Artificial Neural Networks
Lecture Notes

Prof. Dr. Sen Cheng

Winter Semester 2019/20

5 Classification

Regression: predict values, continuous output

Classification: predict class, discrete output

An algorithm that makes classification judgements is called a *classifier*. In its simplest form we would like to determine whether an instance belongs to a class or not. This is called *binary classification*. It is more common than one might think. Examples: spam filters, predicting pass/fail on tests, credit card fraud detection, developing cancer or not, etc.

5.1 Logistic regression (logit regression)

The first binary classifier.

Data: continuous input X, discrete output Y

Model class: The model consists of two components: a function that computes a probability $\hat{p}(x)$ that a particular item x belongs to the class, and a threshold process that makes a decision based on the probability. The latter component is straight-forward and takes the form

$$\hat{y}(x) = \begin{cases} 0 & \hat{p} < p_0 \\ 1 & \hat{p} \geq p_0 \end{cases} \tag{1}$$

Usually $p_0 = 0.5$, but we can set it to other values to be more conservative or more liberal – depending on the problem at hand.

The first component, a function that calculates the probability, is more complicated. It would be nice if we could use a linear function $f(x) = \theta^T x$ (as we did in linear regression) to parametrize the predicted probability $\hat{p}(x)$ because it's simple and it takes into account and combines different factors or features, x_j, i.e. $\theta^T x = \sum \theta_j x_j$. However, the linear function takes on values from $-\infty$ to ∞ (Fig. 1a), and probabilities are constraint: $0 \leq \hat{p}(x) \leq 1$. That's why we relate another quantity of the stochastic process to $\theta^T x$.

Assume we have a Bernoulli process with probability p, such that $0 < p < 1$. The *log odds ratio*, $\log\left(\frac{p}{1-p}\right)$, is continuous and ranges from $-\infty$ to $+\infty$, so it can be equated to $\theta^T x$.

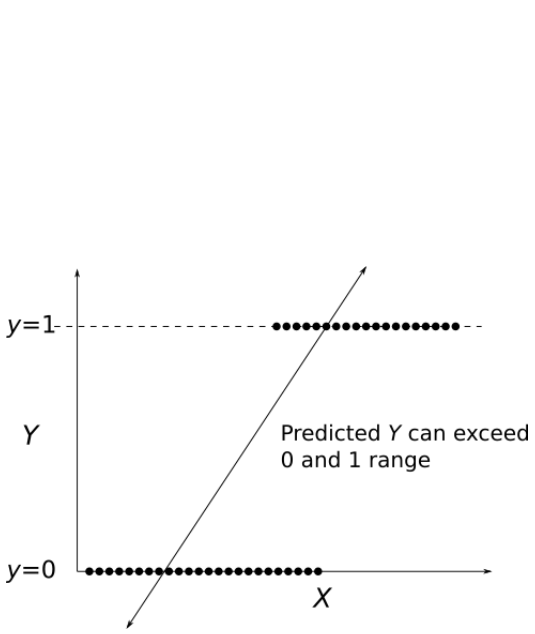
$$\log\left(\frac{p}{1-p}\right) = \theta^T x \tag{2a}$$

$$\frac{p}{1-p} = e^{\theta^T x} \tag{2b}$$

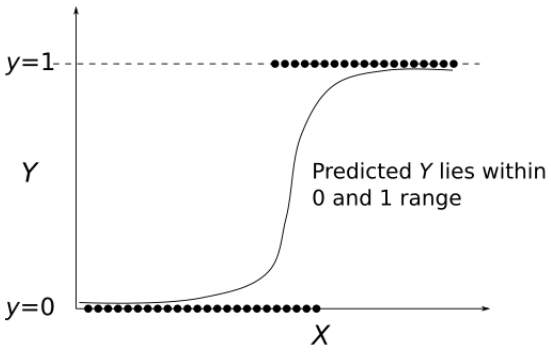
$$p = (1-p)e^{\theta^T x} \tag{2c}$$

$$p\left(1 + e^{\theta^T x}\right) = e^{\theta^T x} \tag{2d}$$

$$p = \frac{e^{\theta^T x}}{1 + e^{\theta^T x}} \tag{2e}$$



(a) Linear regression.



(b) Logistic regression.

Hence, the probability in logistic regression is (Fig. 1b)

$$p(x) = \frac{1}{1+e^{-\theta^T x}} \tag{3}$$

That's why the function

$$\sigma(t) = \frac{1}{1+e^{-t}} \tag{4}$$

is called the *logistic function*. It's sometimes referred to as *the* sigmoid function, even though a *sigmoid function* is generally any s-shaped curve that is bounded.

