Contents

Chapter 1

The Training/Validation/Testing Process

In machine learning, it's essential to assess the performance of a model accurately. This assessment helps us choose the best model and tune its parameters. The data used to develop machine learning models is typically divided into three sets: training, validation, and testing.

1.0.1 Training Set

The training set is used to train the model, i.e., to adjust the model's weights and biases in the case of neural networks, or to determine the best split in decision trees, among other things. The model learns from this data, which is why it's called the "training" set.

1.0.2 Validation Set

The validation set is used to tune model parameters (hyperparameters), to choose the model architecture (for example, the number of hidden layers in a neural network), or to determine the degree of the polynomial in polynomial regression, among other uses. This set provides an unbiased evaluation of a model fit on the training dataset while tuning model hyperparameters.

1.0.3 Testing Set

The testing set is used to provide an unbiased evaluation of the final model fit on the training dataset. The test set serves as a proxy for real-world data that the model has not seen before. It's important to only use the test set once, after all training and validation is complete, to avoid "leaking" information from the test set into the model.

The key to this process is that each set of data is separate and independent. Mixing data between the sets can lead to overly optimistic or pessimistic assessments of a model's performance.

In practice, the division of data into these three sets can be done randomly (often with 70%

for training, 15% for validation, and 15% for testing), or using more structured methods like cross-validation, depending on the amount and nature of the available data.

CHAPTER 2

Evaluating Classification Systems

The confusion matrix is a tabular method used in machine learning to evaluate the performance of a classification model. It allows for the visualization of the model's performance and to compute various performance metrics.

2.1 Definition of a Confusion Matrix

A confusion matrix is a specific table layout that presents the performance of a classification model. For a binary classification problem, it is a 2x2 matrix that compares the actual and the predicted classifications.

	Actual Positive	Actual Negative
Predicted Positive	True Positive (TP)	False Positive (FP)
Predicted Negative	False Negative (FN)	True Negative (TN)

2.2 Performance Metrics

Using the confusion matrix, we can compute several performance metrics:

- **Accuracy:** The proportion of correct predictions (both true positives and true negatives) among the total number of cases examined. It is calculated as $(TP + TN)/(TP + TN + FP + FN)$.
- **Precision:** The proportion of positive identifications that were actually correct. It is calculated as $TP/(TP + FP)$.
- **Recall (Sensitivity):** The proportion of actual positives that were identified correctly. It is calculated as $TP/(TP + FN)$.
- **Specificity:** The proportion of actual negatives that were identified correctly. It is calculated as $TN/(TN + FP)$.
- **F1 Score:** The harmonic mean of precision and recall. It tries to find the balance between precision and recall. $F1 = 2 * (Precision * Recall)/(Precision + Recall)$.

CHAPTER 3

Evaluation Binary Classifiers

Accuracy, recall, precision, and the F1 score are widely used metrics to measure and compare the performance of binary classifiers. This chapter will delve into these evaluation measures, providing insights into their interpretation and practical applications.

3.0.1 Binary Classification

Before diving into the evaluation metrics, let's clarify the concept of binary classification. In binary classification, we aim to assign each instance in a dataset to one of two mutually exclusive classes. For example, classifying emails as spam or not spam, identifying whether a patient has a specific medical condition or not, or predicting whether a credit card transaction is fraudulent or legitimate are common binary classification tasks.

To evaluate the performance of a binary classifier, we need metrics that can provide insights into how well the classifier performs in distinguishing between the two classes.

3.0.2 Accuracy

Accuracy is a widely used metric for evaluating binary classifiers. It measures the overall correctness of the classifier's predictions by calculating the ratio of correctly classified instances to the total number of instances in the dataset. Mathematically, accuracy can be expressed as:

$$\text{Accuracy} = \frac{\text{True Positives} + \text{True Negatives}}{\text{True Positives} + \text{True Negatives} + \text{False Positives} + \text{False Negatives}}$$

While accuracy provides a general overview of the classifier's performance, it may not be sufficient in certain scenarios. This is especially true when the dataset is imbalanced, meaning that one class significantly outweighs the other in terms of the number of instances.

3.0.3 Precision and Recall

Precision and recall are evaluation metrics that provide insights into the classifier's performance on specific classes, allowing us to identify potential trade-offs between false positives and false negatives.

