and points

Description automatically generated

**Problem 3. Gaussian Naive Bayes**

**1. Represent the data on 2D figure**

A graph with numbers and points

Description automatically generated

**2. Use gaussian naive bayes to classify the dataset**

To classify the dataset using Gaussian Naive Bayes problem, we'll follow these steps:

- Calculate the mean and variance for each feature for both classes.

- Use the Gaussian probability density function to estimate the likelihood of a given feature value belonging to each class.

- Apply Bayes' theorem to calculate the posterior probability for each class.

- Classify a new data point by choosing the class with the highest posterior probability.

For class (-1):

- Mean of feature 1:

- Variance of feature 1:

- Mean of feature 2:

- Variance of feature 2:

For class (+1):

- Mean of feature 1:

- Variance of feature 1:

- Mean of feature 2:

- Variance of feature 2:

Now, we have the mean and variance for each feature in both classes. To classify a new data point, we would calculate the likelihood of the point belonging to each class using the Gaussian probability density function for each feature, then multiply these likelihoods together (since Naive Bayes assumes feature independence) and multiply by the prior probabilities of each class (which are equal in this case since we have the same number of examples for each class).

For example, with new point :

- With class -1

+ Likelihood for feature 1:

+ Likelihood for feature 2:

So

- With class +1

+ Likelihood for feature 1:

+ Likelihood for feature 2:

So

Therefore, we can make conclusion point (6,6) will be classify into **class +1**

**3. If Bernoulli naive bayes is used to classify the dataset, what should be changed from (2) ?**

If Bernoulli Naive Bayes is used instead of Gaussian Naive Bayes to classify the dataset, we would need to make some significant changes to the approach, as Bernoulli Naive Bayes is designed for binary/boolean features. It models the presence or absence of a feature, rather than modeling the feature distribution with a Gaussian distribution.

Here's what we would need to change:

1. Binarize the Features: Bernoulli Naive Bayes requires the features to be binary (i.e., 0 or 1). This means we would need to transform our continuous features into binary features. One common approach is to choose a threshold and set the feature to 1 if the feature value is greater than the threshold, and 0 otherwise. However, choosing this threshold can be arbitrary and may not make sense for all types of data, especially if the data does not have a natural binary interpretation.

2. Model Update: Instead of calculating the mean and variance for each feature, we would calculate the probabilities of a feature being 1 (present) or 0 (absent) for each class. For example, for a given feature and class , we would calculate the probability and

3. Probability Calculation: The probability of observing a particular binary feature vector given a class is calculated by taking the product of the individual feature probabilities. For a feature that is present , we use , and for a feature that is absent , we use .

4. Prediction: To make a prediction for a new data point, we calculate the posterior probability for each class using the Bernoulli feature probabilities and then choose the class with the highest posterior probability.

Given that our dataset consists of continuous features rather than binary features, using Bernoulli Naive Bayes would not be appropriate without a meaningful way to binarize the data.

If we were to binarize the data arbitrarily, we would lose a lot of information contained in the continuous features, and the performance of the classifier would likely be poor compared to using Gaussian Naive Bayes, which is designed for continuous data and assumes that the features follow a normal distribution.

**Problem 4. Naive Bayes**

**1. Use the provided dataset to calculate: and**

From the given dataset, we have:

- Priori

-

-

-

- Posteori

**2. With a new observation , predict if it is a banana or not**

- We will calculate the posteori for new observation

- We must calculate and then compare it. However, we saw that 2 probability has the same denominator. Therefore, we can compare it by just compare the numerator

Since , so we can make conclusion that the new observation is not a banana (belong to class **no**)

**Problem 5. Support vector machine**

**1. Write the loss function of SVM**

The loss function of a Support Vector Machine (SVM) classifier, specifically for the linear case, is typically composed of two parts: the hinge loss and the regularization term. The hinge loss is used to maximize the margin between the different classes, and the regularization term is used to prevent overfitting by penalizing large weights.

The decision function for a linear SVM is given by:

where is the bias term, is the weight vector, and is the input feature vector.

