**Section 1**

**Classification: Thresholding**

Logistic regression returns a probability. You can use the returned probability "as is" (for example, the probability that the user will click on this ad is 0.00023) or convert the returned probability to a binary value (for example, this email is spam).

A logistic regression model that returns 0.9995 for a particular email message is predicting that it is very likely to be spam. Conversely, another email message with a prediction score of 0.0003 on that same logistic regression model is very likely not spam. However, what about an email message with a prediction score of 0.6? To map a logistic regression value to a binary category, you must define a classification threshold (also called the decision threshold). A value above that threshold indicates "spam"; a value below indicates "not spam." It is tempting to assume that the classification threshold should always be 0.5, but thresholds are problem-dependent, and are therefore values that you must tune.

The following sections take a closer look at metrics you can use to evaluate a classification model's predictions, as well as the impact of changing the classification threshold on these predictions.

Note: "Tuning" a threshold for logistic regression is different from tuning hyperparameters such as learning rate. **Part of choosing a threshold is assessing how much you'll suffer for making a mistake.** For example, mistakenly labeling a non-spam message as spam is very bad. However, mistakenly labeling a spam message as non-spam is unpleasant, but hardly the end of your job.

**Key Terms**

|  |  |
| --- | --- |
|  [binary classification](https://developers.google.com/machine-learning/glossary#binary_classification) |  [classification model](https://developers.google.com/machine-learning/glossary#classification_model) |

**Section 2**

**Classification: True vs. False and Positive vs. Negative**

**Building blocks of the metrics we'll use to evaluate classification models**

* Positive Class – Is target variable – Ex. Wolf
* Negative Class – Not Target variable – Ex. No Wolf

We can summarize our "wolf-prediction" model using a 2x2 [confusion matrix](https://developers.google.com/machine-learning/glossary#confusion_matrix) that depicts all four possible outcomes:

Chart, treemap chart

Description automatically generated

A **true positive** is an outcome where the model correctly predicts the positive class.

* It gave “1” as prediction when target was “1”.
* Thus, “1” prediction is correct

A **true negative** is an outcome where the model correctly predicts the negative class.

* It gave “0” as prediction when target was “0”
* Thus, “0” prediction is correct

A **false positive** is an outcome where the model incorrectly predicts the positive class.

* It gave “1” as prediction when target was “0”.
* Thus, “1” prediction is incorrect

A **false negative** is an outcome where the model incorrectly predicts the negative class.

* It gave “0” prediction when target was “1”
* Thus, “0” prediction is incorrect.

**Key Terms**

|  |  |  |
| --- | --- | --- |
|  [confusion matrix](https://developers.google.com/machine-learning/glossary#confusion_matrix) |  [negative class](https://developers.google.com/machine-learning/glossary#negative_class) |  [positive class](https://developers.google.com/machine-learning/glossary#positive_class) |
|  [false negative](https://developers.google.com/machine-learning/glossary#FN) |  [false positive](https://developers.google.com/machine-learning/glossary#FP) |  |
|  [true negative](https://developers.google.com/machine-learning/glossary#TN) |  [true positive](https://developers.google.com/machine-learning/glossary#TP) |  |

**Section 3**

**Accuracy** is one metric for evaluating classification models. Informally, accuracy is the fraction of predictions our model got right. Formally, accuracy has the following definition:

Text

Description automatically generated with medium confidence

For binary classification, accuracy can also be calculated in terms of positives and negatives as follows:

Text

Description automatically generated with medium confidence

Where TP = True Positives, TN = True Negatives, FP = False Positives, and FN = False Negatives.

Let's try calculating accuracy for the following model that classified 100 tumors as [malignant](https://wikipedia.org/wiki/Malignancy) (the positive class) or [benign](https://wikipedia.org/wiki/Benign_tumor) (the negative class):

Treemap chart

Description automatically generated with low confidence



Accuracy comes out to 0.91, or 91% (91 correct predictions out of 100 total examples). That means our tumor classifier is doing a great job of identifying malignancies, right?

Actually, let's do a closer analysis of positives and negatives to gain more insight into our model's performance.

