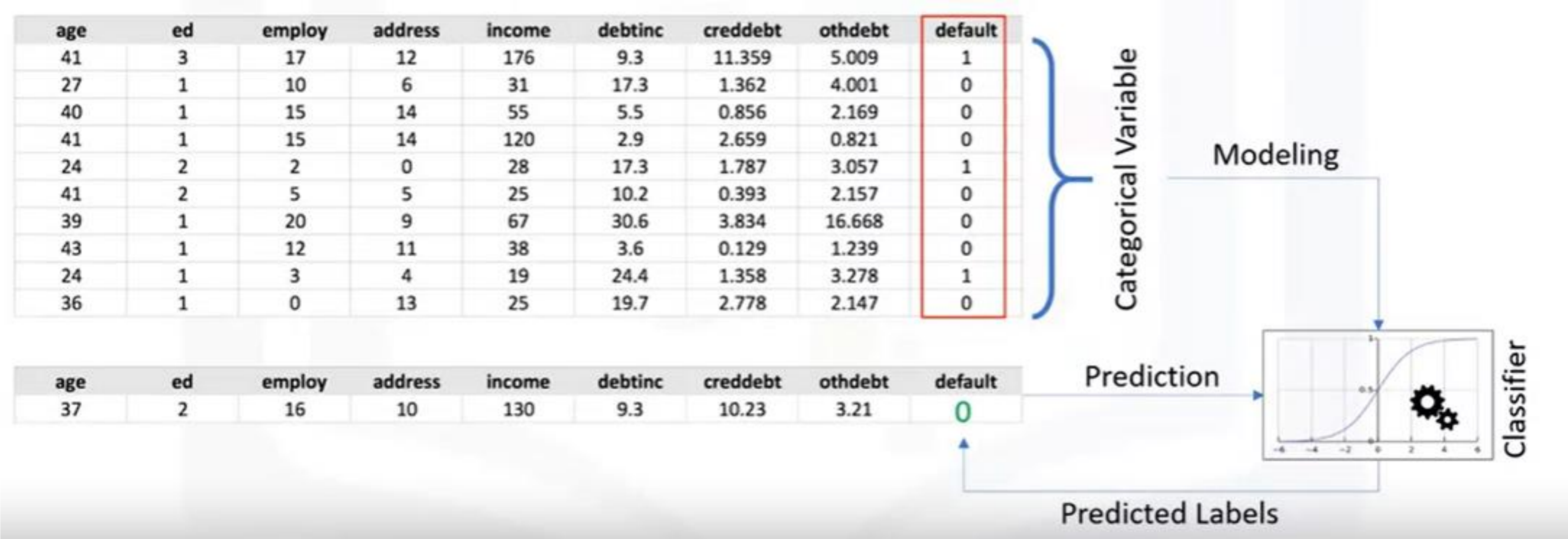
Classification and Evaluation

What is classification

- Two most common supervised learning approaches are:
  - Regression
  - Classification
- So, classification is a supervised learning approach
- The main objective is to categorizing some unknown items into a discrete set of classes or categories
- The target field is a categorical variable with discrete values

How does classification work



age	ed	employ	address	income	debtinc	creddebt	othdebt	default
41	3	17	12	176	9.3	11.359	5.009	1
27	1	10	6	31	17.3	1.362	4.001	0
40	1	15	14	55	5.5	0.856	2.169	0
41	1	15	14	120	2.9	2.659	0.821	0
24	2	2	0	28	17.3	1.787	3.057	1
41	2	5	5	25	10.2	0.393	2.157	0
39	1	20	9	67	30.6	3.834	16.668	0
43	1	12	11	38	3.6	0.129	1.239	0
24	1	3	4	19	24.4	1.358	3.278	1
36	1	0	13	25	19.7	2.778	2.147	0

age	ed	employ	address	income	debtinc	creddebt	othdebt	default
37	2	16	10	130	9.3	10.23	3.21	0

- You have a set of labeled dataset
- You train the model
- Then use the model to determine the label for a new unlabeled instance
- For example, in the above table, we have a data set of customers where were able to pay the loan or not
- After building a classifier model, we will lbe able to use the model to assess a new customer will be able to pay the loan or not
  - Based on that we can deny loan application
  - Or recommend them for other available options

Binary Classification

- The previous example was a binary classification.
- The goal of classification is to learn a mapping from input x to outputs y, where $\mathrm{y} \in \{1, \ldots.. \mathrm{C}\}$, with C being the number of classes.
- If C = 2, then this is called binary classification
- If C > 2, then this is called multiclass classification
- If the class labels are not mutually exclusive, then this is called **multi-label** classification
- - You can think of this as predicting multiple related binary class labels
- Difference is that in multi-class problems the classes are mutually exclusive, whereas for multi-label problems each label represents a different classification task, but the tasks are somehow related (so there is a benefit in tackling them together rather than separately).
- For example, in the famous leptograspus crabs dataset there are examples of males and females of two color forms of crab. You could approach this as a multi-class problem with four classes (male-blue, female-blue, male-orange, female-orange) or as a multi-label problem, where one label would be male/female and the other blue/orange. Essentially in multi-label problems a pattern can belong to more than one class.

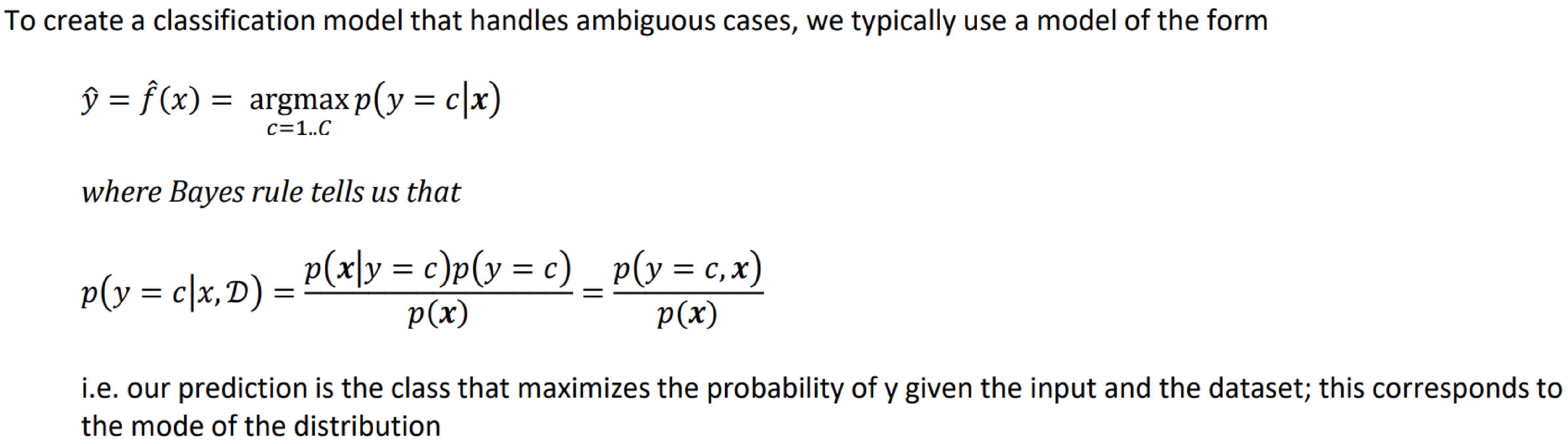
Classification

- Classification can be formalized as function approximation:
- we assume that ***y* = *f(x)*** for some unknown function ***f***, and the goal of learning is to estimate the function f given a labeled training set, and then to make predictions using

$$\hat{y} = \hat{f}(\boldsymbol{x})$$

- Our main goal is to find an $\hat{f}$ that generalizes (makes accurate predictions on novel inputs)
- Assume that we have a binary classification model $\hat{f}(x) = 0$ or $\hat{f}(x) = 1$
  - How useful would this model be? You feed it an input and it either that says the input is or is not the class.
  - Is it good to always be absolutely certain?
  - All models are wrong, but some of them are useful. To be useful, a model needs have some level of stochasticity. (Read about stochasticity if needed: https://machinelearningmastery.com/stochastic-in-machine-learning/)

Classification



To create a classification model that handles ambiguous cases, we typically use a model of the form

$$\hat{y} = \hat{f}(x) = \underset{c=1..C}{\operatorname{argmax}}\, p(y = c|\boldsymbol{x})$$

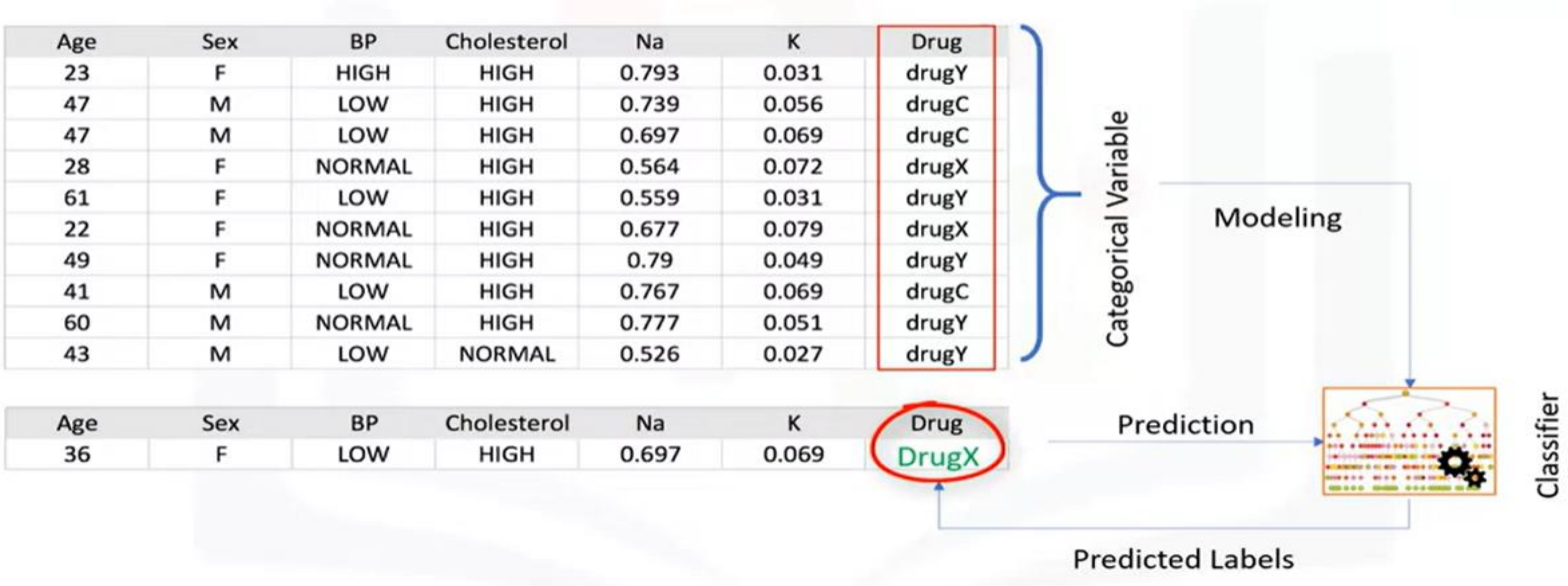
where Bayes rule tells us that

$$p(y = c|x, \mathcal{D}) = \frac{p(\boldsymbol{x}|y = c)p(y = c)}{p(\boldsymbol{x})} = \frac{p(y = c, \boldsymbol{x})}{p(\boldsymbol{x})}$$

i.e. our prediction is the class that maximizes the probability of y given the input and the dataset; this corresponds to the mode of the distribution

- Python’s sklearn library provides a number of well-known algorithms to modeling classification problems.
- To use them, you select a model, use the fit function to train the model, and then using the predict function to get an answer
  - You will need to refer to the book and the data camp for specific examples.

