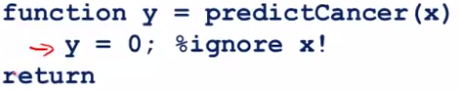
***Handling Skewed Data***

**I. ERROR METRICS FOR SKEWED CLASSES**

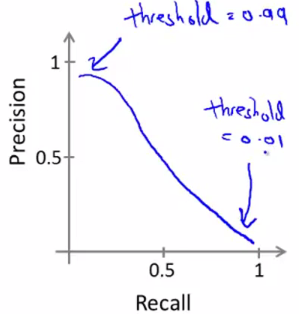
* So, it’s important to have **error metrics** = a single real number evaluation metric for a learning algorithm to tell how well it's doing.
* In the context of evaluation of error metrics, there’s 1 important case, where it's particularly tricky to come up with an appropriate error/evaluation metric for a learning algorithm = **skewed classes**.
* Consider the problem of cancer classification, where we have features of medical patients + we want to decide whether they have cancer or not (y = 1 if patient has cancer)
* We train regression classifier + on a test set we get 1% error.
* 99% correct diagnoses seems like a really impressive result, but say we find out only 0.5% of patients in training + test sets actually have cancer.
* In this case, the 1% error no longer looks so impressive.
* In particular, here's a piece of non-learning code that takes as input features x + it ignores it:



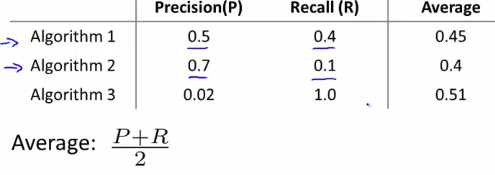
* It just sets y = 0 + always predicts “nobody has cancer”
* This algorithm would actually get 0.5% error, which is even better than the 1% error we got
* So when the ratio of positive to negative examples is very close to 1 of 2 extremes, we call it the **case of skewed classes** (ex: # of positive examples is much, much smaller than # of negative examples b/c y = 1 so rarely)
* In it, we have a lot more of examples from 1 class than from the other class.
* By just predicting y = 0 or y = 1 all the time, an algorithm can do pretty well.
* The problem with using classification error/accuracy as our evaluation metric is the following:
* Say you have 1 learning algorithm w/ 99.2% accuracy (a 0.8% error)
* You make a change to the algorithm + now are getting 99.5% accuracy (0.5% error)
* Is this an improvement to the algorithm or not?
* 1 of the nice things about having a single real number evaluation metric is it helps us quickly decide if we made a good change to the algorithm or not.
* *By going from 99.2% accuracy to 99.5% accuracy, did we do something useful or just replace code w/ something that just predicts y = 0?*
* So, w/ very skewed classes, it becomes much harder to use *just* classification accuracy, b/c we can get very high classification accuracies/very low errors + it's not always clear if doing so is really improving quality of the classifier
* Predicting y = 0 all the time doesn't seem like a particularly good classifier.
* Just predicting y = 0 more often can bring error down to maybe as low as 0.5%.
* When faced w/ such skewed classes, we’d want to come up w/ a different error/evaluation metric.
* 1 such evaluation metric is **precision recall.**
* Say we’re evaluating a binary classifier on a test set
* What our learning algorithm will do is predict some class value for each example in the test set
* If the actual class = 1 + predicted class = 1 🡪a **true positive**
* If our learning algorithm predicts negative/class = 0 + the actual class is also class 0, 🡪 **true negative**
* If our learning algorithm predicts class = one but actual class = zero, that's a **false positive**.
* A **false negative** = our algorithm predicted 0, but the actual class was 1.
* Now we compute 2 numbers:
* **Precision** = # of true positives / # we predicted as positive (TP + FP)
* Of all patients where we've predicted they *have* cancer, what fraction *actually has* cancer?
* High precision is good, b/c it means that of the group of patients we said had cancer, most actually do have cancer.
* **Recall** = # of true positives / # *actual* positive (TP + FN)
* Say all patients in in test or CV set actually have cancer, what fraction of them did we correctly detect as having cancer?
* Once again, having a high recall would be a good thing.
* Computing precision + recall will usually give us a better sense of how well our classifier is doing.
* In particular, w/ have a learning algorithm that predicts y = 0 all the time, this classifier will have a recall = 0 b/c there won't be any TP’s
* That's a quick way to recognize that a classifier that predicts y = 0 all the time isn't a very good one
* More generally, even for settings w/ very skewed classes, it's not possible for an algorithm to "cheat" + somehow get a very high precision + very high recall by doing something simple like predicting y = 0 or predicting y = 1 all the time.
* We're much surer a classifier of w/ high precision or high recall is a good classifier
* This gives us a more useful evaluation metric + a more direct way to understand whether our algorithm may be doing well.
* In the definition of precision and recall, usually we use the convention that y = 1 in the presence of the *rarer class*.
* If trying to detect rare conditions such as cancer, precision + recall are defined setting y = 1, rather than y = 0, b/c it’s the presence of that rare class that we're trying to detect.
* By using precision + recall, we find that what happens is, even w/ very skewed classes, it's not possible for an algorithm to "cheat" + predict y = 1 or predict y = 0 all the time
* If a classifier is getting high precision + high recall, we are confident the algorithm must be doing well, even w/ very skewed classes.

