

Metrics

Last modified: 2026-10-25

i Source

Adapted from Brian Hutchinson's Fall 2025 lecture notes, and drawing on Logan Sizemore's Spring 2025 lectures on classification metrics.

1 Metrics Overview

- Throughout these notes, we have trained models to make accurate predictions
- Two questions:
 - *how do we determine if a model is good?*
 - *Is model A better than model B?*
- We have seen some metrics to compare models (e.g. accuracy, cross-entropy, sample mean squared error (sMSE))
- The goal of this lecture:
To introduce a larger set of metrics to assess model performance.
- *In general*, these metrics are not the losses we optimize
 - Losses are for machines to train/optimize
 - Metrics are for humans to assess model performance

i Watch

von Luxburg discusses evaluation in the machine learning pipeline^a [1, Pt. 55].

^a<https://www.youtube.com/watch?v=POPXWdSaC70>

2 Classification and Detection

2.1 Non-differentiable

Confusion Matrices

- A confusion matrix is a matrix $M \in \mathbb{R}^{C \times C}$
 - M_{ij} is the number of datapoints in class i that the model predicts as class j
 - Some sources flip the axes (M^T)
 - Compute them with `from sklearn.metrics import confusion_matrix`

- We can treat this as a detection task for any of the classes. Instead of $\{dog, cat\}$, we detect dog ($h(x) = isDog(x) \in \{T, F\}$). Write $\hat{y}_{(i)} = h(x_{(i)})$ for the prediction on example i . Then M has the following form:

$$M = \begin{bmatrix} T_p & F_n \\ F_p & T_n \end{bmatrix}$$

- $T_p = \sum_{i=1}^N I(y_{(i)} = T)I(\hat{y}_{(i)} = T)$: true pos. Is dog, predict dog.
- $F_p = \sum_{i=1}^N I(y_{(i)} = F)I(\hat{y}_{(i)} = T)$: false pos. Is $\neg$ dog, predict dog. (Type I error).
- $F_n = \sum_{i=1}^N I(y_{(i)} = T)I(\hat{y}_{(i)} = F)$: false neg. Is dog, predict $\neg$ dog. (Type II error).
- $T_n = \sum_{i=1}^N I(y_{(i)} = F)I(\hat{y}_{(i)} = F)$: true neg. Is $\neg$ dog, predict $\neg$ dog.

Definition 2.1 (Confusion matrix). For a detection task on N labelled examples, the **confusion matrix** counts the examples by true label and predicted label [2]:

$$M = \begin{bmatrix} T_p & F_n \\ F_p & T_n \end{bmatrix}, \quad T_p + F_n + F_p + T_n = N.$$

The entries are the numbers of true positives T_p (positives predicted positive), false negatives F_n (positives predicted negative), false positives F_p (negatives predicted positive), and true negatives T_n (negatives predicted negative). The row sums $T_p + F_n$ and $F_p + T_n$ are fixed by the data, whatever the rule.

Exercise 2.1 (Count the four ways a detector can be right or wrong). In the course overview we wrote a spam rule by hand from five columns of Hastie, Tibshirani and Friedman’s spam data:

if %george < 0.6 and %you > 1.5, then spam; else email.

That is the same 4601 messages, and the data are downloadable: the collection is the **Spambase** data set of Mark Hopkins, Erik Reeber, George Forman and Jaap Suermondt, at <https://archive.ics.uci.edu/dataset/94/spambase>. Of the 4601 messages, 1813 are spam and 2788 are email.

Running that rule over all 4601 messages gives these counts:

Table 1: The rule’s predictions against the truth.

	Actually spam	Actually email
Predicted spam	1217	726
Predicted email	596	2062

Name each of the four cells: true positive, false positive, false negative, true negative. Take *positive* to denote “this message is spam”.

Solution

Solution 2.1. With *positive* meaning spam:

$$\begin{aligned} M &= \begin{bmatrix} T_p & F_n \\ F_p & T_n \end{bmatrix} \\ &= \begin{bmatrix} 1217 & 726 \\ 596 & 2062 \end{bmatrix}. \end{aligned}$$

$T_p = 1217$ messages are spam and were called spam. $F_n = 726$ are spam and were called email — the rule missed them. $F_p = 596$ are email and were called spam. $T_n = 2062$ are email and were called email. The names are worth reading as two words each rather than as four labels. The second word gives what the detector *said*; the first word gives whether it was *right*. A false negative is a negative call that was false.

Exercise 2.2 (Count the four ways a detector can be right or wrong (part 2)). Use the setup of Exercise 2.1.

Which sums must hold among the four cells and the totals?

Solution

Solution 2.2. Three sums must hold:

- the true positives and the false negatives sum to the number of spam messages,
- the false positives and the true negatives sum to the number of email messages,
- all four cells sum to the number of messages.

Exercise 2.3 (Count the four ways a detector can be right or wrong (part 3)). Use the setup of Exercise 2.1.

Check that the four cells are consistent with the totals — 1813 spam, 2788 email, 4601 messages.

Solution

Solution 2.3. All three do:

$$\begin{aligned}T_p + F_n &= 1217 + 596 \\ &= 1813 \quad (\text{the spam}),\end{aligned}$$

$$\begin{aligned}F_p + T_n &= 726 + 2062 \\ &= 2788 \quad (\text{the email}),\end{aligned}$$

$$T_p + F_n + F_p + T_n = 4601 \quad (\text{everything}).$$

The row sums of M are fixed by the data and do not depend on the rule at all. Only the split *within* each row is the rule's doing. Remember this: any metric computed down a column mixes the two classes together, and any metric computed along a row does not.

Exercise 2.4 (Count the four ways a detector can be right or wrong (part 4)). Use the setup of Exercise 2.1.

Two of the four cells are errors. Describe each one in a sentence a user of the mail system would recognize, with no jargon in it.

Solution

Solution 2.4. A false negative is a spam message that arrives in your inbox. A false positive is a real message that arrives in your spam folder.

Exercise 2.5 (Count the four ways a detector can be right or wrong (part 5)). Use the setup of Exercise 2.1.

One of those two errors is much worse than the other for a mail system. Which, and why? Nothing about the table tells you the answer, so say what does.

Solution

Solution 2.5. The false positive is worse, for almost any mail system. A spam message in the inbox costs a second of attention; a real message in the spam folder may never be read at all, and the sender has no way to know.

Notice that nothing in Table 1 says so. The table counts both errors and weighs neither. The judgment

comes from outside the data — from what the system is for, and from who bears the cost of each mistake. This is the reason a single number is never enough on its own, and the reason the rest of the chapter is a list of different numbers rather than a search for the best one.

Exercise 2.6 (Count the four ways a detector can be right or wrong (part 6)). Use the setup of Exercise 2.1.

A **confusion matrix** for C classes is a $C \times C$ matrix M with M_{ij} the number of points whose true class is i and whose predicted class is j . Write Table 1 in that form, and say which entries are the correct predictions in general.

Solution

Solution 2.6. Classes are “spam” and “email”; take $i = 1$ for spam and $i = 2$ for email. Then $M_{11} = T_p$, $M_{12} = F_n$, $M_{21} = F_p$, $M_{22} = T_n$, which is the matrix in Solution 2.1. In general the correct predictions are the **diagonal**^a entries M_{ii} : true class i , predicted class i . Everything off the diagonal is an error, and its position says which error — M_{ij} counts class i mistaken for class j .

^a<https://morrison-lab.github.io/mds/linear-algebra-special-matrices.html#def-diagonal-matrix>

Exercise 2.7 (Count the four ways a detector can be right or wrong (part 7)). Use the setup of Exercise 2.1.

Some textbooks and some software write the transpose^a of this matrix. How would you tell, from a printed matrix alone, which convention was used?

^a<https://morrison-lab.github.io/mds/linear-algebra-vectors.html#def-transpose>

Solution

Solution 2.7. You can tell from the row and column sums whenever the two classes have different sizes. Here the rows sum to 1813 and 2788, which are the numbers of spam and email messages; the columns sum to 1943 and 2658, which are the numbers of messages the rule *called* spam and email. If you know how many messages of each class the data actually contain, the margin that matches is the one indexed by the true class.

Exercise 2.8 (Count the four ways a detector can be right or wrong (part 8)). Use the setup of Exercise 2.1.

Give a **confusion matrix** whose convention you could not tell from the printed matrix alone, as in Exercise 2.7.

Solution

Solution 2.8. You could not tell if the two margins happened to agree — for instance if the detector called exactly as many messages spam as there are spam messages. A symmetric^a matrix is the extreme case: it equals its own transpose^b, so no reading of the margins can separate the conventions. When in doubt, read the documentation of whatever produced the matrix, or feed it a case whose answer you know.

^a<https://morrison-lab.github.io/mds/linear-algebra-special-matrices.html#def-symmetric-matrix>

^b<https://morrison-lab.github.io/mds/linear-algebra-vectors.html#def-transpose>

i Watch

Starmer explains the confusion matrix^a [3].

^a<https://www.youtube.com/watch?v=Kdsp6soqA7o>

i Read

GeeksforGeeks explains the confusion matrix^a [4].

^a<https://www.geeksforgeeks.org/machine-learning/confusion-matrix-machine-learning/>

Definition 2.2 (Accuracy). The **accuracy** of a classifier on N examples is the fraction that it classifies correctly [5]. If M_{ij} is the number of examples in class i that it predicts as class j , with C classes,

$$\text{Acc} = \frac{\text{tr}(M)}{N} = \frac{\sum_{c=1}^C M_{cc}}{N}.$$

For a detection task (Definition 2.1), $\text{Acc} = (T_p + T_n)/N$.

Definition 2.3 (Error rate). The **error rate** of a classifier is the fraction of examples that it classifies incorrectly,

$$\text{Err} = 1 - \text{Acc},$$

where Acc is its accuracy (Definition 2.2).

Exercise 2.9 (Ask what a metric would say about a detector that does nothing). Accuracy is the fraction of messages the rule got right,

$$\begin{aligned} \text{Acc} &= \frac{\sum_i M_{ii}}{N} \\ &= \frac{T_p + T_n}{N}, \end{aligned} \quad \text{Err} = 1 - \text{Acc}.$$

Compute Acc and Err for the rule in Exercise 2.1.

Solution

Solution 2.9.

$$\begin{aligned} \text{Acc} &= (1217 + 2062)/4601 \\ &= 3279/4601 \\ &= 0.7127, \end{aligned}$$

so the rule labels about 71% of the 4601 messages correctly, and $\text{Err} = 0.2873$.

Exercise 2.10 (Ask what a metric would say about a detector that does nothing (part 2)). Use the setup of Exercise 2.9.

Now consider two detectors that ignore the message entirely: one calls everything spam, the other calls everything email. Give the **confusion matrix** of each, and compute the **accuracy** of each.

Solution

Solution 2.10. Call-everything-spam has $T_p = 1813$, $F_p = 2788$, and no negative calls at all:

$$M_{\text{all spam}} = \begin{bmatrix} 1813 & 0 \\ 2788 & 0 \end{bmatrix}, \quad \begin{aligned} \text{Acc} &= \frac{1813}{4601} \\ &= 0.3940. \end{aligned}$$

Call-everything-email is its mirror image:

$$M_{\text{all email}} = \begin{bmatrix} 0 & 1813 \\ 0 & 2788 \end{bmatrix}, \quad \begin{aligned} \text{Acc} &= \frac{2788}{4601} \\ &= 0.6060. \end{aligned}$$

Exercise 2.11 (Ask what a metric would say about a detector that does nothing (part 3)). Use the setup of Exercise 2.9.

Of the two do-nothing detectors of Exercise 2.10, which has the higher **accuracy**, and why?

Solution

Solution 2.11. Calling everything email wins, because email is the larger class.

Exercise 2.12 (Ask what a metric would say about a detector that does nothing (part 4)). Use the setup of Exercise 2.9.

State the rule that predicts which do-nothing detector of Exercise 2.10 wins, without computing anything.

Solution

Solution 2.12. The rule in general: a detector that always guesses one class gets exactly the frequency of that class, so the best do-nothing detector gets the frequency of the *majority class*. That number is the baseline any real detector has to beat, and it is a property of the data rather than of the detector.

Exercise 2.13 (Ask what a metric would say about a detector that does nothing (part 5)). Use the setup of Exercise 2.9.

How much **accuracy** did the hand-written rule gain over the better of the two do-nothing detectors?

Solution

Solution 2.13. $0.7127 - 0.6060 = 0.1067$. So two hand-picked columns out of 57, with two hand-picked thresholds, gain about eleven percentage points over guessing “email” every time. The rule’s **accuracy** has a 95% interval of [70.0, 72.6]% and the baseline’s [59.2, 62.0]%, each the proportion plus or minus 1.96 standard errors^a. They are scored on the same messages, so compare them message by message: the difference in the fraction right is 10.7 percentage points, with a 95% t interval of [8.8, 12.5], which excludes 0.

^a<https://morrison-lab.github.io/sds/estimation.html#def-SE>

Exercise 2.14 (Ask what a metric would say about a detector that does nothing (part 6)). Use the setup of Exercise 2.9.

Is the gain in **accuracy** found in Exercise 2.13 a lot?

Solution

Solution 2.14. Whether that is a lot depends on what you were expecting. It is a real improvement, and it is a modest one: a rule that got only these two numbers to look at is doing better than nothing and nowhere near well enough to run a mail system. Both halves of that sentence matter, and quoting 0.7127 on its own hides the second.

Exercise 2.15 (Ask what a metric would say about a detector that does nothing (part 7)). Use the setup of Exercise 2.9. Suppose instead that 50 messages in a million were spam. What **accuracy** does the “everything is email” detector get?

Solution

Solution 2.15. With 50 spam in 1,000,000 messages, calling everything email is right 999,950 times:

$$\begin{aligned} Acc &= \frac{999,950}{1,000,000} \\ &= 0.99995. \end{aligned}$$

Exercise 2.16 (Ask what a metric would say about a detector that does nothing (part 8)). Use the setup of Exercise 2.9. What does the **accuracy** found in Exercise 2.15 say about reporting **accuracy** on a rare event?

Solution

Solution 2.16. A detector that has never detected anything reports 99.995% **accuracy**. The number is not wrong — it correctly answers a question that nobody cares about. Accuracy weighs every message equally, and when almost every message is negative, it is almost entirely a measurement of the negatives. This is the single most common way a metric misleads, and it has a name: *class imbalance*. Exercise 2.59 returns to it once there are metrics that survive it.

Exercise 2.17 (Ask what a metric would say about a detector that does nothing (part 9)). Use the setup of Exercise 2.9. Prevalence is the fraction of the data that is positive. Express the **accuracy** of each do-nothing detector in terms of the prevalence π , and say which one wins as a function of π .

Solution

Solution 2.17. With prevalence $\pi = (T_p + F_n)/N$:

$$Acc_{\text{all spam}} = \pi, \quad Acc_{\text{all email}} = 1 - \pi.$$

Calling everything positive wins when $\pi > 1/2$, calling everything negative wins when $\pi < 1/2$, and at $\pi = 1/2$ they tie at 0.5. So the do-nothing baseline is $\max(\pi, 1 - \pi)$, which is never below 0.5 and rises toward 1 as the classes become more unbalanced. Here

$$\begin{aligned} \pi &= 1813/4601 \\ &= 0.3940, \end{aligned}$$

and the baseline is 0.6060.

i Other names for these four rates

Each of these four rates carries several names, and all of them are still in current use.

This chapter	Also called
Precision	positive predictive value (PPV)
Recall	sensitivity, true positive rate (TPR), hit rate
Specificity	true negative rate (TNR), selectivity
False positive rate (FPR)	false alarm rate, fall-out

The names come from three different fields. *Sensitivity*, *specificity* and *positive predictive value* are the medical and epidemiological names; *hit rate* and *false alarm rate* come from signal detection; *precision* and *recall* come from information retrieval. A paper reporting *sensitivity* is reporting recall.