Multi class classification

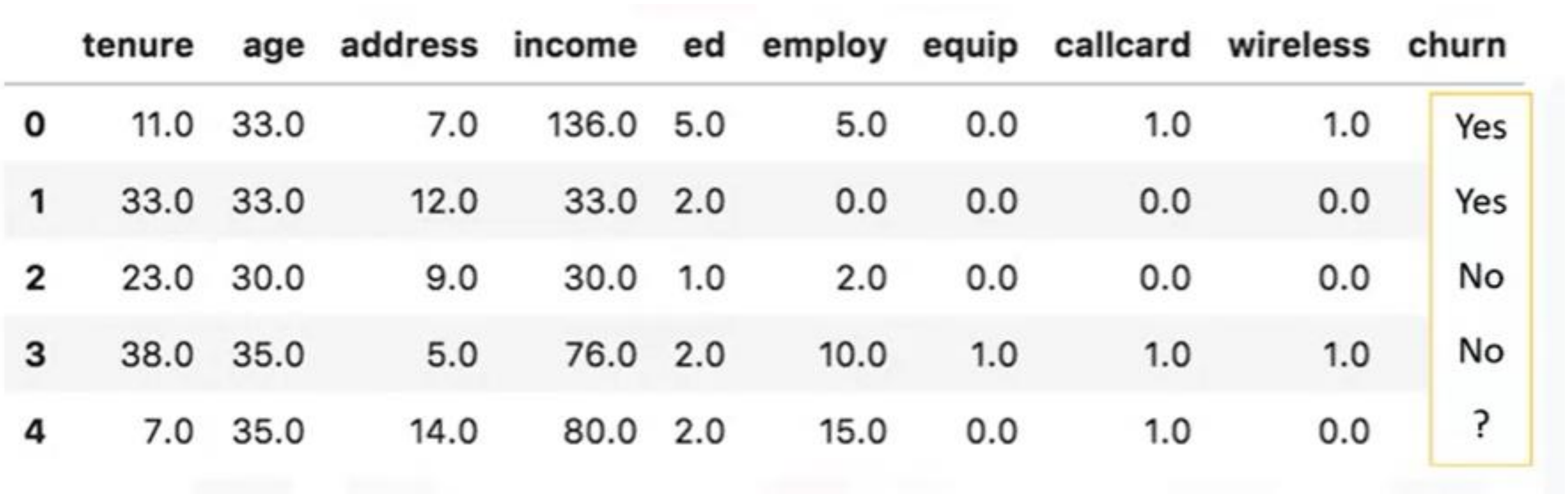


Age	Sex	BP	Cholesterol	Na	K	Drug
23	F	HIGH	HIGH	0.793	0.031	drugY
47	M	LOW	HIGH	0.739	0.056	drugC
47	M	LOW	HIGH	0.697	0.069	drugC
28	F	NORMAL	HIGH	0.564	0.072	drugX
61	F	LOW	HIGH	0.559	0.031	drugY
22	F	NORMAL	HIGH	0.677	0.079	drugX
49	F	NORMAL	HIGH	0.79	0.049	drugY
41	M	LOW	HIGH	0.767	0.069	drugC
60	M	NORMAL	HIGH	0.777	0.051	drugY
43	M	LOW	NORMAL	0.526	0.027	drugY

Age	Sex	BP	Cholesterol	Na	K	Drug
36	F	LOW	HIGH	0.697	0.069	DrugX

- If you see in the above example, we have a patient data set where for each patent it is labeled which drug was effective.
- There are three classes of drugs. So, the prediction for a new patient will be one of the three drugs. So, it is a multi classification problem

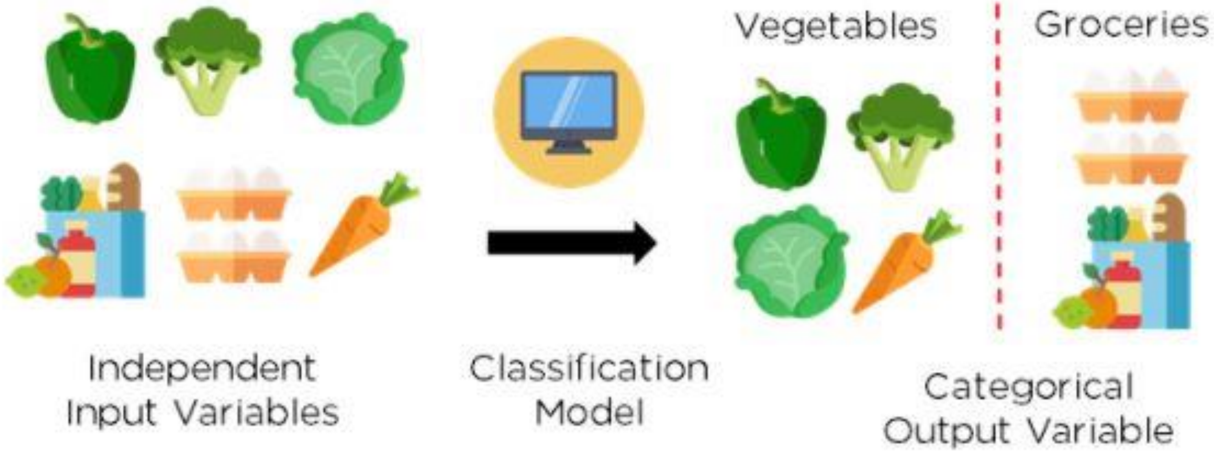
Other classification use cases

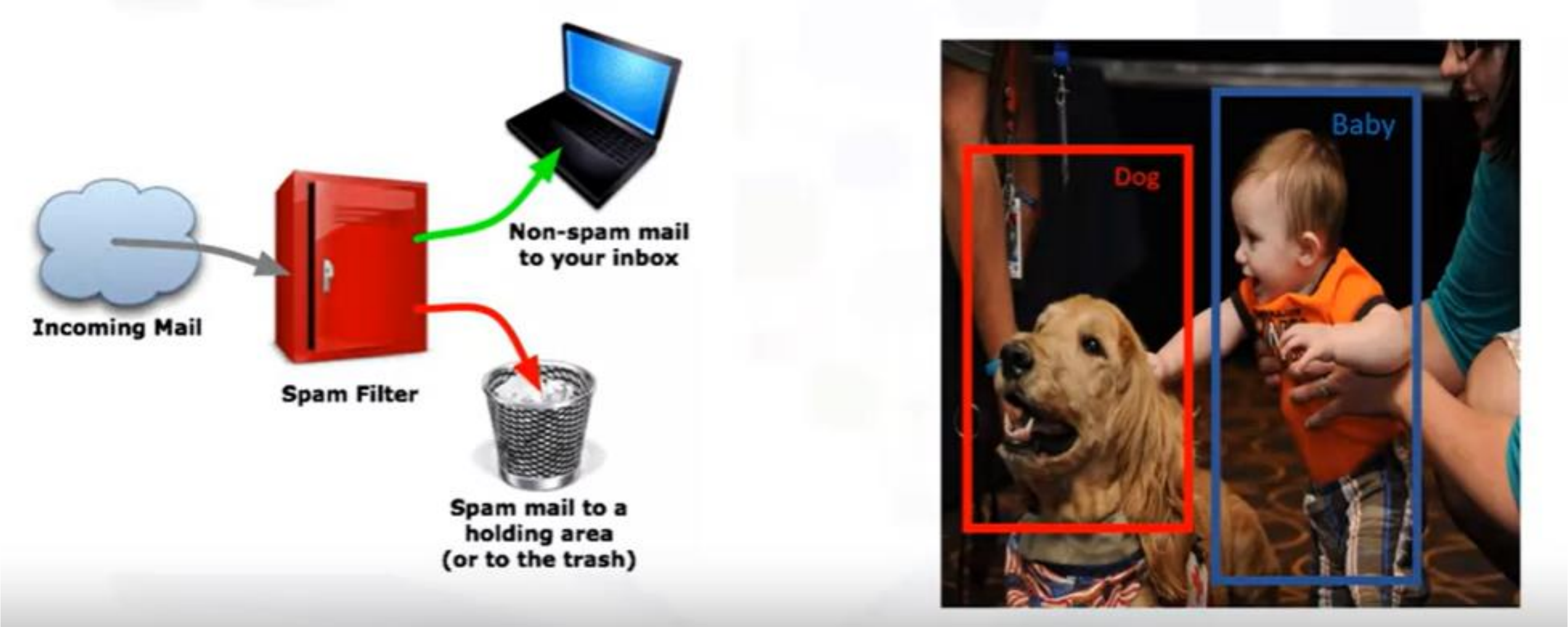


	tenure	age	address	income	ed	employ	equip	callcard	wireless	churn
0	11.0	33.0	7.0	136.0	5.0	5.0	0.0	1.0	1.0	Yes
1	33.0	33.0	12.0	33.0	2.0	0.0	0.0	0.0	0.0	Yes
2	23.0	30.0	9.0	30.0	1.0	2.0	0.0	0.0	0.0	No
3	38.0	35.0	5.0	76.0	2.0	10.0	1.0	1.0	1.0	No
4	7.0	35.0	14.0	80.0	2.0	15.0	0.0	1.0	0.0	?

- Classifying customers into given categories
- Which customer will move to another service provider
- Getting target advertising so that advertisement can be relevant and customer response can be maximized

Other classification use cases




Incoming Mail
Spam Filter
Non-spam mail to your inbox
Spam mail to a holding area (or to the trash)
Dog
Baby

- Also, speech recognition
- Hand-writing recognition
- Document classification
- Categorizing fake and real news

Some common classification algorithms

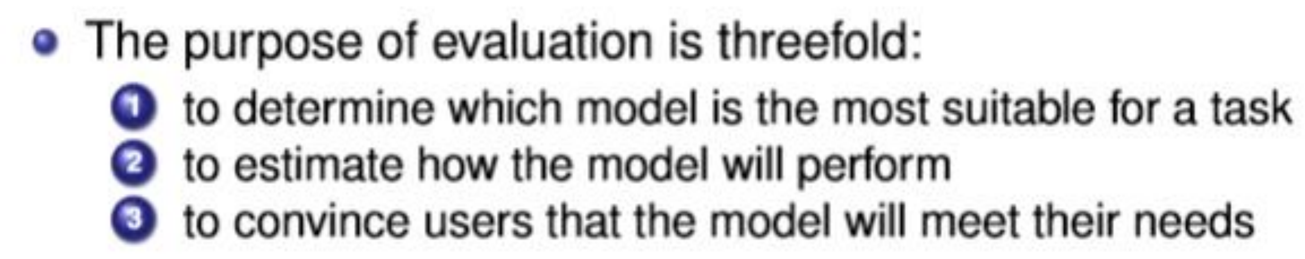
- k-Nearest Neighbors
- Naïve Bayes
- Logistic Regression
- Support Vector Machines (SVM)
- Decision Trees
- Neural Networks
- Graphical Models
- Etc.

Performance Measures

- Building a model is easy enough, building the right model is more difficult.
- In order to choose the model to actually be useful, it must be evaluated.
- Assume we have a trained classifier model and are able to obtain class probabilities for a given input vector.
- We must decide at what threshold to consider a point belonging to one class over another.
- In many cases, this will simply be 0.5, but it is allowable to vary acceptance depending on the problem.
- For example, if you are trying to classify whether or not a bag will be mishandled (delayed or sent to wrong airport), 0.5 might not be a good threshold for classification
  - Because, if you use 0.5, you might miss a mishandle bag to be predicted as correctly handled bag which will be against the main purpose of the classification
  - We don’t want to miss a mishandled bags
    - It is okay if a correct bag is predicted as mishandled sometimes

- - Once we have a method for determining which input belongs to which class, we can then evaluate the performance of our algorithm.

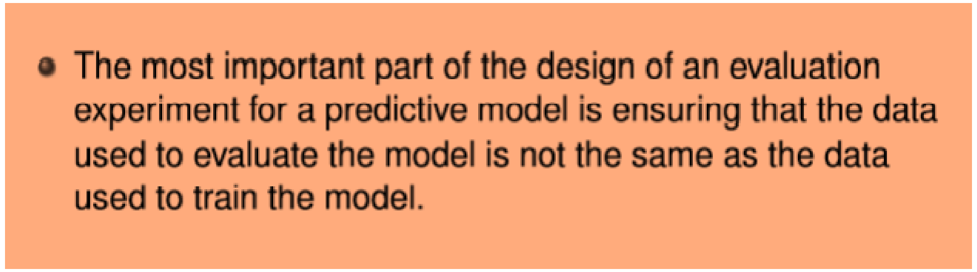
Evaluation



- The purpose of evaluation is threefold:
  1. to determine which model is the most suitable for a task
  2. to estimate how the model will perform
  3. to convince users that the model will meet their needs

- These three purposes all support one goal:
  - to select the best model for the problem; where the "best" varies by context.
  - Some models need to be more explainable than accurate
  - Some models need to be more accurate than explainable

Evaluation



- The most important part of the design of an evaluation experiment for a predictive model is ensuring that the data used to evaluate the model is not the same as the data used to train the model.

- Evaluation of a model is done by calculating a performance measure.
- To calculate the performance measure, you must run your model against a dataset.
- But which dataset?
  - The one you trained it on? No.
- - When building a model, it is critical that you split the dataset up into at least 2 components:
  - a training set,
  - and a test set
- - And preferable to also split the data into a third "validation" set
- - Why a third set? So that you can adjust your model parameters without consulting the test data
- You want to train your model on the training data, and only after it is trained do you evaluate it against the test data.
- If you are unhappy and want to continue tweaking the model, you will need to resample the training and test data and start again
- The test set is never used to train the model
- Why do we do this? To reduce overfitting and increase generalization.
- The whole reason we build machine learning models to make predictions on data we've never seen before. If we use that data as part of the training process, then we're biasing the model by peeking ahead.