Of the 100 tumor examples, 91 are benign (90 TNs and 1 FP) and 9 are malignant (1 TP and 8 FNs).

Of the 91 benign tumors, the model correctly identifies 90 as benign. That's good. However, of the 9 malignant tumors, the model only correctly identifies 1 as malignant—a terrible outcome, as 8 out of 9 malignancies go undiagnosed!

While 91% accuracy may seem good at first glance, another tumor-classifier model that always predicts benign would achieve the exact same accuracy (91/100 correct predictions) on our examples. In other words, our model is no better than one that has zero predictive ability to distinguish malignant tumors from benign tumors.

Accuracy alone doesn't tell the full story when you're working with a class-imbalanced data set, like this one, where there is a significant disparity between the number of positive and negative labels.

**Key Terms**

|  |  |
| --- | --- |
|  [accuracy](https://developers.google.com/machine-learning/glossary#accuracy) |  [class-imbalanced data set](https://developers.google.com/machine-learning/glossary#class_imbalanced_data_set) |

**Section 4**

**Classification: Precision and Recall**

**Precision**

Precision attempts to answer the following question:

What proportion of positive identifications was actually correct?

**Precision is defined as follows:**

**Text

Description automatically generated**

Note: A model that produces no false positives has a precision of 1.0.

Let's calculate precision for our ML model from the previous section that analyzes tumors:

Timeline

Description automatically generated with low confidence

Our model has a precision of 0.5—in other words, when it predicts a tumor is malignant, it is correct 50% of the time.

**Recall**

**Recall** attempts to answer the following question**:**

What proportion of actual positives was identified correctly?

Text

Description automatically generated

Note: A model that produces no false negatives has a recall of 1.0.

Timeline

Description automatically generated with low confidence

Our model has a recall of 0.11—in other words, it correctly identifies 11% of all malignant tumors.

**Section 5**

**Precision and Recall: A Tug of War**

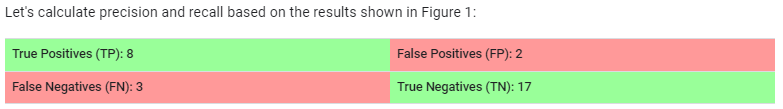
To fully evaluate the effectiveness of a model, you must examine both precision and recall. Unfortunately, precision and recall are often in tension.

**That is, improving precision typically reduces recall and vice versa.**

Explore this notion by looking at the following figure, which shows 30 predictions made by an email classification model. Those to the right of the classification threshold are classified as "spam", while those to the left are classified as "not spam."

A picture containing timeline

Description automatically generated



**Precision** measures the percentage of emails flagged as spam that were correctly classified—

that is, the percentage of dots to the right of the threshold line that are green in Figure 1:

Text

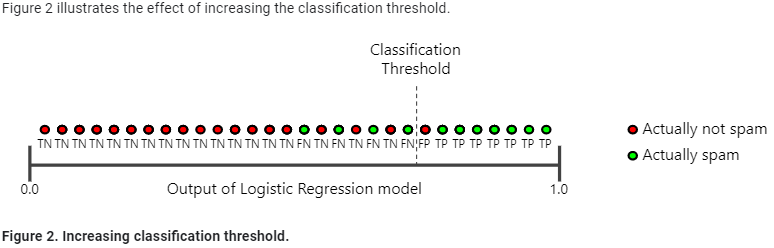
Description automatically generated with medium confidence

**Recall** measures the percentage of actual spam emails that were correctly classified—

that is, the percentage of green dots that are to the right of the threshold line in Figure 1:

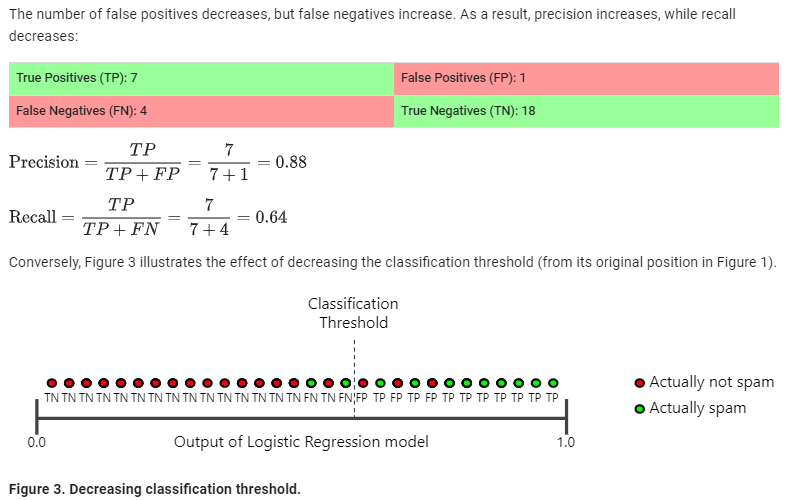
Text

Description automatically generated



**The number of false positives decreases, but false negatives increase.**

**As a result, precision increases, while recall decreases:**



**False positives increase, and false negatives decrease.**

**As a result, this time, precision decreases and recall increases:**

A picture containing timeline

Description automatically generated

**Key Terms**

|  |  |
| --- | --- |
|  [precision](https://developers.google.com/machine-learning/glossary#precision) |  [recall](https://developers.google.com/machine-learning/glossary#recall) |

**Section 6**

**Classification: ROC Curve and AUC**

**ROC curve - Receiver Operating Characteristic curve**

**An ROC curve (receiver operating characteristic curve)** is a graph showing the performance of a classification model at all classification thresholds. This curve plots two parameters:

* True Positive Rate
* False Positive Rate

Graphical user interface, text, application

Description automatically generated

An ROC curve plots TPR vs. FPR at different classification thresholds.

Lowering the classification threshold classifies more items as positive,

thus increasing both False Positives and True Positives.

The following figure shows a typical ROC curve.

Diagram

Description automatically generated

To compute the points in an ROC curve, we could evaluate a logistic regression model many times with different classification thresholds, but this would be inefficient. Fortunately, there's an efficient, sorting-based algorithm that can provide this information for us, called AUC.

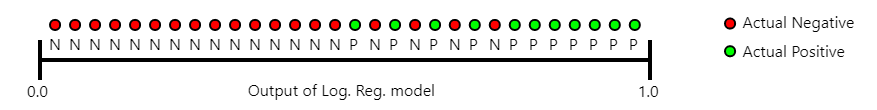
**AUC: Area Under the ROC Curve**

AUC stands for "Area under the ROC Curve." That is, AUC measures the entire two-dimensional area underneath the entire ROC curve (think integral calculus) from (0,0) to (1,1).

Chart, pie chart

Description automatically generated

AUC provides an aggregate measure of performance across all possible classification thresholds. One way of interpreting AUC is as the probability that the model ranks a random positive example more highly than a random negative example. For example, given the following examples, which are arranged from left to right in ascending order of logistic regression predictions:



AUC represents the probability that a random positive (green) example is positioned to the right of a random negative (red) example.

AUC ranges in value from 0 to 1. A model whose predictions are 100% wrong has an AUC of 0.0; one whose predictions are 100% correct has an AUC of 1.0.

**AUC is desirable for the following two reasons:**

AUC is scale-invariant. It measures how well predictions are ranked, rather than their absolute values.

AUC is classification-threshold-invariant. It measures the quality of the model's predictions irrespective of what classification threshold is chosen.

**However, both these reasons come with caveats, which may limit the usefulness of AUC in certain use cases:**

Scale invariance is not always desirable. For example, sometimes we really do need well calibrated probability outputs, and AUC won’t tell us about that.

Classification-threshold invariance is not always desirable. In cases where there are wide disparities in the cost of false negatives vs. false positives, it may be critical to minimize one type of classification error. For example, when doing email spam detection, you likely want to prioritize minimizing false positives (even if that results in a significant increase of false negatives). AUC isn't a useful metric for this type of optimization.