The hinge loss for a single training example where is the true label (usually +1 or -1), is defined as:

The loss function for the entire training set is the sum of the hinge losses over all training examples, plus the regularization term. The regularization term is often the L2-norm of the weight vector, which is the squared Euclidean norm of . The complete loss function, known as the primal form of the SVM objective, can be written as:

Here, is a hyperparameter that controls the trade-off between maximizing the margin and minimizing the classification error. A larger value of puts more emphasis on minimizing the classification error, while a smaller value of puts more emphasis on maximizing the margin and potentially allows for more misclassifications.

The term is used instead of to simplify the derivative during the optimization process. The optimization problem is to find the weight vector and bias that minimize the loss function , subject to the constraints imposed by the hinge loss. This is typically done using quadratic programming techniques or other optimization methods suitable for convex problems.

**2. Explain the use of slack variable for the non-separatable cases as below**

A diagram of different shapes and sizes

Description automatically generated

In the case of non-separable data, where it is not possible to find a hyperplane that perfectly separates the classes without any misclassification, slack variables are introduced to allow for some degree of misclassification or violation of the margin. These slack variables are used to soften the margin constraints, making the SVM more flexible and capable of handling cases where the data are not linearly separable.

The slack variable for a training example measures the degree to which that example is allowed to violate the margin. The modified constraints with slack variables for each training example are:

where .

If , the example is correctly classified and lies outside or on the margin.

If , the example is within the margin or misclassified. Specifically:

, the example is correctly classified but lies within the margin boundary.

, the example is on the decision boundary.

, the example is misclassified.

The objective function of the SVM with slack variables becomes:

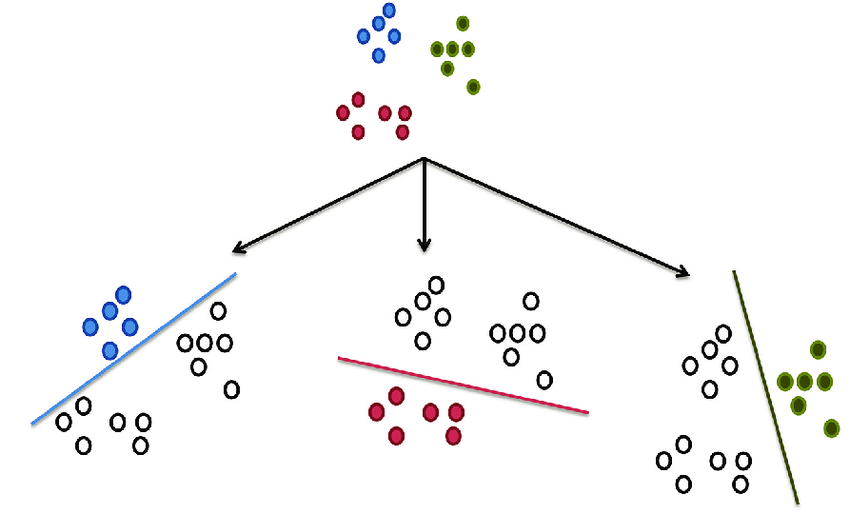
The constant is a hyperparameter that determines the trade-off between maximizing the margin and minimizing the sum of the slack variables. A larger value of places a higher penalty on the slack variables, thus forcing the classifier to try to correctly classify more points at the expense of a potentially smaller margin. A smaller value of allows for more margin violations but can result in a larger margin.

The complete optimization problem for the SVM with slack variables is minimize:

Subject to:

This formulation allows the SVM to handle non-separable data by finding a balance between maximizing the margin and minimizing classification errors. The optimization problem is still convex, and it can be solved using quadratic programming or similar optimization techniques.

**3. Explain how SVM can be used for multiclassification**



Support Vector Machines (SVM) are inherently binary classifiers, which means they are designed to distinguish between two classes. However, they can be extended to handle multi-class classification problems through various strategies. The two most common approaches for using SVM for multi-class classification are the "one-vs-one" (OvO) approach and the "one-vs-rest" (OvR) approach.

