**Evaluating Machine Learning Models**

For any given prediction task, there are multiple machine learning models that one can employ: logistic regression, random forest, decision tree, KNN, etc. And we’d want some way to compare them all, to see which is better.

**Training/Testing**

To that end, you’d usually split the data into groups: a training group and a testing group. Often make the training group comprise 75% of the data and test on 25% of it. For sake of illustration, of the 12 rows of data, 9 will be for training and 3 for testing.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| xa1 | xb1 | xc1 | xd1 | y1 |
| xa2 | xb2 | xc2 | xd2 | y2 |
| xa3 | xb3 | xc3 | xd3 | y3 |
| xa4 | xb4 | xc4 | xd4 | y4 |
| xa5 | xb5 | xc5 | xd5 | y5 |
| xa6 | xb6 | xc6 | xd6 | y6 |
| xa7 | xb7 | xc7 | xd7 | y7 |
| xa8 | xb8 | xc8 | xd8 | y8 |
| xa9 | xb9 | xc9 | xd9 | y9 |
| xa10 | xb10 | xc10 | xd10 | y10 |
| xa11 | xb11 | xc11 | xd11 | y11 |
| xa12 | xb12 | xc12 | xd12 | y12 |

The set of rows of data in the training part that goes into the model we’ll call Xtrain = (**x**1, **x**2, **x**3, **x**4, **x**5, **x**6, **x**7, **x**8, **x**9). And the set of predictions the model is trying to fit, we’ll call ytrain = (y1, y2, y3, y4, y5, y6, y7, y8, y9). From this we’ll get some ML model y = f(**x**) = f(xa,xb,xc,xd). Then to test, we’d take rows from Xtest = (**x**10, **x**11, **x**12) and compare the ML model’s prediction ypred = [f(**x**10­­), f(**x**11), f(**x**12)] to ytest = (y10, y11, y12). To compare, we could find the sum of the squared errors if the predictions are quantitative, or maybe the confusion matrix score if the predictions are categorical.

**N-fold Cross Validation**

But our results (SSE, CMS, etc.) will often vary quite a bit depending on how we split the data up into training and testing sets. So one often employs n-fold cross validation. For instance, four fold cross-validation would split the data into 4 parts and assign the testing set to one of the four parts, and the training set to the other three parts. So they’d be split up like this:



And we’d train on the ¾ , test on the ¼, etc. And then take the average of the Confusion Matrix Scores, as our assessment of the model’s accuracy.

If have a small dataset, then you could use **LOOCV** (*Leave One Out Cross-Validation*). Here, you’d split your data into n parts (n being the size of the data itself). You’d set aside the first row, train on the other n-1, and then test on the first row. Next you’d set aside the second row, train on the other n-1, and then test on the second row, etc., until the end. So we’d have n tests on a single (different) row each time.

**N-fold Cross Validation Stratified**

I think in this option one tries to keep the proportion of positive to negative outcomes the same in the training and testing sets. This is so that we don’t train our model on a set with mostly negative outcomes, and test on a set with mostly positive outcomes, say. This is probably a bigger concern when positive or negative outcome is rare. This is also a concern when have a lot of different classification outcomes, but not a whoooole lot of data. And so you might end up with an Xtrain, ytrain with few examples of a certain class. So Stratified Cross Validation would be helpfull here. Also, you will probably want to use n ~ 10 folds or so, to increase the amount of data you have to train on.

In general, we’ll see the need for large folds, and stratified training sets when the variation in our cross validated accuracy scores seems inordinately high.