To summarize the classification model in logistic regression:

$$\hat{y}(x) = \begin{cases} 0 & \sigma(\theta^T x) < p_0 \\ 1 & \sigma(\theta^T x) \geq p_0 \end{cases} \tag{5}$$

Later, we will need the derivatives of this function.

$$\frac{\mathrm{d}}{\mathrm{d}t}\sigma(t) = \frac{-1}{(1+e^{-t})^2} e^{-t}(-1) \tag{6a}$$

$$= \frac{1+e^{-t}-1}{(1+e^{-t})^2} \tag{6b}$$

$$= \frac{1}{1+e^{-t}} - \frac{1}{(1+e^{-t})^2} \tag{6c}$$

$$\frac{\mathrm{d}}{\mathrm{d}t}\sigma(t) = \sigma(t)\,(1-\sigma(t)) \tag{6d}$$

5.1.1 Cross entropy

The loss function should capture the difference between two probability distributions.

	in class	not in class
true	p_1	p_2
predicted	$\hat{p}_1$	$\hat{p}_2$

Since probabilities have to sum up to 1, we can simplify this to

	in class	not in class
true	p	$1-p$
predicted	$\hat{p}$	$1-\hat{p}$

The *cross entropy* is frequently used in *information theory*, and now machine learning, to quantify the difference between two statistical distributions. We don't have time to derive the equation for cross entropy.

$$H(p,\hat{p}) = -\sum_c p_c \log(\hat{p}_c) \tag{7}$$

Applied to binary classification of one item

$$\begin{aligned} H(p,\hat{p}) &= -p_1 \log(\hat{p}_1) - p_2 \log(\hat{p}_2) &(8)\\ &= -y \log(\hat{p}) - (1-y)\log(1-\hat{p}) &(9)\end{aligned}$$

The cross entropy works better for fitting probabilities because it increases much more steeply for deviations than the

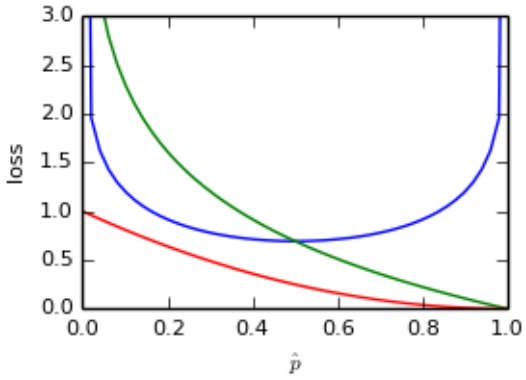


Figure 1: Loss functions as a function of estimated probability. Binary cross entropy, $p = 0.5$ (blue), $p = 1$ (green). Square loss, $p = 1$ (red).

square loss, which is rather small in the interval $[0,1]$ (Fig. 1).

Since the cross entropy is calculated for individual items, we have to sum the cross entropies over all items to get the **loss function** for the entire data:

$$L(\theta,X,Y) = -\frac{1}{N}\sum_{i=1}^{N}\left[y_i \log(\hat{p}(x_i)) + (1-y_i)\log(1-\hat{p}(x_i))\right] \tag{10}$$

5.1.2 Incremental algorithm

Optimization: Unlike in linear regression, no closed form solution is known for the parameters $\hat{\theta}$ that will minimize the loss function in logistic regression. So we have to use a numerical method, such as gradient descent, to find the global minimum. The gradient of the loss function is

$$\nabla_\theta L(\theta,X,Y) = \frac{1}{N}\sum_{i=1}^{N}\left(\sigma(\theta^T x_i) - y_i\right)x_i \tag{11}$$

The derivaiton of Eq. 11 is left as a homework excercise.

If the loss function Eq. 10 is convex, gradient descent is guaranteed to converge to the global minimum, if the learning rate is chosen appropriately. The loss function is convex, if the second derivative, the Hessian matrix, is positive semi-definite.

$$\nabla^2_\theta L(\theta,X,Y) = \frac{1}{N}\sum_{i=1}^{N}\sigma(\theta^T x_i)\left(1-\sigma(\theta^T x_i)\right)x_i x_i^T \tag{12}$$

where we have used Eq. 6d and the chain rule. Since the sigmoid is constrained between 0 and 1, the first two terms are positive. The matrix xx^T is positive semi-definite because for any vector z: $z^T xx^T z = (x^T z)^2 \geq 0$. Therefore, the Hessian is positive semi-definite.

		predicted	
		in class	**not in class**
true	**in class**	true positive (TP)	false negative (FN)
	not in class	false positive (FP)	true negative (TN)

Table 1: Types of correct responses and errors in binary classsification. False positives are also known as type 1 error, and false negatives as type 2 error.