The false positive rate is also written $1 - \text{specificity}$, which is a formula rather than a further name, and Exercise 2.18 asks you to compute it for a worked rule.

Definition 2.4 (Precision (also called positive predictive value)). The **precision**, also called the **positive predictive value** (PPV), of a detector is the fraction of its detections that are correct [6]. Write P for it:

$$P = \frac{T_p}{T_p + F_p}.$$

If the detector makes no detections, then $T_p + F_p = 0$ and the precision is undefined.

Definition 2.5 (Recall (also called sensitivity or true positive rate)). The **recall**, also called the **sensitivity** or the **true positive rate** (TPR), of a detector is the fraction of the positives that it detects [6]. Write R for it:

$$R = \frac{T_p}{T_p + F_n}.$$

If there are no positives, then $T_p + F_n = 0$ and the recall is undefined.

Definition 2.6 (Specificity (also called true negative rate)). The **specificity**, also called the **true negative rate** (TNR), of a detector is the fraction of the negatives that it correctly leaves undetected [7],

$$\text{TNR} = \frac{T_n}{F_p + T_n}.$$

If there are no negatives, then $F_p + T_n = 0$ and the specificity is undefined.

Exercise 2.18 (Split the errors: precision and recall). Accuracy adds the two kinds of error together. The next two metrics each look at one of them.

$$\text{Precision} = \frac{T_p}{T_p + F_p}, \quad \text{Recall} = \frac{T_p}{T_p + F_n}.$$

Compute both for the rule in Exercise 2.1.

Solution

Solution 2.18.

$$\begin{aligned}\text{Precision} &= 1217/(1217 + 726) \\ &= 1217/1943 \\ &= 0.6264\end{aligned}$$

and

$$\begin{aligned}\text{Recall} &= 1217/(1217 + 596) \\ &= 1217/1813 \\ &= 0.6713.\end{aligned}$$

So about 63% of the messages the rule sent to the spam folder belonged there, and about 67% of the spam was caught. Treating the messages as a sample, the 95% intervals are [60.5, 64.8]% for precision, from the 1943 flagged messages, and [65.0, 69.3]% for recall, from the 1813 spam messages. Each is the proportion plus or minus 1.96 standard errors, $\sqrt{\hat{p}(1-\hat{p})/n}$ with its own n , and it describes which messages happened to be in the data, since the rule was fixed in advance.

Exercise 2.19 (Split the errors: precision and recall (part 2)). Use the setup of Exercise 2.18. Each of the two is a fraction of one *margin* of the [confusion matrix](#). Which margin is which, and which of the two depends on the number of spam messages in the data?

Solution

Solution 2.19. [Precision](#) divides by $T_p + F_p = 1943$, the number of messages the detector *called* spam — a *column* sum, which the detector controls. [Recall](#) divides by $T_p + F_n = 1813$, the number of messages that *are* spam — a *row* sum, which is fixed by the data.

So recall has a denominator nobody can change, and precision has one the detector can change by flagging more or fewer messages. That asymmetry makes precision the harder of the two to interpret, so the next exercise sweeps a threshold rather than quoting a single pair.

Exercise 2.20 (Split the errors: precision and recall (part 3)). Use the setup of Exercise 2.18. A detector flags 20 messages, and 15 of them are spam. The data hold 60 spam messages. Compute its [precision](#) and [recall](#).

Solution

Solution 2.20. Before the change,

$$\begin{aligned}\text{Precision} &= 15/20 & \text{Recall} &= 15/60 \\ &= 0.75, & &= 0.25.\end{aligned}$$

Exercise 2.21 (Split the errors: precision and recall (part 4)). Use the setup of Exercise 2.18. Loosen the detector of Exercise 2.20 so that it flags 10 more messages, and 4 of those 10 are spam. Does [precision](#) go up or down?

Solution

Solution 2.21. After it, the detector flags 30 messages and $15 + 4 = 19$ are spam, so

$$\begin{aligned}\text{Precision} &= 19/30 & \text{Recall} &= 19/60 \\ &\approx 0.63, & &\approx 0.32.\end{aligned}$$

Precision goes down. The 10 new flags are spam only 40% of the time, which is below the old precision of 75%, so they pull the precision down.

Exercise 2.22 (Split the errors: precision and recall (part 5)). Use the setup of Exercise 2.18. After the detector of Exercise 2.21 is loosened, does **recall** go up or down?

Solution

Solution 2.22. **Recall** goes up. The 10 new flags catch 4 more of the 60 spam messages, so recall rises.

Exercise 2.23 (Split the errors: precision and recall (part 6)). Use the setup of Exercise 2.18. Why do **precision** and **recall** move as they do when the detector of Exercise 2.21 flags more messages?

Solution

Solution 2.23. Flagging more messages can only add true positives, so **recall** never falls; **precision** can move either way.

A mnemonic: precision is about the things you *said*, recall is about the things that *were*.

Exercise 2.24 (Split the errors: precision and recall (part 7)). Use the setup of Exercise 2.18. Compute **precision** and **recall** for the two do-nothing detectors of Exercise 2.9.

Solution

Solution 2.24. Calling everything spam gives $T_p = 1813$, $F_p = 2788$, $F_n = 0$:

$$\begin{aligned}\text{Precision} &= \frac{1813}{4601} & \text{Recall} &= 1. \\ &= 0.3940,\end{aligned}$$

Perfect **recall**, which is exactly as impressive as it sounds — you can have it at no cost. Its **precision** is the prevalence, which is the general rule: flag everything and your precision is whatever fraction of the data was positive.

Calling everything email gives $T_p = 0$, $F_p = 0$, $F_n = 1813$:

$$\begin{aligned}\text{Recall} &= \frac{0}{1813} & \text{Precision} &= \frac{0}{0} \text{ undefined.} \\ &= 0,\end{aligned}$$

Exercise 2.25 (Split the errors: precision and recall (part 8)). Use the setup of Exercise 2.18. Of the four numbers found in Exercise 2.24, one is not defined. Which one?

Solution

Solution 2.25. **Precision** of the “everything is email” detector is the number that is not defined.

Exercise 2.26 (Split the errors: precision and recall (part 9)). Use the setup of Exercise 2.18. Why is the number found in Exercise 2.25 not defined?

Solution

Solution 2.26. **Precision** is undefined because the detector never made a positive call, so there is no set of flagged messages to take a fraction of.

Exercise 2.27 (Split the errors: precision and recall (part 10)). Use the setup of Exercise 2.18. What would you report in place of the number found in Exercise 2.25?

Solution

Solution 2.27. The honest report is “**precision** undefined; no positive predictions”, not 0 and not 1. Software will often return 0 here, which is a convention rather than a computation — and it is a consequential one, because a metric of 0 reads as *tested and bad* rather than as *not tested*.

Exercise 2.28 (Split the errors: precision and recall (part 11)). Use the setup of Exercise 2.18. **Specificity** is $T_n/(T_n + F_p)$. Compute it for the rule. It looks like a mirror image of **precision** — say precisely what it measures that precision does not, using the row-and-column observation from Solution 2.7.

Solution

Solution 2.28.

$$\begin{aligned}\text{Specificity} &= 2062/(2062 + 726) \\ &= 2062/2788 \\ &= 0.7396.\end{aligned}$$

Specificity divides by $F_p + T_n = 2788$, the number of messages that *are* email — so, like **recall**, it is a row and not a column. **Precision** and specificity both react to false positives, but they normalize by different things: precision asks what fraction of the *flags* were wrong, and specificity asks what fraction of the *real email* got flagged.

The difference matters most under imbalance. Flag 726 of 2788 emails and specificity is a moderate 0.74; flag 726 of a million emails and specificity is 0.9993, while precision is unchanged if the spam count is unchanged. Specificity has the big class in its denominator, so a rare-event problem inflates it.

Exercise 2.29 (Split the errors: precision and recall (part 12)). Use the setup of Exercise 2.18. **Recall** is also called **sensitivity**, and $1 - \text{specificity}$ is the **false positive rate**. Give the false positive rate for the rule, and say which of the two classes each of sensitivity and specificity is computed from.

Solution

Solution 2.29.

$$\begin{aligned}\text{FPR} &= 1 - 0.7396 \\ &= 0.2604,\end{aligned}$$

or directly $726/2788$, with a 95% interval of $[24.4, 27.7]\%$ for the **false positive rate** (the same interval, reflected, for the **specificity**).

Sensitivity is computed entirely from the *positive* class: it is the spam row of the matrix. Specificity is computed entirely from the *negative* class: it is the email row. Neither one can be changed by adding or removing messages of the other class, which makes the pair the natural axes for the ROC curve in Exercise 2.52.

Exercise 2.30 (Split the errors: precision and recall (part 13)). Use the setup of Exercise 2.18. A colleague reports that their spam detector has 99% **precision**. Give two questions you would ask before believing the detector is good.

Solution

Solution 2.30. Two worth asking:

- **What is the recall?** Precision of 99% is trivial to achieve by flagging one message you are certain about and nothing else.
- **What is the prevalence in the data it was measured on?** If the test set was half spam, 99% precision is a real achievement. If it was tuned and measured on the same messages, or if the test set was built by collecting spam until there was a lot of it, the number does not transfer to a real inbox.

A third question: *on which messages?* A detector measured on last year's spam is being asked an easier question than the one it will face.

i Watch

Starmer explains sensitivity and specificity^a [8].

^a<https://www.youtube.com/watch?v=vP06aMoz4v8>

i Read

GeeksforGeeks explains precision and recall^a [9].

^a<https://www.geeksforgeeks.org/machine-learning/precision-and-recall-in-machine-learning/>

Multi-class metrics

Precision, recall and F_β are all defined for a detection task, so for $C > 2$ we reduce to detection first. The standard reduction is **one-vs-rest**. We take each class c in turn, relabel every point as c or $\neg c$, and read the resulting 2×2 matrix directly from M ,

$$T_p^{(c)} = M_{cc}, \quad F_n^{(c)} = \sum_{j \neq c} M_{cj}, \quad F_p^{(c)} = \sum_{i \neq c} M_{ic}, \quad T_n^{(c)} = N - T_p^{(c)} - F_n^{(c)} - F_p^{(c)}$$

The diagonal entry counts the true positives. The rest of row c counts the class- c points the model mislabeled, and the rest of column c counts the other points it mislabeled as c .

That leaves C binary problems and C values of whichever metric you wanted. Two conventions collapse them to one number. They differ only in when the averaging happens.

- **Macro-averaging** computes the metric per class, then averages:

$$P_{macro} = \frac{1}{C} \sum_{c=1}^C \frac{T_p^{(c)}}{T_p^{(c)} + F_p^{(c)}}, \quad R_{macro} = \frac{1}{C} \sum_{c=1}^C \frac{T_p^{(c)}}{T_p^{(c)} + F_n^{(c)}}$$

Every class contributes equally, however rare it is.

- **Micro-averaging** pools the counts, then computes the metric once:

$$P_{micro} = \frac{\sum_c T_p^{(c)}}{\sum_c T_p^{(c)} + \sum_c F_p^{(c)}}, \quad R_{micro} = \frac{\sum_c T_p^{(c)}}{\sum_c T_p^{(c)} + \sum_c F_n^{(c)}}$$

Every *point* contributes equally, so large classes dominate.

When every point gets exactly one predicted label, each mistake is a false positive for exactly one class and a false negative for exactly one other, so $\sum_c F_p^{(c)} = \sum_c F_n^{(c)}$ and the two micro quantities coincide with each other and with accuracy: $P_{micro} = R_{micro} = \text{Acc}$.

The same two conventions apply to F_β , and the order of operations matters more there than it does for P and R . We compute micro- F_1 from the pooled counts, so it is the F_1 formula applied to P_{micro} and R_{micro} — which makes it accuracy again when every point gets exactly one predicted label, since then $P_{micro} = R_{micro} = \text{Acc}$. Macro- F_1 is the mean of the C per-class F_1 scores, and that is **not** the F_1 formula applied to P_{macro} and R_{macro} : the formula is not linear in P and R , so averaging before combining and combining before averaging give different numbers.

- Discussion: under what class proportions and per-class scores do macro- and micro-averaging agree? What has to be true of the errors for the two to diverge sharply?

Balanced accuracy (higher is better)

R_{macro} has its own name, balanced accuracy (BAcc), because it fixes the most common failure of accuracy:

$$\text{BAcc} = \frac{1}{C} \sum_{c=1}^C \frac{T_p^{(c)}}{T_p^{(c)} + F_n^{(c)}}$$

Averaging the per-class recalls gives each class equal weight regardless of size, so a model that ignores a rare class scores 0 on that class's term however few points it contains [10]. A classifier guessing uniformly at random scores $1/C$ whatever the class proportions are, which accuracy does not.

Take $C = 3$ and $N = 90$, with M_{ij} the number of class- i points predicted as class j — rows are the true class, columns the prediction:

true class	pred 1	pred 2	pred 3	row total
1	3	7	0	10
2	0	50	0	50
3	0	12	18	30

The diagonal¹ sums to 71, so $\text{Acc} = 71/90 \approx 0.789$, which looks fairly high. The per-class **recalls** are $3/10 = 0.3$, $50/50 = 1.0$ and $18/30 = 0.6$, giving $\text{BAcc} = 1.9/3 \approx 0.633$. Class 2 accounts for the whole difference: it holds 50 of the 90 points and is classified perfectly, so under **accuracy** it dominates the score, while class 1 — right less than a third of the time — can only ever move the total by 10 points.

¹<https://morrison-lab.github.io/mds/linear-algebra-special-matrices.html#def-diagonal-matrix>

i Other names for F_1 , and one number that is not F_1

Balanced F-score is a fourth name for F_1 , alongside the three the chapter already gives. Image segmentation adds a fifth, the **Dice coefficient**, usually written as an agreement between two sets rather than as a combination of two rates:

$$\text{Dice} = \frac{2|A \cap B|}{|A| + |B|}.$$

Take A to be the items the model called positive and B the items that actually were. Then $|A \cap B| = T_p$, $|A| = T_p + F_p$, $|B| = T_p + F_n$, and the fraction becomes $2T_p/(2T_p + F_p + F_n)$, to which $2PR/(P + R)$ reduces. The two are one number under two names, not two measures that happen to agree.

The **Jaccard index**, also called **intersection over union** (IoU), looks like a sixth name and is not one:

$$\begin{aligned} J &= \frac{|A \cap B|}{|A \cup B|} \\ &= \frac{T_p}{T_p + F_p + F_n}. \end{aligned}$$

Both denominators count each mistake once. They differ in the true positives, which J counts once and F_1 counts twice. The two always rank a set of models the same way, since $J = F_1/(2 - F_1)$ is increasing in F_1 , but they are different numbers, and $J \leq F_1$ for every model, with equality only at 0 and at 1. How far apart they are depends on the value of F_1 . The gap $F_1 - J$ is widest in the middle of the range, reaching $3 - 2\sqrt{2} \approx 0.17$ at $F_1 = 2 - \sqrt{2} \approx 0.59$, and closes at both ends. The *proportion* $J/F_1 = 1/(2 - F_1)$ behaves differently: it falls steadily as the model gets worse, so a model scoring $F_1 = 0.99$ reports a Jaccard index 99% as large, while one scoring $F_1 = 0.1$ reports one only 53% as large. Quote whichever you like, and say which one it is.

Definition 2.7 (F-measure (also called F-score)). For the precision P and recall R of a detector (Definition 2.4, Definition 2.5) and a weight $\beta > 0$, the **F-measure**, also called the **F-score** [11], is

$$F_\beta = (1 + \beta^2) \frac{P \cdot R}{\beta^2 P + R}.$$

If $P = R = 0$, the denominator is 0 and the F-measure is undefined.