**Confusion Matrix**

To describe the accuracy of a categorical predictive model, we can create a confusion matrix, which tabulates all Outcomes in rows and crosses them with Predictions in colums. This table compares outcomes and predictions of a test for illness.

|  |  |  |
| --- | --- | --- |
| n = 165 | **Prediction: Sick** | **Prediction: Well** |
| **Outcome: Sick** | 50 | 10 |
| **Outcome: Well** | 5 | 100 |

So in row 1 we see there were 50 + 10 = 60 sick people. And in row 2 we see there were 5 + 100 = 105 well people. And each of these row groups are split according to how the test predicted their status. And the matrix below compares outcomes and predictions for favorite movie (out of three possible).

|  |  |  |  |
| --- | --- | --- | --- |
| n = 611 | **Prediction: Troll 2** | **Prediction: Gore Police** | **Prediction: Cool as Ice** |
| **Outcome: Troll 2** | 12 | 102 | 93 |
| **Outcome: Gore Police** | 112 | 23 | 77 |
| **Outcome: Cool as Ice** | 83 | 92 | 17 |

The first row indicates that 12 + 102 + 93 = 207 people preferred Troll 2 most. The second row indicates 212 people liked Gore Police best. And the third row indicates 192 people liked Cool as Ice the most. And each of these rows groups is split according to how the test predicted their status. The off-diagonal elements give us false-positives and false-negatives. And sum of all entries should be the number of trials, n. Let’s introduce some notation:



And note following correspondances,



where · mean intersection of sets, and + means union of sets. A graphical representation of these is given below. Consider a universal set. If the feature is present, then it’s in the diamond-hatched box, otherwise, it’s in the surounding square-hatched area. If the feature is predicted to be present, then it’s in the rose box, otherwise, outside the rose box. True Positives are the rose diamond-hatched area. False Positives are the rose square-hatched area. True Negatives are the grey square-hatched area. And False Negatives are the green diamond-hatched area.

A diagram of false positive and true positive

Description automatically generated

All of these guys can be determined from the Confusion Matrix. And there are some additional quantities of interest we could calculate. For instance, let’s consider Accuracy,



I’ve seen people write this as,



where Tr(M) is the trace of the confusion matrix, M, and the norm, |M|, is the sum of all elements of the confusion matrix M. This would be the number of true predictions divided by the total number of predictions. So can consider this to be the probability of the prediction being true. It seems to treat False Positives and False Negatives as equally bad.

So this is the test we most often use for evaluating a model’s performance. But this isn’t always the best metric. One guy says that if have rare disease that afflicts only 0.1% of population, then a test which predicts no one has the disease would still be 99.9% accurate. So generally, imbalanced data sets may not be good candidates for this metric. It’s a little awkward to draw, but geometrically, it’d be the sum of the rose and grey areas, divided by the total universal set’s area,

A person standing next to a grid

Description automatically generated with medium confidence

Another one is:



Graphically, this would be the ratio of the rose diamond-hatched area to the entire diamond-hatched area. *It’s true positives/actual positives*. This is also called the True Positives Rate. The higher the sensitivity, the greater fraction of true cases our ‘test’ picks up. So given that the case is true, it’s the probability that test will predict it’s true. Can also say it’s higher the fewer False Negatives we have.

A diagram of a positive and false negative

Description automatically generated

and,



Graphically, this would be ratio of the green diamond-hatched area to the total diamond-hatched area.

A diagram of a positive and false negative

Description automatically generated

Next we have the Specificity.



Graphically, this would be the ratio of the grey square-hatched area to the entire square-hatched area. *It’s* *true negatives/actual negatives*. The higher the specificity, the greater fraction of negatives cases our ‘test’ picks up. So given that the case is negative, it’s the probability the test will say it’s negative. Can also say it’s higher the fewer False Positives we have.

A screen shot of a computer screen

Description automatically generated

And then:



which is the ratio of the rose square-hatched area to the total square-hatched area. This is also called the False Positives Rate.

A screen shot of a computer screen

Description automatically generated

And there are two more that are often discussed.



Graphically, this would be the ratio of the rose diamond-hatched area to the entire rose area. *It’s true positives/predicted positives.* I guess you could say it is the probability that case is positive if test says positive, *P(Outcat|Predcat)*. So assesses how accurate our positive predictions are. Can also say it’s higher the fewer False Positives we have.

