week 3: ROC curves, Normalization and PCA

http://mlvu.github.io

February 8, 2021

1 Exam questions

At this point in the course, it's a good idea to have a look at the practice exam. The homework prepares you for the exam, but the format of the homework questions is different from that of the exam questions. Here are some questions you should be able to answer based on the material covered so far. You can find two complete practice exams on Canvas.

1. What is the advantage of gradient descent over random search?
   A In gradient descent, parallel searches are allowed to communicate.
   B Gradient descent is less likely to get stuck in local minima.
   C Gradient descent computes the direction of steepest descent, random search approximates it.✓
   D Gradient descent is easier to parallelise.

2. Why is accuracy a bad loss function to use in gradient descent?
   A It is expensive to compute.
   B It makes the gradient zero almost everywhere. ✓
   C It is unreliable in situations with high class imbalance.
   D The confidence interval is high on small test sets.

3. I'm performing spam classification. I represent each email by three numbers: how often the word *pill* occurs, how often the word *hello* occurs and how often the word *congratulations* occurs. What are these three attributes called?
   A The instances
   B The classes
   C The features ✓
   D The principal components

Here we see the derivation of the gradient of the squared-error loss for linear regression.

$$\frac{\partial \frac{1}{2}\sum_i (f(x_i) - y_i)^2}{\partial b} = \frac{1}{2}\frac{\partial \sum_i (x_i w + b - y_i)^2}{\partial b} \tag{1}$$

$$= \frac{1}{2}\sum_i \frac{\partial (x_i w + b - y_i)^2}{\partial b} \tag{2}$$

$$= \frac{1}{2}\sum_i \frac{\partial (x_i w + b - y_i)^2}{\partial (x_i w + b - y_i)} \frac{\partial (x_i w + b - y_i)}{\partial b} \tag{3}$$

$$= \sum_i (x_i w + b - y_i) \frac{\partial (x_i w + b - y_i)}{\partial b} \tag{4}$$

$$= \sum_i (x_i w + b - y_i) \tag{5}$$

4. To get from line (1) to line (2), we use the
   A Chain rule **B** Product rule **C** Exponent rule **D** Sum rule ✓

5. To get from line (2) to line (3), we use the
   A Chain rule ✓ **B** Product rule **C** Exponent rule **D** Sum rule

6. In line 5, the correct result is
   A $\sum_i \left(x_i{}^2 w + b - y_i\right)$
   B $\sum_i x_i (x_i w + b - y_i)$
   C $\sum_i (x_i w + b - y_i)$ ✓
   D $\sum_i (x_i w + b - y_i)^2$

2 ROC Curves and confusion matrices

In this section, we'll look at evaluation metrics. This section covers the lecture "Methodology 1" and section 2.2 of Peter Flach's *Machine Learning* (see the PDF on Canvas).

As a running example, we will use the following classification dataset:

	x_1	x_2	label
a	1	0	Neg
b	2	0	Pos
c	3	0	Pos
d	$1/2$	1	Pos
e	2	2	Pos
f	0	3	Neg
g	1	3	Neg
h	2	3	Neg

If you have trouble with the exercises below, we recommend drawing the data in its feature space, and drawing the decision boundary of each classifier.

2.1 A linear classifier

question 1: As a first classifier, c_{lin}, we will use a diagonal line crossing all the points where $x_1 = x_2$. Points above this line will be classified as Neg, points below or on the line as Pos. How do we describe this classifier mathematically? Fill in the blanks:

$$c_{lin}(x_1, x_2) = \begin{cases} \text{Pos} & \text{if } x_1 \geqslant x_2 \\ \text{Neg} & \text{otherwise} \end{cases}.$$

The diagonal that the question asks for (the decision boundary) is the set of points where $x_1 = x_2$. If $x_1 > x_2$, we're above this line, so we should classify the point as Pos.
c_{lin} is a simple classifier. Let's try a more complex one before we move on.

question 2: Let c_2 be the classifier whose decision boundary crosses the points $(0, 1)$ and $(1, 1.5)$, with the class *Neg* above the line. Draw this line first. How do we define this classifier?