5.2 Analyzing performance

The performance of a classifier is more difficult to assess than that of a regression algorithm. In linear regression, the square loss function immediately gives you an intuitive sense of how far off your estimates are. The cross entropy is somewhat opaque. Also, the deviations in the probability distribution that it captures might be far from the needs in an application. To understand this, note that there are two types of errors in binary classification (Table 1). The two different types of errors (false positives and false negatives) can be traded off, i.e., one can choose which error rate to minimize, but that will increase the rate of the other error. This choice is introduced by the parameter p_0. The larger p_0, the harder it is to classify an item as in the class, so FP goes down and FN goes up.

In some cases, it might be better to put as many item as possible in the class, i.e. minimize the rate of FN. However, that will come at the price of raising the rate of FP. For instance, if your test detects cancer, patients prefer that you catch all real instances of cancer, even if that means that there are many false alarms (FP), since early treatment increases survival rates. In some other cases, one might prefer to only put items in the class if they truely belong there (minimize FP), at the expense of missing a few (increasing FN). This might be the case if you decide whether to perform a heart transplant. The procedure is so risky and expensive that you don't want to perform it unnecessarily, even if that means that in a few cases you don't perform a surgery that could have helped the patient. Finally, in yet other cases, it might be best to find lowest overall error rate, irrespective of which type the error is.

It is useful to have a way to measure how well our classifier describes the data regardless of our choice of which error to minimize. We turn to such methods next. They are also useful to select an appropriate threshold.

5.2.1 Precision-Recall curve

The *precision* is defined as the fraction of "yes" responses that are correct, i.e.,

$$\text{precision} = \frac{TP}{TP+FP} \tag{13}$$

Recall is defined as the fraction of class items that are correctly classified, i.e.,

$$\text{recall} = \frac{TP}{TP+FN} \tag{14}$$

This is also known as the *hit rate*. For both precision and recall, higher values are better.

The precision-recall curve is a plot of precision against recall that is obtained by running the classification with different decision thresholds (Fig. 2). A given value for the decision threshold leads to a certain point on the precision-recall curve. Since the curve is monotonically decreasing, you can either have high precision or high recall, but not both at the same time. This is known as the *precision-recall tradeoff* and it is a direct consequence of the tradeoff between FN's and FP's. One strategy to pick a decision threshold that is a good tradeoff is the following method: start on the left most point (highest precision), move along the curve, and stop just before the curve descends steeply. Precision and and recall are sometimes combined into a single F_1 score, which is the harmonic mean of the two variables;

$$F_1 = \frac{2}{\frac{1}{\text{precision}} + \frac{1}{\text{recall}}} = \frac{TP}{TP + \frac{FN+FP}{2}} \tag{15}$$

Hence, the F_1 score takes into account both error types. One could pick a decision threshold to maximize F_1.

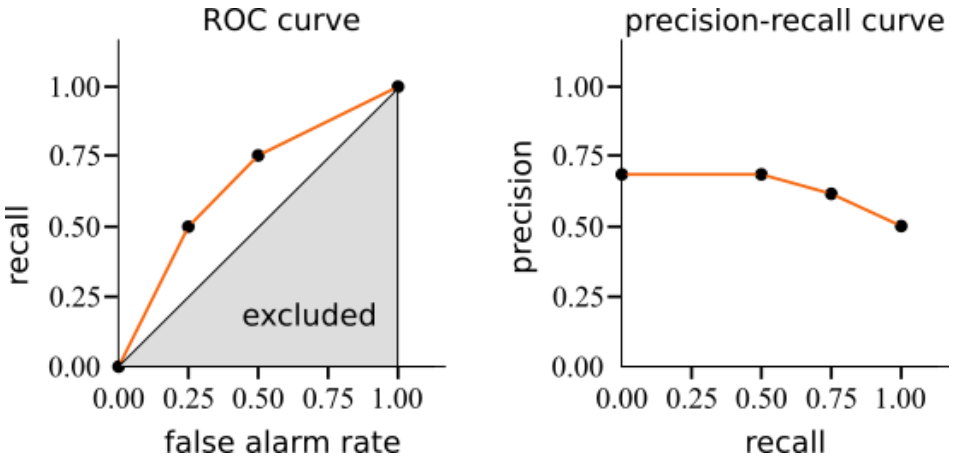


Figure 2: Precision-Recall and ROC curve.