Precision measures the proportion of correctly predicted positive instances (true positives) out of all instances predicted as positive (true positives + false positives). It can be expressed as:

$$\text{Precision} = \frac{\text{True Positives}}{\text{True Positives} + \text{False Positives}}$$

Recall, also known as sensitivity or true positive rate, measures the proportion of correctly predicted positive instances (true positives) out of all actual positive instances (true positives + false negatives). Mathematically, recall can be represented as:

$$\text{Recall} = \frac{\text{True Positives}}{\text{True Positives} + \text{False Negatives}}$$

Precision and recall are complementary metrics. Precision focuses on the quality of positive predictions, while recall emphasizes the classifier's ability to identify positive instances. The choice between precision and recall depends on the specific requirements of the problem at hand.

3.0.4 F1 Score

The F1 score combines precision and recall into a single metric, providing a balanced evaluation measure that considers both false positives and false negatives. It is the harmonic mean of precision and recall, and it can be calculated as:

$$\text{F1} = \frac{2 \times \text{Precision} \times \text{Recall}}{\text{Precision} + \text{Recall}}$$

The F1 score ranges between 0 and 1, where a value of 1 represents perfect precision and recall. It is particularly useful when we want to strike a balance between precision and recall, considering both the false positives and false negatives in the classifier's predictions.

Chapter 4

The k-Nearest Neighbor Classifier

The k-nearest neighbors (k-NN) algorithm is a type of instance-based learning algorithm used for classification and regression. Given a new, unknown observation, k-NN algorithm searches through the entire dataset to find the 'k' training examples that are closest to the new instance, and predicts the label based on these 'k' nearest neighbors.

4.1 The k-NN Algorithm

The algorithm can be summarized as follows:

1. Given a new observation $\mathbf{x}$, compute the distance between $\mathbf{x}$ and all points in the training set.
2. Identify the 'k' points in the training data that are closest to $\mathbf{x}$.
3. If k-NN is used for classification, output the most common class among these 'k' points as the prediction. If k-NN is used for regression, output the average of the values of these 'k' points as the prediction.

The distance between points can be calculated using various metrics, the most common one being the Euclidean distance:

$$d(\mathbf{x}, \mathbf{y}) = \sqrt{\sum_{i=1}^{n} (x_i - y_i)^2}$$

where n is the number of features, and x_i and y_i are the corresponding features of $\mathbf{x}$ and $\mathbf{y}$.

4.2 Choosing the Right 'k'

The choice of 'k' has a significant impact on the k-NN algorithm. A small 'k' (like 1) can capture a lot of noise and lead to overfitting, while a large 'k' can smooth over many

details and potentially lead to underfitting. Cross-validation is typically used to select an optimal ‘k‘.

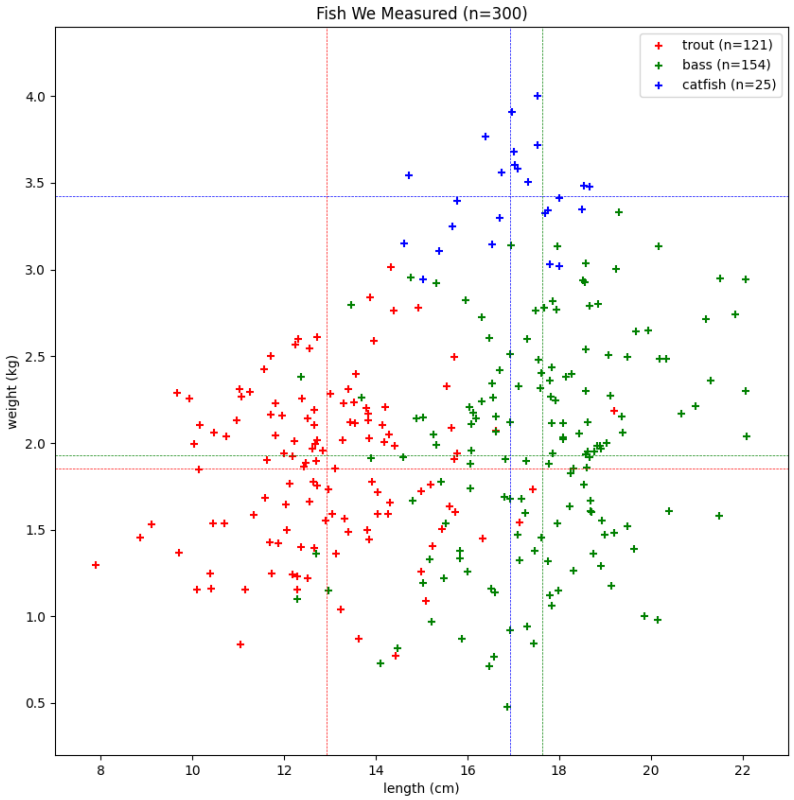
4.3 Considerations

Although the k-NN algorithm is simple to understand and implement, it can be computationally intensive for large datasets, as it requires computing the distance between every pair of points. Additionally, it’s sensitive to the choice of the distance metric and the scale of the features.