Definition 2.8 (F1 score). The **F1 score** of a detector is the F-measure with $\beta = 1$ (Definition 2.7), the harmonic mean of its precision P and recall R ,

$$F_1 = \frac{2PR}{P + R}.$$

Exercise 2.31 (Combine precision and recall into one number, carefully). We would like one number, and we have two. The usual way to combine them is the **F-measure**:

$$F_\beta = (1 + \beta^2) \frac{P \cdot R}{\beta^2 P + R}, \quad F_1 = 2 \frac{P \cdot R}{P + R}.$$

Compute F_1 for the rule in Exercise 2.1.

Solution

Solution 2.31. For the rule, $P = 0.6264$ and $R = 0.6713$, so

$$F_1 = 2 \cdot \frac{0.6264 \times 0.6713}{0.6264 + 0.6713} = 0.6480.$$

Exercise 2.32 (Combine precision and recall into one number, carefully (part 2)). Use the setup of Exercise 2.31.

Compute F_1 for the do-nothing detector of Exercise 2.9 that calls every message spam.

Solution

Solution 2.32. Calling everything spam has $P = 0.3940$ and $R = 1$, giving

$$F_1 = 2(0.3940)(1)/(1.3940) = 0.5653.$$

A detector that flags every single message scores $F_1 = 0.5653$, and the hand-written rule scores 0.6480. The rule is better, and not by as much as one might hope, and F_1 shows this in a way that accuracy did not.

```
# No formula gives a simple interval for F1, which is a ratio of counts,
# so resample the messages with replacement and recompute F1 each time.
set.seed(1)
spam_f1_boot <- replicate(1000, {
  i <- sample(spam_n, replace = TRUE)
  tp <- sum(spam_data$spam[i] == 1 & spam_pred_hand[i] == 1)
  fp <- sum(spam_data$spam[i] == 0 & spam_pred_hand[i] == 1)
  fn <- sum(spam_data$spam[i] == 1 & spam_pred_hand[i] == 0)
  2 * tp / (2 * tp + fp + fn)
})
spam_f1_ci <- quantile(spam_f1_boot, c(0.025, 0.975))
```

A 95% bootstrap interval for the rule's F_1 is [0.6308, 0.6663], the middle 95% of 1000 recomputations on messages drawn with replacement. It lies above the F_1 of 0.5653 for flagging everything.

Exercise 2.33 (Combine precision and recall into one number, carefully (part 3)). Use the setup of Exercise 2.31.

Compute F_1 for the do-nothing detector of Exercise 2.9 that calls every message email.

Solution

Solution 2.33. Calling everything email makes no positive calls, so P is 0/0 and F_1 cannot be computed.

Exercise 2.34 (Combine precision and recall into one number, carefully (part 4)). Use the setup of Exercise 2.31.

Is F_1 defined for the detector of Exercise 2.33, and why?

Solution

Solution 2.34. The precision P is undefined, so F_1 is undefined too. Even if you substitute $P = 0$ by convention, $R = 0$ as well, and the formula gives 0/0.

Exercise 2.35 (Combine precision and recall into one number, carefully (part 5)). Use the setup of Exercise 2.31.

Show that F_1 is the **harmonic mean** of P and R , that is, $F_1 = 2/(1/P + 1/R)$.

Solution

Solution 2.35. Substitute and simplify:

$$\begin{aligned}\frac{2}{\frac{1}{P} + \frac{1}{R}} &= \frac{2}{\frac{R+P}{PR}} \\ &= \frac{2PR}{P+R} \\ &= F_1.\end{aligned}$$

Exercise 2.36 (Combine precision and recall into one number, carefully (part 6)). Use the setup of Exercise 2.31.

Compare F_1 with the ordinary average $(P + R)/2$ for the pair $P = 0.99$, $R = 0.01$.

Solution

Solution 2.36. With $P = 0.99$ and $R = 0.01$:

$$\begin{aligned}\frac{P+R}{2} &= 0.50, & F_1 &= 2 \frac{0.99 \times 0.01}{1.00} \\ & & &= 0.0198.\end{aligned}$$

Exercise 2.37 (Combine precision and recall into one number, carefully (part 7)). Use the setup of Exercise 2.31.

Which of the two summaries in Exercise 2.36, F_1 or the ordinary average, would you rather a colleague reported, and why?

Solution

Solution 2.37. The arithmetic mean^a says “moderate”; F_1 says “this detector is nearly useless”, which is the truth — it flags almost nothing, and is right when it does. The harmonic mean is pulled toward the *smaller* of its arguments, so a good score on one axis cannot make up for a bad score on the other. That is the whole reason it is the mean used here.

In the extreme case this is clear: if either P or R is 0, then $F_1 = 0$, whereas the arithmetic mean would still report half of whatever the other one was.

^a<https://morrison-lab.github.io/sds/exploratory-descriptive.html#def-sample-mean>

Exercise 2.38 (Combine precision and recall into one number, carefully (part 8)). Use the setup of Exercise 2.31.

Take the limits $\beta \rightarrow 0$ and $\beta \rightarrow \infty$ in the formula for F_β . Which metric does each one recover?

Solution

Solution 2.38. As $\beta \rightarrow 0$ the β^2 terms vanish:

$$F_\beta \rightarrow \frac{PR}{R} = P.$$

As $\beta \rightarrow \infty$, divide numerator and denominator by β^2 :

$$F_\beta = \frac{(\frac{1}{\beta^2} + 1)PR}{P + \frac{R}{\beta^2}} \rightarrow \frac{PR}{P} = R.$$

So small β recovers [precision](#) and large β recovers [recall](#).

Exercise 2.39 (Combine precision and recall into one number, carefully (part 9)). Use the setup of Exercise 2.31.

From Exercise 2.38, a colleague concludes that β is “how much you care about [precision](#)”. Are they right or backwards?

Solution

Solution 2.39. They are backwards. Large β favors [recall](#), which is the opposite of what the symbol’s position in the formula suggests at a glance — β^2 multiplies P in the denominator, and a bigger denominator term for P makes P matter less.

Exercise 2.40 (Combine precision and recall into one number, carefully (part 10)). Use the setup of Exercise 2.31.

State in one sentence what the weight β in F_β controls.

Solution

Solution 2.40. The rule, in one sentence: β is how many times as much you care about [recall](#) as about [precision](#). F_2 counts recall twice as heavily; $F_{0.5}$ counts precision twice as heavily.

Exercise 2.41 (Combine precision and recall into one number, carefully (part 11)). Use the setup of Exercise 2.31.

A medical screening test is followed by a cheap confirmatory test, so a false positive costs a second test and a false negative costs a missed diagnosis. Would you tune β above or below 1?

Solution

Solution 2.41. Above 1. A false negative is the expensive error, [recall](#) is the metric that counts false negatives, and large β favors recall — so F_2 or a similar value. The cheap confirmatory test makes the false positives affordable.

Exercise 2.42 (Combine precision and recall into one number, carefully (part 12)). Use the setup of Exercise 2.31.

F_1 ignores T_n entirely — it appears nowhere in P or R . Give a situation where that is exactly what you want, and one where it hides something you need to know.

Solution

Solution 2.42. F_1 ignores the true negatives, so it is unmoved by how many uninteresting things the detector correctly ignored.

You want that when the negatives are arbitrary in number: a web search over ten billion pages that

returns ten good results has ten billion true negatives, and any metric containing T_n would be a measurement of the size of the web rather than of the search. Precision, recall and F_1 all stay the same if you add a billion irrelevant pages, which is the property wanted here.

It hides something when you need to know what fraction of the negatives were harmed. F_1 never divides by the number of emails, so the rule's $F_1 = 0.6480$ is unchanged whether its 726 false positives were drawn from the 2788 real emails here or from a million. Those are very different mail systems: in the first, $726/2788 = 26.04\%$ of your real mail is in the spam folder; in the second, 0.07% of it is. Specificity and the false positive rate see that difference, because the negative count is in their denominator, and F_1 cannot. When both classes matter, report the confusion matrix, not a summary of it.

i Read

GeeksforGeeks explains the F_1 score^a [12].

^a<https://www.geeksforgeeks.org/machine-learning/f1-score-in-machine-learning/>

Precision-recall curve

If the model outputs $P(\text{true}|x)$, then we can trade off precision against recall by adjusting the threshold, θ :

$$h(x) = I(P(\text{true}|x) > \theta)$$

- The area under the precision-recall curve (PR AUC) summarizes the curve in a single number

A random scorer's PR AUC is not 0.5; Exercise 2.59 asks what it is.

Exercise 2.43 (Move the threshold and watch the metrics move). Every metric so far was computed from one confusion matrix, which came from one rule. A model that outputs a probability does not give you one rule — it gives you a family of them, one per threshold:

$$h_\theta(\tilde{x}) = I(P(\text{spam} | \tilde{x}) > \theta).$$

Fitting the logistic regression of the logistic regression chapter to the same two columns, %george and %you, over all 4601 messages gives

$$P(\text{spam} | \tilde{x}) = \text{expit}(-0.6967 - 8.1739\% \text{george} + 0.3225\% \text{you}),$$

and sweeping θ gives:

Table 2: The same model at five thresholds.

θ	T_p	F_n	F_p	T_n	Precision	Recall	F_1	Accuracy
0.30	1805	8	2024	764	0.4714	0.9956	0.6398	0.5584
0.40	1435	378	906	1882	0.6130	0.7915	0.6909	0.7209
0.50	910	903	499	2289	0.6458	0.5019	0.5649	0.6953
0.60	398	1415	258	2530	0.6067	0.2195	0.3224	0.6364
0.70	94	1719	111	2677	0.4585	0.0518	0.0932	0.6023

Describe what happens to recall as θ rises in the table.

Solution

Solution 2.43. Recall falls all the way down the column: 0.9956, 0.7915, 0.5019, 0.2195, 0.0518.

Exercise 2.44 (Move the threshold and watch the metrics move (part 2)). Use the setup of Exercise 2.43. Prove it: show that raising θ can never increase [recall](#), whatever the data.

Solution

Solution 2.44. It has to. Raising θ can only *remove* messages from the set the model calls spam — a message with $P(\text{spam} \mid \tilde{x}) > \theta'$ also satisfies $P > \theta$ for any $\theta < \theta'$. So T_p can only fall. Meanwhile $T_p + F_n = 1813$ is the number of spam messages in the data, which no threshold can change. A quantity that can only fall, divided by a constant, can only fall.

Exercise 2.45 (Move the threshold and watch the metrics move (part 3)). Use the setup of Exercise 2.43. Now describe what happens to [precision](#). It is not monotone. Explain why your proof in Exercise 2.43 does not apply to precision.

Solution

Solution 2.45. [Precision](#) goes 0.4714, 0.6130, 0.6458, 0.6067, 0.4585: up, then down. The proof in Solution 2.43 worked because [recall](#)'s denominator was fixed. Precision's is not: its denominator is $T_p + F_p$, the number of messages the model *called* spam, and that shrinks as θ rises too. Both the numerator and the denominator fall, so the ratio can move in either direction.

Exercise 2.46 (Move the threshold and watch the metrics move (part 4)). Use the setup of Exercise 2.43. Give a concrete reason why [precision](#) might *fall* at a high threshold for this particular model, given that it is looking at only two words.

Solution

Solution 2.46. Look at what a high score tells you about this model. With only `%george` and `%you` to go on, the score is large when `george` never appears and `you` appears very often. The highest-scoring message in the data has `%you` = 18.75 and is *real email*; of the twenty highest-scoring messages, sixteen are real email. Among the top fifty, the spam fraction is 0.22 — *below* the overall prevalence of 0.3940, so at the extreme the model is worse than a random guess that says spam with probability equal to the overall spam rate.

Those messages are short personal notes. A model looking at two words cannot tell a chatty email from a sales pitch, and the higher you push θ , the more the surviving messages are selected for the one property the model mistakes for spam.

There is a second lesson underneath. `%george` averages 1.27 in email and 0.00 in spam, which made it look so useful in the course overview. But 72.31% of the real emails contain the word `george` zero times; the average comes from the minority that do, where it averages 4.57. A feature can separate the two classes *on average* and still say nothing about most individual cases.

Exercise 2.47 (Move the threshold and watch the metrics move (part 5)). Use the setup of Exercise 2.43. Which threshold in Table 2 maximizes [accuracy](#)?

Solution

Solution 2.47. [Accuracy](#) peaks at $\theta = 0.40$ with 0.7209, not at 0.50.

Exercise 2.48 (Move the threshold and watch the metrics move (part 6)). Use the setup of Exercise 2.43. Which maximizes F_1 ?

Solution

Solution 2.48. F_1 also peaks at $\theta = 0.40$ among these five, at 0.6909, and the true maximum over all thresholds is 0.7008 at $\theta = 0.3573$.

Exercise 2.49 (Move the threshold and watch the metrics move (part 7)). Use the setup of Exercise 2.43. Sweeping every distinct threshold, not just these five, the best F_1 is 0.7008 at $\theta = 0.3573$, with $P = 0.5908$ and $R = 0.8610$. What does that say about the habit of thresholding a probability at 0.5?

Solution

Solution 2.49. So 0.5 is not the best threshold by either measure here, and it is not better by accident: 0.5 is the threshold that minimizes the *error count* only when the two errors cost the same and the classes are balanced. Neither holds here. Thresholding at 0.5 is a default, not a decision — treat the threshold as a setting to be chosen from the cost of the two errors, and report which value you used.

Exercise 2.50 (Move the threshold and watch the metrics move (part 8)). Use the setup of Exercise 2.43. A **precision-recall curve** plots precision against recall as θ is swept. Sketch it from Table 2, mark the five points, and say in plain words what the bottom-right corner of such a plot would indicate.

Solution

Solution 2.50. The five points, as (recall, precision):

(0.9956, 0.4714), (0.7915, 0.6130), (0.5019, 0.6458), (0.2195, 0.6067), (0.0518, 0.4585).

Plotting **precision** up and **recall** across, the curve rises from the left, peaks near recall 0.5, and falls to the right (Figure 1).

The bottom-right corner is high recall with low precision: a detector that catches nearly all the spam by flagging nearly everything. The top-left corner is the opposite — flag one message you are sure of and nothing else. Both corners are reachable at no cost by any model at all, so the interesting part of the curve is the middle, and the corner values say more about θ than about the model.

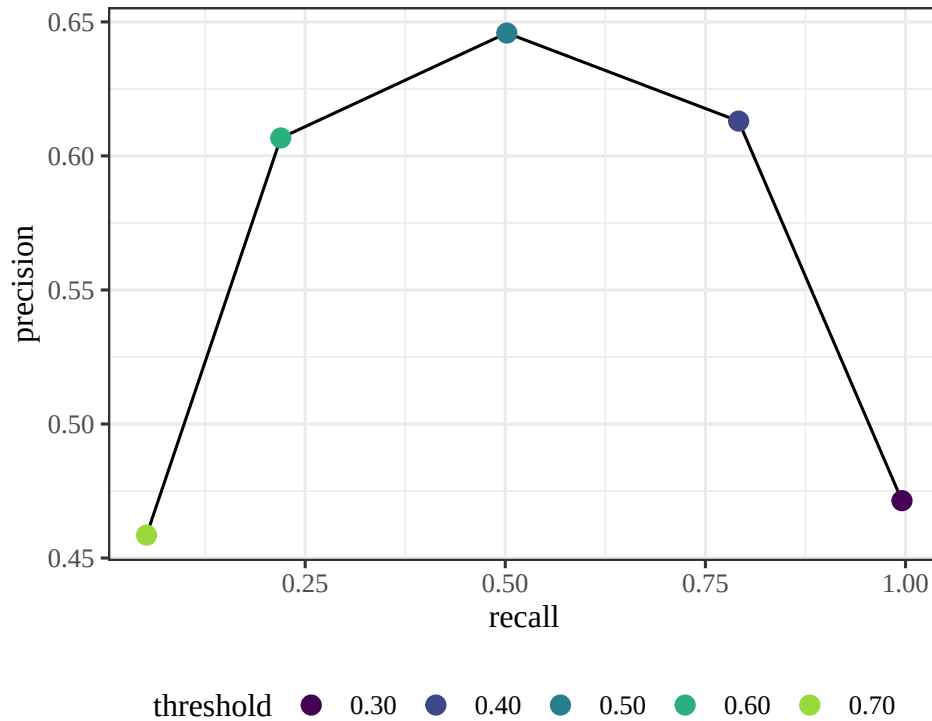


Figure 1: The precision-recall curve through the five thresholds 0.30 to 0.70 of Table 2, with recall across, precision up, and one point per threshold.

