MLP, Softmax as MaxEnt classifier, F1 Score

Jindřich Libovický (reusing materials by Milan Straka)

Charles University
ÚFAL

Charles University in Prague
Faculty of Mathematics and Physics
Institute of Formal and Applied Linguistics

CC BY SA
unless otherwise stated

Today's Lecture Objectives

After this lecture you should be able to

- Implement training of multi-layer perceptron using SGD
- Explain the theoretical foundation behind the softmax activation function (including the necessary math)
- Choose a suitable evaluation metric for various classification tasks

Multilayer Perceptron

Multilayer Perceptron

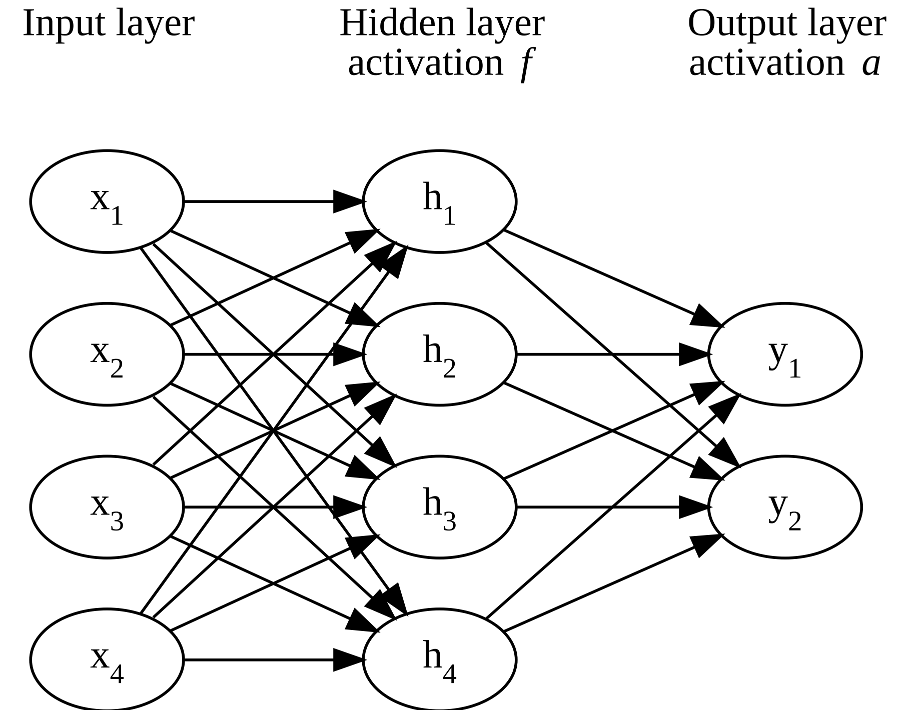
- The computation is performed analogously:

$$h_i = f\left(\sum\nolimits_j x_j w_{j,i}^{(h)} + b_i^{(h)}\right),$$
$$y_i = a\left(\sum\nolimits_j h_j w_{j,i}^{(y)} + b_i^{(y)}\right),$$

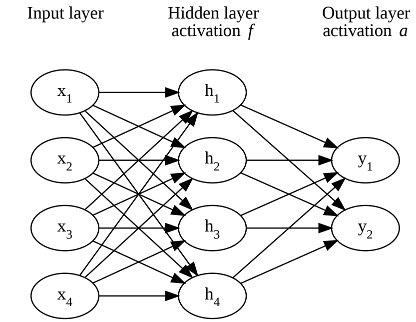
or in matrix form

$$\boldsymbol{h} = f\left(\boldsymbol{x}^T \boldsymbol{W}^{(h)} + \boldsymbol{b}^{(h)}\right),$$
$$\boldsymbol{y} = a\left(\boldsymbol{h}^T \boldsymbol{W}^{(y)} + \boldsymbol{b}^{(y)}\right),$$

and for batch of inputs $\boldsymbol{H} = f\left(\boldsymbol{X}\boldsymbol{W}^{(h)} + \boldsymbol{b}^{(h)}\right)$ and $\boldsymbol{Y} = a\left(\boldsymbol{H}\boldsymbol{W}^{(y)} + \boldsymbol{b}^{(y)}\right)$.