* One-vs-One (OvO) Approach:

In the one-vs-one approach, a separate SVM is trained for every pair of classes. If there are classes, this results in binary classifiers. Each classifier is trained on data from two classes and learns to distinguish between them.

During prediction, an unseen sample is presented to all classifiers. The class that wins the most "duels" (i.e., the class that is predicted the most times by the classifiers) is the one assigned to the sample. This method can be computationally expensive for datasets with a large number of classes because it requires training many classifiers.

* One-vs-Rest (OvR) Approach:

In the one-vs-rest approach, one SVM is trained for each class to separate that class from all other classes. Therefore, if there are classes, this results in binary classifiers. Each classifier is trained on data from one class as the positive class and all other data as the negative class.

During prediction, an unseen sample is presented to all classifiers, and the classifier that outputs the highest confidence score (distance from the decision boundary) for the positive class is the one whose associated class label is assigned to the sample.

Both OvO and OvR have their advantages and disadvantages. OvO tends to be more computationally expensive due to the larger number of classifiers that need to be trained, but it can be more accurate since each classifier is trained on a smaller, more manageable subset of the data. OvR is computationally cheaper because it involves fewer classifiers, but it can suffer from class imbalance since each classifier is trained against all other classes combined.

There are also other methods for multi-class SVM, such as the Directed Acyclic Graph SVM (DAGSVM) and error-correcting output codes, but OvO and OvR are the most commonly used.

In addition to these strategies, some SVM implementations, like those found in the popular machine learning libraries scikit-learn or LIBSVM, handle multi-class classification internally, abstracting away the details from the user and providing a simple interface to train and predict on multi-class datasets.

**4. In Scikit-learn library, explain the use of parameters for:**

**a. LinearSVC**

**b. SVC**

**c. When LinearSVC and SVC are equivalent ?**

In the scikit-learn library, LinearSVC and SVC are classes used to perform classification with Support Vector Machines. Each class has a set of parameters that control the behavior of the SVM. Here's an explanation of some key parameters for both:

**a. LinearSVC**

LinearSVC is an implementation of Support Vector Classification for the case of a linear kernel. It is based on the liblinear library and is more suitable for large datasets.

Key parameters include:

- C: Regularization parameter. The strength of the regularization is inversely proportional to C. Must be strictly positive.

- loss: Specifies the loss function. ‘hinge’ is the standard SVM loss (used e.g., by the `SVC` class) while 'squared\_hinge' is the square of the hinge loss.

- penalty: Specifies the norm used in the penalization. The ‘l2’ penalty is the standard used by the SVC class. The ‘l1’ leads to ‘coef\_’ vectors that are sparse.

- dual: Select the algorithm to either solve the dual or primal optimization problem. Prefer dual=False when n\_samples > n\_features.

- tol: Tolerance for stopping criteria.

- max\_iter: The maximum number of iterations to be run.

**b. SVC**

SVC is an implementation of Support Vector Classification based on the **libsvm** library. It can perform linear and non-linear classification using various kernel functions.

Key parameters include:

- C: Regularization parameter, similar to LinearSVC.

- kernel: Specifies the kernel type to be used in the algorithm. It can be ‘linear’, ‘poly’, ‘rbf’, ‘sigmoid’, ‘precomputed’, or a callable.

- degree: Degree of the polynomial kernel function (‘poly’). Ignored by all other kernels.

- gamma: Kernel coefficient for ‘rbf’, ‘poly’, and ‘sigmoid’. If ‘gamma=scale’ (default), it uses 1 / (n\_features \* X.var()) as the value of gamma. If auto, it uses 1 / n\_features.

- coef0: Independent term in kernel function. It is only significant in ‘poly’ and ‘sigmoid’.

- tol: Tolerance for stopping criteria.

- max\_iter: The maximum number of iterations to be run.