5.2.2 ROC curves

The receiver operating characteristics (ROC) curve is a plot of the recall rate (hit rate) against the rate of false positives (false alarm rate), which is

$$\text{FPR} = \frac{\text{FP}}{\text{TN} + \text{FP}} \tag{16}$$

The ROC curve is monotonically increasing due to the tradeoff between the two types of errors. The identity line ($y = x$) indicates random guessing. The further away the ROC curve is from the identity line, the better the performance of the classifier. The lower triangle in Fig. 2 is excluded, because performance in this area would indicate that out-of-class items were predicted to be in-class more often than in-class items were. This would only be possible, if the classifier knew what the right answers were, but for some reason gave the wrong answer instead.

Picking a decision threshold selects a point along the ROC curve, but to move to a different curve requires changing the data or the classifier. That's why you can compare the performance of different classifiers by looking at the *area under the curve (AUC)*. The higher the AUC, the better. The maximum is AUC=1.

Rule of thumb: Use the precision-recall curve, when the positive class is rare, or when you care more about the FP than the FN. Otherwise use the ROC curve.

5.3 Multiclass classification

In many applications, the feature vectors can be classified into more than two classes (Fig. 3). We will discuss true multiclass classifiers later. However, multiclass classification can be achieved simply by using multiple binary classifiers. There are two strategies: one-vs-all (a.k.a. one-vs-all-other, one-vs-rest) and one-vs-one.

One-vs-all: One binary classifier is trained for each class $c_i, i = 1, \ldots, k$. Training includes the entire dataset, where all items in c_i belong to the class and all items in other classes do not. This method only works for binary classifiers that can return a real value that represents how well a given item matches the characteristics of the class. For instance, one can extract a probability from logistic regression. Given a feature vector x, the output of each binary classifier $\hat{p}_i(x)$ is treated as a vote for the class c_i, which the classifier was trained on. The class receiving the highest vote is chosen.

$$\hat{y} = \underset{i}{\text{argmax}}\ \hat{p}_i(x) \tag{17}$$

This method is frequently used. One of the biggest problems with this approach is that the training sets for the binary classifiers contain many more negative than positive items. This is not ideal for training binary classifiers.

One-vs-one: In this method, $k(k-1)/2$ binary classifiers are trained for multiclass classification with c classes. Each binary classifier is trained on items from a pair of classes to distinguish between these two. Given a feature vector x, each binary classifier gives a vote for one of the two classes it was trained on. The class receiving the highest number of votes is chosen as the final output. Unlike one-vs-all, this method works when the binary classifiers only ouptput

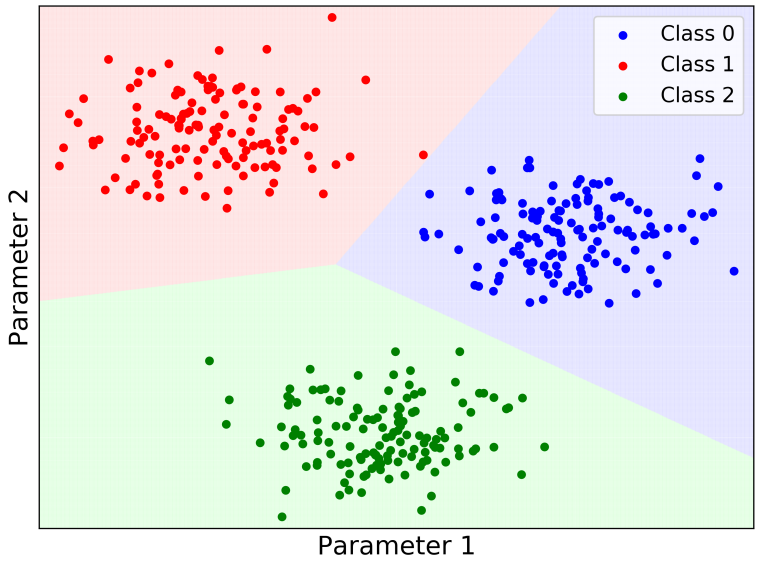


Figure 3: Classification with multiple classes.

binary decisions. One issue with this approach is that in some region of feature space, there might be ambiguity, i.e., more than one class receives the highest number of votes.