**II. TRADING OFF PRECISION AND RECALL**

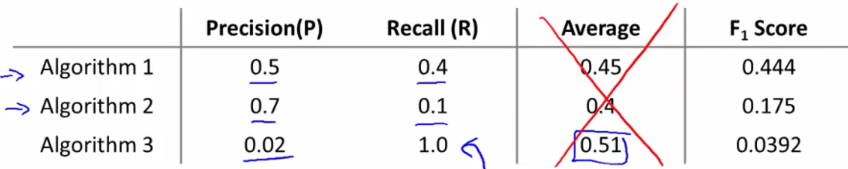
* For many applications, we'll want to somehow control the trade-off between precision + recall
* Let's continue our cancer classification example, where y = 1 if patient has cancer + y = 0 otherwise.
* Say we've trained in logistic regression classifier which outputs probability between 0-1.
* As usual, predict y = 1 if h(x) >= 0.5 + predict y = 0 if h(x) < 0.5
* But, suppose we want to predict that a patient has cancer *only if we're very confident they really do*.
* B/c if you tell a patient they have cancer, it's going to give them a huge shock
* Its very bad news, + they may end up going through a pretty painful treatment process + so on.
* 1 way to do this would be to modify the algorithm + set the threshold to 0.7
* This is like saying “tell someone they have cancer only if we think there's a >= 70% chance they do”
* We end up w/ a classifier w/ *higher precision*, b/c out of all patients we think have cancer, a higher fraction will actually turn out to have it b/c we made predictions only if we were very confident
* In contrast, this classifier will have *lower recall* b/c now we predict y = 1 on a *smaller number* of patients
* Now, can even take this further + set the threshold to 0.9 predict y = 1 only if we’re > 90% certain a patient has cancer.
* A large fraction of those patients will turn out to have cancer, which makes it a *higher precision classifier* w/ *lower recall* b/c we want to *correctly* detect that those patients have cancer.
* Now consider a different example + suppose we want to avoid *missing* too many *actual* cases of cancer (avoid false negatives)
* If a patient actually has cancer, but we fail to tell them, that can be really bad as they're not going to go for treatment + they die
* So, suppose that, when in doubt, we want to predict that y = 1/predict they have cancer so that they at least look further into it + can get treated in case they do turn out to have cancer.
* In this case, rather than setting a higher probability threshold, we set it to a lower value, like 0.3
* If we think there's > 30% chance they have cancer, we are more conservative + tell them they may have cancer so they can seek treatment if necessary.
* In this case we’d have a *higher recall* classifier, b/c we're going to be *correctly* flagging a higher % of all patients that *actually do* have cancer.
* But we're going to end up w/ lower precision b/c a higher fraction of patients we said have cancer will turn out not to have cancer after all.
* Why we might want have higher precision or higher recall can work both ways.
* The general principle is🡺 depending on whether you want higher precision + lower recall, or higher recall + lower precision, you end up predicting y = 1 when h(x) is greater than *some* threshold.
* In general, for most classifiers, there is going to be a trade-off between precision and recall
* As you vary the threshold, we can actually plot out a curve that trades off precision and recall where higher thresholds have higher precision/lower recall and vice versa