A pink paper with red text

Description automatically generated

and last,



Graphically, this would be the ratio of the rose diamond-hatched area to the entire diamond-hatched area. *So this is the same as sensitivity*. *It’s true positives/actual positives*. It’s the probability the test will say positive if case is actually positive, *P(Predcat|Outcat)*. So in this sense, it’s the converse of precision. Can also say it’s higher, the fewer False Negatives we have.

A diagram of a positive and false negative

Description automatically generated

These are related via Baye’s Rule-ish:



Last, we have the so-called F1 score. This is the ‘harmonic mean’ of Precision (P), and Recall (R):



So F1 = 2RP/(R+P). We’d want R and P to be close to 1, and so F1 to also be close to 1. A bad F1 score would be F1 ~ 0.

So if Sensitivity = 1 → False Neg = 0. And if Specificity = 1 → False Pos = 0. Now if our ML model just made the prediction of Cat for every single case, then can see Sensitivity = 1 and Specificity = 0 (can always identify the Category but never properly identify its absence). On the other hand, if it never made a prediction of Cat, but always of , then Sensitivity = 0 and Specificity = 1 (always properly identify absence of Category, but never properly identify its presence). The best ML model would get 1 for Sensitivity and 1 for Specificity, as its predictions would always match the outcomes. Equivalently the best models would have 0 for False Positives and 0 for False Negatives.

**Why might we be interested in one metric vs. the other?**

Accuracy is our go-to metric. But this isn’t always the best. One guy says that if have rare disease that afflicts only 0.1% of population, then a test which predicts no one has the disease would still be 99.9% accurate. So generally, imbalanced data sets may not be good candidates for this metric.

Consider Sensitivity = Predicted Positive/Actual Positive. And Specificity = Predicted Negative/ Actual Negative. Ideally both fractions would be 1. This would be perfect prediction. But this can’t usually be managed. Often we have to err on one side or the other. We trade greater Sensitivity for lesser Specificity and vice versa. If we were handing out a drug to cure a non-fatal, but severely debilitating disease. Then if the side effects of our drug were benign, then maybe we’d want the model with greatest Sensitivity, as there is no harm in treating someone without the disease. In other words, we’d want to be minimizing False Negatives. But if the drug were fatal to people without the disease, then maybe we’d rather a higher Specificity instead, as we’d want to minimize False Positives.

**Example**

Here’s a confusion matrix plotted via Seaborn.

A diagram of a dog

Description automatically generated

So TP = 4, TN = 1, FP = 3, FN = 2. And then,

Sensitivity = TP/(TP + FN) = 4/(4+2) = 0.67.

Specificity = TN/(TN + FP) = 1/(1+3) = 0.25.

Precision = TP/(TP + FP) = 4/(4+3) = 0.43.

Recall = TP/(TP + FN) = 4/(4+2) = 0.67.

F1 → F1-1 = (P-1 + R-1)/2 = (7/4 + 6/4)/2 = 13/8 → F1 = 8/13 ≈ 0.62

**Example**

Consider our medicinal example,

|  |  |  |
| --- | --- | --- |
| n = 165 | **Prediction: Sick** | **Prediction: Well** |
| **Outcome: Sick** | 50 | 10 |
| **Outcome: Well** | 5 | 100 |

Highlighting in green Outcome = Sick, and in red Outcome = . And we have:

The sensitivity and specificity of identifying a sick person would be:



**Example**

Say we had n = 100. And say 90 are actually sick, 10 are healthy. But our test says all are sick. What would this look like?

|  |  |  |
| --- | --- | --- |
| n = 100 | **Predicted: Sick** | **Predicted: Healthy** |
| **Outcome: Sick** | 90 | 0 |
| **Outcome: Healthy** | 10 | 0 |

Then we have:



And now say our test says all are healthy. What would this look like?

|  |  |  |
| --- | --- | --- |
| n = 100 | **Predicted: Sick** | **Predicted: Healthy** |
| **Outcome: Sick** | 0 | 90 |
| **Outcome: Healthy** | 0 | 10 |

Then we have:



**Example**

Now consider our movie matrix. What is the sensitivity and specificity of predicting whether someone prefers Troll 2 to all the other movies?