**c. When LinearSVC and SVC are equivalent:**

LinearSVC` and SVC are equivalent when SVC is used with a linear kernel. In this case, both are intended to solve the same problem, which is linear classification. However, even when they are set to perform linear classification, there are differences in their implementation that can lead to different behaviors, particularly in terms of performance and the handling of large datasets.

However, LinearSVC does not support the kernel parameter because it is specifically designed for linear classification and uses a different underlying library (liblinear rather than libsvm). Additionally, LinearSVC tends to be faster on large datasets with many features, whereas SVC with a linear kernel can be slower but may be more suitable for smaller datasets.

**Problem 6. Classification**

**1. Explain: Accuracy, Precision, Recall and F1-Score. What is the precision-recall trade-off ?**

Accuracy, Precision, Recall, and F1-Score are metrics used to evaluate the performance of classification algorithms:

- Accuracy: This is the most intuitive performance measure and it is simply a ratio of correctly predicted observation to the total observations. It is suitable for symmetric datasets where values of false positive and false negatives are almost same. It is calculated as:

- Precision: Also known as Positive Predictive Value, it is the ratio of correctly predicted positive observations to the total predicted positive observations. It is a measure of a classifier's exactness. High precision relates to a low false positive rate. It is particularly useful in cases where False Positives are a bigger concern than False Negatives. Precision is calculated as:

- Recall: Also known as Sensitivity or True Positive Rate, it is the ratio of correctly predicted positive observations to the all observations in actual class - yes. It is a measure of a classifier's completeness. High recall relates to a low false negative rate. Recall is calculated as:

- F1-Score: It is also called the F Score or the F Measure. It is a way of combining the precision and recall of the model, and it is defined as the harmonic mean of the model's precision and recall. The F1 score is a number between 0 and 1 and is a measure of the test's accuracy. The F1 Score is the weighted average of Precision and Recall. Therefore, this score takes both false positives and false negatives into account. It is a good way to show that a classifer has a good value for both recall and precision.

The precision-recall trade-off

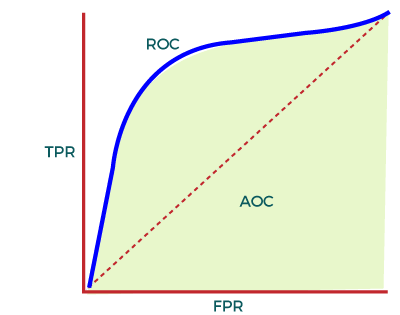
The precision-recall trade-off is a concept that comes from the fact that increasing precision typically reduces recall and vice versa. This is due to the threshold used to classify a prediction as positive. By adjusting this threshold, one can trade precision for recall or the other way around. For instance, in a spam detection system, if we set a high threshold to ensure that only the most likely messages are classified as spam, we will have high precision (few legitimate emails will be classified as spam), but we might miss some actual spam emails (lower recall). Conversely, if we lower the threshold, we will catch more spam (higher recall) but we might start flagging more legitimate emails as spam (lower precision).

In practice, the best balance between precision and recall depends on the specific costs of false positives and false negatives in the given application. Often, a domain-specific trade-off point is chosen depending on which metric is more important to optimize.

**2. Explain:**

**a. Structure of a ROC**

**b. Meaning of ROC**



- Structure of a ROC:

The ROC (Receiver Operating Characteristic) curve is a graphical plot that illustrates the diagnostic ability of a binary classifier system as its discrimination threshold is varied. The curve is created by plotting the True Positive Rate (TPR, also known as recall or sensitivity) against the False Positive Rate (FPR, where FPR = 1 - specificity) at various threshold settings.

- The structure of an ROC curve includes the following:

+ X-axis (False Positive Rate): The false positive rate is plotted along the x-axis and is calculated as FPR = FP / (FP + TN), where FP is the number of false positives and TN is the number of true negatives.

+ Y-axis (True Positive Rate): The true positive rate is plotted along the y-axis and is calculated as TPR = TP / (TP + FN), where TP is the number of true positives and FN is the number of false negatives.