Exercise 2.51 (Move the threshold and watch the metrics move (part 9)). Use the setup of Exercise 2.43. Suppose the mail system may not put more than 1% of real email in the spam folder. Which column of Table 2 does that constrain, and which of the five thresholds satisfy it?

Solution

Solution 2.51. “No more than 1% of real email in the spam folder” is a cap on the false positive rate, $F_p/(F_p + T_n)$, with $F_p + T_n = 2788$. So it caps F_p at 27 messages.

None of the five thresholds satisfies it: the smallest F_p in the table is 111 at $\theta = 0.70$, a false positive rate of 0.0398. Raising the threshold further does not fix this. At $\theta = 0.85$ the model still lets $F_p = 29$ real emails through, a false positive rate of 0.0104, which is over the cap, and it catches $T_p = 5$ spam messages out of 1813 to do it.

That is the correct answer to the question, and it is a verdict on the model rather than on the threshold: two words cannot meet this requirement at any threshold. Report it that way.

2.2 Receiver operating characteristic curve and AUC ROC (Detection Only)

- The receiver operating characteristic (ROC) curve is similar to the precision-recall curve.
- As we sweep θ , the curve plots the tradeoff between
 - TPR, the true positive rate, which is the same as recall
 - FPR, the false positive rate, also called the false alarm rate, which is the fraction of the negatives that the detector wrongly detects
 - Show a ROC curve

- Exercise 2.55 asks what the curve looks like for a model that ignores its inputs² entirely.
- The area under the ROC curve (ROC AUC) summarizes the curve in a single number
- When should we use PR AUC, and when ROC AUC? Exercise 2.59 applies this question to a rare-event detection problem.

i Other names for ROC AUC, and one name to distrust

ROC AUC, *AUC ROC* and *AUROC* all name the area under the ROC curve. Biostatistics calls the same number the *c-statistic*, the **concordance statistic**, or the **concordance index**, and those names say what the area measures: draw one positive and one negative at random, and the area is the probability the model scores the positive higher, counting a tie as half. Nothing in that description mentions a curve, so a paper can report a *c-statistic* without plotting anything.

Distrust the bare word **AUC**. This chapter has two areas under two curves, ROC AUC and PR AUC, and a source that writes only *AUC* leaves you to guess which. The guess is not safe. The two areas have different denominators, they do not share a baseline, and on a rare-event problem the two views can describe the same pair of detectors very differently.

Name the curve whenever you report an area.

Definition 2.9 (False positive rate (also called false alarm rate)). The **false positive rate** (FPR), also called the **false alarm rate**, of a detector is the fraction of the negatives that it wrongly detects [13],

$$\text{FPR} = \frac{F_p}{F_p + T_n} = 1 - \text{TNR},$$

with the true negative rate of Definition 2.6. The ROC curve plots the true positive rate (the recall of Definition 2.5) against it as the threshold varies.

Exercise 2.52 (Read a ROC curve, and say what its diagonal means). Sweeping the threshold θ traces out a second curve, the **receiver operating characteristic**. It plots the true positive rate against the false positive rate:

$$\begin{aligned} \text{TPR} &= \frac{T_p}{T_p + F_n} & \text{FPR} &= \frac{F_p}{F_p + T_n} \\ &= R, \end{aligned}$$

Both axes of a ROC curve are rates, and both denominators are fixed before the model sees anything. Say what each denominator counts, and why neither one changes as θ moves.

Solution

Solution 2.52. *TPR* divides by $T_p + F_n$, the number of items that really are positive. *FPR* divides by $F_p + T_n$, the number that really are negative. Both counts are properties of the labelled data, not of the detector, so moving θ only moves items between the numerator and the rest of its own column. So the two axes are comparable across detectors on the same data.

Exercise 2.53 (Read a ROC curve, and say what its diagonal means (part 2)). Use the setup of Exercise 2.52.

Locate the end of the curve where a detector sits when $\theta \rightarrow 1$.

²<https://morrison-lab.github.io/sds/model-validation.html#def-data-set>

Solution

Solution 2.53. At $\theta \rightarrow 1$ the detector calls nothing positive, so

$$\begin{aligned} T_p &= F_p \\ &= 0 \end{aligned}$$

and the curve is at the origin $(0, 0)$.

Exercise 2.54 (Read a ROC curve, and say what its diagonal means (part 3)). Use the setup of Exercise 2.52.

Locate the end of the curve where a detector sits when $\theta \rightarrow 0$.

Solution

Solution 2.54. At $\theta \rightarrow 0$ it calls everything positive, so

$$\begin{aligned} F_n &= T_n \\ &= 1 \end{aligned}$$

and

$$\begin{aligned} TPR &= FPR \\ &= 1, \end{aligned}$$

the far corner $(1, 1)$. Every ROC curve therefore runs between the same two points, so the area between them is a fair comparison.

Exercise 2.55 (Read a ROC curve, and say what its diagonal means (part 4)). Use the setup of Exercise 2.52.

Suppose the model assigns every message an independent^a uniform random score, ignoring its contents. Show that the expected^b curve is the line $TPR = FPR$, and compute the area under it.

^a<https://morrison-lab.github.io/pds/independence.html#def-indpt>

^b<https://morrison-lab.github.io/pds/expectation.html#def-expectation>

Solution

Solution 2.55. If the score is independent^a of the label, then for any threshold θ the event “score exceeds θ ” is independent of the class. So the fraction of positives above the threshold and the fraction of negatives above it are the same number, namely $1 - \theta$ for a uniform score. Then

$$\begin{aligned} TPR(\theta) &= 1 - \theta \\ &= FPR(\theta), \end{aligned}$$

which is the line $y = x$ drawn from $(1, 1)$ down to $(0, 0)$. The area under it is the area of a triangle with base and height 1, so $ROC\ AUC = 1/2$.

This is the baseline the number is read against: 0.5 is worthless, and a detector scoring 0.5 has learned nothing that separates the classes.

^a<https://morrison-lab.github.io/pds/independence.html#def-indpt>

Exercise 2.56 (Read a ROC curve, and say what its diagonal means (part 5)). Use the setup of Exercise 2.52.

A detector’s ROC curve dips below that diagonal over some stretch. Describe a change you could make

to the detector, without retraining it, that would move the curve above the diagonal there.

Solution

Solution 2.56. Invert its decisions on that stretch. A curve below the diagonal^a says the detector is ranking negatives above positives there, so swapping the labels it emits, or equivalently replacing the score s by $-s$, reflects the curve through the diagonal.

A detector reliably worse than chance is still informative, because you can always read it backwards. This is also why *ROC AUC* below 0.5 should prompt you to look for a sign error before you conclude anything about the data.

^a<https://morrison-lab.github.io/mds/linear-algebra-special-matrices.html#def-diagonal-matrix>

Exercise 2.57 (Read a ROC curve, and say what its diagonal means (part 6)). Use the setup of Exercise 2.52.

Two detectors have the same ROC AUC, and their curves cross. Explain why the one you should deploy depends on something the AUC does not contain.

Solution

Solution 2.57. The AUC is an average over every threshold, including thresholds you would never deploy. Two curves that cross show that one detector is better in the low-*FPR* region and the other is better in the high-*TPR* region, and which region you operate in is a fact about your costs, not about the model. A mail system operates at very low *FPR*, because deleting real mail is expensive, so it wants the detector that wins on the left of the plot. A cheap screening test feeding a confirmatory test operates at high *TPR* and wants the other one. The single number averaged those two verdicts together and reported that they tie.

Exercise 2.58 (Read a ROC curve, and say what its diagonal means (part 7)). Use the setup of Exercise 2.52.

The precision-recall curve shares one axis with the ROC curve and replaces the other. Say which axis is shared, and give the denominator of the axis that changed.

Solution

Solution 2.58. **Recall** is shared: it is the y axis of the ROC curve and the x axis of the precision-recall curve.

The axis that changed is **precision**, whose denominator is $T_p + F_p$, the number of items the detector *called* positive. That denominator is chosen by the detector rather than fixed by the data, which is exactly the difference that Exercise 2.59 turns on.

i Watch

Starmer builds confusion matrices at different thresholds, from 4:03^a, then the ROC curve, from 7:12^b, and the AUC, from 13:44^c [14].

^a<https://www.youtube.com/watch?v=4jRBRDbJemM&t=243s>

^b<https://www.youtube.com/watch?v=4jRBRDbJemM&t=432s>

^c<https://www.youtube.com/watch?v=4jRBRDbJemM&t=824s>

i Watch

Hastie and Tibshirani draw ROC curves in Python^a [15], video 9.Py.

^a<https://www.youtube.com/watch?v=1fmHMmoa47g>

i Read

GeeksforGeeks explains the ROC curve and AUC^a [16].

^a<https://www.geeksforgeeks.org/machine-learning/auc-roc-curve/>

Exercise 2.59 (Choose between PR AUC and ROC AUC on a rare event). A factory's inspection model runs over $N = 400,000$ parts, of which 40 are defective. Two candidate models catch the same defects and differ only in how often they flag a good part:

	T_p	F_n	F_p	T_n
Model A	32	8	64	399,896
Model B	32	8	3,200	396,760

Check that each row accounts for all 400,000 parts.

Solution

Solution 2.59. Model A: $32 + 8 + 64 + 399,896 = 400,000$. Model B: $32 + 8 + 3,200 + 396,760 = 400,000$.

Exercise 2.60 (Choose between PR AUC and ROC AUC on a rare event (part 2)). Use the setup of Exercise 2.59.

Check that both models have the same [recall](#).

Solution

Solution 2.60. Both have $T_p = 32$ and $F_n = 8$, so both have

$$\begin{aligned} R &= \frac{32}{32 + 8} \\ &= 0.8. \end{aligned}$$

[Recall](#) cannot tell them apart, because recall never looks at the negatives.

Exercise 2.61 (Choose between PR AUC and ROC AUC on a rare event (part 3)). Use the setup of Exercise 2.59.

Compute FPR for each model.

Solution

Solution 2.61. There are 40 defective parts and therefore 399,960 good parts, which is the denominator of FPR for both rows:

$$\begin{aligned} FPR_A &= \frac{64}{399,960} & FPR_B &= \frac{3,200}{399,960} \\ &= 1.60 \times 10^{-4}, & &= 8.00 \times 10^{-3}. \end{aligned}$$

Exercise 2.62 (Choose between PR AUC and ROC AUC on a rare event (part 4)). Use the setup of Exercise 2.59. Compute [precision](#) for each model.

Solution

Solution 2.62. [Precision](#) divides by what the model flagged:

$$\begin{aligned} P_A &= \frac{32}{96} & P_B &= \frac{32}{3,232} \\ &= 0.3333, & &= 0.009901. \end{aligned}$$

Exercise 2.63 (Choose between PR AUC and ROC AUC on a rare event (part 5)). Use the setup of Exercise 2.59. Report the ratio between the two models on *FPR* and on [precision](#).

Solution

Solution 2.63. Model B's [false positive rate](#) is 50 times A's, and its [precision](#) is $0.3333/0.009901 = 33.7$ times worse. The two measures agree on the direction and disagree on the size.

Exercise 2.64 (Choose between PR AUC and ROC AUC on a rare event (part 6)). Use the setup of Exercise 2.59.

A ROC curve through the operating points of the two models would be hard to tell apart by eye. Explain why, using the denominators you wrote down in Exercise 2.52.

Solution

Solution 2.64. *FPR* divides by 399,960, a number so much larger than either F_p that both points sit essentially on the left edge of the plot: one at $x = 0.00016$ and the other at $x = 0.008$, against an axis that runs to 1. Plotted at any ordinary scale, a fiftyfold difference between two quantities that are both nearly zero is invisible.

[Precision](#) divides by $T_p + F_p$, which is 96 for A and 3,232 for B. That denominator is small, and it is set by the model rather than by the data, so the same change in F_p moves precision across most of its range. The lesson is not that *FPR* is wrong. It is that when the good parts outnumber the defective ones by four orders of magnitude, the count of good parts in the denominator hides the very difference you are trying to see.

Exercise 2.65 (Choose between PR AUC and ROC AUC on a rare event (part 7)). Use the setup of Exercise 2.59.

You are the engineer who must pull the flagged parts for hand inspection. How many parts do you pull per defect you find, under each model?

Solution

Solution 2.65. Model A flags 96 parts and 32 are defective, so you pull $96/32 = 3$ parts per defect found. Model B flags 3,232 parts for the same 32 defects, so you pull $3,232/32 = 101$ parts per defect found. That number is $1/P$: [precision](#)'s reciprocal is the work per success, and it is the quantity the engineer experiences directly. Switching from A to B multiplies the inspection queue by about 34 while catching not one additional defect.

Exercise 2.66 (Choose between PR AUC and ROC AUC on a rare event (part 8)). Use the setup of Exercise 2.59.

A random-scoring model has $ROC\ AUC = 1/2$ whatever the data. What is the corresponding baseline for PR AUC on this dataset?

Solution

Solution 2.66. A random-scoring model that flags a fraction θ of parts catches that same fraction of the defects, so its **precision** is the prevalence, whatever θ is:

$$\frac{40}{400,000} = 1 \times 10^{-4}.$$

The baseline for PR AUC is therefore the prevalence, and it moves with the dataset, where the ROC baseline is always 1/2.

Exercise 2.67 (Choose between PR AUC and ROC AUC on a rare event (part 9)). Use the setup of Exercise 2.59.

Why does the difference between the two baselines of Exercise 2.66 matter when you report a PR AUC of, say, 0.30?

Solution

Solution 2.67. So a PR AUC of 0.30 cannot be interpreted until you say what the prevalence is. Here it would be 3000 times the random baseline, which is a strong model. On a balanced dataset the baseline is 0.5 and the same 0.30 is worse than guessing. Always report the prevalence beside a PR AUC.

Exercise 2.68 (Choose between PR AUC and ROC AUC on a rare event (part 10)). Use the setup of Exercise 2.59.

State the rule you would give a colleague for choosing between the two curves, in terms of which quantity the problem makes large.

Solution

Solution 2.68. Use the precision-recall curve when the negatives are the large class and the cost of a false positive falls on someone who has to handle each one. Use the ROC curve when the two classes are comparable in size, or when you genuinely care about the rate at which negatives are disturbed rather than the composition of the flagged set.

The compact form: if T_n is enormous and uninteresting, keep it out of the denominator.

i Watch

Starmer also draws precision-recall curves, from 14:28^a [14].

^a<https://www.youtube.com/watch?v=4jRBRDbJemM&t=868s>

Exercise 2.69 (Rank single features of real emails by ROC AUC). The Spambase data [17] describe 4601 emails by 57 numeric features, with a label `Class` that is 1 for spam and 0 for email. Download it from the UCI repository^a, or use the copy `data/spambase.csv`^b on this site. Any computer with R or Python can run the code in the solution.

A single feature can act as a score: call an email spam when the feature is at least θ . Take four features: `word_freq_remove`, `word_freq_george`, `char_freq_!` and `capital_run_length_average`.

Before computing anything, predict which feature gives the highest ROC AUC. Predict which gives an AUC below 0.5, and say what that implies for the rule “spam when the feature is large”.