Chapter 5

Bayesian Classifiers

You drop a net in several places in a lake. You measure, weigh, and identify every fish to be one of the following 3: `trout`, `bass`, or `catfish`. Here is a scatter plot of what you find.



(Data: `make_data.py`, Graph: `1_scatter_fish.py`)

The vertical and horizontal lines represent the mean length and width (respectively) of each type of fish.

You are writing a system that someone on the lake could use to identify their fish.

5.1 The Prior

Even before the person measures and weighs the fish, they can make a decent guess at what any randomly selected fish is:

Fish	Count	Percent
trout	121/300	40.3%
bass	154/300	51.3%
catfish	25/300	8.3%

If you know nothing about the fish, there is a 51.3% chance it is a bass. The probability without any observations is known as the *prior*:

$$\begin{aligned} \mathrm{P}(\text{fish} = \text{trout}) &= .403 \\ \mathrm{P}(\text{fish} = \text{bass}) &= .513 \\ \mathrm{P}(\text{fish} = \text{trout}) &= .083 \end{aligned}$$

But, now the user makes an observation: The fish is 16 inches long and weighs 2.1 kg. Can we update our beliefs based on this? Sounds like a job for conditional probability.

By the definition of conditional probability:

$$\mathrm{P}(\text{fish} = \text{trout} | \text{length} = 16, \text{weight} = 2.1) = \frac{\mathrm{p}(\text{fish} = \text{trout}, \text{length} = 16, \text{weight} = 2.1)}{\mathrm{p}(\text{length} = 16, \text{weight} = 2.1)}$$

I'm going to run out of space, so I will use F to represent fish, L to represent length, and W to represent weight.

We can calculate the denominator by summing up all the possibilities. Now the right-hand side is:

$$\frac{\mathrm{p}(\mathrm{F}{=}\text{trout}, \mathrm{L}{=}16, \mathrm{W}{=}2.1)}{\mathrm{p}(\mathrm{F}{=}\text{trout}, \mathrm{L}{=}16, \mathrm{W}{=}2.1) + \mathrm{p}(\mathrm{F}{=}\text{bass}, \mathrm{L}{=}16, \mathrm{W}{=}2.1) + \mathrm{p}(\mathrm{F}{=}\text{catfish}, \mathrm{L}{=}16, \mathrm{W}{=}2.1)}$$

We can calculate the probability of all three possibilities (and they will add up to 1.0).

5.2 Naive Bayes

The assumption in naive Bayes is that the probability distributions of length and weight are conditionally independent given the breed of the fish. That is:

$$p(\text{F=trout}, \text{L=16}, \text{W=2.1}) = p(\text{L=16}|\text{F=trout})p(\text{W=2.1}|\text{F=trout})P(\text{F=trout})$$

Within a particular species, things like length and weight, which represent of the sum of a lot of genetic and environmental factors, are often normally distributed.

What if we assume that length has a normal distribution for each kind of fish? What is the most likely estimator for the mean of each trait?

$$\mu = \frac{1}{n}\sum_{i=1}^{n} x_i$$

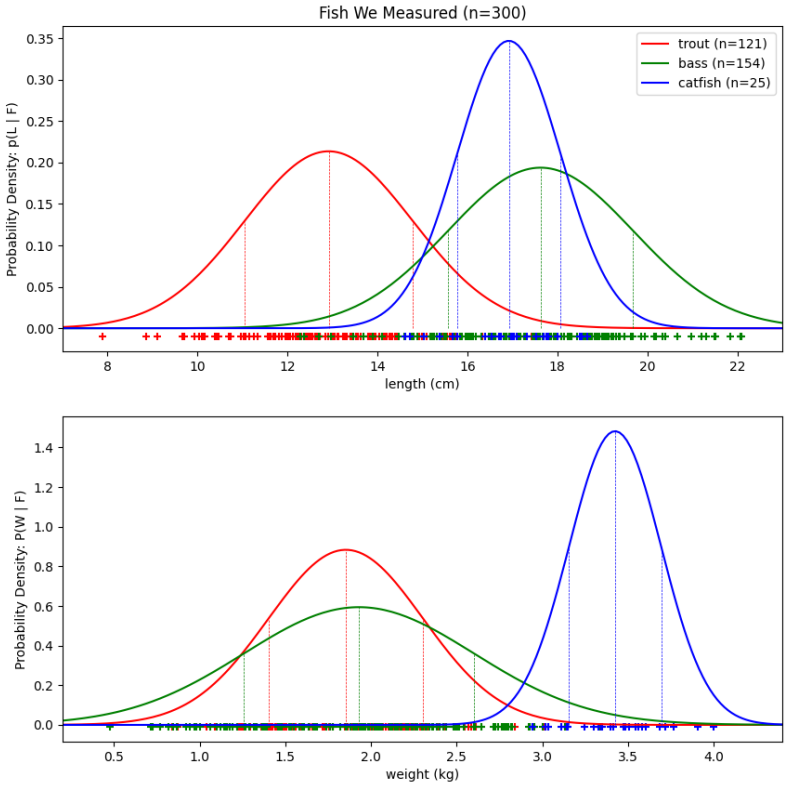
And variance?