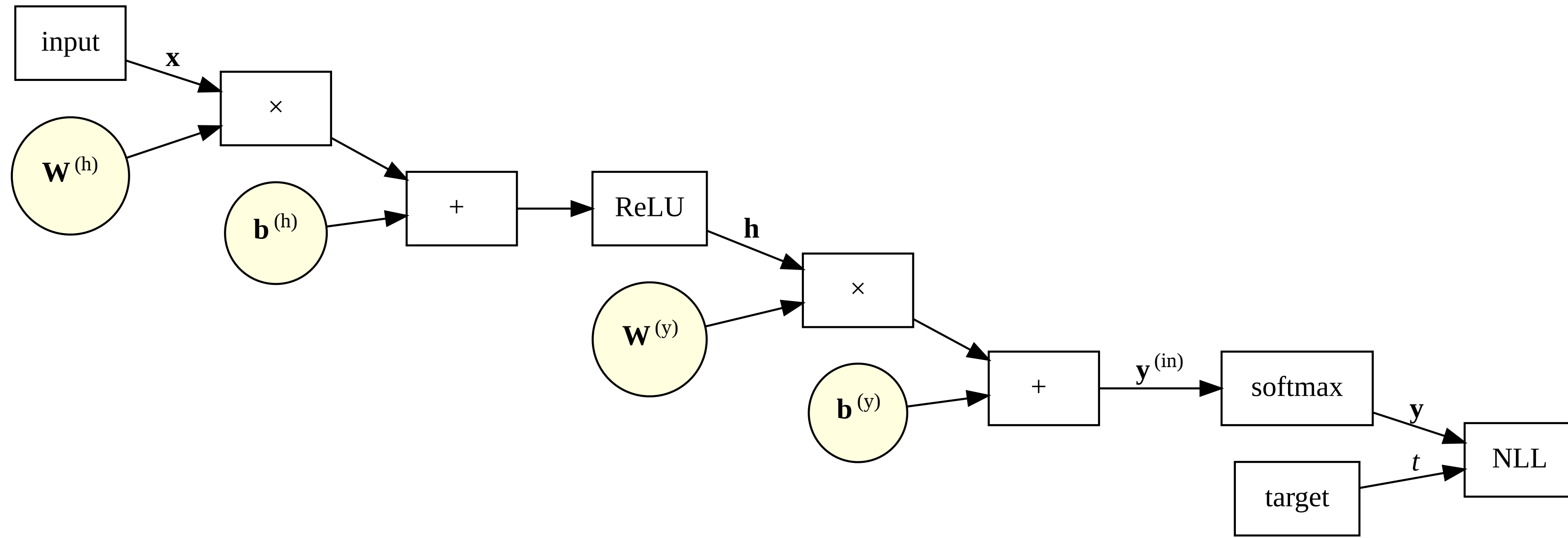
Assume we have an MLP with input of size D, weights $\boldsymbol{W}^{(h)} \in \mathbb{R}^{D\times H}$, $\boldsymbol{b}^{(h)} \in \mathbb{R}^H$, hidden layer of size H and activation f with weights $\boldsymbol{W}^{(y)} \in \mathbb{R}^{H\times K}$, $\boldsymbol{b}^{(y)} \in \mathbb{R}^K$, and finally an output layer of size K with activation a.



To compute the gradient of the loss L with respect to all weights, proceed gradually:

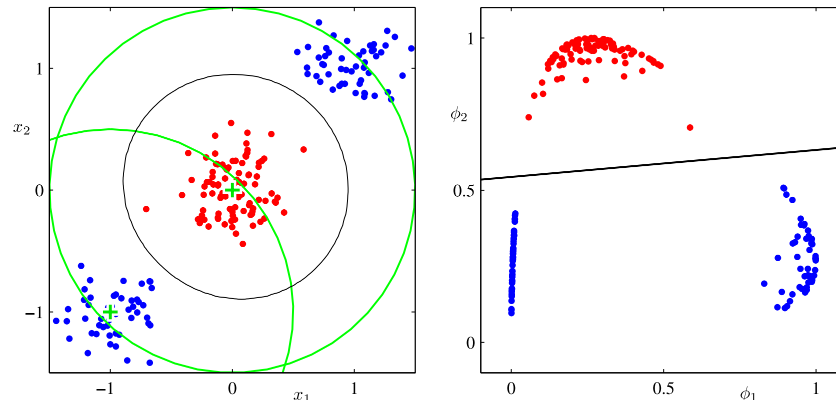
- first compute $\frac{\partial L}{\partial \boldsymbol{y}}$,
- then compute $\frac{\partial \boldsymbol{y}}{\partial \boldsymbol{y}^{(in)}}$, where $\boldsymbol{y}^{(in)}$ are the inputs to the output layer (i.e., before applying activation function a; in other words, $\boldsymbol{y} = a(\boldsymbol{y}^{(in)})$),
- then compute $\frac{\partial \boldsymbol{y}^{(in)}}{\partial \boldsymbol{W}^{(y)}}$ and $\frac{\partial \boldsymbol{y}^{(in)}}{\partial \boldsymbol{b}^{(y)}}$, which allows us to obtain $\frac{\partial L}{\partial \boldsymbol{W}^{(y)}} = \frac{\partial L}{\partial \boldsymbol{y}} \cdot \frac{\partial \boldsymbol{y}}{\partial \boldsymbol{y}^{(in)}} \cdot \frac{\partial \boldsymbol{y}^{(in)}}{\partial \boldsymbol{W}^{(y)}}$ and analogously $\frac{\partial L}{\partial \boldsymbol{b}^{(y)}}$,
- followed by $\frac{\partial \boldsymbol{y}^{(in)}}{\partial \boldsymbol{h}}$ and $\frac{\partial \boldsymbol{h}}{\partial \boldsymbol{h}^{(in)}}$,
- and finally using $\frac{\partial \boldsymbol{h}^{(in)}}{\partial \boldsymbol{W}^{(h)}}$ and $\frac{\partial \boldsymbol{h}^{(in)}}{\partial \boldsymbol{b}^{(h)}}$ to compute $\frac{\partial L}{\partial \boldsymbol{W}^{(h)}}$ and $\frac{\partial L}{\partial \boldsymbol{b}^{(h)}}$.

General Backpropagation Algorithm



- The computation can be represented as a graph with tensors and operations
- Computing the gradients corresponds to a backward path in the graph, nodes replaced with their derivatives

Hidden Layer Interpretation and Initialization



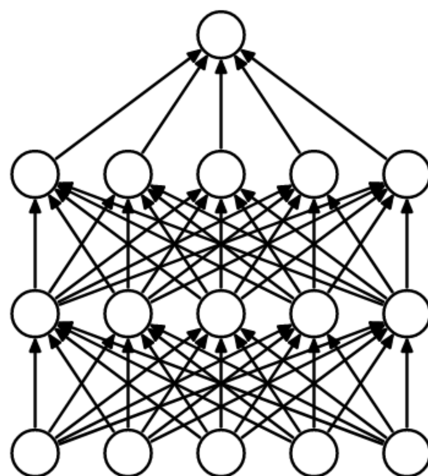
One way how to interpret the hidden layer is:

- the part from the hidden layer to the output layer is the previously used generalized linear model (linear regression, logistic regression, …);
- the part from the inputs to the hidden layer can be considered automatically constructed features. The features are a linear mapping of the input values followed by a nonlinearity, and the theorem on the next slide proves they can always be constructed to achieve as good a fit of the training data as is required.

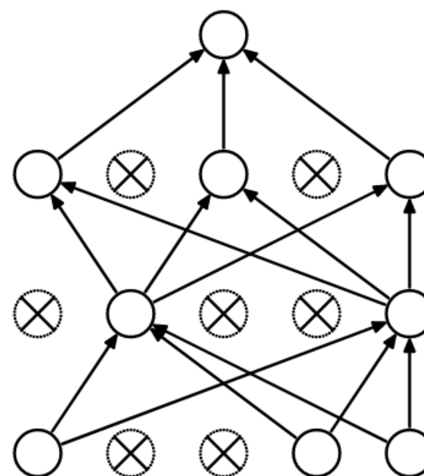
Note that the weights in an MLP must be initialized randomly. If we used just zeros, all the constructed features (hidden layer nodes) would behave identically and we would never distinguish them.

Using random weights corresponds to starting with random features, which allows the SGD to make progress (improve the individual features).

Regularization Using Dropout



(a) Standard Neural Net



(b) After applying dropout.

Srivastava et al. (2014), Dropout: A Simple Way to Prevent Neural Networks from Overfitting, Fig. 1

- A trick by Srivastava et al., 2014 -- randomly zero-out neurons during training, typically 10-50%
- Decreases the efficient capacity during training, keeps the capacity at test time
- Intuition: more robust features because the information needs to survive damaging a part of the network

Universal Approximation Theorem

Universal Approximation Theorem '89

Let $\varphi(x): \mathbb{R} \rightarrow \mathbb{R}$ be a nonconstant, bounded and nondecreasing continuous function.
(Later a proof was given also for $\varphi = \mathrm{ReLU}$ and even for any nonpolynomial function.)