^a<https://doi.org/10.24432/C53G6X>

^b<https://morrison-lab.github.io/lds/chapters/data/spambase.csv>

Solution

Solution 2.69. Spam often contains words like “remove” and many exclamation marks, and long runs of capital letters. Emails to George (the recipient whose name appears in the data) are rarely spam, so a large `word_freq_george` points to a real email. That feature should have an AUC below 0.5: the rule “spam when the feature is large” ranks spam below email more often than not.

Exercise 2.70 (Rank single features of real emails by ROC AUC (part 2)). Use the setup of Exercise 2.69. Compute the ROC AUC of each feature. Use the fact that the AUC equals the probability that a randomly chosen spam email scores higher than a randomly chosen email that is not spam, counting ties as one half.

Solution

Solution 2.70.

2.2.1 Python

```
import numpy as np
import pandas as pd

spam_py = pd.read_csv("data/spambase.csv")
feats = ["word_freq_remove", "word_freq_george",
         "char_freq!", "capital_run_length_average"]

def auc_py(score, y):
    r = score.rank()          # ties get their average rank
    n1 = int((y == 1).sum())
    n0 = int((y == 0).sum())
    return (r[y == 1].sum() - n1 * (n1 + 1) / 2) / (n1 * n0)

roc_auc_py = pd.Series(
    {f: auc_py(spam_py[f], spam_py["Class"]) for f in feats})
roc_auc_py

word_freq_remove          0.703130
word_freq_george          0.363332
char_freq!                0.829046
capital_run_length_average 0.788200
dtype: float64
```

```
# No simple formula gives an interval for an AUC, so resample the emails with
# replacement and recompute each feature's AUC every time.
rng = np.random.default_rng(1)
auc_boot_py = np.array([
    auc_py(spam_py[f].iloc[i].reset_index(drop=True),
           spam_py["Class"].iloc[i].reset_index(drop=True)) for f in feats]
    for i in (rng.integers(0, len(spam_py), len(spam_py)) for _ in range(500))
])
print(pd.DataFrame(np.percentile(auc_boot_py, [2.5, 97.5], axis=0),
                    index=["2.5%", "97.5%"], columns=feats).round(3))
```

	word_freq_remove	...	capital_run_length_average
2.5%	0.692	...	0.775
97.5%	0.713	...	0.802

[2 rows x 4 columns]

2.2.2 R

```
spam_all <- read.csv("data/spambase.csv", check.names = FALSE)
feats <- c("word_freq_remove", "word_freq_george",
          "char_freq_!", "capital_run_length_average")

auc_r <- function(score, y) {
  r <- rank(score) # ties get their average rank
  n1 <- sum(y == 1)
  n0 <- sum(y == 0)
  (sum(r[y == 1]) - n1 * (n1 + 1) / 2) / (n1 * n0)
}

roc_auc <- sapply(feats, function(f) auc_r(spam_all[[f]], spam_all$Class))
roc_auc
```

	word_freq_remove	word_freq_george
	0.7031303	0.3633321
	char_freq_!	capital_run_length_average
	0.8290461	0.7882004

```
# No simple formula gives an interval for an AUC, so resample the emails with
# replacement and recompute each feature's AUC every time.
set.seed(1)
auc_boot <- replicate(500, {
  i <- sample(nrow(spam_all), replace = TRUE)
  sapply(feats, function(f) auc_r(spam_all[[f]][i], spam_all$Class[i]))
})
auc_ci <- apply(auc_boot, 1, quantile, c(0.025, 0.975))
round(auc_ci, 3)
```

	word_freq_remove	word_freq_george	char_freq_!	capital_run_length_average
2.5%	0.693	0.355	0.817	0.775
97.5%	0.715	0.373	0.841	0.803

```
# The claims the prose makes: both languages agree, one feature is below 0.5,
# and the flipped rule has AUC 1 - AUC.
stopifnot(
  max(abs(roc_auc - unlist(reticulate::py$roc_auc_py)[feats])) < 1e-9,
  sum(roc_auc < 0.5) == 1,
  abs(auc_r(-spam_all$word_freq_george, spam_all$Class) -
    (1 - roc_auc[["word_freq_george"]])) < 1e-9
)
best <- names(which.max(roc_auc))
worst <- names(which.min(roc_auc))
```

The best feature is `char_freq_!`, with AUC 0.83. The worst is `word_freq_george`, with AUC 0.36. The AUCs of the four features, with 95% bootstrap intervals, are `word_freq_remove` 0.70 [0.69, 0.71], `word_freq_george` 0.36 [0.35, 0.37], `char_freq_!` 0.83 [0.82, 0.84], `capital_run_length_average` 0.79 [0.77, 0.80]. Each interval is the middle 95% of 500 AUCs recomputed on emails drawn with replacement. Figure 2 shows the curves for the best and worst features.

Exercise 2.71 (Rank single features of real emails by ROC AUC (part 3)). Use the setup of Exercise 2.69. Plot the ROC curves of the best and worst features on one set of axes, with the diagonal for reference.

Solution

Solution 2.71. See Figure 2. The curve of the best feature rises above the diagonal. The curve of the worst feature stays below it.

```

roc_points <- function(score, y) {
  th <- c(Inf, sort(unique(score), decreasing = TRUE))
  tibble::tibble(
    fpr = sapply(th, function(t) mean(score[y == 0] >= t)),
    tpr = sapply(th, function(t) mean(score[y == 1] >= t))
  )
roc_best <- roc_points(spam_all[[best]], spam_all$Class)
roc_worst <- roc_points(spam_all[[worst]], spam_all$Class)
library(ggplot2)
roc_df <- dplyr::bind_rows(
  dplyr::mutate(roc_best, feature = best),
  dplyr::mutate(roc_worst, feature = worst)
)

# A bootstrap band: the true positive rate at each false positive rate on a grid,
# recomputed on emails drawn with replacement. Sorting by score gives every
# threshold's rates at once, so each resample is quick.
fpr_grid <- seq(0, 1, length.out = 101)
tpr_on_grid <- function(score, y) {
  ord <- order(score, decreasing = TRUE)
  # Keep one point per distinct score, the last of each tie block, so a block of
  # tied scores moves in one jump, as in the curve drawn from roc_points().
  last_of_tie <- !duplicated(score[ord], fromLast = TRUE)
  fpr <- c(0, (cumsum(y[ord] == 0) / sum(y == 0))[last_of_tie])
  tpr <- c(0, (cumsum(y[ord] == 1) / sum(y == 1))[last_of_tie])
  approx(fpr, tpr, xout = fpr_grid, ties = "ordered")$y
}
set.seed(1)
band_df <- dplyr::bind_rows(lapply(c(best, worst), function(f) {
  boot <- replicate(200, {
    i <- sample(nrow(spam_all), replace = TRUE)
    tpr_on_grid(spam_all[[f]][i], spam_all$Class[i])
  })
  tibble::tibble(
    fpr = fpr_grid,
    lower = apply(boot, 1, quantile, 0.025),
    upper = apply(boot, 1, quantile, 0.975),
    feature = f
  )
}))

ggplot(roc_df, aes(fpr, tpr, linetype = feature)) +
  annotate("segment", x = 0, y = 0, xend = 1, yend = 1, linetype = "dotted") +
  geom_ribbon(aes(y = NULL, ymin = lower, ymax = upper, fill = feature),
    data = band_df, alpha = 0.2, colour = NA, show.legend = FALSE) +
  geom_line() +
  scale_linetype_manual(values = setNames(c("solid", "dashed"), c(best, worst))) +
  coord_equal() +
  labs(x = "false positive rate", y = "true positive rate", linetype = NULL) +
  theme(legend.position = "bottom")

```

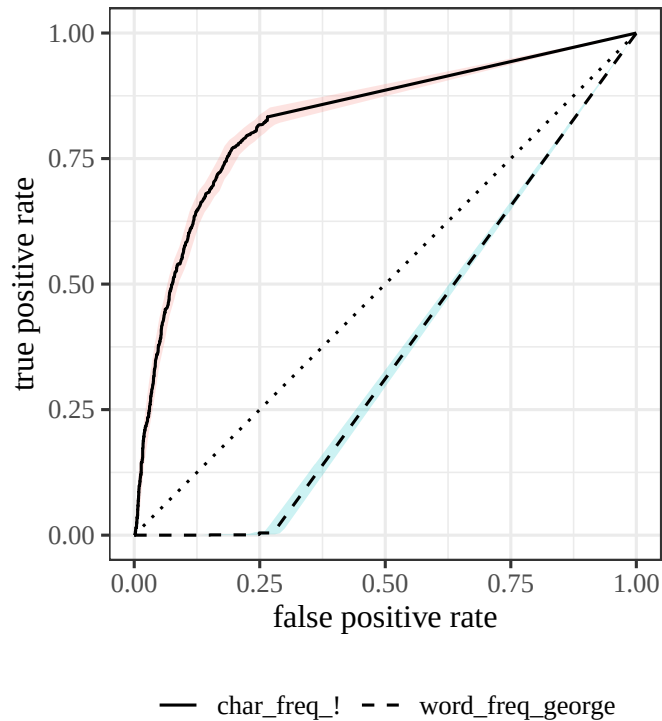


Figure 2: ROC curves on the Spambase emails for the best single feature (solid) and the worst (dashed), with the diagonal for a score that ignores the email (dotted). The worst feature’s curve lies below the diagonal. Each shaded band is a pointwise 95% bootstrap interval for the true positive rate at each false positive rate, from 200 resamples of the emails.

Exercise 2.72 (Rank single features of real emails by ROC AUC (part 4)). Use the setup of Exercise 2.69. The feature with AUC below 0.5 can still be useful. Say what you would change in the rule, and what its AUC becomes.

Solution

Solution 2.72. Flip the rule: call an email spam when the feature is *small*. Negating the score swaps every pair of a spam and an email, so the new AUC is $1 - \text{old AUC}$, here 0.64 for word_freq_george. An AUC below 0.5 does not imply a feature is useless; it implies the direction of the rule is backwards. A feature with an AUC near 0.5 is the one that carries no information.

2.3 Differentiable

2.3.1 Cross-entropy (lower is better)

If $h(x_{(i)}) \in \mathbb{R}^C$ contains the posterior probabilities of the C classes and $y_{(i)} \in \mathbb{R}^C$ is a one hot vector, then

$$H(y, h(x)) = - \sum_{i=1}^N \sum_{j=1}^C y_{(i)j} \log_b h(x_{(i)})_j$$

Units depend on b :

- $b = 2$: bits
- $b = e$: nats

- $b = 10$: hartleys

Exercise 2.73 (Reading the cross-entropy sum). A classifier outputs $h(x_{(i)}) \in \mathbb{R}^C$, a vector of posterior probabilities over the C classes, and the label $y_{(i)} \in \mathbb{R}^C$ is one-hot: it holds a 1 in the position of the true class and 0 everywhere else. The cross-entropy over a set of N examples is

$$H(y, h(x)) = - \sum_{i=1}^N \sum_{j=1}^C y_{(i)j} \log_b h(x_{(i)})_j. \quad (1)$$

The inner sum runs over all C classes, but only one term of it is nonzero. Say which term.

Solution

Solution 2.73. Only the true class contributes. Write c_i for the index where $y_{(i)}$ holds its 1. Every other $y_{(i)j}$ is 0, so every other term of the inner sum is $0 \cdot \log_b h(x_{(i)})_j = 0$.

Exercise 2.74 (Reading the cross-entropy sum (part 2)). Use the setup of Exercise 2.73. Write what Equation 1 reduces to once the one-hot structure is used.

Solution

Solution 2.74. Only the term for c_i is left, so

$$H(y, h(x)) = - \sum_{i=1}^N \log_b h(x_{(i)})_{c_i}.$$

The probabilities the classifier assigns to the wrong classes never appear. They matter only because the vector sums to 1, so probability spent on a wrong class is probability taken from the right one.

Exercise 2.75 (Reading the cross-entropy sum (part 3)). Use the setup of Exercise 2.73. What is the smallest value H can take, and what must the classifier do to reach it?

Solution

Solution 2.75. The smallest value is 0. A probability is at most 1, so $\log_b h(x_{(i)})_{c_i} \leq 0$ and each term of the sum above is at least 0. Equality needs $h(x_{(i)})_{c_i} = 1$ for every i : the classifier must put all of its probability on the true class, every time.

Exercise 2.76 (Reading the cross-entropy sum (part 4)). Use the setup of Exercise 2.73. What is the largest value H can take?

Solution

Solution 2.76. There is no largest value. As $h(x_{(i)})_{c_i} \rightarrow 0$, $-\log_b h(x_{(i)})_{c_i} \rightarrow \infty$. One confident mistake is enough: a classifier that assigns the true class a probability of 10^{-12} on a single example contributes 12 hartleys from that example alone. This asymmetry is the metric working as intended. [Accuracy](#) charges 1 for a mistake whether the classifier was hesitant or certain; cross-entropy^a charges for the certainty.

^a<https://morrison-lab.github.io/pds/expectation.html#def-cross-entropy>

Exercise 2.77 (Reading the cross-entropy sum (part 5)). Use the setup of Exercise 2.73. The base b names the units: 2 gives bits, e gives nats, 10 gives hartleys. Show that changing b multiplies H by a positive constant, and say what that implies for comparing two classifiers.

Solution

Solution 2.77. The base is a scale factor. By the change-of-base identity, $\log_b z = \ln z / \ln b$, so

$$H_b = \frac{H_e}{\ln b},$$

and $\ln b > 0$ for any $b > 1$. So the three unit choices give three numbers that are positive multiples of each other: 1.386 nats is 2 bits is 0.602 hartleys.

Multiplying every score by the same positive constant^a leaves their order alone, so a classifier that beats another in bits beats it in nats too, and the base cannot change which model you pick. It changes only the units in which the number is expressed.

^a<https://morrison-lab.github.io/mds/algebra-exponentials.html#def-constant-function>

Theorem 2.1 (Cross-entropy with one-hot labels). *Let each label $y_{(i)}$ be one-hot, with its 1 at position c_i . Then the cross-entropy of Equation 1 reduces to the negative log probability that the classifier assigns to the true class of each example:*

$$H(y, h(x)) = - \sum_{i=1}^N \log_b h(x_{(i)})_{c_i}. \quad (2)$$

i Proof

Proof. By Solution 2.73. □

Example 2.1. A classifier with $C = 3$ classes outputs $h(x) = (\frac{1}{2}, \frac{1}{4}, \frac{1}{4})$ for one example. In bits ($b = 2$), Equation 2 gives $-\log_2 \frac{1}{2} = 1$ if the true class is the first, and $-\log_2 \frac{1}{4} = 2$ if it is the second or the third. The probability on the other two classes never enters.

Theorem 2.2 (Cross-entropy with one-hot labels is nonnegative). *With one-hot labels, the cross-entropy of Equation 1 satisfies $H(y, h(x)) \geq 0$. It equals 0 exactly when the classifier puts probability 1 on the true class of every example.*

i Proof

Proof. This is the solution to Exercise 2.75: see Solution 2.75. □

Example 2.2 (Cross-entropy of two classifiers). On two examples, a classifier assigns the true class the probabilities 0.8 and 0.5. A second classifier assigns it 1 and 1.

```
cn_true_prob <- list(first = c(0.8, 0.5), second = c(1, 1))
vapply(cn_true_prob, \(p) -sum(log2(p)), numeric(1))
```

```
first second
1.321928 0.000000
```

By Theorem 2.2 both values are at least 0. The first classifier scores 1.322 bits. The second scores exactly 0, because it puts all its probability on the true class every time.