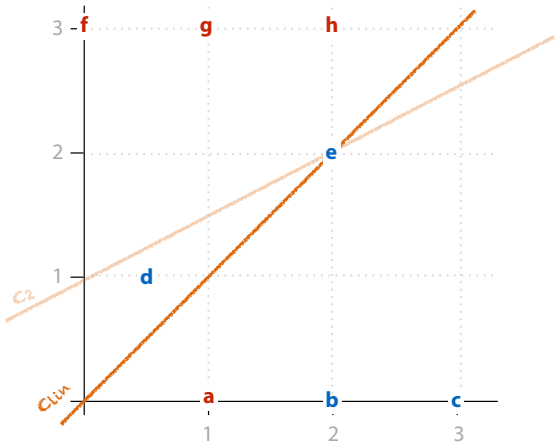
$$c_2(x_1, x_2) = \begin{cases} \text{Pos} & \text{if } -0.5x_1 + x_2 - 1 \leqslant 0 \\ \text{Neg} & \text{otherwise} \end{cases}$$

We could fill the given values in to the equation $ax_1 + bx_2 + c = 0$ and solve for zero. But that would give us two problems: it's a lot of work, and we have two equations for three variables. There are infinitely many pairs a and b that would give us this decision boundary. Let's make things easier

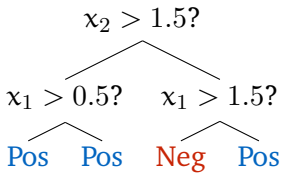
by assuming that b is 1, and rewriting to $x_2 = -ax_1 - c$. This is just the familiar function for a line in the plane: $-a$ is the slope, and $-c$ is the bias.

The given points tell us that the line crosses the x_2 axis at 1, and the slope should be 0.5. Thus, $-a = 0.5$ and $-c = 1$. A quick check tells us that for a point below the line, $(0, 0)$, the quantity $-0.5x_1 + x_2 - 1$ is negative, so such points should be classified as Pos.

Here is a diagram of the data and the boundaries for the two classifiers:



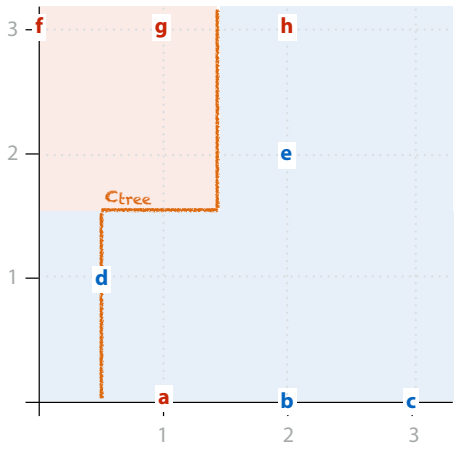
question 3: We will compare the linear classifier c_{lin} with a *decision tree* classifier, c_{tree}. Here is the decision tree:



If the inequality on the node is true, move right, otherwise move left.

This tree divides the feature space into four **segments** (A, B, C and D). To each segment we assign the majority class in that segment (using Pos for a draw). Label the leaves A, B, C and D with classes.

Here is a diagram of the data and the boundaries for the tree classifier:



question 4: In this exercise we don't do any training (we just evaluate the given classifiers), so we'll evaluate all metrics on a single given dataset.

If you do train, you would *split* your dataset. You would compute your evaluation metrics on the validation data during hyperparameter selection and on the test data during final testing.

Evaluating on the training data is the worst sin you can commit in Machine Learning. Can you think of a situation when you would still want to *compute* the training loss?

Getting good training loss, but poor validation loss is a sure sign of overfitting. If overfitting is likely (as it is in decision trees and neural networks), you may want to compare the training loss to the validation loss to see if overfitting explains the poor performance.

question 5: Give the *confusion matrices* for classifiers c_{lin} and c_{tree}.

c_{lin}		predicted Pos	predicted Neg
true	Pos	3	1
	Neg	1	3

c_{tree}		predicted Pos	predicted Neg
true	Pos	4	0
	Neg	2	2

question 6: From the confusion matrices, we can compute several metrics. Compute the accuracy, precision, recall, true positive rate and false positive rate