$$\sigma^2 = \frac{1}{n}\sum_{i=1}^{n} (\mu - x_i)^2$$

We can compute those pretty easily:

Fish	Trait	μ	σ^2
Trout	Length	12.91522571	3.49054363
	Weight	1.85290281	0.20387574
Bass	Length	17.62628327	4.23881747
	Weight	1.92960088	0.45092986
Catfish	Length	16.92187387	1.32301935
	Weight	3.42423185	0.07244546

If we plot those probability distributions out:



(Source:`2_univariates.py`)

Note that the area under each one of those curves is exactly 1.0. They are probability distributions.

Now we can use the formula for the normal distribution:

$$p(x) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{(x-\mu)^2}{2\sigma^2}}$$

Thus,

Given	$p(L = 16|F)$	$p(W = 2.1|F)$
F=Trout	.0546	.7607
F=Bass	.1418	.5753
F=Catfish	.2516	0.01

Now, using the naive assumption, we can compute the joint probability for any type of fish:

$$p(F = ?, L = 16, W = 2.1) = p(W = 2.1|F = ?)p(L = 16|F = ?)p(F = ?)$$

For	$p(L = 16, W = 2.1, F = ?)$
F=Trout	.016763
F=Bass	.041886
F=Catfish	0.00001

The sum of these is $p(L = 16, W = 2.1)$: .0586

By the definition of conditional probability:

$$p(F = ?|L = 16, W = 2.1) = \frac{p(F = ?, L = 16, W = 2.1)}{p(L = 16, W = 2.1)}$$

So

For	$p(F = ?|L = 16, W = 2.1)$
F=Trout	.286
F=Bass	.714
F=Catfish	≈ 0

If you pull a random fish out of the lake and its length is 16 and its weight is 2.1, there is a 71.4% chance that it is bass. There is a 28.6% chance it is a trout.

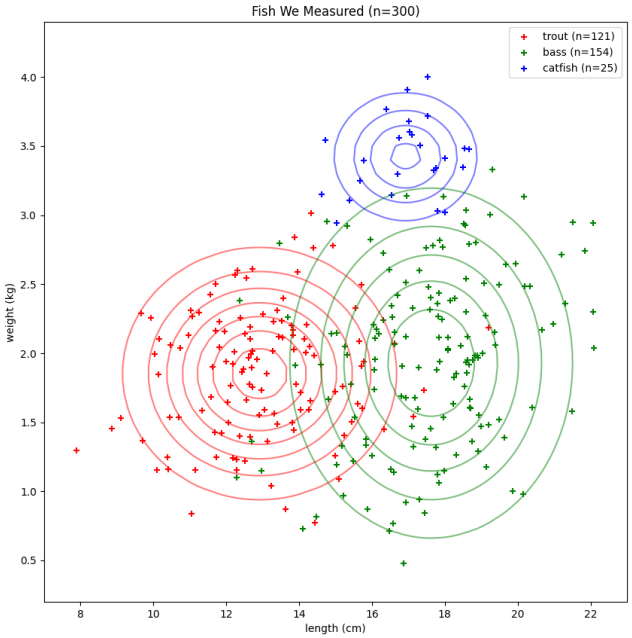
5.3 Using the multivariate Gaussian distribution

Give the kind of fish, length and width are independent!? That makes no sense. Longer fish tend to be heavier, right?