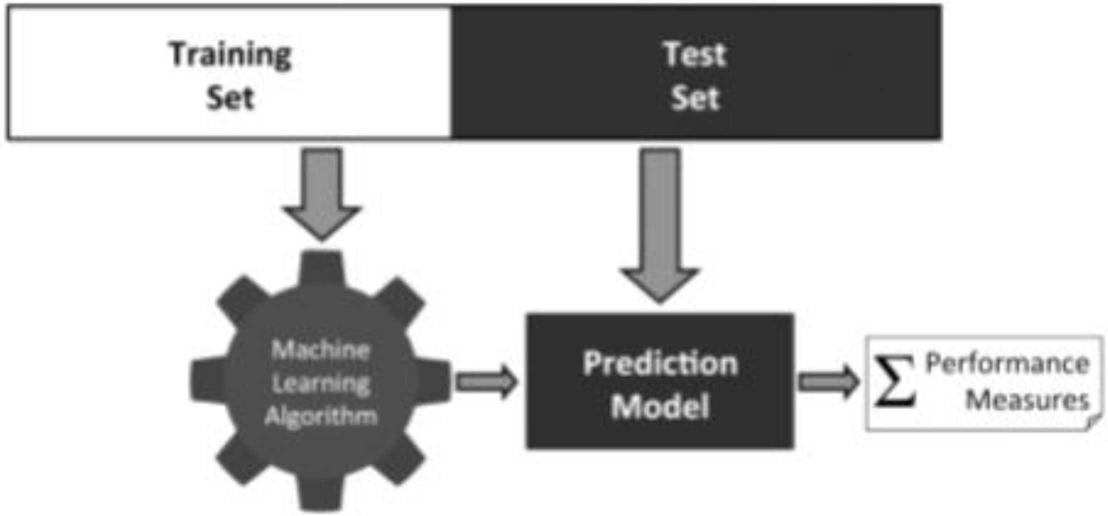


Figure: The process of building and evaluating a model using a **hold-out test set**.

Misclassification Rate

- The most straight-forward performance measure that we can use is the misclassification rate.

$$\text{misclassification rate} = \frac{\text{number incorrect predictions}}{\text{total predictions}}$$

The misclassification rate is fine as a singular metric, but it hides a lot of information.

Misclassification Rate

- For binary prediction problems, there are 4 possible outcomes:
- - True Positive (you predict true and it really is true) **(TP)**
- - True Negative (you predict false and it really is false) **(TN)**
- - False Positive (you predict true but its really false) **(FP)**
- - False Negative (you predict false but its really true**)** **(FN)**

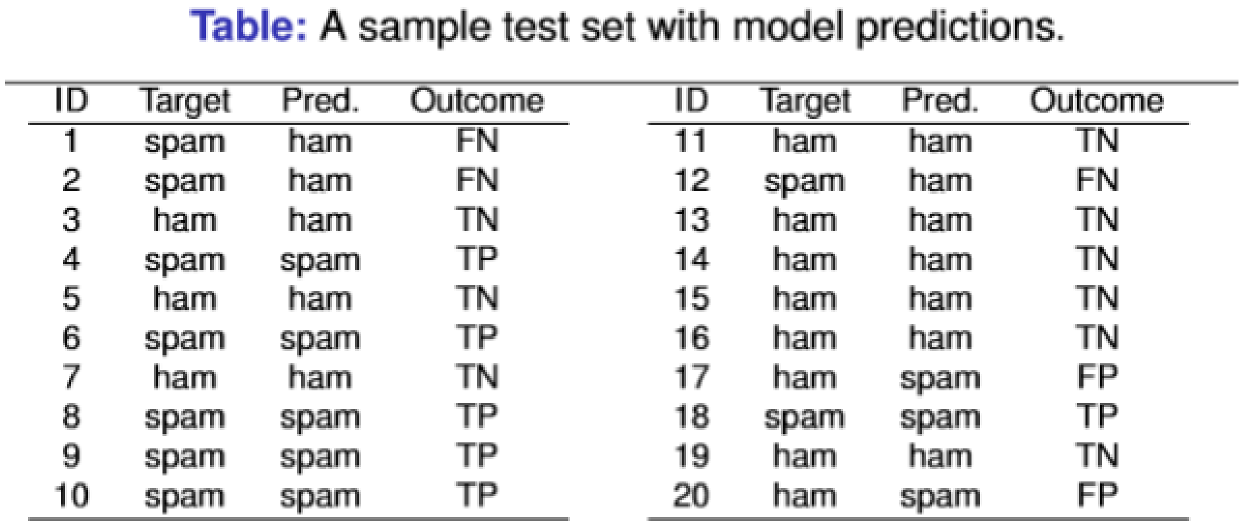


Table: A sample test set with model predictions.

ID	Target	Pred.	Outcome
1	spam	ham	FN
2	spam	ham	FN
3	ham	ham	TN
4	spam	spam	TP
5	ham	ham	TN
6	spam	spam	TP
7	ham	ham	TN
8	spam	spam	TP
9	spam	spam	TP
10	spam	spam	TP
11	ham	ham	TN
12	spam	ham	FN
13	ham	ham	TN
14	ham	ham	TN
15	ham	ham	TN
16	ham	ham	TN
17	ham	spam	FP
18	spam	spam	TP
19	ham	ham	TN
20	ham	spam	FP

$$\text{misclassification rate} = \frac{\text{number incorrect predictions}}{\text{total predictions}} \qquad (1)$$

$$\text{misclassification rate} = \frac{(2+3)}{(6+9+2+3)} = 0.25$$

Confusion Matrix

- The misclassification rate is fine as a singular metric, but it hides a lot of information.
- To better understand where a classification algorithm goes right or wrong, it is more useful to create a confusion matrix:

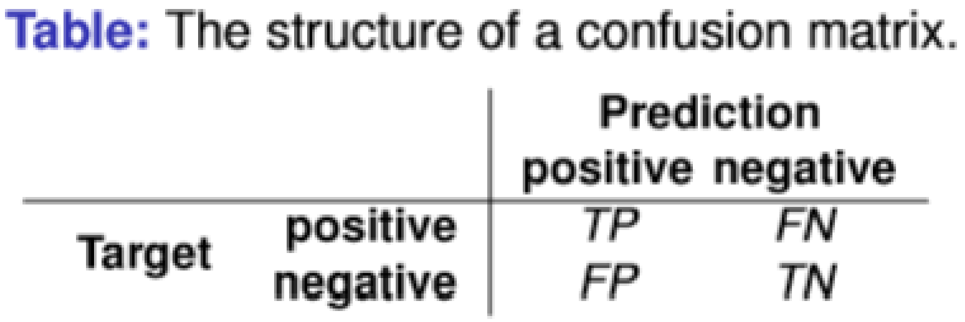


Table: The structure of a confusion matrix.

		Prediction	
		positive	**negative**
Target	**positive**	*TP*	*FN*
	negative	*FP*	*TN*

The confusion matrix is nice because we can tell at a glance if the algorithm is performing well - the values on the TP/TN diagonal will be large.

Confusion Mat

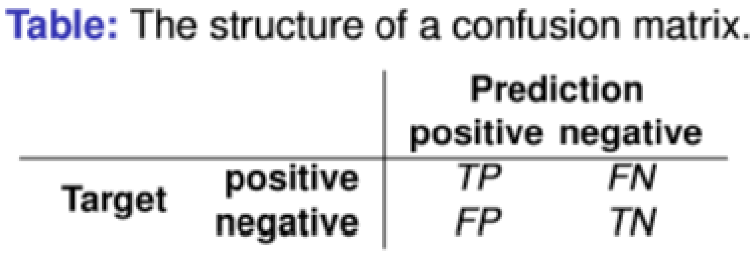


Table: The structure of a confusion matrix.

		Prediction	
		positive	negative
Target	positive	*TP*	*FN*
	negative	*FP*	*TN*

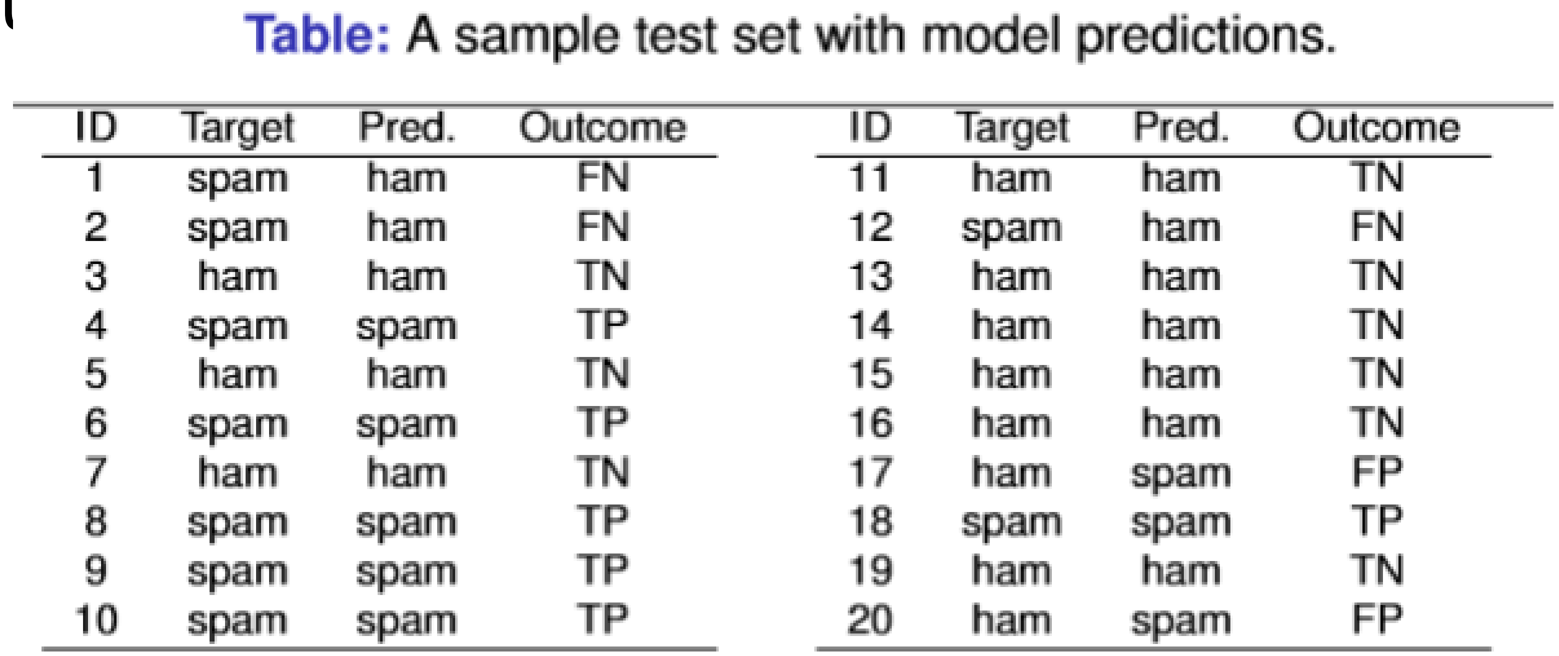


Table: A sample test set with model predictions.

ID	Target	Pred.	Outcome	ID	Target	Pred.	Outcome
1	spam	ham	FN	11	ham	ham	TN
2	spam	ham	FN	12	spam	ham	FN
3	ham	ham	TN	13	ham	ham	TN
4	spam	spam	TP	14	ham	ham	TN
5	ham	ham	TN	15	ham	ham	TN
6	spam	spam	TP	16	ham	ham	TN
7	ham	ham	TN	17	ham	spam	FP
8	spam	spam	TP	18	spam	spam	TP
9	spam	spam	TP	19	ham	ham	TN
10	spam	spam	TP	20	ham	spam	FP

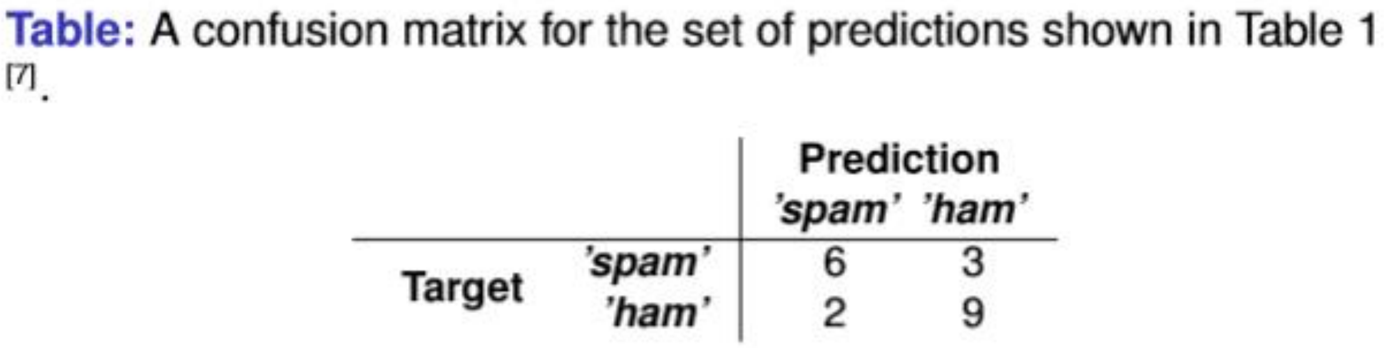


Table: A confusion matrix for the set of predictions shown in Table 1 [7].