We compute these metrics from the four values of the confusion matrix: the number of true positives (TP), true negatives (TN), false positives (FP) and false negatives(FN):

		predicted	
		Pos	Neg
true	Pos	TP	FN
	Neg	FP	TN

		c_{lin}	c_{tree}
accuracy	$\frac{TP+TN}{total}$	$3/4$	$3/4$
precision	$\frac{TP}{TP+FP}$	$3/4$	$2/3$
recall	$\frac{TP}{TP+FN}$	$3/4$	1
true positive rate	$\frac{TP}{TP+FN}$	$3/4$	1
false positive rate	$\frac{FP}{TN+FP}$	$1/4$	$1/2$

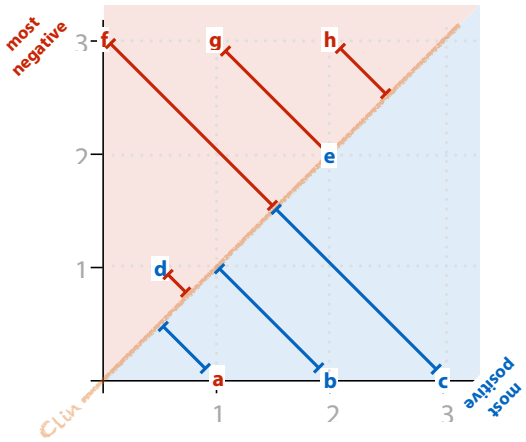
Most of these are pretty intuitive, if you work out the meaning of the various sums. For instance, the false positive rate is the proportion of actual negatives, that were predicted to be positive.

To plot a ROC and precision/recall curves, and to compute AUC metrics, we need to turn our simple binary classifier into a *ranking* classifier. That is, we don't just want classifications, we want it to rank the data from most Neg to most Pos. How we do this depends on the classifier.

question 7: How do we turn a linear classifier into a ranking classifier? Give the ranking that c_{lin} assigns to the training data.

We compute the distance to the classification boundary (the negative distance on one side, and the positive on the other), and rank points by this distance. For this course (the homework and the exam), you don't actually need to compute the distance, it's fine to draw the dataset and "measure" the distances. We'll make sure that all examples allow this approach.

Here's the sort of diagram you should end up with:



ranking:

So, for c_{lin} the ranking from negative to positive is

f g h d e a b c .

question 8: How do we turn a decision tree classifier into a ranking classifier? Give the ranking that c_{tree} assigns to the training data.

We group the points by the way the decision tree segments the feature space. We then assign each segment a class probability based on the relative frequencies of *training* data points. We sort the groups by class probability. Points within the same group are ranked equal.

For c_{tree} the ranking from negative to positive is

(**f g**) (**h e**) (**a b c**) (**d**).

Brackets indicate equal rank.

We can't tell whether our ranking is perfect, because the training data doesn't give us a correct *ranking*, only a correct *labeling*. But for some pairs of instances we *do* know how they should be ranked: all pairs of positive and negative examples.

We can visualize this in a matrix with negative instances on the horizontal axis, and positive instances on the vertical, both sorted with the most positive (according to the classifier's ranking) in the bottom left corner. The

matrix is usually colored green for pairs that are ranked correctly, red for pairs ranked incorrectly, and yellow for pairs that are ranked equal.

question 9:

1. Draw the coverage matrices of c_{lin} and c_{tree}.

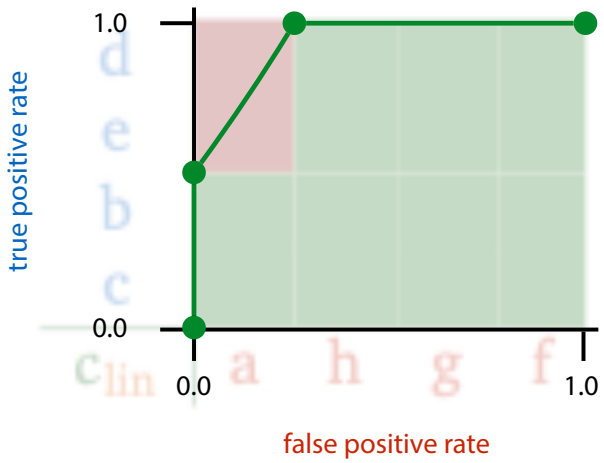
2. How many ranking errors does each classifier make on the training data? c_{lin} makes 2 ranking errors. c_{tree} makes 2.5, counting each tie as half a ranking error.

3. If we have class imbalance (one class has more instances than the other), how can we tell by the coverage matrix?

   It will be non-square. The more stretched out, the higher the class imbalance.