+ Threshold: Each point on the ROC curve represents a different threshold used to classify instances as positive or negative. The threshold is not shown on the curve but is implied: as we move from the bottom left to the top right, the threshold typically decreases, meaning the classifier is more likely to predict positive.

+ Diagonal line: The diagonal line from the bottom left to the top right represents the performance of a random classifier (no better than random guessing). A classifier with points above this line is considered to have some ability to separate the positive and negative classes, while a classifier that lies below this line performs worse than random guessing.

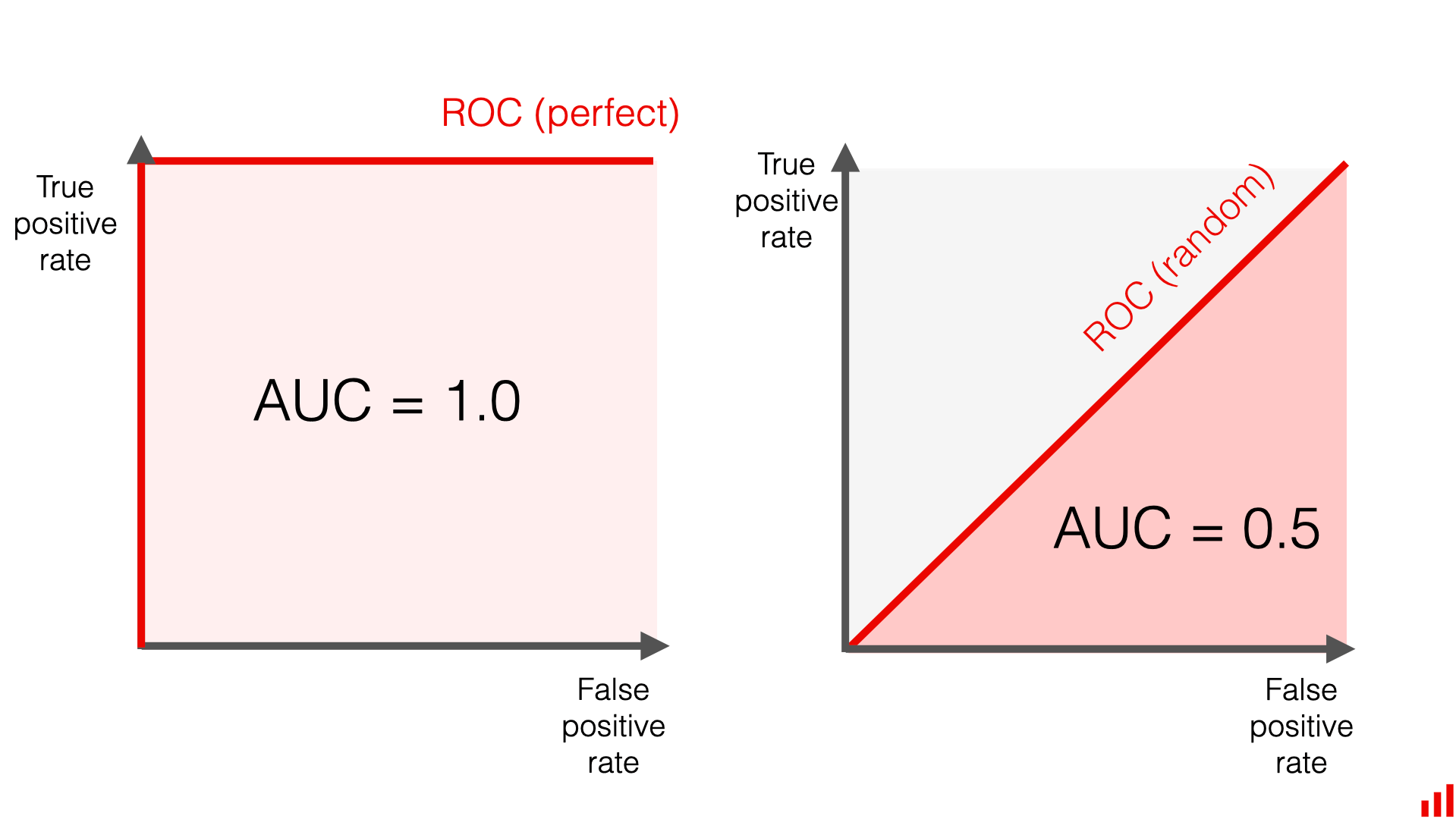
- Meaning of ROC:

+ ROC stands for Receiver Operating Characteristic. It originated from signal detection theory and was first used during World War II for the analysis of radar signals before it was adopted for other types of psychological tests and later for machine learning classification problems.