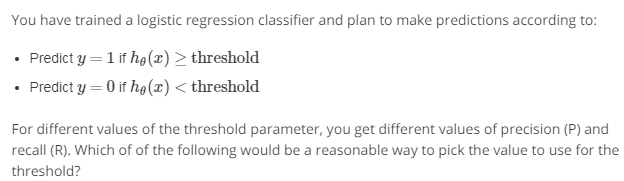
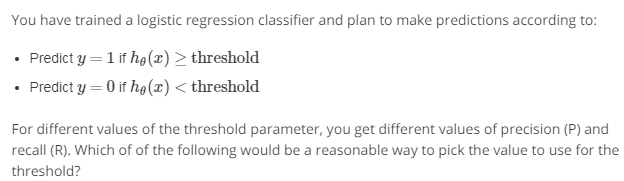
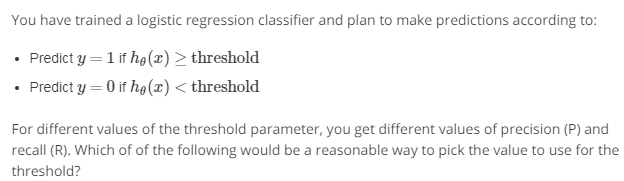
* A Precision-recall curve can have many different shapes, depending on the details of the classifier.
* So, this raises another interesting question: *is there a way to choose this threshold automatically?* Or
* More generally, if we have a few different algorithms/ideas for algorithms, how do we compare different precision-recall numbers?
* Suppose we have 3 different learning algorithms/the same algorithm w/ 3 different threshold values
* *How do we decide which of these algorithms is best?*
* 1 of the things we talked about earlier is the importance of a single real number evaluation metric that tells you how well is your classifier doing.
* But by switching to the precision-recall metric, we've actually lost that + now have *two* real numbers
* Often, if we trying to compare Algorithm 1 + Algorithm 2, we end up asking ourselves “is precision of 0.5 + a recall of 0.4 better or worse than precision of 0.7 + recall of 0.1?”
* If you end up having to sit around + think + make these kinds of decisions, it really slows down the decision-making process for what changes are useful to incorporate into an algorithm.
* Whereas in contrast, w/ a single real number evaluation metric, like a number that just tells us if algorithm 1 or algorithm 2 is better, it helps us quickly decide which algorithm to go w/ + to quickly evaluate different changes we may be contemplating for an algorithm.
* *So how can we get a single real number evaluation metric?*
* 1 natural thing that you might try is to look at what classifier has the highest average precision + recall value



* This turns out not to be such a good solution b/c w/ a classifier that predicts y = 1 all the time, then we can get a very high recall + very low value of precision.
* Conversely, w/ a classifier that predicts y = 0 almost all the time, we actually end up w/ a very high precision w/ a very low recall.
* So, neither of the 2 extremes of either a very high or a very low threshold will give a particularly good classifier
* The average is actually highest for Algorithm 3, even though we can get that same performance by predicting y = 1 all the time, + that's just not a very good classifier
* A classifier that just prints out y = 1 is not good
* So Algorithm 1 or Algorithm 2 would be more useful than Algorithm 3, but Algorithm 3 has a higher average value of precision-recall
* So this average of precision + recall is not a particularly good way to evaluate a learning algorithm.
* In contrast, there's a different way for combining precision and recall, **the F Score**





* Algorithm 1 has the highest F Score, Algorithm 2 has the 2nd highest, Algorithm 3 has the lowest.
* The F or **F1 Score** is a little bit like taking the average of precision and recall, but gives a higher weight to the lower value of the two
* The numerator takes a product of precision and recall, so if either is 0, the F Score will be = 0.
* So in that sense, it kind of combines precision + recall, but for the F Score to be large, *both* precision + recall have to be pretty large.
* There are many different possible formulas for combing precision + recall, and this F Score formula is 1 out of a much larger number of possibilities
* But traditionally this is what people in ML seem to use, as it usually gives the effect you want
* If either a precision or recall = 0, we get a very low F Score, so to have a high F Score, we need precision or recall = 1
* Concretely, if P or R=0, the F Score = 0, whereas a perfect F Score occurs when precision + recall both = 1
* Intermediate values between 0-1 usually give a reasonable rank ordering of different classifiers.
*   
* 