Theorem 2.3 (Changing the base of cross-entropy rescales it). Let H_b be the cross-entropy of Equation 1 in base $b > 1$. Then

$$H_b = \frac{H_e}{\ln b}.$$

Two classifiers are therefore ordered the same way in every base.

i Proof

Proof. This is the solution to Exercise 2.77: see Solution 2.77. □

Example 2.3 (One classifier in three units). A classifier assigns the true class the probabilities 0.8 and 0.5 on two examples.

```
cb_p <- c(0.8, 0.5)
cb_h <- c(bits = -sum(log2(cb_p)), nats = -sum(log(cb_p)), hartleys = -sum(log10(cb_p)))
rbind(value = cb_h, nats_over_ln_b = cb_h[["nats"]] / log(c(2, exp(1), 10)))
```

	bits	nats	hartleys
value	1.321928	0.9162907	0.39794
nats_over_ln_b	1.321928	0.9162907	0.39794

The second row is $H_e / \ln b$ for $b = 2, e, 10$, and by Theorem 2.3 it matches the first row. Every classifier's score is multiplied by the same $1/\ln b$, so the ordering of classifiers does not change with the base.

i Other names for cross-entropy, and two conventions to state

Software and statistics each renamed this quantity. **Log loss** and **logistic loss** are the same number, and so is the **negative log-likelihood** of the labels under the model, since $-\log h(x_{(i)})_j$ at the true class j is exactly the log-likelihood of that label with its sign reversed. Generalized linear models add **deviance**, which for one-hot labels is twice the summed negative log-likelihood.

Settle two conventions before comparing one of these numbers with another source's. The notation choices page^a explains both.

Which base. The definition of H leaves the base b free. Theorem 2.3 shows the base cannot change which of two classifiers you would pick, but it does change the number itself, so state the base whenever you quote one.

Summed or averaged. The chapter's $H(y, h(x))$ sums over all N examples. Most software reports the mean, $\bar{H} = H/N$, under the same name. The chapter is explicit about this where it matters: perplexity is defined from $\bar{H}$, not from H .

^a<https://morrison-lab.github.io/macros/notation-choices.html#sec-cross-entropy-conventions>

i Watch

Starmer explains cross-entropy^a [18].

^a<https://www.youtube.com/watch?v=6ArSys5qHAU>

i Read

GeeksforGeeks explains the cross-entropy loss^a [19].

^a<https://www.geeksforgeeks.org/machine-learning/what-is-cross-entropy-loss-function/>

2.3.2 Perplexity (lower is better)

$$ppl = b^{\bar{H}}, \quad \bar{H} = \frac{1}{N} H(y, h(x))$$

- Monotonic function of H .
- Sometimes easier to interpret.
- Equivalent “difficulty” as picking among ppl equally-likely options.
- “Per-word” often used in language modeling.

Exercise 2.78 (What perplexity counts). Perplexity undoes the logarithm in the cross-entropy,

$$ppl = b^{\bar{H}}, \quad \bar{H} = \frac{1}{N} H(y, h(x)), \quad (3)$$

where $\bar{H}$ is the per-example cross-entropy of Equation 1 rather than the sum.

Take a classifier that spreads its probability evenly over all C classes on every example. Compute $\bar{H}$ and then ppl . Do it in bits and again in hartleys.

Solution

Solution 2.78. The even classifier has perplexity C . Every predicted probability is $1/C$, so by Equation 2 each example contributes $-\log_b(1/C) = \log_b C$, and the average is $\bar{H} = \log_b C$. Then

$$\begin{aligned} ppl &= b^{\log_b C} \\ &= C. \end{aligned}$$

In bits with $C = 4$, the cross-entropy is

$$\begin{aligned} \bar{H} &= \log_2 4 \\ &= 2 \text{ bits} \end{aligned}$$

and the perplexity is

$$\begin{aligned} ppl &= 2^2 \\ &= 4. \end{aligned}$$

In hartleys, the cross-entropy is

$$\begin{aligned} \bar{H} &= \log_{10} 4 \\ &= 0.602 \end{aligned}$$

hartleys, and the perplexity is

$$\begin{aligned} ppl &= 10^{0.602} \\ &= 4. \end{aligned}$$

Exercise 2.79 (What perplexity counts (part 2)). Use the setup of Exercise 2.78.

Exercise 2.78 should give the same ppl under both bases. Show that this holds for any classifier, not just the even one.

Solution

Solution 2.79. Perplexity does not depend on the base. By Theorem 2.3, $\bar{H}_b = \bar{H}_e / \ln b$. So

$$\begin{aligned}
 b^{\bar{H}_b} &= b^{\bar{H}_e / \ln b} \\
 &= e^{(\bar{H}_e / \ln b) \ln b} \\
 &= e^{\bar{H}_e}.
 \end{aligned}$$

The base cancels: it enters once in the units of $\bar{H}$ and once as the base of the exponent, and the two uses undo each other. So perplexity is quoted as a bare number where cross-entropy^a has to be quoted with its units.

^a<https://morrison-lab.github.io/pds/expectation.html#def-cross-entropy>

Exercise 2.80 (What perplexity counts (part 3)). Use the setup of Exercise 2.78. What is the smallest value *ppl* can take, and which classifier reaches it?

Solution

Solution 2.80. **The smallest value is 1.** $\bar{H} \geq 0$ by Theorem 2.2, so

$$\begin{aligned}
 ppl &= b^{\bar{H}} \\
 &\geq b^0 \\
 &= 1,
 \end{aligned}$$

with equality exactly when $\bar{H} = 0$ — the classifier that is always certain and always right. Perplexity is bounded below by 1 rather than 0 because it counts options, and there is always at least one.

Exercise 2.81 (What perplexity counts (part 4)). Use the setup of Exercise 2.78.

Exercise 2.78 is where the usual explanation comes from: a perplexity of k implies the classifier is as uncertain as someone choosing among k equally likely options. A language model reports a per-word perplexity of 20 over a vocabulary of 50,000 words. Say what that does and does not tell you.

Solution

Solution 2.81. **It reports difficulty, not vocabulary.** A perplexity of 20 says the model is, on average, as uncertain about the next word as someone choosing uniformly among 20 candidates. The gap between 20 and 50,000 is the model's whole contribution: context has narrowed 50,000 options down to about 20. Three things it does not tell you. It is an average, so a model that is nearly certain on function words and uncertain about content words can report the same 20 as one that is moderately uncertain about every word. It says nothing about whether the 20 candidates include the right word at all. And it is comparable across two models only when they share a vocabulary and a tokenizer: a model that splits words into smaller pieces is predicting easier things more often, so its per-token perplexity is lower without the model being better.

i Read

GeeksforGeeks explains perplexity^a [20].

^a<https://www.geeksforgeeks.org/nlp/perplexity-for-llm-evaluation/>

3 Regression

3.1 SSE, sMSE and sRMSE

The sum of squared errors (SSE) adds up the squared distances between predictions and labels. Dividing by N gives the sample mean squared error (sMSE), and its square root is the sample root mean squared error (sRMSE):

$$\begin{aligned}SSE &= \sum_{i=1}^N \|\hat{y}_{(i)} - y_{(i)}\|_2^2 \\sMSE &= SSE/N \\sRMSE &= \sqrt{sMSE}\end{aligned}$$

i Other names for these three

SSE is also the **sum of squared residuals** or the **residual sum of squares** (RSS). sMSE is also the **quadratic loss**, and sRMSE is the **root mean square** of the residuals.

L_2 loss names this family too, and is the one to define before using. Some sources mean sMSE by it and some mean SSE . Those two losses rank models identically on a fixed dataset, since they differ by the constant factor N , and sRMSE ranks with them for the same reason: the square root is applied once, at the end, and is increasing.

A third reading is a different objective, not a different scale. Some sources take the root *inside* the sum, scoring a model by $\sum_i \|\hat{y}_{(i)} - y_{(i)}\|_2$. Moving the root inside changes which model wins, because each un-squared distance then enters the sum on its own rather than being squared first.

Exercise 3.1 (Three names for one quantity). The squared-error family is three reports of the same residuals:

$$SSE = \sum_{i=1}^N \|\hat{y}_{(i)} - y_{(i)}\|_2^2, \quad sMSE = \frac{SSE}{N}, \quad sRMSE = \sqrt{sMSE}. \quad (4)$$

Suppose you are predicting apartment rents in dollars.
Give the units of each of the three.

Solution

Solution 3.1. Only sRMSE is in dollars. A residual is a difference of rents, so it is in dollars; squaring gives dollars squared. SSE and sMSE are therefore both in dollars squared, and the square root returns sRMSE to dollars.

That is the whole case for sRMSE: a report of \$180 can be compared with a rent of \$1,400, where a report of 32,400 dollars-squared can be compared with nothing the reader knows.

Exercise 3.2 (Three names for one quantity (part 2)). Use the setup of Exercise 3.1.

Two analysts fit the same model to the same data. One evaluates on all 1,000 apartments, the other on a random half of them. Which of the three quantities do you expect to differ, and which to agree?

Solution

Solution 3.2. SSE differs; sMSE and sRMSE roughly agree. SSE adds one non-negative term per example, so halving the evaluation set roughly halves it. It measures the dataset as much as the model. sMSE divides that total by N , which cancels the dependence on how many apartments were scored, and sRMSE inherits the cancellation.

So SSE is the quantity to minimize during fitting, where N is fixed and the constant^a factor is irrelevant,

and sMSE or sRMSE is the quantity to report, where the reader's evaluation set is not yours.

^a<https://morrison-lab.github.io/mds/algebra-exponentials.html#def-constant-function>

Exercise 3.3 (Three names for one quantity (part 3)). Use the setup of Exercise 3.1.

A residual of 0.5 and a residual of 2 both contribute to SSE . Compare each contribution to the size of the residual itself, and state the general rule.

Solution

Solution 3.3. Squaring shrinks residuals below 1 and stretches those above. $0.5^2 = 0.25$, which is half of 0.5. $2^2 = 4$, which is twice 2. In general $e^2 < |e|$ when $|e| < 1$ and $e^2 > |e|$ when $|e| > 1$, with the two agreeing at $|e| = 1$.

Note what the crossover depends on: the unit. The same rents measured in cents put every residual far above 1, and the same rents measured in thousands of dollars put most of them below it. The rule is about the number, not about the error being small.

Exercise 3.4 (Three names for one quantity (part 4)). Use the setup of Exercise 3.1.

Exercise 3.3 is often summarized as “squaring penalizes large errors.” Name a situation where that is the behavior you want, and one where it is not.

Solution

Solution 3.4. It depends on what a large error costs. Squaring is right when the cost of being wrong grows faster than the error does. A delivery estimate^a that is 10 minutes late is an annoyance; one that is 100 minutes late may cause the customer to leave, and that is worse than ten annoyances. It is wrong when the large residuals are not real. One mistyped rent of \$14,000 instead of \$1,400 contributes a residual near 12,600, whose square exceeds the combined squares of four thousand ordinary residuals of \$180 — so the fit moves to accommodate a typo. Exercise 3.5 covers what to do instead.

^a<https://morrison-lab.github.io/sds/estimation.html#def-estimate>

i Read

GeeksforGeeks explains regression metrics^a [21], including MSE, RMSE and MAE (its names for the sMSE, sRMSE and sMAE).

^a<https://www.geeksforgeeks.org/machine-learning/regression-metrics/>

3.2 sMAE

The sample mean absolute error (sMAE) replaces the squares with absolute values:

$$\text{sMAE} = \frac{1}{N} \sum_{i=1}^N \|\hat{y}_{(i)} - y_{(i)}\|_1$$

Exercise 3.5 (What changes when you fit with absolute error). The sMAE averages the absolute residuals without squaring them:

$$\text{sMAE} = \frac{1}{N} \sum_{i=1}^N \|\hat{y}_{(i)} - y_{(i)}\|_1. \quad (5)$$

Take the simplest possible regression, $C = 1$ and no inputs at all, so the model is a single constant c

and $\|h - y\|_1 = |c - y|$. The five observations are

2, 3, 4, 5, 100.

Find the c that minimizes sMSE, and the c that minimizes sMAE. Name each in one word.

Solution

Solution 3.5. The mean and the median. Squared error is minimized by the mean, $(2 + 3 + 4 + 5 + 100)/5 = 22.8$. Absolute error is minimized by the median, 4.

The gap is the whole lesson. Four of the five observations lie between 2 and 5, and the squared-error fit lands at 22.8 — above every one of them. No observation is anywhere near the value that best summarizes them under sMSE.

Exercise 3.6 (What changes when you fit with absolute error (part 2)). Use the setup of Exercise 3.5. Differentiate $\sum_i |c - y_{(i)}|$ with respect to c at a point where c equals none of the observations, and say what setting the derivative to zero requires.

Solution

Solution 3.6. Zero requires a tie in the counts. Away from the observations, $|c - y_{(i)}|$ is differentiable with derivative +1 when $c > y_{(i)}$ and -1 when $c < y_{(i)}$. Summing,

$$\frac{d}{dc} \sum_{i=1}^N |c - y_{(i)}| = \#\{i : y_{(i)} < c\} - \#\{i : y_{(i)} > c\}.$$

Setting this to zero would require as many observations below c as above it, and with $N = 5$ the two counts sum to an odd number, so their difference is never zero. No off-observation point is stationary. The derivative is -1 just left of 4 and $+1$ just right of it, so the sum falls until c reaches 4 and rises after: the minimum sits at that corner, not at a stationary point^a.

Notice what the derivative does not contain: the values $y_{(i)}$ themselves, only their side of c . Moving the 100 to 10,000 leaves every count unchanged, so the sMAE fit does not move at all. The sMSE derivative is $2 \sum (c - y_{(i)})$, which carries every value, so the same edit moves the mean^b from 22.8 to 2,002.8.

^a<https://morrison-lab.github.io/mds/optimization.html#def-stationary-point>

^b<https://morrison-lab.github.io/sds/exploratory-descriptive.html#def-sample-mean>

Exercise 3.7 (What changes when you fit with absolute error (part 3)). Use the setup of Exercise 3.5. Which of the two answers in Exercise 3.5 would you rather report as the typical rent, and why?

Solution

Solution 3.7. The median, for a typical rent. A reader who hears “typical” expects a value that describes most of the apartments. 4 does; 22.8 describes none of them. The mean^a is answering a different question — what you would pay per apartment if you rented all five — which is the right question for a budget and the wrong one for a description.

^a<https://morrison-lab.github.io/sds/exploratory-descriptive.html#def-sample-mean>

Exercise 3.8 (What changes when you fit with absolute error (part 4)). Use the setup of Exercise 3.5. Give one reason to prefer squared error anyway.

Solution

Solution 3.8. Squared error is smoother and has a closed-form solution^a. sMSE is differentiable everywhere, and for linear models the minimizer is a formula. sMAE has a corner at every observation, where the derivative jumps from -1 to $+1$ without passing through intermediate values, so gradient methods need care and there is generally no closed-form solution.

The choice is a statement about the data. Squared error assumes the extreme observations are real and worth fitting; absolute error assumes they are noise and should not move the answer. With 100 as a genuine penthouse, the mean^b is informative. With 100 as a typo for 10.0, it is a fit ruined by one keystroke.

^a<https://morrison-lab.github.io/mds/expressions-algorithms.html#def-closed-form-solution>

^b<https://morrison-lab.github.io/sds/exploratory-descriptive.html#def-sample-mean>

i Other names for sMAE, and one abbreviation that is not it

sMAE is also the L_1 **loss** and the **mean absolute deviation**.

That second name suggests the abbreviation **MAD**, which is ambiguous: forecasting uses it for the mean absolute deviation, and much statistical writing uses it for the *median* absolute deviation, $\text{median}_i |y_{(i)} - \text{median}(y)|$, which measures how spread out a single sample is. That second quantity takes no model and no predictions, where sMAE measures how far a model's predictions fall from the truth. The two readings of MAD share an abbreviation, two of their three words, and nothing else. Expand MAD the first time you write it.