+ The ROC curve is a useful tool for evaluating the performance of a classification model at various threshold settings. It provides a comprehensive view of the trade-off between the true positive rate and the false positive rate of the classifier. By analyzing the ROC curve, one can select an optimal threshold that balances the TPR and FPR for their specific needs, or compare the performance of different classifiers based on the area under their ROC curves (AUC).

+ The ROC curve is particularly useful when dealing with imbalanced datasets or when the costs of different types of errors (false positives and false negatives) vary. It allows for the comparison of classifiers independently of the class distribution or error costs.

**3. Meaning of a AUC**



AUC stands for "Area Under the Curve." When used in the context of a ROC curve, it refers to the Area Under the Receiver Operating Characteristic curve. The AUC provides a single scalar value that summarizes the performance of a binary classification model across all possible threshold values. Here's what the AUC represents:

- Overall Performance: The AUC measures the overall ability of the model to discriminate between the positive and negative classes. A higher AUC value means that the model is better at correctly classifying positive and negative examples.

- Probability Interpretation: The AUC can be interpreted as the probability that the classifier will rank a randomly chosen positive instance higher than a randomly chosen negative instance. This is based on the assumption that positive and negative instances are chosen randomly from the dataset.

- Aggregate Measure: Unlike accuracy, which can be skewed by imbalanced class distributions, the AUC is an aggregate measure of performance across all classification thresholds. It is not affected by the proportion of positive to negative class instances.

- Range: The AUC ranges from 0 to 1:

+ An AUC of 0.5 suggests that the model has no discriminative ability and performs no better than random guessing.

+ An AUC of 1.0 suggests that the model has perfect discriminative ability and can perfectly distinguish between positive and negative class instances.

+ An AUC less than 0.5 suggests that the model is performing worse than random guessing, which usually indicates a problem with the model or the data.

- Comparison: The AUC is often used to compare the performance of two or more classification models. A model with a higher AUC is generally considered to have better performance.

In summary, the AUC is a widely used metric in machine learning for binary classification problems because it is insensitive to class imbalance and provides a single number that captures the model's performance across all thresholds. It is especially useful when we need to compare different models or when there is no clear preference for either recall or precision in the problem domain.

**4. Meaning of confusion matrix**

A confusion matrix is a table often used to describe the performance of a classification model (or "classifier") on a set of test data for which the true values are known. It allows easy identification of confusion between classes, i.e., how often instances of class A are classified as class B, and is a powerful tool for summarizing the performance of a classification algorithm.

The confusion matrix itself is a simple layout to understand, but the terms used can be a bit confusing. Here's a breakdown of a confusion matrix for a binary classifier:

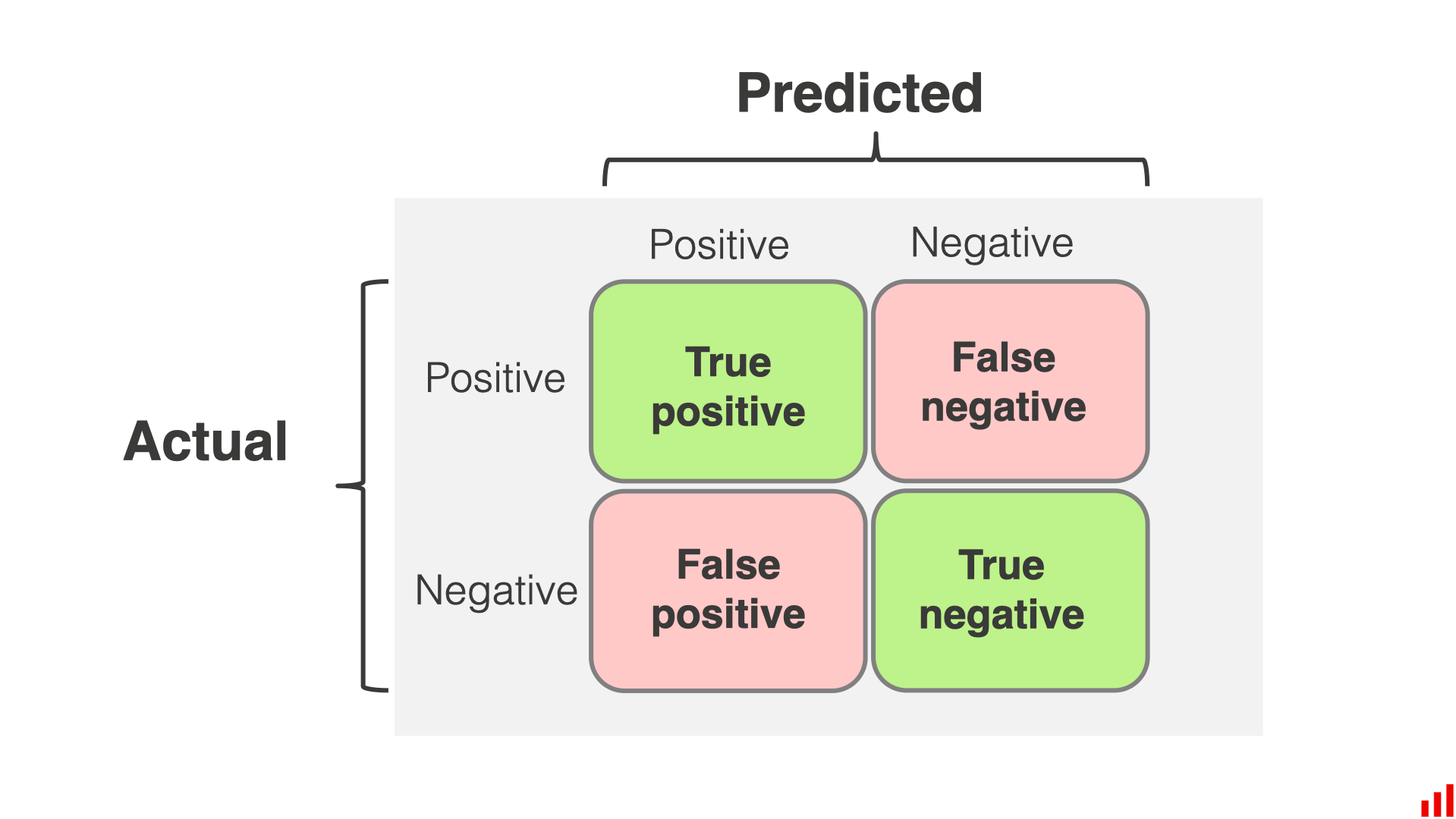
- True Positives (TP): These are cases in which the model correctly predicted the positive class.

- True Negatives (TN): These are cases in which the model correctly predicted the negative class.

- False Positives (FP): These are cases in which the model incorrectly predicted the positive class (also known as a "Type I error").

- False Negatives (FN): These are cases in which the model incorrectly predicted the negative class (also known as a "Type II error").

The confusion matrix looks like this:



Each row of the matrix represents the instances in an actual class while each column represents the instances in a predicted class. Here's what each term represents:

- The first row is about the positive class (the class of interest). The first cell (TP) is the number of correct predictions that an instance is positive, and the second cell (FN) is the number of incorrect predictions that an instance is negative.

- The second row is about the negative class. The first cell (FP) is the number of incorrect predictions that an instance is positive, and the second cell (TN) is the number of correct predictions that an instance is negative.