**Section 8**

**Classification: Prediction Bias**

Logistic regression predictions should be unbiased. That is:

"Average of predictions" should ≈ "average of observations"

Prediction bias is a quantity that measures how far apart those two averages are. That is:



**Note**: "Prediction bias" is a different quantity than bias (the b in wx + b).

A significant nonzero prediction bias tells you there is a bug somewhere in your model, as it indicates that the model is wrong about how frequently positive labels occur.

For example, let's say we know that on average, 1% of all emails are spam. If we don't know anything at all about a given email, we should predict that it's 1% likely to be spam. Similarly, a good spam model should predict on average that emails are 1% likely to be spam. (In other words, if we average the predicted likelihoods of each individual email being spam, the result should be 1%.) If instead, the model's average prediction is 20% likelihood of being spam, we can conclude that it exhibits prediction bias.

**Possible root causes of prediction bias are:**

* **Incomplete feature set**
* **Noisy data set**
* **Buggy pipeline**
* **Biased training sample**
* **Overly strong regularization**

You might be tempted to correct prediction bias by post-processing the learned model—that is, by adding a **calibration** **layer** that adjusts your model's output to reduce the prediction bias. For example, if your model has +3% bias, you could add a calibration layer that lowers the mean prediction by 3%. However, adding a calibration layer is a bad idea for the following reasons:

* You're fixing the symptom rather than the cause.
* You've built a more brittle system that you must now keep up to date.

If possible, avoid calibration layers. Projects that use calibration layers tend to become reliant on them—using calibration layers to fix all their model's sins. Ultimately, maintaining the calibration layers can become a nightmare.

**Note:** A good model will usually have near-zero bias.

That said, a low prediction bias does not prove that your model is good. A really terrible model could have a zero-prediction bias. For example, a model that just predicts the mean value for all examples would be a bad model, despite having zero bias.

**Bucketing and Prediction Bias**

Logistic regression predicts a value between 0 and 1. However, all labeled examples are either exactly 0 (meaning, for example, "not spam") or exactly 1 (meaning, for example, "spam"). Therefore, when examining prediction bias, you cannot accurately determine the prediction bias based on only one example; you must examine the prediction bias on a "**bucket**" of examples. That is, prediction bias for logistic regression only makes sense when grouping enough examples together to be able to compare a predicted value (for example, 0.392) to observed values (for example, 0.394).

**You can form buckets in the following ways**:

**Linearly breaking up the target predictions.**

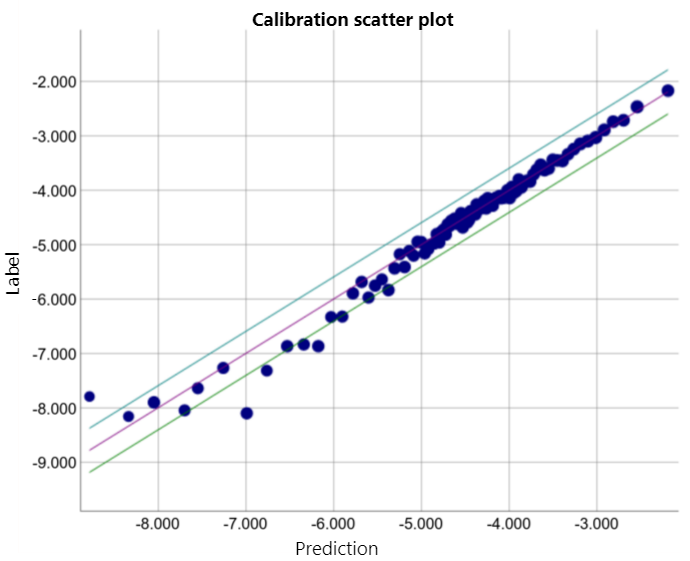
**Forming quantiles.**

Consider the following calibration plot from a particular model. Each dot represents a bucket of 1,000 values. The axes have the following meanings.

The x-axis represents the **average of values the model predicted for that bucket**.

The y-axis represents the **actual average of values in the data set for that bucket**.

Both axes are logarithmic scales.