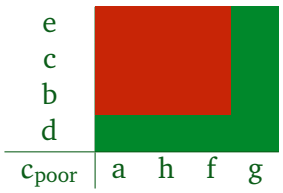
question 10: How do we convert a coverage matrix to an **ROC plot**? How does the green area in a coverage matrix relate to the ROC plot? How do they differ?

We can convert the coverage matrix to ROC space by normalizing the axes. That is, if we have a green cell at $(2, 3)$, counting from the bottom left, we can draw a point in ROC space at $(2/4, 3/4)$. That is, we know we have a classifier that can achieve a true positive rate of $3/4$ and a false negative rate of $2/4$.



The green area gives us an estimate of the probability of a ranking error. The same is true of the *Area Under the Curve*. For the latter, we draw the convex hull of all achievable classifiers, and compute the area.

An important difference is that the green area in a coverage matrix may be *concave*. For instance:



The corresponding ROC curve is convex (it covers the bottom right half of the ROC space).

Since a classifier with AUC below 0.5 is pretty useless, we don't usually worry about this. The differences between the (normalized) coverage matrix and the ROC space are not important for reasonable classifiers and large datasets.

3 Whitening

Basis systems The *standard basis system* (in 2D) is just the vectors $b_1 = \begin{pmatrix}1\\0\end{pmatrix}$ and $b_2 = \begin{pmatrix}0\\1\end{pmatrix}$. When a point has coordinates (x, y), that just means that we can get to the point by adding x times the b_1 vector to y times the b_2 vector.

For the standard basis, we just end up at the point $\begin{pmatrix}x\\y\end{pmatrix}$. But we can also use a nonstandard basis. If a point p has coordinates (x, y) in the coordinate system with basis vectors b_1, b_2, it means that p has coordinates $xb_1 + yb_2$ in the standard basis system.

question 11: Convert the following points to a representation in the standard basis system.

1. The point $\begin{pmatrix}1\\1\end{pmatrix}$ expressed in the coordinate system with basis vectors $b_1 = \begin{pmatrix}2\\0\end{pmatrix}$, $b_2 = \begin{pmatrix}0\\2\end{pmatrix}$. $x = 1, y = 1$, so we get to the standard basis point by computing $1 \cdot b_1 + 1 \cdot b_1 = \begin{pmatrix}2\\2\end{pmatrix}$
2. The point $\begin{pmatrix}5\\3\end{pmatrix}$ expressed in the coordinate system with basis vectors $b_1 = \begin{pmatrix}0\\1\end{pmatrix}$, $b_2 = \begin{pmatrix}1\\0\end{pmatrix}$. $\begin{pmatrix}3\\5\end{pmatrix}$
3. The point $\begin{pmatrix}1\\2\end{pmatrix}$ expressed in the coordinate system with basis vectors $b_1 = \begin{pmatrix}1\\2\end{pmatrix}$, $b_2 = \begin{pmatrix}3\\4\end{pmatrix}$. $\begin{pmatrix}7\\10\end{pmatrix}$

question 12: A basis system is **orthonormal** if two conditions hold:

1. The basis vectors all have norm (i.e. length) 1. (normality)
2. The angles between any two basis vectors are $90\,\text{deg}$. (orthogonality)

Consider the following three bases:

$$\mathbf{A} = \begin{pmatrix}0 & 1\\1 & 1\end{pmatrix} \quad \mathbf{B} = \begin{pmatrix}1 & -1\\1 & 1\end{pmatrix} \quad \mathbf{C} = \sqrt{1/2}\begin{pmatrix}1 & -1\\1 & 1\end{pmatrix}$$

The columns of each matrix are the basis vectors (it may help to draw them in standard coordinates). Which of these have orthogonal basis vectors? Which one is orthonormal? The columns of $\mathbf{A}$ are neither normal nor orthogonal. Matrix $\mathbf{B}$ has orthogonal columns, but they are not unit vectors (vectors of length one). Matrix $\mathbf{C}$ represents an orthonormal basis.

question 13: Why is it useful to have an orthonormal basis? Matrices representing an orthonormal basis are *orthogonal*. This means that $\mathbf{C}^{-1} = \mathbf{C}^{\mathsf{T}}$. This means that we can easily transform from and to the basis without having to compute an expensive matrix inverse.