		Prediction	
		’spam’	*’ham’*
Target	*’spam’*	6	3
	’ham’	2	9

Misclassification Rate from confusion matrix

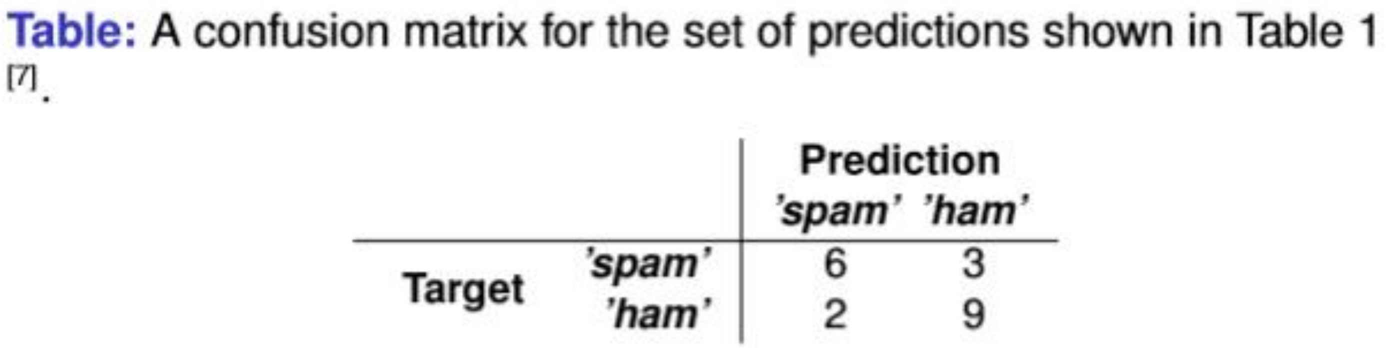
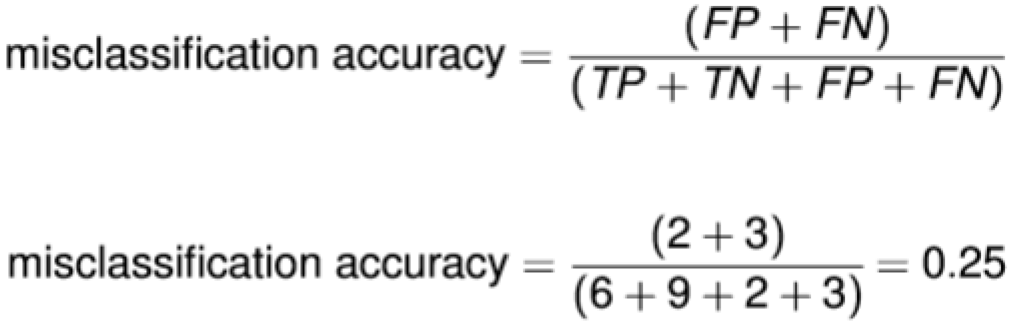


Table: A confusion matrix for the set of predictions shown in Table 1 [7].

		Prediction	
		’spam’	*’ham’*
Target	*’spam’*	6	3
	’ham’	2	9

- The misclassification rate can be easily calculated from the confusion matrix:



$$\text{misclassification accuracy} = \frac{(FP + FN)}{(TP + TN + FP + FN)}$$

$$\text{misclassification accuracy} = \frac{(2 + 3)}{(6 + 9 + 2 + 3)} = 0.25$$

Accuracy

- Another way to summarize the data is by the accuracy - how many instances are classified correctly?

$$\text{classification accuracy} = \frac{(TP + TN)}{(TP + TN + FP + FN)}$$

$$\text{classification accuracy} = \frac{(6 + 9)}{(6 + 9 + 2 + 3)} = 0.75$$

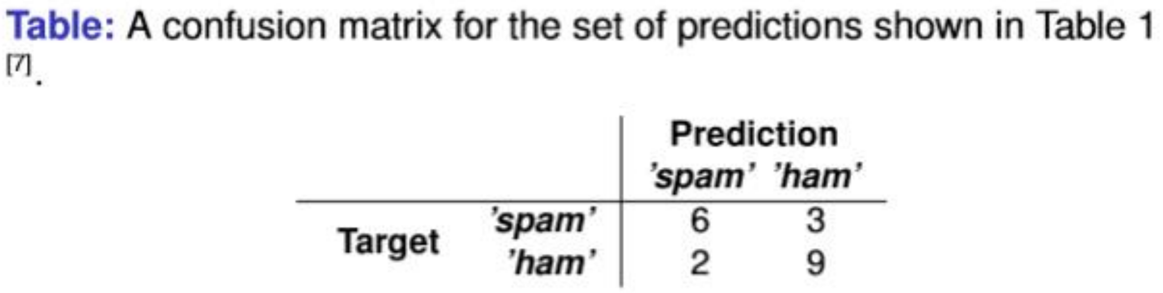


Table: A confusion matrix for the set of predictions shown in Table 1 [7].

		Prediction	
		'spam'	*'ham'*
Target	*'spam'*	6	3
	'ham'	2	9

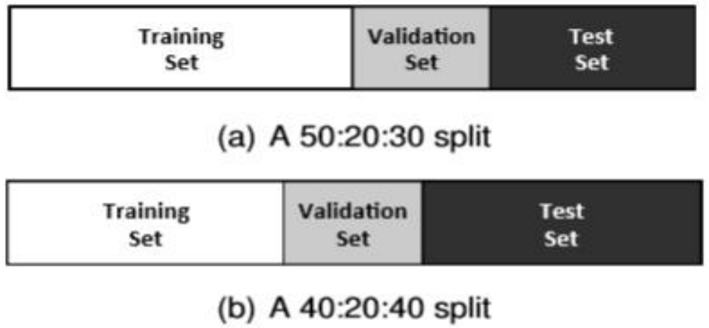
- Like the misclassification rate, the accuracy also hides a lot of information.
- For example, imagine if your target vector has 99 "1"s and only a single 0. If you predict "1" for everything, then you have a 99% accuracy. (just like our baggage tracking example)

Designing Evaluation Experiments

- - It is crucial to evaluate a model using data that was not used for training
- The overall performance of a model can be captured in a single performance measure
  - This is useful, and sometimes required, but not always the best approach for complete understanding
- So how do we split the data into training, validation, and test datasets?
  - The easiest thing to do is select percentages of the data to act as each set, such as all three sum to 100% -
  - - Some common splits are 50, 20, 30 and 40, 20, 40.
  - It is also not uncommon to see 70, 15, 15.

Experiment with Hold-out sampling

- What happens if we start to look at how our performance measure (e.g. misclassification) changes as we train our models?
- We see that the error for the training set approaches 0, but the error on the validation data reduces and then curves back up
  - The closer the training error gets to zero, the more likely your model is overfitting the data
  - The overfitting is why the validation error starts to increase, the model begins to perform poorly on data it was not trained on
  - Ideally, we want to stop training out model just before the validation error begins to increase
- Without a validation (and test set), our model would easily overfit the data and we wouldn't be aware until it was too late.



(a) A 50:20:30 split

(b) A 40:20:40 split

Figure: Hold-out sampling can divide the full data into training, validation, and test sets.

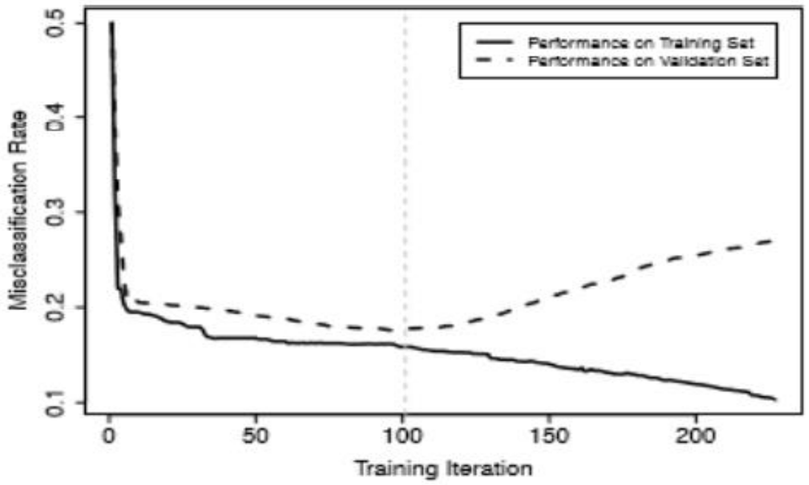


Figure: Using a validation set to avoid overfitting in iterative machine learning algorithms.

Problem with random sampling

- Randomly sampling the data is a good approach, but there are two issues:
  - - You need enough data
  - There is the risk of a "lucky split", where one of the sets contains all the difficult to classify data, and the other sets do not
  - - call it chance imbalance.
- - One way to address these issues is to use a slightly more involved process called k-fold cross validation

Experimenting with K-fold cross validation

- Under k-fold cross validation, you break your data up in to k equally sized chunks, and train the algorithm k times.
- Each time you train the algorithm, you select 1 chunk as the validation set and the other k-1 chunks as the training set.
- The validation set changes each time; think of it as for i=1..k, val_set = chunk[i] and train_set = chunk[-i]

- For each fold that you create a model on, you calculate the performance metric
- If you'd like, you can also calculate a confusion matrix for each fold
- One way to get the score of the whole cross validation set is to take the average of the performance measure
- Equivalently, you can add up all the confusion matrices and use the sum matrix to calculate the measure

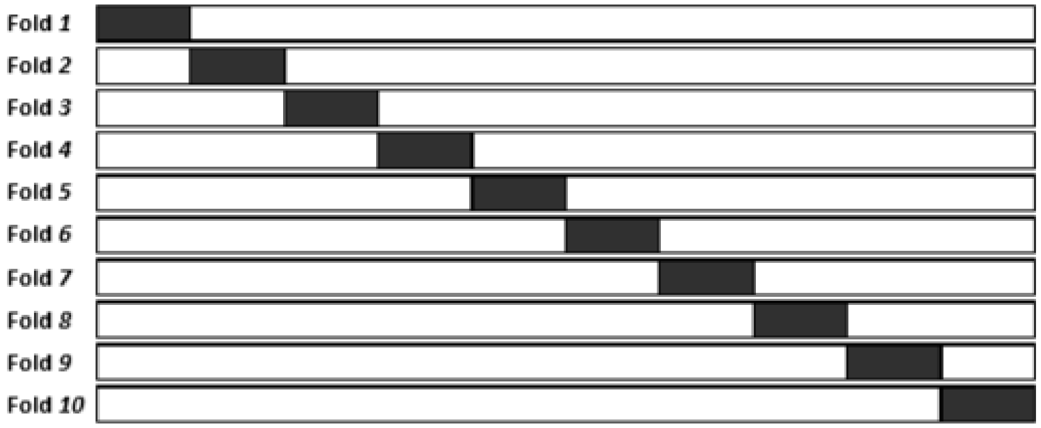
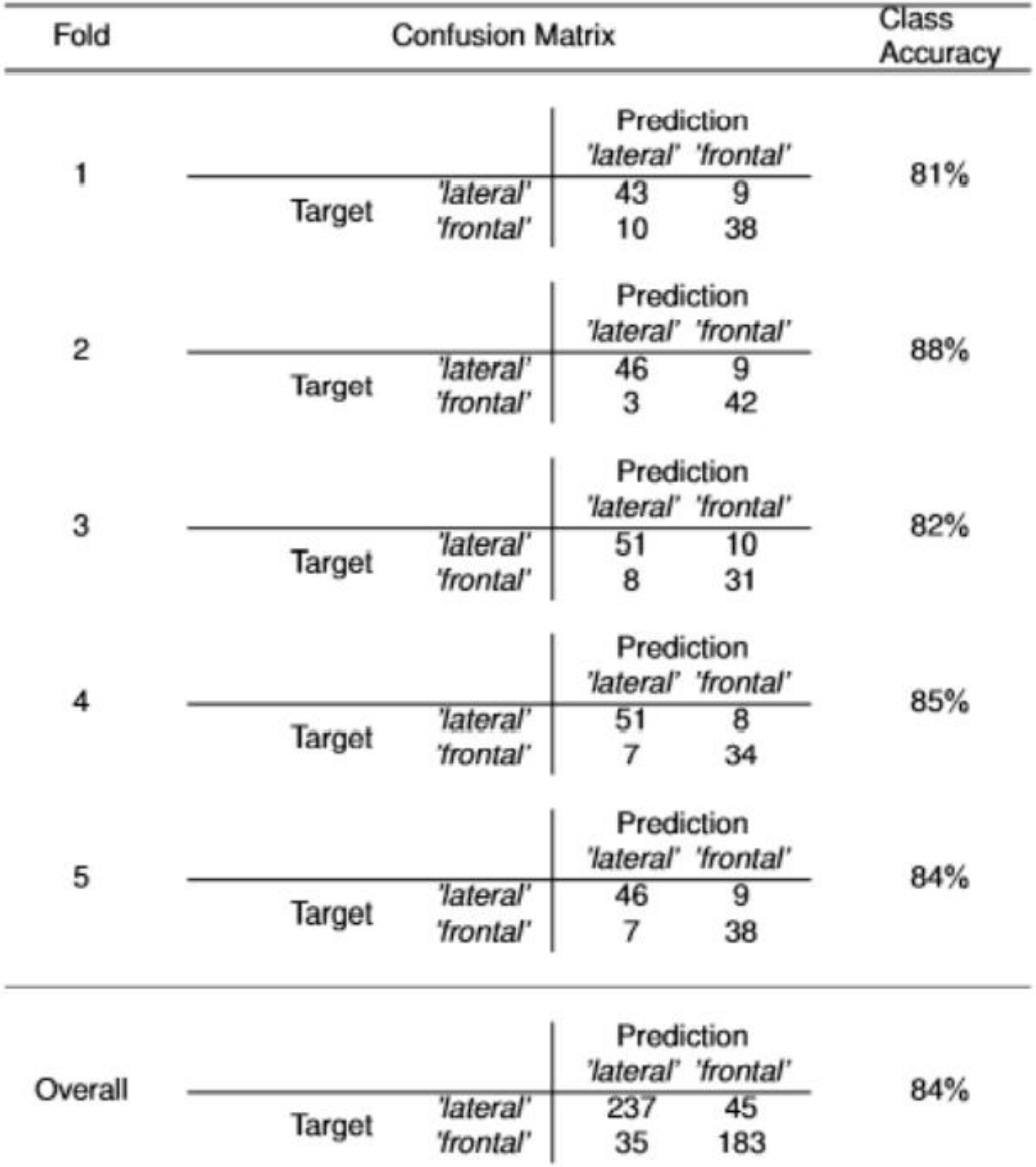