3.3 R^2

The coefficient of determination (sometimes written r^2) measures how much of the variance in the output we can predict from the input variable(s).

$$R^2 = 1 - \frac{SSE}{SS_{tot}}$$
$$SS_{tot} = \sum_{i=1}^N (y_{(i)} - \bar{y})^2$$
$$\bar{y} = \frac{1}{N} \sum_{i=1}^N y_{(i)}$$

i Two near-synonyms for R^2 , and two names that mean different things in different sources

Squared correlation. Writing R^2 as r^2 suggests it is the Pearson correlation squared, and for a least-squares fit with an intercept it is. Outside that case the two are different quantities, and Exercise 3.10 examines the range of R^2 . Anything that reports r^2 for a model you did not fit by least squares with an intercept is reporting a different quantity under a familiar name.

Explained variance. Some libraries report a score by that name which replaces SSE/N with the *variance* of the residuals. The two agree whenever the residuals average to zero, which a least-squares fit with an intercept guarantees, so the distinction stays invisible until a model is biased. Take $y = (2, 4, 6, 8, 10)$ and a model that predicts $y + 1$ for every point. Every residual is -1 , so the residuals have zero variance and a score computed that way is 1, a perfect fit, while R^2 is $1 - 5/40 = 0.875$. One score is measuring the shape of the predictions and the other is measuring the predictions.

SSR and ESS . SS_{tot} splits into the part a least-squares fit with an intercept explains and the part it leaves over, and the abbreviations for those two pieces are not standardized. SSR is read as the *regression* sum of squares by some sources and as the *residual* sum of squares by others, which are the two different pieces; ESS is read as *explained* by some and as *error* by others, likewise. Do not infer either from the letters. Read which piece the source subtracts from SS_{tot} , or which one it puts in the numerator of $1 - (\cdot)/SS_{tot}$.

Exercise 3.9 (When the coefficient of determination goes negative). The coefficient of determination compares your model's squared error against the squared error of one particular comparison model:

$$\begin{aligned} R^2 &= 1 - \frac{SSE}{SS_{tot}}, \\ SS_{tot} &= \sum_{i=1}^N (y_{(i)} - \bar{y})^2, \\ \bar{y} &= \frac{1}{N} \sum_{i=1}^N y_{(i)}. \end{aligned} \tag{6}$$

Name the comparison model whose error is SS_{tot} , and say what $R^2 = 0$ therefore implies.

Solution

Solution 3.9. The comparison model is the constant model that always predicts $\bar{y}$. Substituting $\hat{y}_{(i)} = \bar{y}$ into SSE gives exactly SS_{tot} . So R^2 asks one question: by what fraction does your model reduce the squared error of a model that ignores the inputs entirely?

$R^2 = 0$ means it reduces it by none. The model is worth exactly what the average of the labels is worth, whatever its inputs, parameters or training cost.

Exercise 3.10 (When the coefficient of determination goes negative (part 2)). Use the setup of Exercise 3.9.

R^2 is often introduced as a proportion of variance^a, which would put it between 0 and 1. Show that Equation 6 can go negative, using $y = (1, 2, 3, 4)$ and a model that predicts 10 everywhere.

^a<https://morrison-lab.github.io/pds/variance-covariance.html#def-variance>

Solution

Solution 3.10. It goes to -45 . Here $\bar{y} = 2.5$ and

$$SS_{tot} = 1.5^2 + 0.5^2 + 0.5^2 + 1.5^2 \\ = 5.$$

The model predicting 10 has

$$SSE = 9^2 + 8^2 + 7^2 + 6^2 \\ = 81 + 64 + 49 + 36 \\ = 230,$$

so

$$R^2 = 1 - 230/5 \\ = -45.$$

Nothing is out of range, because R^2 is not a proportion of anything. It is 1 minus a ratio of two error sums, and that ratio is unbounded above. The proportion-of-variance reading is a special case, true only when the model was fit by least squares with an intercept^a and is being scored on the data it was fit to. If any of those three conditions fails, the lower bound of 0 no longer holds.

^a<https://morrison-lab.github.io/mds/algebra-exponentials.html#def-intercept>

Exercise 3.11 (When the coefficient of determination goes negative (part 3)). Use the setup of Exercise 3.9.

A model always predicts the mean^a of its training labels. It is scored on a test set whose label mean is different. Is R^2 on the test set positive, zero, or negative?

^a<https://morrison-lab.github.io/sds/exploratory-descriptive.html#def-sample-mean>

Solution

Solution 3.11. Negative, unless the two label means^a agree.

^a<https://morrison-lab.github.io/sds/exploratory-descriptive.html#def-sample-mean>

Exercise 3.12 (When the coefficient of determination goes negative (part 4)). Use the setup of Exercise 3.9.

Show why.

Solution

Solution 3.12. Write $\bar{y}_{\text{train}}$ for the training mean^a and $\bar{y}$ for the test mean. The model predicts $\bar{y}_{\text{train}}$ for every test point, so

$$SSE = \sum_{i=1}^N (y_{(i)} - \bar{y}_{\text{train}})^2 \\ = \sum_{i=1}^N (y_{(i)} - \bar{y})^2 + N(\bar{y} - \bar{y}_{\text{train}})^2 \\ = SS_{tot} + N(\bar{y} - \bar{y}_{\text{train}})^2.$$

The middle term vanishes because the differences $y_{(i)} - \bar{y}$ sum to zero. Then

$$R^2 = 1 - \frac{SSE}{SS_{tot}} \\ = -\frac{N(\bar{y} - \bar{y}_{\text{train}})^2}{SS_{tot}}.$$

This is zero only when the means^b agree. Otherwise it is negative: the model is worse than the model that predicts the sample mean^c of the test set's labels for every point.

^a<https://morrison-lab.github.io/sds/exploratory-descriptive.html#def-sample-mean>

^b<https://morrison-lab.github.io/sds/exploratory-descriptive.html#def-sample-mean>

^c<https://morrison-lab.github.io/sds/exploratory-descriptive.html#def-sample-mean>

Exercise 3.13 (When the coefficient of determination goes negative (part 5)). Use the setup of Exercise 3.9.

A test set has $N = 50$ points and $SS_{tot} = 200$. A model scores $R^2 = 0.75$ on it. Find SSE and $sMSE$ on the test set.

Solution

Solution 3.13. Rearranging Equation 6, $SSE = SS_{tot}(1 - R^2)$, so

$$\begin{aligned} SSE &= 200(1 - 0.75) & sMSE &= SSE/N \\ &= 50, & &= 50/50 \\ & & &= 1. \end{aligned}$$

On a fixed test set, a higher R^2 implies a smaller $sMSE$. Across different problems that link breaks. R^2 is scaled by how much the labels vary, so predicting rents in a city where they range from \$800 to \$5,000 is an easier way to earn a high R^2 than predicting them in a block where they range from \$1,400 to \$1,500. The second model can have the far smaller $sMSE$ while reporting the far worse R^2 . Report both.

i Watch

Starmer explains R^2 ^a [22].

^a<https://www.youtube.com/watch?v=2AQKmw14mHM>

i Read

GeeksforGeeks explains R^2 ^a [23].

^a<https://www.geeksforgeeks.org/machine-learning/ml-r-squared-in-regression-analysis/>

3.4 Predicted against true outputs

Exercise 3.14 (Plot predictions against the true outputs). The Auto data give the fuel economy `mpg` and the `horsepower` of 392 cars. Use the copy `data/auto.csv`^a on this site. Any computer with R or Python can run the code in the solution.

Fit a line to predict `mpg` from `horsepower` by least squares.

^a<https://morrison-lab.github.io/lds/chapters/data/auto.csv>

Solution

Solution 3.14.

3.4.1 Python

```
import numpy as np
import pandas as pd

auto_py = pd.read_csv("data/auto.csv")
x_py = auto_py["horsepower"].to_numpy(dtype=float)
y_py = auto_py["mpg"].to_numpy(dtype=float)

fits_py = {}
fits_py[1] = np.polyval(np.polyfit(x_py, y_py, 1), x_py)
print(fits_py[1][:3].round(2))

[19.42 13.89 16.26]
```

3.4.2 R

```
auto_r <- read.csv("data/auto.csv")

fit_line <- lm(mpg ~ horsepower, data = auto_r)
print(round(unname(coef(fit_line)), 3))

[1] 39.936 -0.158
```

Exercise 3.15 (Plot predictions against the true outputs (part 2)). Use the setup of Exercise 3.14. Compute the sample R^2 of the fitted line on the same cars.

Solution

Solution 3.15.

3.4.3 Python

```
def r2_py(y, pred):
    return 1 - np.sum((y - pred) ** 2) / np.sum((y - y.mean()) ** 2)

print(f"line: sample R^2 = {r2_py(y_py, fits_py[1]):.3f}")

line: sample R^2 = 0.606
```

3.4.4 R

```
r2_r <- function(y, pred) 1 - sum((y - pred)^2) / sum((y - mean(y))^2)

cat("line R^2 =", round(r2_r(auto_r$mpg, fitted(fit_line)), 3), "\n")

line R^2 = 0.606
```

Exercise 3.16 (Plot predictions against the true outputs (part 3)). Use the setup of Exercise 3.14. Plot the fitted line's prediction for each car against its true mpg, with the line where the prediction equals the true value for reference.

Solution

Solution 3.16.

3.4.5 Python

```
import matplotlib.pyplot as plt

fig, ax = plt.subplots(figsize=(3.5, 3.5))
ax.scatter(y_py, fits_py[1], s=8, alpha=0.6);
ax.plot([5, 47], [5, 47], "k--");
ax.set_xlabel("True mpg");
ax.set_ylabel("Predicted mpg");
plt.tight_layout()
plt.show()
```

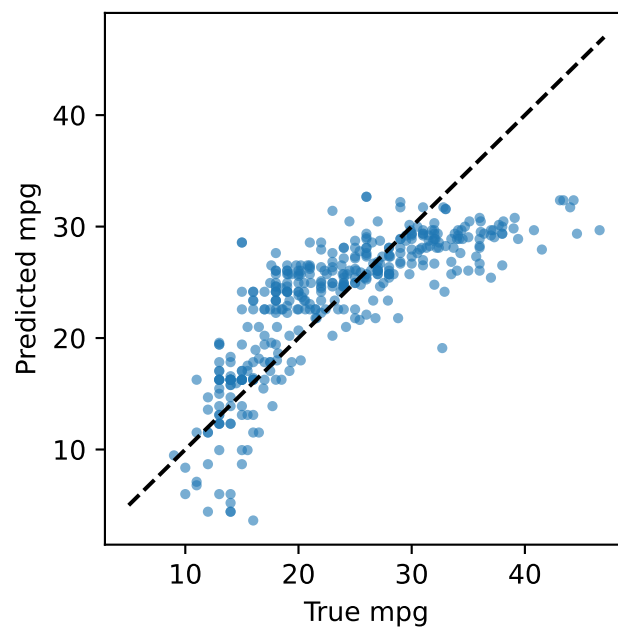


Figure 3: Predicted against true miles per gallon for the Auto cars, from a line in horsepower. The dashed line is where the prediction equals the true value.

3.4.6 R

```
library(ggplot2)

ggplot(data.frame(true = auto_r$mpg, pred = fitted(fit_line)), aes(true, pred)) +
  geom_point(size = 0.8, alpha = 0.6) +
  geom_abline(linetype = "dashed") +
  labs(x = "True mpg", y = "Predicted mpg")
```

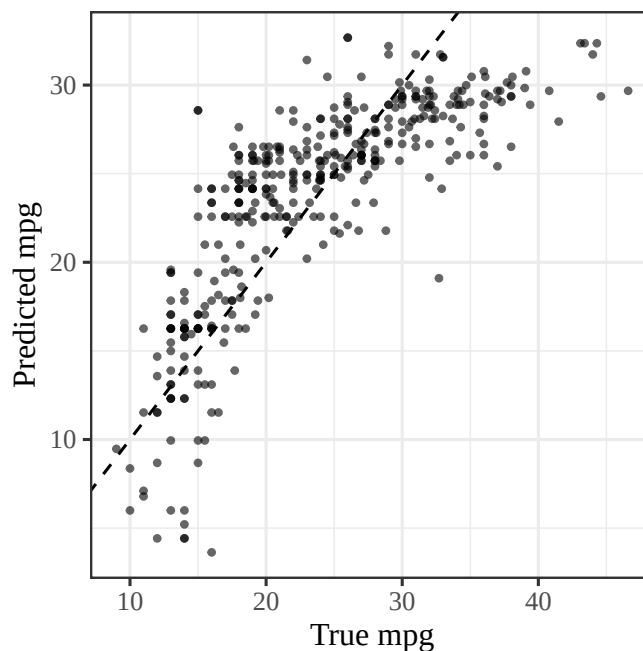


Figure 4: Predicted against true miles per gallon for the Auto cars, from a line in horsepower. The dashed line is where the prediction equals the true value.

Exercise 3.17 (Plot predictions against the true outputs (part 4)). Use the setup of Exercise 3.14. If the model predicted every car perfectly, where would the points lie?

Solution

Solution 3.17. If every prediction were perfect, the points would lie on the dashed line, where the prediction equals the true value. The distance from a point to that line, measured vertically, is the point's residual.

Exercise 3.18 (Plot predictions against the true outputs (part 5)). Use the setup of Exercise 3.14. The plot of Exercise 3.16 should show a pattern. Say what the pattern is, and what it tells you about the line.

Solution

Solution 3.18. In the plot of Exercise 3.16, the points for the line bend away from the dashed line instead of following it. The line predicts too low for the cars with the highest mpg and too high for many cars in the middle of the range. A residual pattern like this shows that the model's mistakes are systematic, not noise: fuel economy falls with horsepower along a curve, and a straight line cannot follow it. A single number such as the sample R^2 of 0.606 does not show this, and the plot does.

Exercise 3.19 (Plot predictions against the true outputs (part 6)). Use the setup of Exercise 3.14. Fit a quadratic in `horsepower` and repeat Exercise 3.14 and Exercise 3.16.

Solution

Solution 3.19.

3.4.7 Python

```
fits_py[2] = np.polyval(np.polyfit(x_py, y_py, 2), x_py)
print(f"quadratic: sample R^2 = {r2_py(y_py, fits_py[2]):.3f}")
```

quadratic: sample R² = 0.688

```
fig, axes = plt.subplots(1, 2, figsize=(7, 3.5), sharey=True)
for ax, degree, name in zip(axes, (1, 2), ("Line", "Quadratic")):
    ax.scatter(y_py, fits_py[degree], s=8, alpha=0.6);
    ax.plot([5, 47], [5, 47], "k--");
    ax.set_title(name);
    ax.set_xlabel("True mpg");
axes[0].set_ylabel("Predicted mpg");
plt.tight_layout()
plt.show()
```

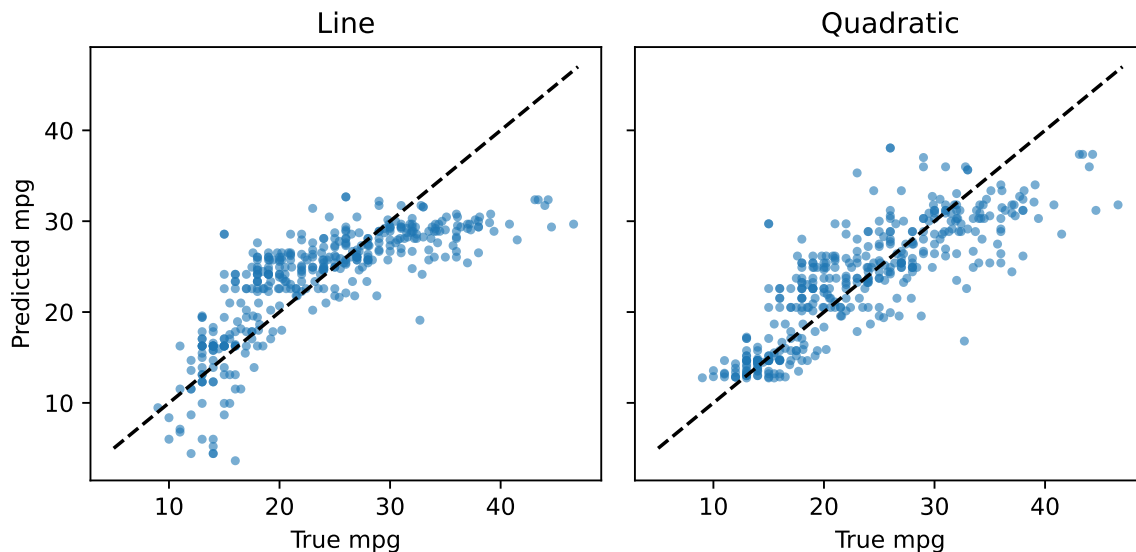


Figure 5: Predicted against true miles per gallon for the Auto cars, from a line (left) and a quadratic (right) in horsepower. The dashed line is where the prediction equals the true value.