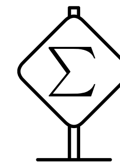
For any $\varepsilon > 0$ and any continuous function $f: [0,1]^D \rightarrow \mathbb{R}$, there exists $H \in \mathbb{N}$, $\boldsymbol{v} \in \mathbb{R}^H$, $\boldsymbol{b} \in \mathbb{R}^H$ and $\boldsymbol{W} \in \mathbb{R}^{D \times H}$, such that if we denote

$$F(\boldsymbol{x}) = \boldsymbol{v}^T \varphi(\boldsymbol{x}^T \boldsymbol{W} + \boldsymbol{b}) = \sum_{i=1}^{H} v_i \varphi(\boldsymbol{x}^T \boldsymbol{W}_{*,i} + b_i),$$

where φ is applied elementwise, then for all $\boldsymbol{x} \in [0,1]^D$:

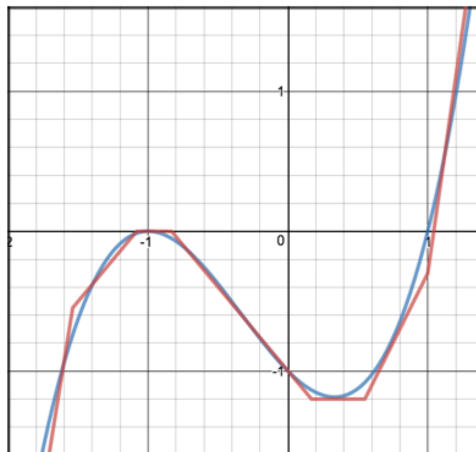
$$|F(\boldsymbol{x}) - f(\boldsymbol{x})| < \varepsilon.$$

Universal Approximation Theorem for ReLUs



Sketch of the proof:

- If a function is continuous on a closed interval, it can be approximated by a sequence of lines to arbitrary precision.



$$n_1(x) = Relu(-5x - 7.7)$$
$$n_2(x) = Relu(-1.2x - 1.3)$$
$$n_3(x) = Relu(1.2x + 1)$$
$$n_4(x) = Relu(1.2x - .2)$$
$$n_5(x) = Relu(2x - 1.1)$$
$$n_6(x) = Relu(5x - 5)$$

$$Z(x) = -n_1(x) - n_2(x) - n_3(x) + n_4(x) + n_5(x) + n_6(x)$$

- However, we can create a sequence of k linear segments as a sum of k ReLU units – on every endpoint a new ReLU starts (i.e., the input ReLU value is zero at the endpoint), with a tangent which is the difference between the target tangent and the tangent of the approximation until this point.

Where Are We

We have seen the gradual development of machine learning systems to neural networks.

- linear regression → Perceptron → (multinomial) logistic regression → MLP

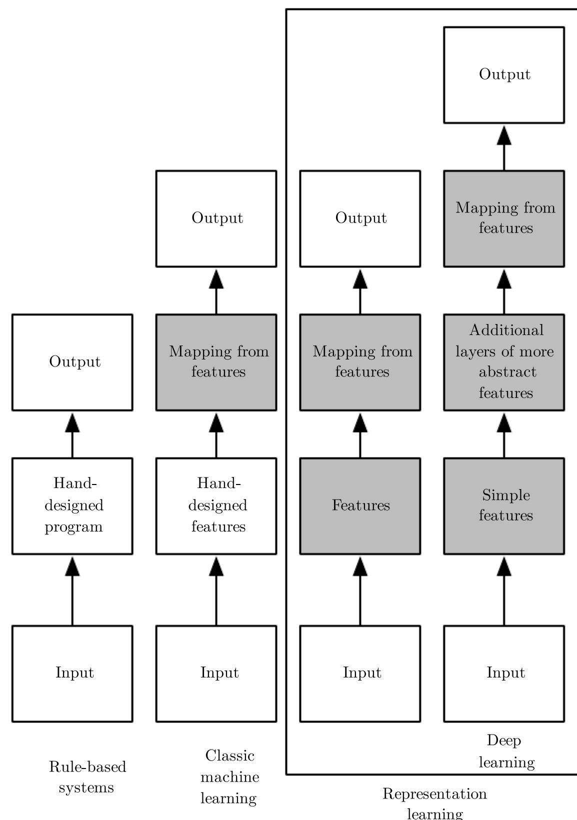


Figure 1.5 of "Deep Learning" book, https://www.deeplearningbook.org.