Figure: The division of data during the ***k*-fold cross validation** process. Black rectangles indicate test data, and white spaces indicate training data.

Experimenting with k-fold cross validation

- Once you have finished cross validation and calculated the overall performance measure, you can then retrain the model using all of the data and deploy that final model, with the assumption that it will perform on par with the overall metric.
    - If you have a test set, you can test your overall model against it to get a final score for generalization



Fold	Confusion Matrix				Class Accuracy
			Prediction		
			'lateral'	'frontal'	
1	Target	'lateral'	43	9	81%
		'frontal'	10	38	
			Prediction		
			'lateral'	'frontal'	
2	Target	'lateral'	46	9	88%
		'frontal'	3	42	
			Prediction		
			'lateral'	'frontal'	
3	Target	'lateral'	51	10	82%
		'frontal'	8	31	
			Prediction		
			'lateral'	'frontal'	
4	Target	'lateral'	51	8	85%
		'frontal'	7	34	
			Prediction		
			'lateral'	'frontal'	
5	Target	'lateral'	46	9	84%
		'frontal'	7	38	
			Prediction		
			'lateral'	'frontal'	
Overall	Target	'lateral'	237	45	84%
		'frontal'	35	183	

Leave one-out cross validation

- When k is equal to the size of the dataset, it's referred to as leave-one-out cross-validation
- - You test n-1 instances against 1 instance, then aggregate the results.

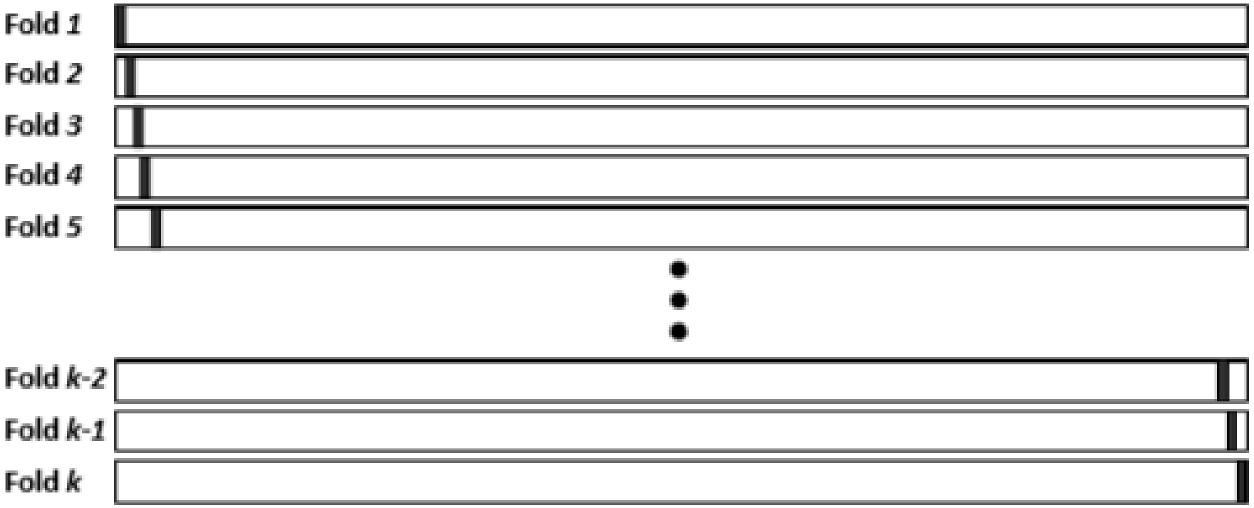


Figure: The division of data during the **leave-one-out cross validation** process. Black rectangles indicate instances in the test set, and white spaces indicate training data.

Bootstrapping

- The basic idea with bootstrapping is to train k different models, but unlike a sequential sampling with kfold cross validation, bootstrapping randomly samples (with replacement) from the dataset to create the training and validation sets each iteration.
- - K is typically much larger than you'd see in cross validation; k > 200
- Bootstrapping seems to work better for smaller datasets than k-fold cross validatio

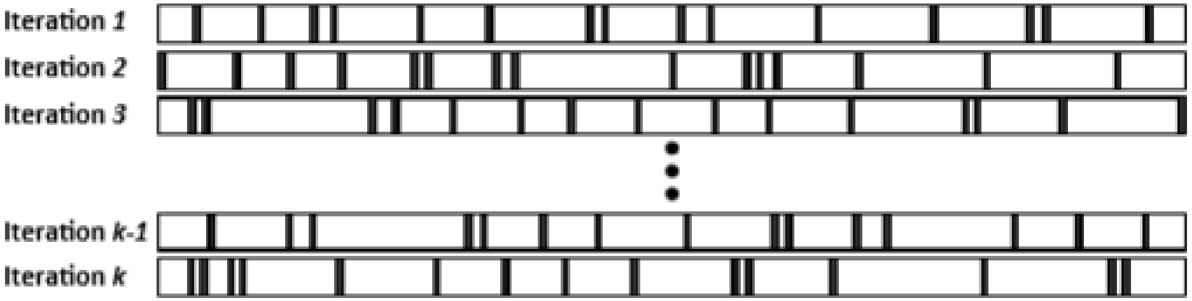


Figure: The division of data during the $\epsilon 0$ bootstrap process. Black rectangles indicate test data, and white spaces indicate training data.

Dealing with time series data

- When dealing with time-series data, you cannot randomly sample because the process destroys the temporal association within the data. (For example, time is an important factor for weather data)
- Instead, you group the data into past and future chunks
- Care must be taken to ensure that your past chunk accounts for all of the seasonality behavior that you'd expect
- - As with cross validation or bootstrapping, you create multiple models and aggregate their results to determine the expected performance.

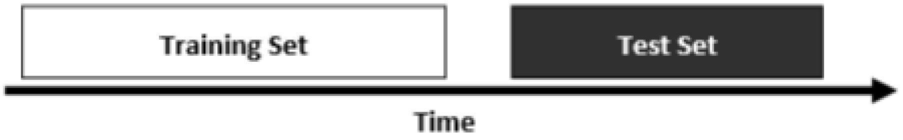


Figure: The **out-of-time sampling** process.

Performance measures for Categorial target: TPR, TNR, FPT, FNR

- The confusion matrix can be used to calculate multiple performance measures for categorical targets
- The most basic measures are the
- - true positive rate (TRP)
- - true negative rate (TNR)
- - false positive rate (FPR)
- - false negative rate (FNR).
- These values vary between [0, 1], with high values of TPR and TNR rate and low values of FPR and FNR being preferred.

$$\text{TPR} = \frac{TP}{(TP + FN)}$$

$$\text{TNR} = \frac{TN}{(TN + FP)}$$

$$\text{FPR} = \frac{FP}{(TN + FP)}$$

$$\text{FNR} = \frac{FN}{(TP + FN)}$$

$$\text{TPR} \quad = \frac{6}{(6+3)} \quad = 0.667$$

$$\text{TNR} \quad = \frac{9}{(9+2)} \quad = 0.818$$

$$\text{FPR} \quad = \frac{2}{(9+2)} \quad = 0.182$$

$$\text{FNR} \quad = \frac{3}{(6+3)} \quad = 0.333$$

Table: A sample test set with model predictions.

ID	Target	Pred.	Outcome
1	spam	ham	FN
2	spam	ham	FN
3	ham	ham	TN
4	spam	spam	TP
5	ham	ham	TN
6	spam	spam	TP
7	ham	ham	TN
8	spam	spam	TP
9	spam	spam	TP
10	spam	spam	TP
11	ham	ham	TN
12	spam	ham	FN
13	ham	ham	TN
14	ham	ham	TN
15	ham	ham	TN
16	ham	ham	TN
17	ham	spam	FP
18	spam	spam	TP
19	ham	ham	TN
20	ham	spam	FP

		'spam'	***'ham'***
Target	***'spam'***	6	3
	'ham'	2	9

		Prediction	
		positive	**negative**
Target	**positive**	*TP*	*FN*
	negative	*FP*	*TN*

Precision and Recall

- Using the confusion matrix, it is possible to create aggregate metrics besides accuracy.
- The **precision** is the accuracy in the positive predictions. **[Precision captures how often, when a model makes a positive prediction, that prediction turns out to actually be correct]** //mix of correctly and incorrectly handled bags that were predicted as correctly handled

$$precision = \frac{TP}{TP + FP}$$

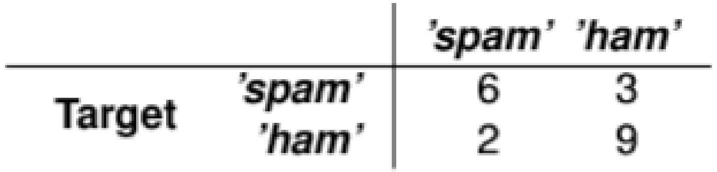
- The **recall** (or sensitivity, or true positive rate) is the ratio of positive instances that are correctly classified **[Recall tells us how confident we can be that all the instances with a positive target level have been found by the model.]** //was your model able to predict **all the** mishandled bags correctly as a mishandled?

$$recall = \frac{TP}{TP + FN}$$

Precision and Recall

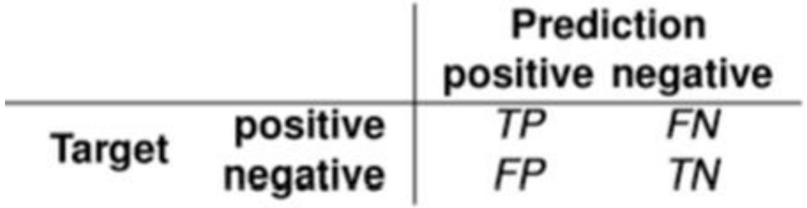
- Returning to the email example, if we assume that "spam" is positive, then
  - precision measures how often the emails marked as spam are actually spam
    - 1 - precision is how likely will a genuine ham email be marked as spam
  - recall measures how often the spam emails were actually marked as spam
    - 1 - recall is how often a spam email will be missed by the system

$$precision = \frac{TP}{TP + FP}$$



		'spam'	'ham'
Target	'spam'	6	3
	'ham'	2	9

$$recall = \frac{TP}{TP + FN}$$



		Prediction	
		positive	negative
Target	positive	*TP*	*FN*
	negative	*FP*	*TN*

$$\text{precision} = \frac{6}{(6+2)} = 0.75$$

$$\text{recall} = \frac{6}{(6+3)} = 0.667$$

F1 Score

- Precision and recall can be combined into a single measure, called the F_1 measure.
- - The F_1 measure is the harmonic mean
- - Harmonic means are less sensitive to outliers than the arithmetic mean

$$\text{F}_1\text{-measure} = 2 \times \frac{(\text{precision} \times \text{recall})}{(\text{precision} + \text{recall})}$$

$$precision = \frac{TP}{TP + FP}$$

		'spam'	'ham'
Target	'spam'	6	3
	'ham'	2	9

$$\begin{aligned}\text{F}_1\text{-measure} &= 2 \times \frac{\left(\frac{6}{(6+2)} \times \frac{6}{(6+3)}\right)}{\left(\frac{6}{(6+2)} + \frac{6}{(6+3)}\right)} \\ &= 0.706\end{aligned}$$

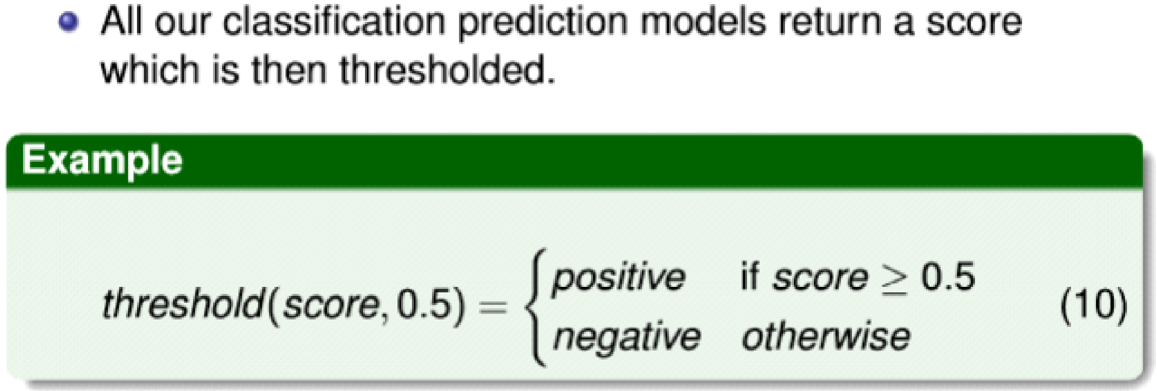
$$recall = \frac{TP}{TP + FN}$$

		Prediction	
		positive	negative
Target	positive	*TP*	*FN*
	negative	*FP*	*TN*

- The F1 score favors classifiers that have a similar precision and recall score, but this is not always what you want.
- Sometimes one component is more important than the other.