Covariance and normal distributions PCA is closely related to the normal distribution. We can think of PCA as a basis transformation that makes it look like the data was sampled from a standard multivariate normal distribution: it makes the variance 1 in every dimension, and the covariance 0 for every pair of dimensions.

question 14: Here is some data.

x_1	x_2
0	2
2	0
0	2
2	0

Compute the sample covariance.

We first compute the sample mean: $\mathbf{m} = \begin{pmatrix} 1 \\ 1 \end{pmatrix}$.

Subtracting the sample mean, and arranging the data in a matrix (with instances as columns), we get

$$\mathbf{X} = \begin{pmatrix} -1 & 1 & -1 & 1 \\ 1 & -1 & 1 & -1 \end{pmatrix}.$$

The sample covariance is

$$\mathbf{S} = \frac{1}{n-1}\mathbf{X}\mathbf{X}^{\mathsf{T}} = \frac{1}{3}\begin{pmatrix} 4 & -4 \\ -4 & 4 \end{pmatrix}$$

Let $\mathbf{y} = \mathbf{A}\mathbf{x} + \mathbf{t}$ be an *affine* transformation (a linear transformation $\mathbf{A}$ followed by a translation $\mathbf{t}$). We draw $\mathbf{x}$ from a standard normal distribution

$$\mathrm{N}(\mathbf{0}, \mathbf{I}) = \mathrm{N}\left(\begin{pmatrix} 0 \\ 0 \end{pmatrix}, \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}\right)$$

and compute $\mathbf{y}$. This is equivalent to sampling from a normal distribution with mean $\boldsymbol{\mu} = \mathbf{t}$ and covariance $\boldsymbol{\Sigma} = \mathbf{A}\mathbf{A}^{\mathsf{T}}$.

question 15: Let

$$\mathbf{A} = \begin{pmatrix} 1 & 0 \\ 0 & 2 \end{pmatrix}, \mathbf{t} = \begin{pmatrix} 1 \\ 1 \end{pmatrix}$$

Sampling $\mathbf{x}$ from $\mathsf{N}(\mathbf{0}, \mathbf{I})$ is easy: we just sample x_1 and x_2 independently from a *uni*variate standard normal distributions. Here are some standard normally distributed numbers:

$$0.5, 1.1, 0.1, 0.9, -0.2, 1.3$$

Turn these into three samples from $\mathsf{N}(\mathbf{0}, \mathbf{I})$ and transform them to samples from $\mathsf{N}(\boldsymbol{\mu}, \boldsymbol{\Sigma})$.

Using samples

$$\boldsymbol{x}^1 = \begin{pmatrix} 0.5, & 1.1 \end{pmatrix}^{\mathsf{T}}, \quad \boldsymbol{x}^2 = \begin{pmatrix} 0.1, & 0.9 \end{pmatrix}^{\mathsf{T}} \quad \boldsymbol{x}^3 = \begin{pmatrix} -0.2, & 1.3 \end{pmatrix}^{\mathsf{T}}$$

and computing $\mathbf{y}^1 = \mathbf{A}\mathbf{x}^1 + \mathbf{t}$, we get:

$$\mathbf{y}^1 = \begin{pmatrix} 1.5, & 3.2 \end{pmatrix}^{\mathsf{T}}, \quad \mathbf{y}^2 = \begin{pmatrix} 1.1, & 2.8 \end{pmatrix}^{\mathsf{T}} \quad \mathbf{y}^3 = \begin{pmatrix} 0.8, & 3.6 \end{pmatrix}^{\mathsf{T}}.$$

Bonus question: Compute the sample covariance of the resulting data, and the actual covariance Σ.

For the sample covariance, follow the same procedure as in the previous question. After a bit of calculating, we get:

$$\mathbf{S} \approx \begin{pmatrix} 0.123 & 0.06 \\ 0.06 & 0.16 \end{pmatrix}$$

The true covariance we get by computing $\boldsymbol{\Sigma} = \mathbf{A}\mathbf{A}^{\mathbf{T}} = \begin{pmatrix} 1 & 0 \\ 0 & 4 \end{pmatrix}$. They're quite different at the moment. If we increase the size of the sample, $\mathbf{S}$ will converge to $\boldsymbol{\Sigma}$.