|  |  |  |  |
| --- | --- | --- | --- |
| n = 611 | **Prediction: Troll 2** | **Prediction: Gore Police** | **Prediction: Cool as Ice** |
| **Outcome: Troll 2** | 12 | 102 | 93 |
| **Outcome: Gore Police** | 112 | 23 | 77 |
| **Outcome: Cool as Ice** | 83 | 92 | 17 |

Highlighting in green Outcome = Troll 2, and in red Outcome = . Note Sensitivity only has to do with green, and Specificity with red. And we have:



What is the sensitivity and specificity of predicting whether someone prefers Gore Police to all the other movies?

|  |  |  |  |
| --- | --- | --- | --- |
| n = 611 | **Prediction: Troll 2** | **Prediction: Gore Police** | **Prediction: Cool as Ice** |
| **Outcome: Troll 2** | 12 | 102 | 93 |
| **Outcome: Gore Police** | 112 | 23 | 77 |
| **Outcome: Cool as Ice** | 83 | 92 | 17 |

Highlighting in green Outcome = Gore Police, and in red Outcome = . Note Sensitivity only has to do with green, and Specificity with red. And we have:



So there.

**ROC, PR, and AUC**

When using logistic regression type models (and could be decision trees based on logistic regression), a definitive classification is not typically rendered. Rather we have probabilities of being in a class. Typically we would say if p > 0.5, then put it in the class, and if p < 0.5, then don’t. But we can adjust this threshold, pcut = 0.5 to other values. And pcut = 0.5, is often not the best one. The considerations below can help us determine which is best. Now that I think about it, really any model with a hyperparameter (like pcut for logistic regression, or α for pruning decision trees, or n\_neighbors for KNN, etc.) is amenable to this sort of analysis.

**ROC (Sensitivity vs. Specificity)**

A convenient way to graphically compare the different models is to create an ROC (Receiver Operator Characteristic – why?) curve. This is a plot of the Sensitivity (i.e. Recall, True Positives Rate) vs. 1 – Specificity (i.e. False Positives Rate) as a parametric function of the probability cutoff, pcut. A perfect model would correspond to Sensitivity = 1 and Specificity = 1. This would correspond to the coordinate (0, 1).

Chart, line chart

Description automatically generated

For instance, let’s consider a few hypothetical models. To start, say we have n elements, n1 of which are ‘Positive’ and n0 of which are ‘Negative’. So there is a p = n1/n chance of a given element to be Positive. Now let’s consider a random decision tree model, with no predictive power. So it would assign random probabilities of being in the Pos category to each of the elements. We’ll take these probabilities to be uniformly distributed between 0 and 1. Below I’ve separately drawn the probability distribution of being classified as positive for negative elements (red line) and positive elements (green line). And let our probability cutoff be pcut, somewhere beween 0 and 1.

A graph with lines and numbers

Description automatically generated with medium confidence

Then the Sensitivity of our random decision tree model + probability cutoff will be:



(In second line, I think I can translate to probabilities as I did because the outcome and prediction are independent in this model, and remember everything to the right of pcut is classified as Pos) and the Specificity will be:



which means,



So Sensitivity and 1 – Specificity are identical. Therefore when plotted on the ROC graph, we just get a straight line which goes from coordinate (1,1) when pcυt = 0, to coordinate (0,0) when pcut = 1. That’s our red diagonal line up above.

Now suppose we had a decent predictive model so that all the Pos cases had probability pT kind of normally (triangularly) disributed about 0.7, say, and all the Neg cases had probability pF normally (triangularly) distributed about 0.4. And let our probability cutoff be pcut, somewhere beween 0 and 1.