Precision/Recall Trade-off

- Generally, a classifier will assign a probability score on each instance of your data set.
- When building a classifier, you have some degree of control over how high of a precision or recall score you want to aim for.
- It would be nice if we could create a model that has both high precision and high recall, but unfortunately this is not possible.
    - - You find all of the positive predictions, and every positive prediction you make is correct (not possible generally)



- All our classification prediction models return a score which is then thresholded.

Example

$$threshold(score, 0.5) = \begin{cases} positive & \text{if } score \geq 0.5 \\ negative & otherwise \end{cases} \qquad (10)$$

Precision/Recall Trade-off and deciding best threshold

- Classification is made by first selecting a decision threshold, and then classifying based on that threshold. Different threshold values will produce different confusion matrices (and thus different precision and recall scores)
  - As the threshold changes, the number of false positives and false negatives will also change.
  - As the threshold decreases, more points are classified positives, so the number of false positives will increase and false negatives will decrease.
  - As the threshold increases, the number of false positives will decrease and false negatives will increase.

Precision-Recall curve and finding the best threshold

- For different values of threshold, you will get a precision and a recall. So, if you plot the value of precision and recall for different values of threshold, you can find the best point in the plot that will help you to know for which threshold you can maximize both precision and recall.

- However, depending on the business, you might want to sacrifice some false positive and you want to cover more actual positive cases and your decision of the threshold depends on that.

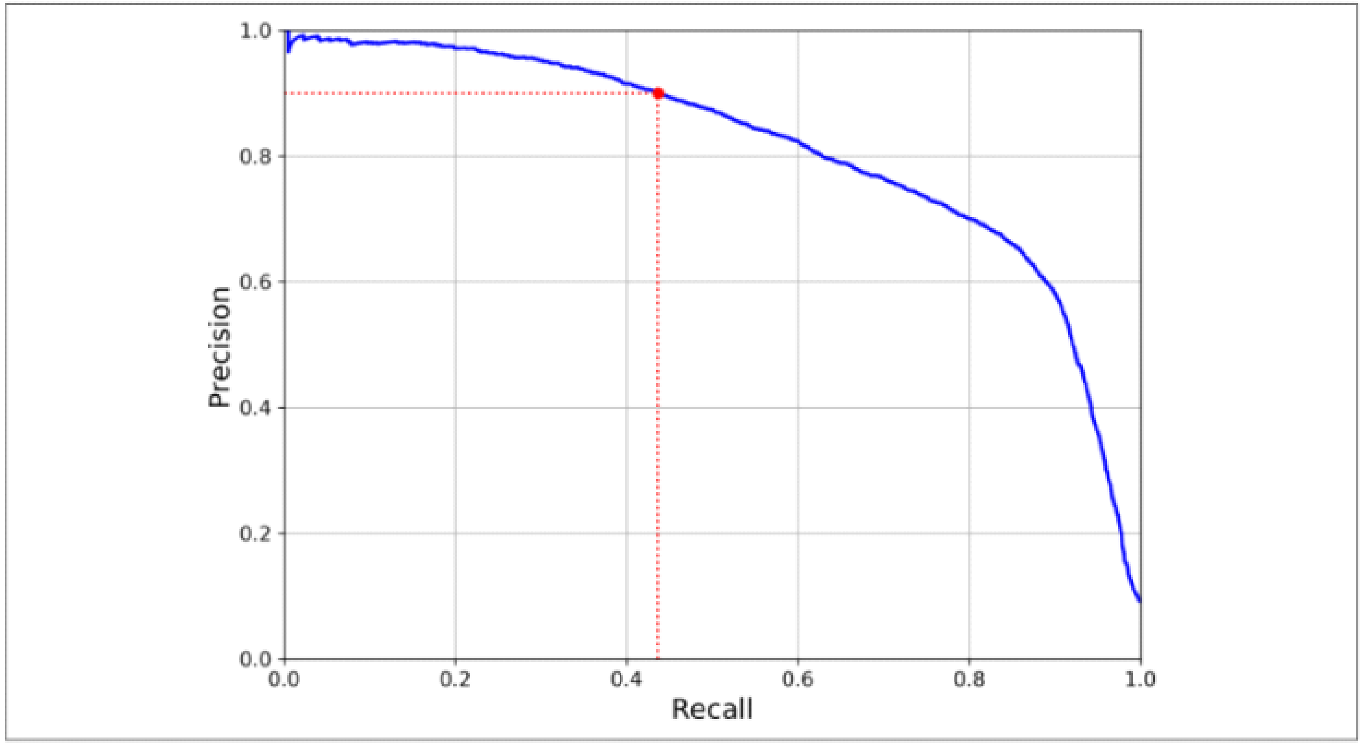


Figure 3-5. Precision versus recall

Measuring profit and loss

- True positive/negative rates, accuracy, precision, recall, average class accuracy… all assume that positive and negative responses are equally important, but this isn't always the case.
- - It may be cheaper to get a false positive than a false negative (example, predicting a correct bag as mishandled could be tolerable up to some extent, however, predicting a mishandled bag as correctly handled is not really what you want as it costs a lot if a bag is mishandled. However, there is still a trade-off as you really don’t want a lot of false positive as well.)
- A profit matrix associates a cost with true positive, true negative, false positive, and false negative outcomes.
  - - The actual values depend on the domain

Measuring profit and loss

- Consider a dataset for loans in which borrowers are classified as either good or bad, and the following profit matrix that shows the cost of each possible outcome:

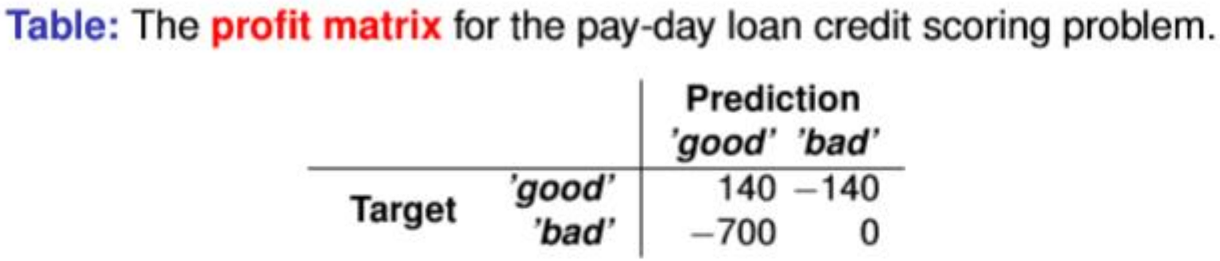


Table: The **profit matrix** for the pay-day loan credit scoring problem.

		Prediction	
		'good'	*'bad'*
Target	*'good'*	140	−140
	'bad'	−700	0

- If the model predicts a bad borrower as "good", the company stands to lose $700 because they'll grant the loan but it won't be repaid.
- If they deny a loan to a good borrowing (by predicting that they're bad), they'll lose $140 - the amount they would have made if the loan had been approved. -

Measuring profit and loss

- For this data, two models were trained and their confusion matrices generated:

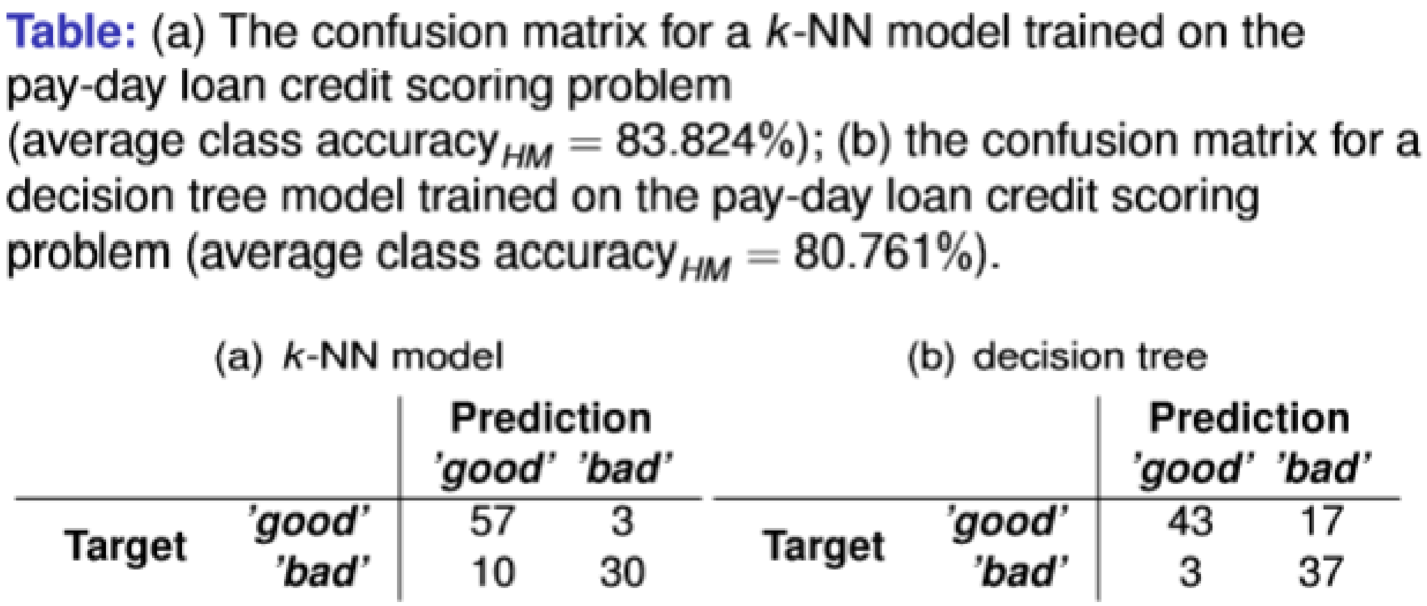


Table: (a) The confusion matrix for a *k*-NN model trained on the pay-day loan credit scoring problem (average class accuracy$_{HM}$ = 83.824%); (b) the confusion matrix for a decision tree model trained on the pay-day loan credit scoring problem (average class accuracy$_{HM}$ = 80.761%).

(a) *k*-NN model

		Prediction	
		'good'	*'bad'*
Target	*'good'*	57	3
	'bad'	10	30

(b) decision tree

		Prediction	
		'good'	*'bad'*
Target	*'good'*	43	17
	'bad'	3	37

So which model is better in light of the cost relationship?

- **We multiple the values from the profit matrix and the confusion matrices**
- **We use element-wise multiplication, not matrix multiplication; 57 * 140 = 7,980**

Measuring profit and loss

- These results indicate that the decision tree model may yield more profit for the company than the k-NN model, and so we would pick that one.
- The numbers in the profit matrix does not need to absolute, they can be relative (you can give more weight to TP twice as much as FN, depending on your business)
- Sometimes this information may not be available and in that case you can use other measures.