5.3.1 Confusion matrix

The *confusion matrix* M_{ij} indicates how often an item from class c_i is (mis-)classified by the algorithm as belonging to class c_j (Fig. 4). The confusion matrix helps visualize the overall performance of the multiclass classifier. If the performance is ideal, i.e. all items were assigned to the right class, the confusion matrix has nonzero entries along the diagonal, and zero entries everywhere else.

(a)

		predicted class			
		1	**2**	$\cdots$	**k**
actual class	**1**	M_{11}	M_{12}	$\cdots$	M_{1k}
	2	M_{21}	M_{22}	$\cdots$	M_{2k}
	:	$\vdots$	$\vdots$	$\ddots$	$\vdots$
	k	M_{k1}	M_{k2}	$\cdots$	M_{kk}

(b)

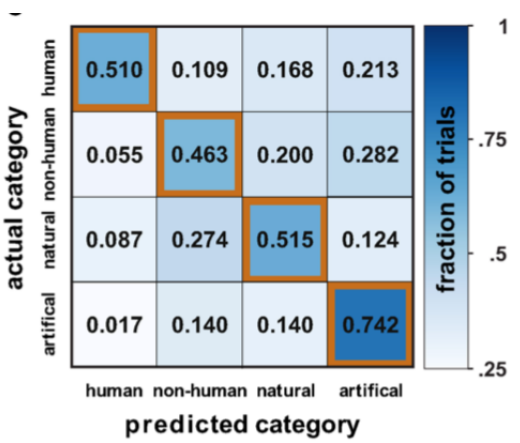


Figure 4: (a) Confusion matrix showing how many items from the acutal class c_i were (mis-)classified as class c_j by the classifier. (b) Example of a confusion matrix from Azizi et al. (2019).

5.4 Bayes classifier and Bayes error rate

Bayes' rule is very useful in cases where two things, e.g., A and B, depend on each other.

$$P(A|B) = \frac{P(B|A)P(A)}{P(B)} \tag{18}$$

Each term in Bayes' rule has a name: $P(A|B)$ is called the *posterior*, meaning the probability of A *after* we have observed B. Accordingly, $P(A)$ is the *prior*, the probability of A before we have observed B. $P(B|A)$ is the *likelihood* of B given A. Finally, $P(B)$ is the *normalization constant* to ensure that the right-hand side of the equation sums/integrates to one, as required for a probability distribution.

$$P(B) = \sum_{A's} P(B|A)P(A) \tag{19}$$

Bayes' rule can be applied to classification. This is then called the *naïve Bayes classifier*. Assume that we have a number of classes, denoted by c_i and feature vectors x. The classification problem could be solved if we knew $P(c_i|x)$ for all i. A given feature vector is assigned to the class c_i for which $P(c_i|x)$ is the highest. According to Bayes' rule

$$P(c_i|x) = \frac{p(x|c_i)P(c_i)}{p(x)} \tag{20}$$

Of course, writing the probability as a posterior only helps if the prior and the likelihood are known. This is sometimes the case, but much of the time it is not. Even though the name naïve Bayes classifier might suggest that it is a specific algorithm for classification, it is actually a general model of what a classification problem is about.

The *Bayes error rate* is a lower bound on the error rate in a classification problem, i.e., the lowest possible error rate that any classifier could theoretically achieve. It is analogous to the irreducible error in regression problems. The Bayes error rate is useful as a comparision to judge how close to optimal a given classifier is performing.

$$E_{\text{Bayes}} = 1 - \sum_i \int_{C_i} p(x|c_i)P(c_i)dx \tag{21}$$

where C_i is the feature region where class c_i has the highest posterior $P(c_i|x)$ (Fig. 3, shaded regions). So the Bayes error rate specifies how often the naïve Bayes classifier does not assign the correct class to a feature vector.

In practice, the Bayes error rate can be calculated only for very simple toy problems because for real data neither the distribution $p(x|c_i)$ nor the regions C_i in Eq. 21 are known. However, the concept is still useful since there are a number of methods to calculate an approximation of the Bayes error rate.

References

Azizi, A. H., Pusch, R., Koenen, C., Klatt, S., Bröker, F., Thiele, S., Kellermann, J., Güntürkün, O., & Cheng, S. (2019). Emerging category representation in the visual forebrain hierarchy of pigeons (Columba livia). *Behavioural Brain Research*, *356*, 423–434.