**Why are the predictions so poor for only part of the model?** Here are a few possibilities:

* **The training set doesn't adequately represent certain subsets of the data space.**
* **Some subsets of the data set are noisier than others.**
* **The model is overly regularized. (Consider reducing the value of lambda.)**

**Key Terms**

|  |  |
| --- | --- |
|  [bucketing](https://developers.google.com/machine-learning/glossary#bucketing) |  [calibration layer](https://developers.google.com/machine-learning/glossary#calibration_layer) |
|  [prediction bias](https://developers.google.com/machine-learning/glossary#prediction_bias) |  |

**Regularization for Sparsity: L₁ Regularization**

Sparse vectors often contain many dimensions. Creating a [feature cross](https://developers.google.com/machine-learning/crash-course/feature-crosses/video-lecture) results in even more dimensions. Given such high-dimensional feature vectors, model size may become huge and require huge amounts of RAM.

In a high-dimensional sparse vector, it would be nice to encourage weights to drop to exactly 0 where possible. A weight of exactly 0 essentially removes the corresponding feature from the model. Zeroing out features will save RAM and may reduce noise in the model.

For example, consider a housing data set that covers not just California but the entire globe. Bucketing global latitude at the minute level (60 minutes per degree) gives about 10,000 dimensions in a sparse encoding; global longitude at the minute level gives about 20,000 dimensions. A feature cross of these two features would result in roughly 200,000,000 dimensions. Many of those 200,000,000 dimensions represent areas of such limited residence (for example, the middle of the ocean) that it would be difficult to use that data to generalize effectively. It would be silly to pay the RAM cost of storing these unneeded dimensions. Therefore, it would be nice to encourage the weights for the meaningless dimensions to drop to exactly 0, which would allow us to avoid paying for the storage cost of these model coefficients at inference time.

We might be able to encode this idea into the optimization problem done at training time, by adding an appropriately chosen regularization term.

Would L2 regularization accomplish this task? Unfortunately, not. L2 regularization encourages weights to be small but doesn't force them to exactly 0.0.

An alternative idea would be to try and create a regularization term that penalizes the count of non-zero coefficient values in a model. Increasing this count would only be justified if there was a sufficient gain in the model's ability to fit the data. Unfortunately, while this count-based approach is intuitively appealing, it would turn our convex optimization problem into a non-convex optimization problem. So, this idea, known as **L0 regularization** isn't something we can use effectively in practice.

However, there is a regularization term called **L1 regularization** that serves as an approximation to L0, but has the advantage of being convex and thus efficient to compute. So we can use L1 regularization to encourage many of the uninformative coefficients in our model to be exactly 0, and thus reap RAM savings at inference time.

Graphical user interface, text, application, email

Description automatically generated

You can think of the derivative of L2 as a force that removes x% of the weight every time. As [Zeno](https://wikipedia.org/wiki/Zeno%27s_paradoxes#Dichotomy_paradox) knew, even if you remove x percent of a number *billions of times*, the diminished number will still never quite reach zero. (Zeno was less familiar with floating-point precision limitations, which could possibly produce exactly zero.) At any rate, L2 does not normally drive weights to zero.

You can think of the derivative of L1 as a force that subtracts some constant from the weight every time. However, thanks to absolute values, L1 has a discontinuity at 0, which causes subtraction results that cross 0 to become zeroed out. For example, if subtraction would have forced a weight from +0.1 to -0.2, L1 will set the weight to exactly 0. Eureka, L1 zeroed out the weight.

L1 regularization—penalizing the absolute value of all the weights—turns out to be quite efficient for wide models.

Note that this description is true for a one-dimensional model.

**Key Terms**

|  |  |
| --- | --- |
|  [convex optimization](https://developers.google.com/machine-learning/glossary#convex_optimization) |  [L1 regularization](https://developers.google.com/machine-learning/glossary#L1_regularization) |
|  [L2 regularization](https://developers.google.com/machine-learning/glossary#L2_regularization) |  [one-hot encoding](https://developers.google.com/machine-learning/glossary#one-hot_encoding) |
|  [weight](https://developers.google.com/machine-learning/glossary#weight) | |