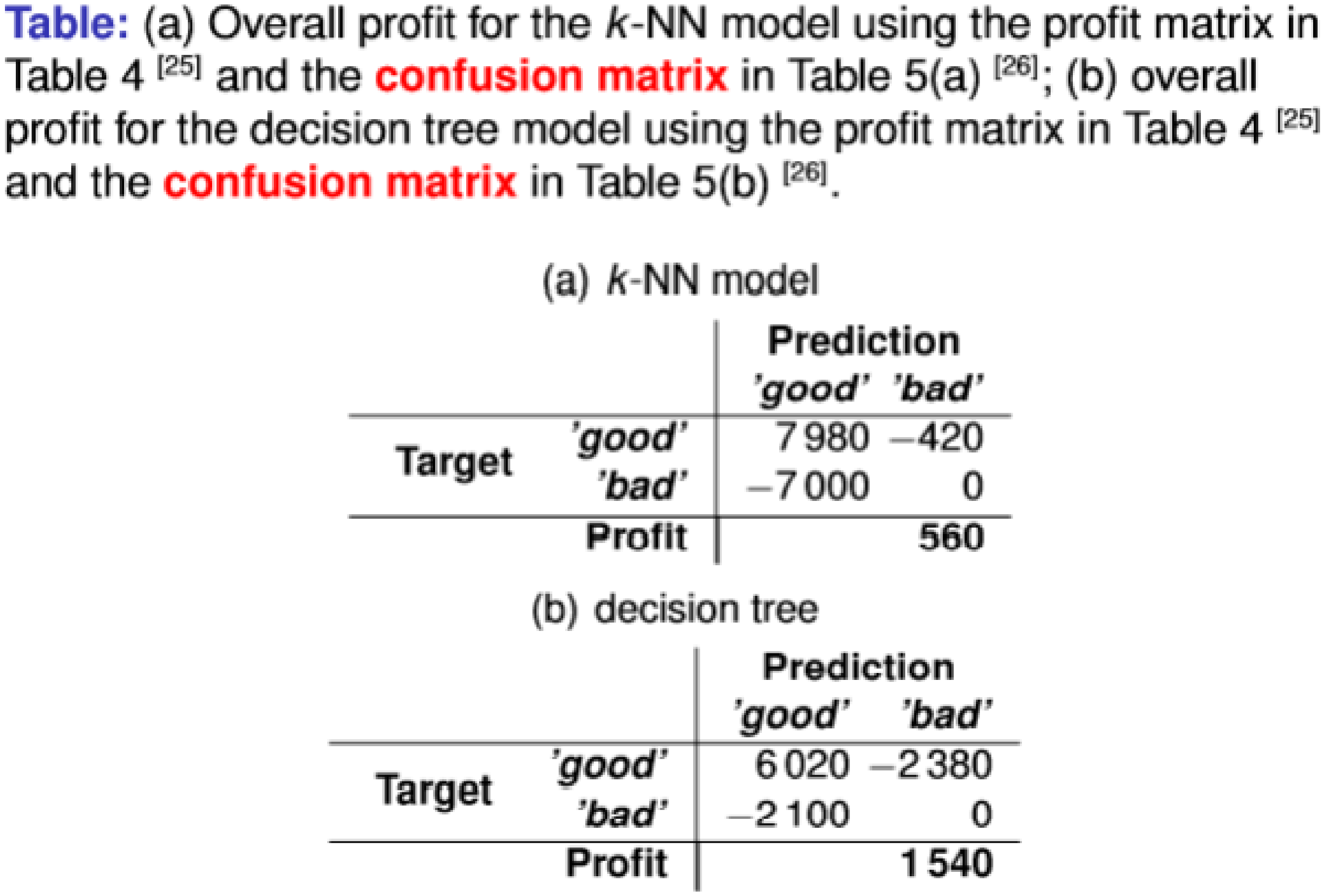


Table: (a) Overall profit for the *k*-NN model using the profit matrix in Table 4 [25] and the **confusion matrix** in Table 5(a) [26]; (b) overall profit for the decision tree model using the profit matrix in Table 4 [25] and the **confusion matrix** in Table 5(b) [26].

(a) *k*-NN model

		Prediction	
		'good'	*'bad'*
Target	*'good'*	7 980	−420
	'bad'	−7 000	0
	Profit		**560**

(b) decision tree

		Prediction	
		'good'	*'bad'*
Target	*'good'*	6 020	−2 380
	'bad'	−2 100	0
	Profit		**1 540**

The ROC curve and area under ROC curve

- The Receiver Operating Characteristic (ROC) curve is another way to assess how well a model performs (specifically binary classifiers)
- The ROC Curve plots the True Positive Rate against the False Positive Rate
  - - The false positive rate is the ratio of negative instances that are incorrectly classified as positive;
- As the True Positive Rate (recall) increases, the classifier produces more false positives.
- The ROC Curve basically helps you figure out how high of a false positive rate you can stomach.

The ROC curve and area under ROC curve (AUC)

- The dotted line on the ROC curve represents a purely random classifier.
- A good classifier should be as far away from the dotted line as possible.
- The area under the ROC Curve (AUC) gives a single value, with a value of 1 indicating a perfect classifier.
- The ROC and P/R curves are similar. You should use the P.R curve when the positive class is rare, or when you care more about false positives than false negatives.
- Otherwise use the ROC curve.

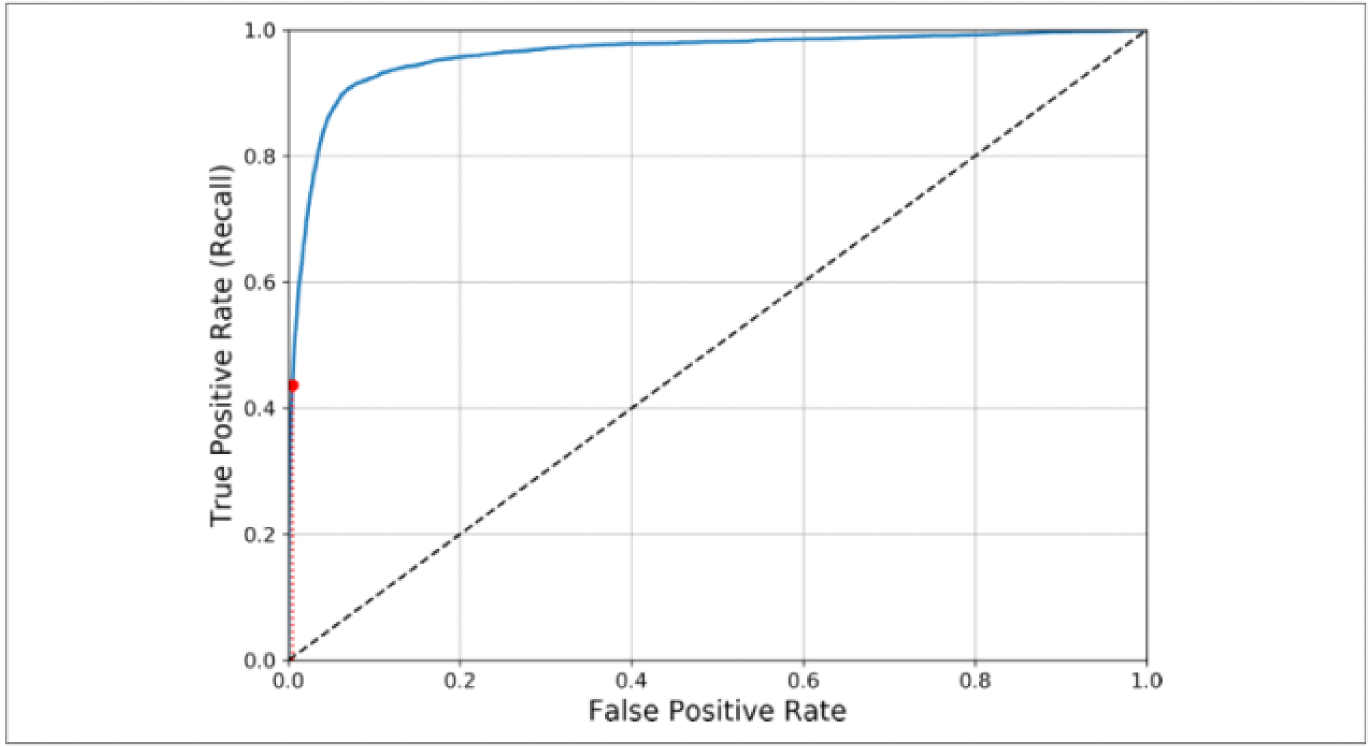
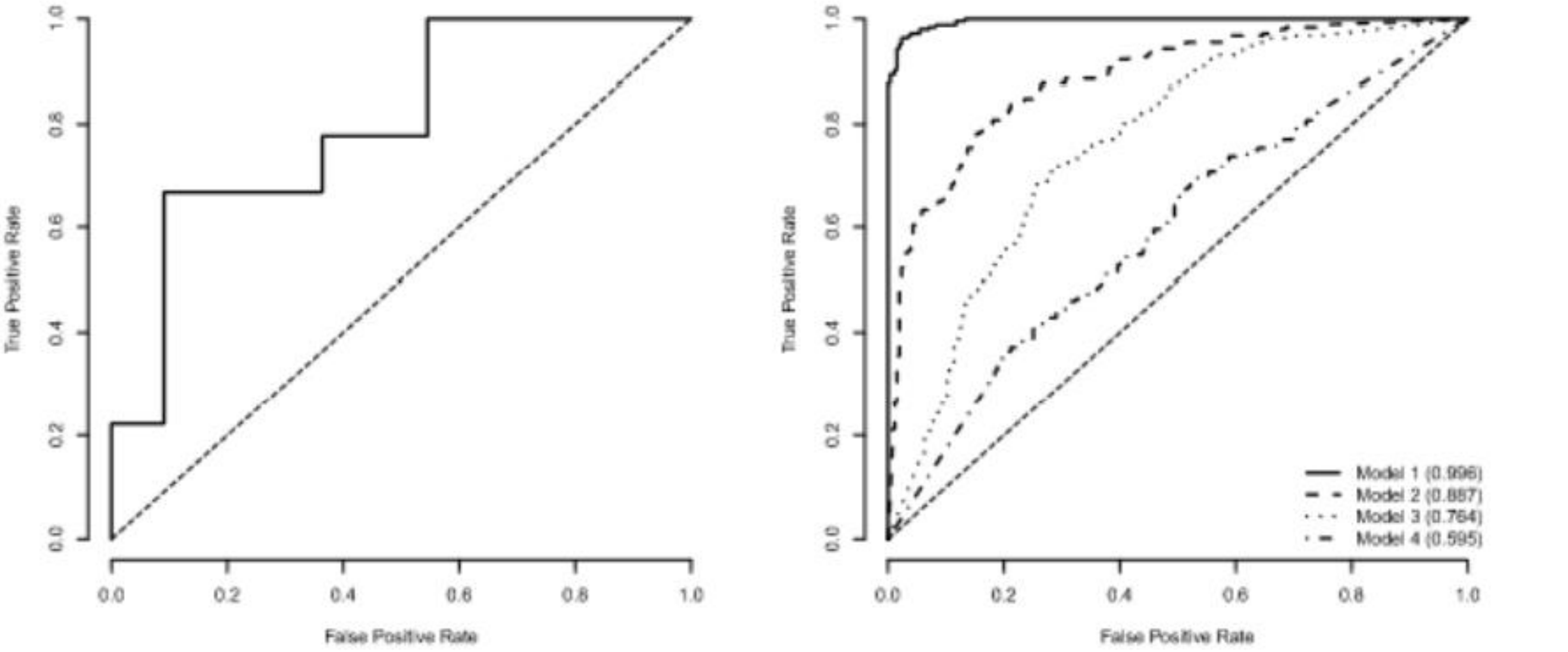


Figure 3-6. This ROC curve plots the false positive rate against the true positive rate for all possible thresholds; the red circle highlights the chosen ratio (at 43.68% recall)



(a)

(b)

Figure: (a) A complete ROC curve for the email classification example; (b) a selection of ROC curves for different models trained on the same prediction task.

Multiclass Classification

- Multiclass classification occurs when a classifier distinguishes between more than 2 classes.
- Some algorithms handle multiple classes directly (decision trees, naïve bayes), others are designed only for binary classification (logistic regression, support vector machines)
- When using binary classifiers, you can train one binary classification for each class (yes/no) and evaluate each classifier against the input, then select the class whose classifier has the highest decision score.
- - **This is called one-vs-rest or one-vs-all**
    - For example, you can keep only class 1, and all the other classes to Not class 1. It can help you to predict whether the items are in class 1 or not.