From the confusion matrix, a variety of metrics can be calculated, such as Accuracy, Precision, Recall, and F1-Score, which provide a more complete picture of how well a classification model is performing.

**Problem 7. Classification**

**1. For threshold 0.5, the result of prediction is**

|  |  |  |  |
| --- | --- | --- | --- |
| **ID** | **True class** | **Prediction** | **Prediction class** |
| 1 | 0 | 0.33 | 0 |
| 2 | 0 | 0.27 | 0 |
| 3 | 0 | 0.11 | 0 |
| 4 | 1 | 0.38 | 0 |
| 5 | 1 | 0.17 | 0 |
| 6 | 1 | 0.63 | 1 |
| 7 | 1 | 0.62 | 1 |
| 8 | 1 | 0.33 | 0 |
| 9 | 0 | 0.15 | 0 |
| 10 | 0 | 0.57 | 1 |

So we can construct the following confusion matrix

|  |  |  |  |
| --- | --- | --- | --- |
|  | | **Predict** | |
| **Negative** | **Positive** |
| **Actual** | **Negative** | 4 | 1 |
| **Positive** | 3 | 2 |

We have

* True positive (TP) = 2
* False positive (FP) = 1
* True negative (TN) = 4
* False negative (FN) = 3

**2. Calculate accuracy, precision, recall, F1 score**

* Accuracy
* Precision =
* Recall
* F1 score =

**3. Draw a ROC**

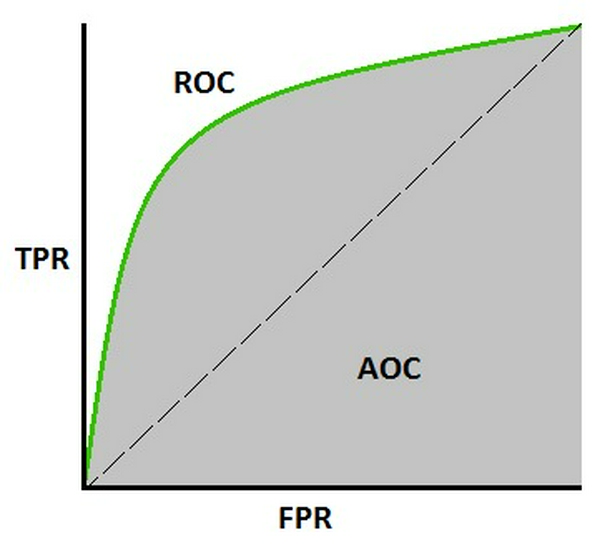
We will calculate the TN, TP, FN, FP for each threshold from 0.0 to 1.0 with interval 0.1. For each threshold, we will also calculate the TPR and FPR (True Positive rate and False Positive Rate)

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Threshold** | **TP** | **FP** | **TN** | **FN** | **TPR** | **FPR** |
| 0.0 | 5 | 5 | 0 | 0 | 1 | 1 |
| 0.1 | 5 | 5 | 0 | 0 | 1 | 1 |
| 0.2 | 4 | 3 | 2 | 1 | 0.8 | 0.6 |
| 0.3 | 4 | 2 | 3 | 1 | 0.8 | 0.4 |
| 0.4 | 2 | 1 | 4 | 3 | 0.4 | 0.2 |
| 0.5 | 2 | 1 | 4 | 3 | 0.4 | 0.2 |
| 0.6 | 2 | 0 | 5 | 3 | 0.4 | 0 |
| 0.7 | 0 | 0 | 5 | 5 | 0 | 0 |
| 0.8 | 0 | 0 | 5 | 5 | 0 | 0 |
| 0.9 | 0 | 0 | 5 | 5 | 0 | 0 |
| 1.0 | 0 | 0 | 5 | 5 | 0 | 0 |

A graph with a line going up

Description automatically generated

**4. Calculate AUC**



To calculate the Area Under the Curve (AUC) for the ROC curve, we can use the trapezoidal rule to approximate the area under the stepwise function formed by the TPR and FPR values at each threshold.