A diagram of a graph

Description automatically generated

Then when pcut = 0, we’d classify everything as Pos, and so, having picked up all the actual Pos cases, we’d have Sensitivity = 1. But we’d have picked up none of the Neg cases, because we would’ve classified nothing as false, so our Specificity = 0. And since then 1-Specificity = 1, this corresponds to coordinate (1,1). So we start in the same spot as our random model. As pcut increases, we’d still be keeping most of the positives, so Sensitivity would still be around 1. And we’d begin to pick up some of the negatives, so Specificity would begin increase, meaning 1- Specificity would begin to decrease. So now we’re at coordinate (something less than 1,1). As pcut continues to increase, we’d begin to pick up (i.e., correctly classify) all the Negatives, and so Specificity would be approaching 1, meaning 1- Specificity would be approaching 0. And we’d begin to lose some of the Positives, so Sensitivity would begin to drop from 1. So now we’d be at coordinate (something close to 0, something less than 1). And finally as pcut → 1, we’d lose all the Positives and have acquired all the Negatives. So Sensitivity would be 0, and Specificity would be 1, which means 1- Specificity would be 0. So our coordinate would now be about (0,0). This more or less traces the paths of the green and blue curves. Now consider this guy,

A diagram of a model

Description automatically generated with medium confidence

As pcut starts from 0, we get Sensitivity = 1, 1-Specificity = 1 – 0 = 1. As pcut increases up to midway between the red and green curves, the Sensitivity will remain 1, and 1-Specificity will gradually decrease from 1 – 0 = 1, to 1 – 1 = 0, as more and more negative cases are picked up. Finally, as pcut increases from midway between the curves to 1, Sensitivity will decrease to 0, while 1-Specificity will remain 1-1 = 0. So the ROC curve it traces out would in fact completely hug the borders of the square. And thus its area (AUC) would be 1. So we can treat the area under the ROC as measure of how well the model resolves the two cases, or in other words, how non-overlapping the model-assigned probability distributions are.

Will also note that the pcut we’d choose, in any event, is the one for which (Sensitivity, 1 – Specificity) is closest to (1, 1 – 1 = 0).

**PR (Precision vs. Recall)**

Another metric I’ve seen, besides ROC is PR = Precision-Recall. Here’s a precision-recall curve, plotted parametrically as a function of pcut. A perfect model would have Precision = 1, Recall = 1.

Chart, line chart

Description automatically generated

For instance, let’s consider a few hypothetical models. To start, say we have n elements, n1 of which are ‘Positive’ and n0 of which are ‘Negative’. So there is a p = n1/n chance of a given element to be Positive. Now let’s consider a random decision tree model of this set of elements – i.e., one which assigns random probabilities to each of the Neg elements, and to each of the Pos elements. We’ll take their probabilities to be uniformly distributed between 0 and 1. And let our probability cutoff be pcut, somewhere beween 0 and 1.

A graph with lines and numbers

Description automatically generated with medium confidence

Then the Precision of our random decision tree model + probability cutoff will be:



And the Recall is Sensitivity, as we recall (ha!), which we’ve already calculated:



So as pcut goes from 0 to 1, our coordinate (Recall, Precision) is going from (1, p) to (0, p). This is our red line.

Now suppose we had a predictive model so that all the Pos cases had probability pT kind of normally (triangularly) disributed about 0.7, say, and all the Neg cases had probability pF normally (triangularly) distributed about 0.4. And let our probability cutoff be pcut, somewhere beween 0 and 1.

A diagram of a graph

Description automatically generated

Then when pcut = 0, we’d classify everything as Pos. And Recall = TP/(TP + FN) = 1/(1+FN/TP) = 1/(1+ 0/p) = 1. So our coordinate so far is (1,p). This is as in the random model. As pcut increases, TP should stay the same roughly, but our FP rate should drop. Thus Precision should go up. And our FN rate should go up, from its initial value of 0, as now we’d begin to be classifying things as negative. So I guess Recall would drop. So now our coordinate is (something less than 1, something greater than p). As pcut continues to increase, to say near x = 0.5, our TP rate is still pretty high, and our FP is getting much lower. So Precision should still increase a bit. Our FN rate should continue to increase as well, and so Recall should get still smaller. So now our coordinate is (something closer to 0, something close to 1). And finally as pcut → 1, we’d stop classifying anything as positive and so TP → 0. FP would go to zero too. This is a little ambiguous. So let’s use formula Precision = Outcat·Predcat/Predcat (remember · means intersection). Well as pcut → 1, I feel like all predictions of True must be likely True. So Outcat·Predcat = Predcat. And this would make Precision → 1. Hmmmm. And as TP → 0, while FN → some constant, can see that Recall would go to 0. So our coordinate would be (0,1). Interesting that it doesn’t go back down to (0,p). This roughly traces the shape of the green and blue curves. Now consider this guy,