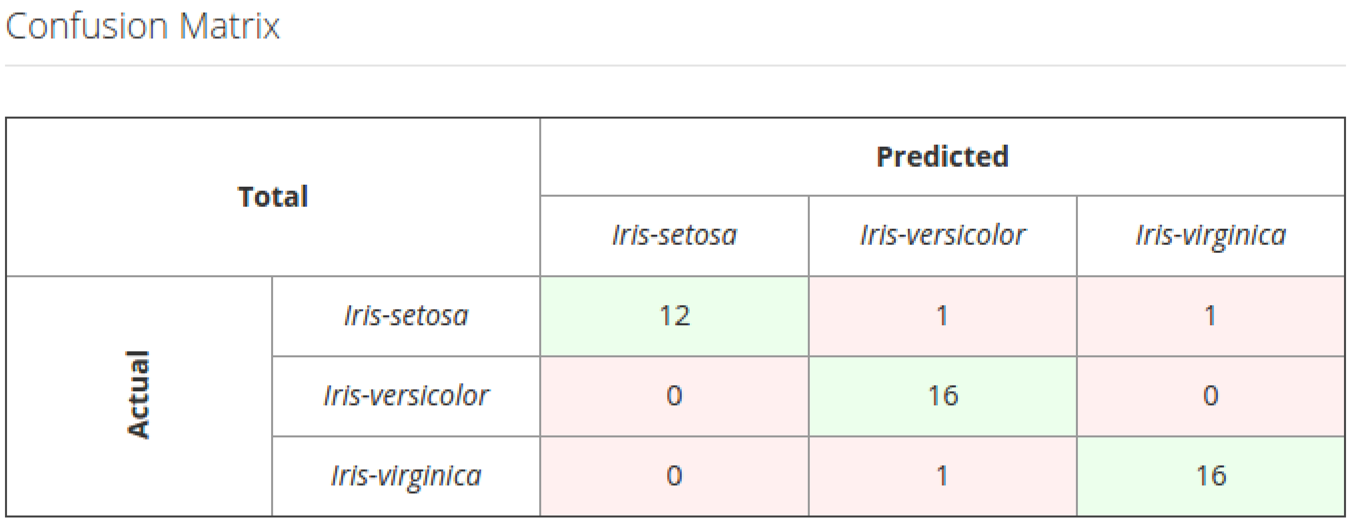
Multiclass Classification

- one-vs-rest or one-vs-all example:
  - Let’s say you would like to classify digits (0-9) from images (E.g., MNIST data set)
  - one way to create a system that can classify the digit images into 10 classes (from 0 to 9) is to train 10 binary classifiers, one for each digit (a 0-detector, a 1-detector, a 2-detector, and so on).
  - Then when you want to classify an image, you get the decision score from each classifier for that image and you select the class whose classifier outputs the highest score.
  - This is called the one-versus-all (OvA) strategy (also called one-versus-the-rest)

Multiclass Classification

- Another strategy is to train a binary classifier for every pair of digits: one to distin guish 0s and 1s, another to distinguish 0s and 2s, another for 1s and 2s, and so on.
- This is called the **one-versus-one (OvO) strategy.**
- If there are N classes, you need to train N × (N – 1) / 2 classifiers.
- e.g. if you have a data set with hand writing recognition for digits, For the MNIST problem, this means training 45 binary classifiers!
- When you want to classify an image, you have to run the image through all 45 classifiers and see which class wins the most duels.
- The main advantage of OvO is that each classifier only needs to be trained on the part of the training set for the two classes that it must distinguish.
- One-vs-one is preferred when the classifiers scale poorly with the size of the training set. Otherwise, one-vs-all is generally better.
- Some algorithms (such as Support Vector Machine classifiers) scale poorly with the size of the training set, so for these algorithms OvO is preferred since it is faster to train many classifiers on small training sets than training few classifiers on large training sets. For most binary classification algorithms, however, OvA is preferred.
- Python’s Sklearn has some of this behavior built in to some classifiers, and automatically does OvO or OvR when it detects multiple classes in the target vector

A confusion matrix with multi-class



Confusion Matrix

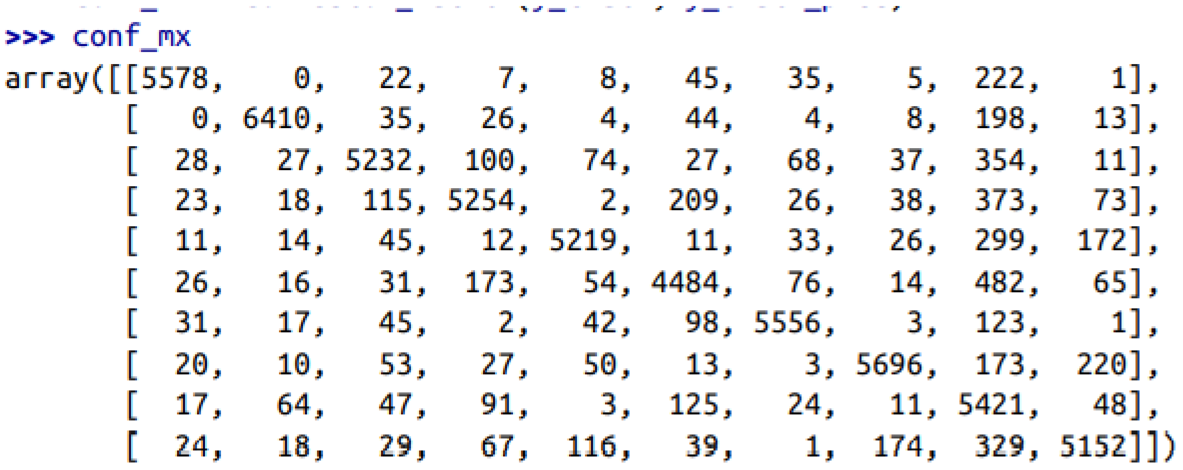
Total		Predicted		
		Iris-setosa	*Iris-versicolor*	*Iris-virginica*
Actual	*Iris-setosa*	12	1	1
	Iris-versicolor	0	16	0
	Iris-virginica	0	1	16

Error Analysis and improvement

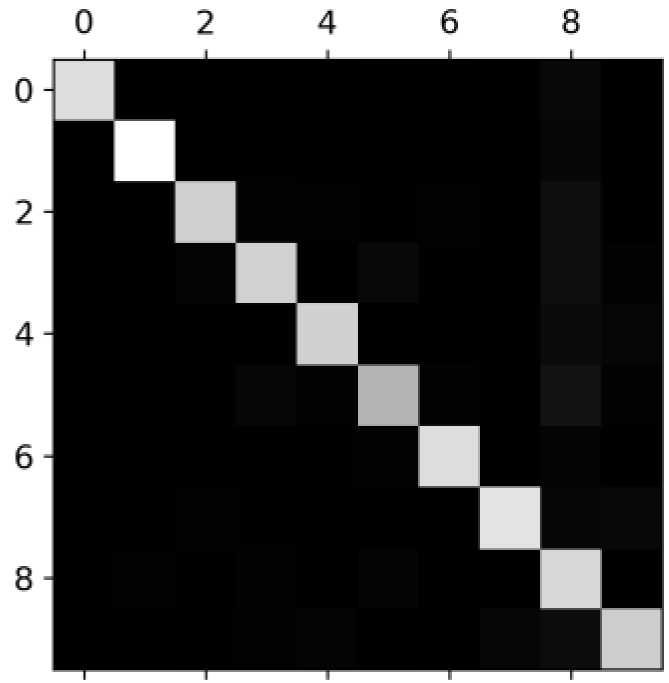
- When you will be doing a real project, you will follow all the steps we have learned so far such as business question, EDA, feature engineering, trying out multiple models, shortlisting the best ones and fine-tuning their hyperparameters, and automating as much as possible.
- Now, you have found a promising model and you want to find ways to improve it.
- One way to do this is to analyze the types of errors it makes
- First you can get the confusing matrix. For a multi class problem, the confusion matrix could be big with lot of different numbers.

Error Analysis and improvement

- A confusion matrix for 10 classes for MNIST dataset



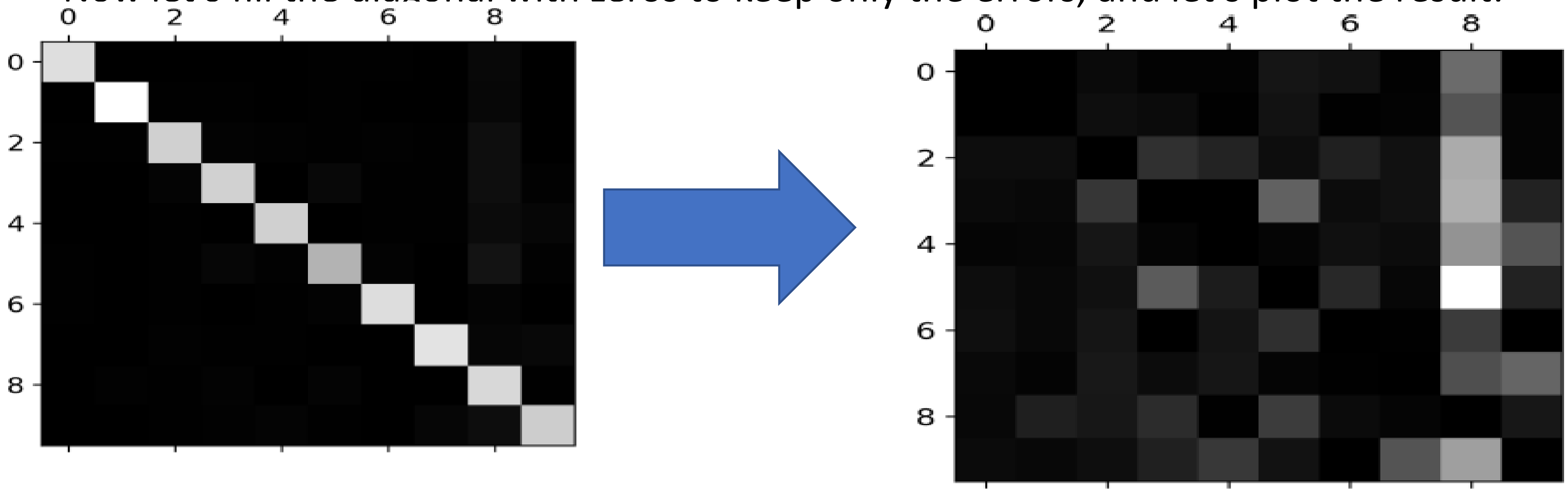
```
>>> conf_mx
array([[5578,    0,   22,    7,    8,   45,   35,    5,  222,    1],
       [   0, 6410,   35,   26,    4,   44,    4,    8,  198,   13],
       [  28,   27, 5232,  100,   74,   27,   68,   37,  354,   11],
       [  23,   18,  115, 5254,    2,  209,   26,   38,  373,   73],
       [  11,   14,   45,   12, 5219,   11,   33,   26,  299,  172],
       [  26,   16,   31,  173,   54, 4484,   76,   14,  482,   65],
       [  31,   17,   45,    2,   42,   98, 5556,    3,  123,    1],
       [  20,   10,   53,   27,   50,   13,    3, 5696,  173,  220],
       [  17,   64,   47,   91,    3,  125,   24,   11, 5421,   48],
       [  24,   18,   29,   67,  116,   39,    1,  174,  329, 5152]])
```



- As there are lot of numbers, we can plot it as an image)Matplotlib’s matshow() function)
- This confusion matrix looks fairly good, since most images are on the main diagonal, which means that they were classified correctly. The 5s look slightly darker than the other digits, which could mean that there are fewer images of 5s in the dataset or that the classifier does not perform as well on 5s as on other digits. In fact, you can verify that both are the case.

Error Analysis and improvement

- Let’s focus the plot on the errors.
- First, you need to divide each value in the confusion matrix by the number of instances in the corresponding class, so you can compare error rates instead of absolute number of errors:
    - e.g. if it classified a 7 as a 1, but there is only 1 7 in the dataset, then it completely failed at that task
- Now let’s fill the diagonal with zeros to keep only the errors, and let’s plot the result:



- Now you can clearly see the kinds of errors the classifier makes.
- Remember that rows represent actual classes, while columns represent predicted classes.
- The column for class 8 is quite bright, which tells you that many images get misclassified as 8s. How- ever, the row for class 8 is not that bad, telling you that actual 8s in general get prop- erly classified as 8s.
- You can also see that 3s and 5s often get confused (in both directions).

Error Analysis and improvement

- Looking at this plot, it seems that your efforts should be spent on reducing the false 8s.
- For example, you could try to gather more training data for digits that look like 8s (but are not) so the classifier can learn to distinguish them from real 8s.
- Or you could engineer new features that would help the classifier—for example, writing an algorithm to count the number of closed loops (e.g., 8 has two, 6 has one, 5 has none).
- Or you could preprocess the images (e.g., using Scikit-Image, Pillow, or OpenCV) to make some patterns stand out more, such as closed loops.
- Analyzing individual errors can also be a good way to gain insights on what your classifier is doing and why it is failing, but it is more difficult and time-consuming.

Multi label classification

- Until now each instance has always been assigned to just one class.
- In some cases you may want your classifier to output multiple classes for each instance.
- For example, consider a face-recognition classifier:
  - what should it do if it recognizes several people on the same picture?
  - Of course it should attach one tag per person it recognizes.
  - Say the classifier has been trained to recognize three faces, Alice, Bob, and Charlie;
  - then when it is shown a picture of Alice and Charlie, it should output [1, 0, 1] (meaning “Alice yes, Bob no, Charlie yes”). Such a classification system that outputs multiple binary tags is called a multilabel classification system

Performance Matrix for Regression

- We will learn this when we will learn regression