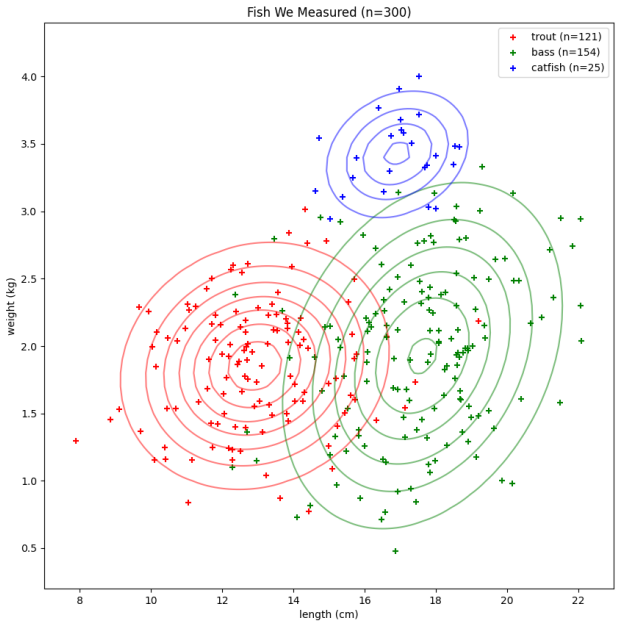
The multivariate Gaussian distribution is a generalization of the normal distribution::

- It deals with data of any dimension d.
- It can includes the idea of covariance: For example, length and width are not independent.

If we use your naive Bayes, we can get two dimensional density plots that would look like this:



With independence, the ovals are circles, ovals that go up, or ovals that sideways. If we use the multivariate Gaussian distribution, we get:



When there is covariance, the ovals can slant in any direction.

If we discard the naive assumption, we should get slightly more accurate results.

The multivariate Gaussian distribution uses vectors and matrices. The mean $\vec{\mu}$ is a vector

of dimension d. The variance of each variable and its covariance with the other variables is captured in a $d \times d$ matrix called *the covariance matrix* usually called $\mathbf{\Sigma}$. (Yes, this is also the symbol for summation. This creates some confusion at times, but you can usually figure out which is intended by the context.)

When you know $\vec{\mu}$ and $\mathbf{\Sigma}$, the probability density for any vector $\vec{x}$ is:

$$p(\vec{x}) = (2\pi)^{-d/2} \det(\mathbf{\Sigma})^{-1/2} \exp\left(-\frac{1}{2}(\vec{x} - \vec{\mu})^T \mathbf{\Sigma}^{-1} (\vec{x} - \vec{\mu})\right)$$

If I have a bunch of data points $\vec{x_1}, ..., \vec{x_n}$. How do I compute the values of $\vec{\mu}$ and $\mathbf{\Sigma}$ for which the observations $\vec{x_1}, ..., \vec{x_n}$ would be most likely?

$$\vec{\mu} = \frac{1}{n} \sum_{i=1}^{d} \vec{x_i}$$

And the covariance matrix? (Here we think of each $\vec{x_i}$ and $\vec{\mu}$ as column vectors.)

$$\mathbf{\Sigma} = \frac{1}{n} \sum_{i=1}^{n} (\vec{x_i} - \vec{\mu})(\vec{x_i} - \vec{\mu})^T$$

Notice that the diagonal entries of the matrix are the variance of each variable. Notice also, that this is a symmetric matrix.

Applying this to our data we get:

Fish	$\vec{\mu}$	$\mathbf{\Sigma}$
Trout	$\begin{bmatrix} 12.91522571 \\ 1.85290281 \end{bmatrix}$	$\begin{bmatrix} 3.51963149 & 0.09975744 \\ 0.09975744, & 0.20557471 \end{bmatrix}$
Bass	$\begin{bmatrix} 17.62628327 \\ 1.92960088 \end{bmatrix}$	$\begin{bmatrix} 4.26652216 & 0.39553268 \\ 0.39553268 & 0.45387711 \end{bmatrix}$
Catfish	$\begin{bmatrix} 16.92187387 \\ 3.42423185 \end{bmatrix}$	$\begin{bmatrix} 1.37814515 & 0.07281453 \\ 0.07281453 & 0.07546403 \end{bmatrix}$

We can use these to compute the likelihood of seeing a fish with length = 16 and weight = 2.1 for each type of fish:

Given	$p(L = 16, W = 2.1 \mid F = ?)$
F=Trout	.04578
F=Bass	.07732
F=Catfish	.000001

Multiplied by the prior to get the joint probability:

Given	$p(L = 16, W = 2.1, F = ?)$
F=Trout	.01847
F=Bass	.03969
F=Catfish	≈ 0

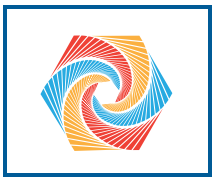
We sum those to get $p(L = 16, W = 2.1) = .0582$

Given	$p(F = ? \mid L = 16, W = 2.1)$
F=Trout	.318
F=Bass	.682
F=Catfish	≈ 0

Given a randomly selected fish that is 19 inches long and 2.1 kg in weight, you would guess that it is a bass with a confidence of 68.2%.

Appendix A

Answers to Exercises



Index