3.4.8 R

```
fit_quad <- lm(mpg ~ poly(horsepower, 2), data = auto_r)
cat("quadratic R^2 =", round(r2_r(auto_r$mpg, fitted(fit_quad)), 3), "\n")
```

quadratic R² = 0.688

```

pred_df <- rbind(
  data.frame(model = "Line", true = auto_r$mpg, pred = fitted(fit_line)),
  data.frame(model = "Quadratic", true = auto_r$mpg, pred = fitted(fit_quad)))

ggplot(pred_df, aes(true, pred)) +
  geom_point(size = 0.8, alpha = 0.6) +
  geom_abline(linetype = "dashed") +
  facet_wrap(~model) +
  labs(x = "True mpg", y = "Predicted mpg")

```

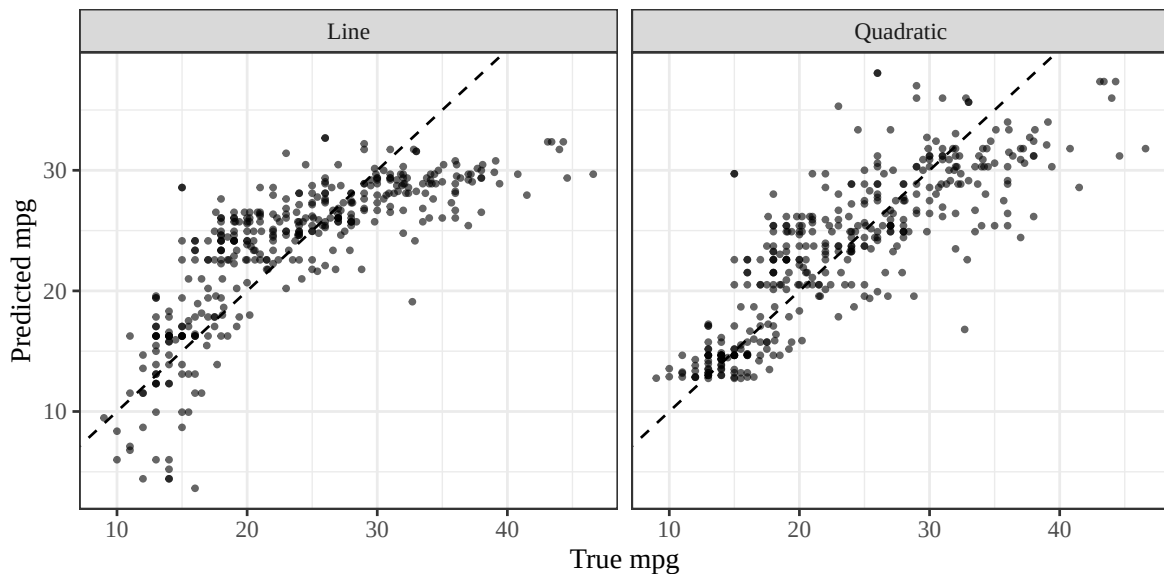


Figure 6: Predicted against true miles per gallon for the Auto cars, from a line (left) and a quadratic (right) in horsepower. The dashed line is where the prediction equals the true value.

Exercise 3.20 (Plot predictions against the true outputs (part 7)). Use the setup of Exercise 3.14. Say what changes in the plot.

Solution

Solution 3.20. With the quadratic, the points scatter more evenly around the dashed line. A predicted-versus-true plot plays the role for regression that the [confusion matrix](#) plays for classification: it shows what kind of mistakes the model makes, not only how many.

Exercise 3.21 (Plot predictions against the true outputs (part 8)). Use the setup of Exercise 3.14. Say what changes in the sample R^2 .

Solution

Solution 3.21. With the quadratic, the sample R^2 rises to 0.688.

4 Token Sequences

4.1 Token Error Rate

- Our model outputs a hypothesis sequence, and we compare it to a reference sequence.
 - Rather than comparing tokens at matching indices, we compute an edit distance. The edit distance lets us substitute, delete or insert tokens, but not reorder them.
 - Commonly used to evaluate speech recognition systems.
1. Perform a minimum edit distance alignment of the reference sequence (length N_{ref}) to the hypothesis sequence (length N_{hyp}).
 2. Count substitutions (S), insertions (I) and deletions (D).
 3. Compute the token error rate (TER) as follows:

$$TER = \frac{S + I + D}{N_{ref}}$$

i Other names for TER, and a second meaning of the acronym

The rate takes its name from whatever the tokens are. Words give **word error rate** (WER), characters give **character error rate** (CER), and phones give **phone error rate** (PER). The distance the alignment step minimizes is an **edit distance** or a **Levenshtein distance**, the two naming one quantity when substituting, inserting and deleting each cost one.

A published number needs two checks before you compare yours to it: what that source counts as an edit, and what it expands TER to. Machine translation uses the same three letters for a *translation edit rate*, which allows a block of tokens to be *moved* as a single edit. The edit distance in this chapter does not: reordered output is charged as deletions and insertions instead. A rate computed under the two rules is not the same measurement. The rule that allows moves can never score higher, since every alignment without a move is still available to it, and it scores strictly lower whenever a move beats the deletions and insertions it replaces.

Exercise 4.1 (Scoring a sequence against a reference). A speech recognizer outputs a hypothesis sequence, which you want to score against a reference sequence. Token error rate aligns the two by minimum edit distance, counting substitutions S , insertions I and deletions D , and reports

$$TER = \frac{S + I + D}{N_{ref}}. \quad (7)$$

The edit distance allows tokens to be substituted, deleted or inserted, but not reordered. Take the reference *the cat sat on the mat* and the hypothesis *the cat sit on mat*. Find the alignment and give S , I and D .

Solution

Solution 4.1. **The alignment finds one substitution and one deletion.**

Table 3: The minimum-edit alignment

Reference	the	cat	sat	on	the	mat
Hypothesis	the	cat	sit	on	—	mat

In Table 3 the token *sat* became *sit*, a substitution, and the second *the* is missing, a deletion. So $S = 1$, $I = 0$ and $D = 1$. No alignment does it in fewer edits: the sequences differ in length, which forces at least one deletion, and *sat* and *sit* differ, which forces at least one more.

Exercise 4.2 (Scoring a sequence against a reference (part 2)). Use the setup of Exercise 4.1. Compute TER by Equation 7 for the alignment you found there.

Solution

Solution 4.2. With $N_{ref} = 6$ reference tokens,

$$\begin{aligned} TER &= \frac{1 + 0 + 1}{6} \\ &= \frac{1}{3}. \end{aligned}$$

Exercise 4.3 (Scoring a sequence against a reference (part 3)). Use the setup of Exercise 4.1. What is the smallest value TER can take?

Solution

Solution 4.3. **The smallest value is 0.** Each of S , I and D is a count, so the numerator is at least 0, and it is 0 exactly when the two sequences are identical.

Exercise 4.4 (Scoring a sequence against a reference (part 4)). Use the setup of Exercise 4.1. What is the largest? If there is no largest, give a hypothesis that exceeds 1.

Solution

Solution 4.4. **There is no largest value.** The numerator counts edits, which insertions make unbounded, while the denominator is fixed by the reference alone. Take a one-token reference and a hypothesis of 100 wrong tokens: one substitution and 99 insertions give

$$\begin{aligned} TER &= 100/1 \\ &= 100. \end{aligned}$$

A rate above 1 is not a paradox here, because TER is not a proportion of anything — the same structure as R^2 in Exercise 3.9, and the same warning. A name ending in *rate* invites the reader to assume a ceiling that the definition does not provide.

Exercise 4.5 (Scoring a sequence against a reference (part 5)). Use the setup of Exercise 4.1. Exercise 4.4 exposes an asymmetry in Equation 7. Name it, and say what it costs a recognizer that is unsure whether it heard a word.

Solution

Solution 4.5. **Inserting is charged, and staying silent is charged the same.** N_{ref} does not grow when the hypothesis does, so a recognizer is charged one unit for a word it guesses wrongly and one unit for a word it omits. Write p for the chance a word really is there, and q for the chance a guess names it correctly when it is. Guessing costs $1 - pq$, since it escapes the charge only when the word is present and the guess is right. Silence costs p , since an omission is charged only when there was something

to omit. So guessing is better when $pq > 1 - p$, which needs p well above a half: a recognizer unsure whether it heard anything at all gains nothing by guessing, whatever q is. The asymmetry rewards a different case, the one where a word is certainly there and only its identity is in doubt. There p is near 1, silence costs almost a full unit, and any guess with $q > 0$ beats it. A system tuned on *TER* will fill those stretches with its best guess rather than leave a gap.

Whether that is the behavior you want depends on the reader of the transcript. For a search index it is: a wrong word is skipped, a missing word cannot be found. For a medical dictation it is not, where a confident wrong drug name is worse than a blank a clinician can see and fill. The metric cannot make that judgment, so it has to be made before the metric is chosen.

i Watch

Andrew Ng explains the BLEU score^a [24].

^a<https://www.youtube.com/watch?v=DejHQYAGb7Q>

i Read

GeeksforGeeks explains minimum edit distance^a [25], the alignment behind the token error rate.

^a<https://www.geeksforgeeks.org/dsa/edit-distance-dp-5/>

5 Further reading

- Hastie et al. [26], Chapter 7
- Brodersen et al. [10] — balanced accuracy, for imbalanced classes
- Starmer [27], on assessing model performance — confusion matrices, sensitivity and specificity, and ROC curves, illustrated

References

- [1] U. von Luxburg, “Statistical machine learning.” University of Tübingen, lecture videos on YouTube, 2020. Available: <https://www.youtube.com/playlist?list=PL05umP7R6ij2XCvrRzLokX6EoHWaGA2cC>
- [2] Wikipedia contributors, “Confusion matrix — Wikipedia, the free encyclopedia.” 2026. Available: https://en.wikipedia.org/wiki/Confusion_matrix
- [3] J. Starmer, “Machine learning fundamentals: The confusion matrix.” StatQuest, video on YouTube, 2018. Available: <https://www.youtube.com/watch?v=Kdsp6soqA7o>
- [4] GeeksforGeeks, “Understanding the confusion matrix in machine learning.” GeeksforGeeks, tutorial article, 2026. Available: <https://www.geeksforgeeks.org/machine-learning/confusion-matrix-machine-learning/>
- [5] Wikipedia contributors, “Evaluation of binary classifiers — Wikipedia, the free encyclopedia.” 2026. Available: https://en.wikipedia.org/wiki/Evaluation_of_binary_classifiers
- [6] Wikipedia contributors, “Precision and recall — Wikipedia, the free encyclopedia.” 2026. Available: https://en.wikipedia.org/wiki/Precision_and_recall
- [7] Wikipedia contributors, “Sensitivity and specificity — Wikipedia, the free encyclopedia.” 2026. Available: https://en.wikipedia.org/wiki/Sensitivity_and_specificity
- [8] J. Starmer, “Machine learning fundamentals: Sensitivity and specificity.” StatQuest, video on YouTube, 2019. Available: <https://www.youtube.com/watch?v=vP06aMoz4v8>
- [9] GeeksforGeeks, “Precision and recall in machine learning.” GeeksforGeeks, tutorial article, 2026. Available: <https://www.geeksforgeeks.org/machine-learning/precision-and-recall-in-machine-learning/>

- [10] K. H. Brodersen, C. S. Ong, K. E. Stephan, and J. M. Buhmann, “The balanced accuracy and its posterior distribution,” in *20th international conference on pattern recognition (ICPR)*, 2010, pp. 3121–3124. doi: 10.1109/ICPR.2010.764³.
- [11] Wikipedia contributors, “F-score — Wikipedia, the free encyclopedia.” 2026. Available: <https://en.wikipedia.org/wiki/F-score>
- [12] GeeksforGeeks, “F1 score in machine learning.” GeeksforGeeks, tutorial article, 2026. Available: <https://www.geeksforgeeks.org/machine-learning/f1-score-in-machine-learning/>
- [13] Wikipedia contributors, “Receiver operating characteristic — Wikipedia, the free encyclopedia.” 2026. Available: https://en.wikipedia.org/wiki/Receiver_operating_characteristic
- [14] J. Starmer, “ROC and AUC, clearly explained.” StatQuest, video on YouTube, 2019. Available: <https://www.youtube.com/watch?v=4jRBRDbJemM>
- [15] T. Hastie, R. Tibshirani, and J. Taylor, “Statistical learning with python.” Stanford Online, lecture videos on YouTube, 2023. Available: <https://www.youtube.com/playlist?list=PLoROMvodv4rPP6braWoRt5UCXYZ71GZIQ>
- [16] GeeksforGeeks, “AUC-ROC curve in machine learning.” GeeksforGeeks, tutorial article, 2026. Available: <https://www.geeksforgeeks.org/machine-learning/auc-roc-curve/>
- [17] M. Hopkins, E. Reeber, G. Forman, and J. Suermondt, “Spambase.” UCI Machine Learning Repository, 1999. doi: 10.24432/C53G6X⁴.
- [18] J. Starmer, “Neural networks part 6: Cross entropy.” StatQuest, video on YouTube, 2021. Available: <https://www.youtube.com/watch?v=6ArSys5qHAU>
- [19] GeeksforGeeks, “What is cross-entropy loss function?” GeeksforGeeks, tutorial article, 2025. Available: <https://www.geeksforgeeks.org/machine-learning/what-is-cross-entropy-loss-function/>
- [20] GeeksforGeeks, “Perplexity for LLM evaluation.” GeeksforGeeks, tutorial article, 2026. Available: <https://www.geeksforgeeks.org/nlp/perplexity-for-llm-evaluation/>
- [21] GeeksforGeeks, “Regression metrics.” GeeksforGeeks, tutorial article, 2026. Available: <https://www.geeksforgeeks.org/machine-learning/regression-metrics/>
- [22] J. Starmer, “R-squared, clearly explained.” StatQuest, video on YouTube, 2015. Available: <https://www.youtube.com/watch?v=2AQKmw14mHM>
- [23] GeeksforGeeks, “R-squared in regression analysis in machine learning.” GeeksforGeeks, tutorial article, 2025. Available: <https://www.geeksforgeeks.org/machine-learning/ml-r-squared-in-regression-analysis/>
- [24] A. Ng, “Bleu score.” DeepLearning.AI, Deep Learning Specialization, Course 5; video on YouTube, 2018. Available: <https://www.youtube.com/watch?v=DejHQYAGb7Q>
- [25] GeeksforGeeks, “Edit distance.” GeeksforGeeks, tutorial article, 2026. Available: <https://www.geeksforgeeks.org/dsa/edit-distance-dp-5/>
- [26] T. Hastie, R. Tibshirani, and J. Friedman, *The elements of statistical learning: Data mining, inference, and prediction*, 2nd ed. Springer, 2009. Available: <https://hastie.su.domains/ElemStatLearn/>
- [27] J. Starmer, *The StatQuest illustrated guide to machine learning*. StatQuest Publications, 2022. Available: <https://statquest.org/statquest-store/>

³<https://doi.org/10.1109/ICPR.2010.764>

⁴<https://doi.org/10.24432/C53G6X>