A diagram of a model

Description automatically generated with medium confidence

Can see this since when pcut = 0, we’ll have Recall = 1, and Precision = p. So we start in the usual place. As pcut increases, Recall (sensitivity) will stay 1, and Precision will increase as we’re excluding more and more negative cases, and so making the likelihood of being Pos, if test says Pos, higher. As pcut gets to around x = 0.55, we’ll have Recall = 1 still, and Precision will also equal 1, as all Neg cases are excluded by the cutoff, and so likelihood of being Pos, if test says Pos, is 1. So we’ll be at coordinate (1,1). Then as pcut increases from 0.55 → 1, we should have Recall (sensitivity) drop to 0, while Precision will stay 1. And so we’ll eventually end up at coordinate (0, 1). Thus our curve will hug the contours of that upper rectangle. And it will have the largest possible area under the curve (AUC). Again, large AUC means greater distinguishability of probability distributions. And we’d choose pcut that brings our model closest to (Recall, Precision) = (1,1).

**AUC (Area Under Curve)**

As we worked out, above, the more resolved the probabilities are for positive and negative cases, in the ML model, the higher the AUC (area under curve) score will be. And so we can say that AUC is a measure of the performance of the model. A guy on Datacamp says that AUC can be interpreted as a probability, namely, the probability that a randomly selected positive data point will have a higher assigned probability value than a randomly selected negative data point. And this makes sense given the curves we drew above. He also says that ROC-AUC is the most commonly used metric to assess the performance of a binary clasasification problem, whereas accuracy, and the classification matrix more generally, is the one most often used for trinary or higher classification problems.

**Accuracy**

Curious as to how Accuracy depends on the probability cutoff. First consider random predictive model,

A graph with lines and numbers

Description automatically generated with medium confidence

Then accuracy is:



Hmmm. Looks like accuracy is maximized when pcut = 0 or 1, depending on the larger of n1 and n0. Moving on to this guy,

A diagram of a graph

Description automatically generated

Accuracy is (pT is green curve, and pF is red curve):

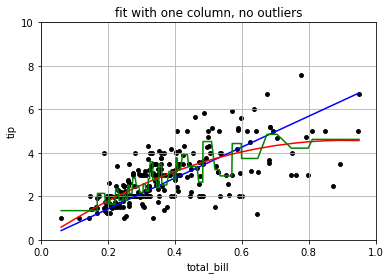


Can kind of see that this is maximized when pcut is roughly in between the curves, assuming n0 and n1 are comparable. If n0 >> n1, then pcut would probably skew more towards the right.

**MSE, MAE, R2**

For regression models, the criteria are a little more straightforward. Typically one uses the mean square error (MSE), mean absolute error (MAE), or R2 = 1 – MSEf/MSEμ coefficient to evaluate how well the curve/model, f, fits the data.

Think I’ll put this observation here. So I did linear, quadratic, and decision tree regression on this dataset. The quadratic regression came out slightly best. But observe that it’s only good within the range of the data. If we were to extrapolate beyond it, we’d get crazy result. It would predict, at best, that the tips no longer increase with total\_bill size. And at worst, if we really followed the quadratic curve, it’d say tips would start to go down. So really, the linear model is best if we’re going to be extrapolating. It’s pretty straightforward to know when you going beyond your data in 1D. But what about in higher dimensions?



**Employing**

If your training error is generally much lower than your testing error, then you’re overfitting the data. These errors should be as low as possible, but about the same.

So once you’ve determined which model gives the best predictions, I presume you would train it on all the data you’ve got (combine all training and testing data into one training data set), and then deploy the model for new real world applications.