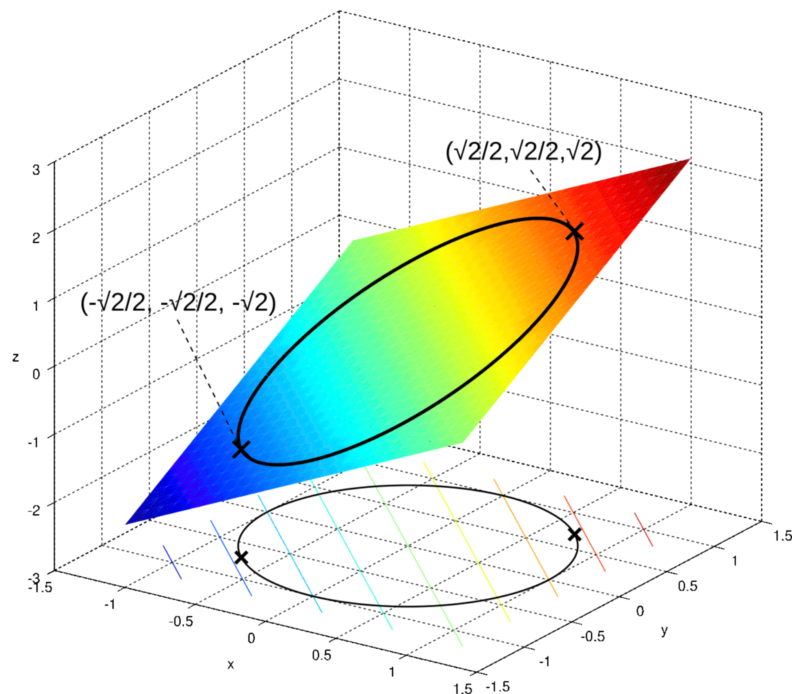
https://imgs.xkcd.com/comics/machine_learning_2x.png

Lagrange Multipliers

Constrained Optimization

Given a function $f(\boldsymbol{x})$, we can find a minimum/maximum with respect to a vector $\boldsymbol{x} \in \mathbb{R}^D$, by investigating the critical points $\nabla_{\boldsymbol{x}} f(\boldsymbol{x}) = 0$.

Consider now finding a minimum subject to a constraint $g(\boldsymbol{x}) = 0$.



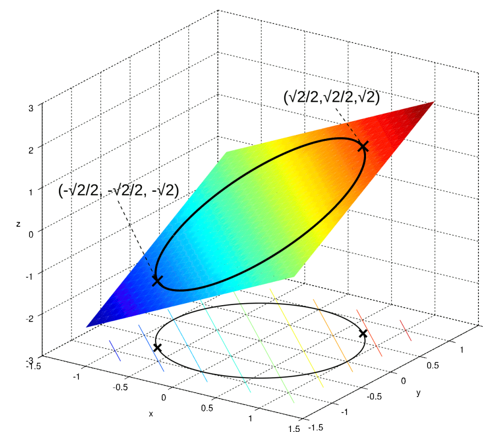
On the left, there is an example with $f(x, y) = x + y$ and the constraint $x^2 + y^2 = 1$, which can be represented as $g(x, y) = x^2 + y^2 - 1$.

https://upload.wikimedia.org/wikipedia/commons/e/ed/Lagrange_very_simple.svg

Let $f(\boldsymbol{x}): \mathbb{R}^D \rightarrow \mathbb{R}$ be a function. We seek its minimum subject to an equality constraint $g(\boldsymbol{x}) = 0$ for $g(\boldsymbol{x}): \mathbb{R}^D \rightarrow \mathbb{R}$.

- Note that $\nabla_{\boldsymbol{x}} g(\boldsymbol{x})$ is orthogonal to the surface of the constraint, because if $\boldsymbol{x}$ and a nearby point $\boldsymbol{x} + \boldsymbol{\varepsilon}$ lie on the surface, from the Taylor expansion $g(\boldsymbol{x} + \boldsymbol{\varepsilon}) \approx g(\boldsymbol{x}) + \boldsymbol{\varepsilon}^T \nabla_{\boldsymbol{x}} g(\boldsymbol{x})$ we get $\boldsymbol{\varepsilon}^T \nabla_{\boldsymbol{x}} g(\boldsymbol{x}) \approx 0$.
- For the desired minimum, $\nabla_{\boldsymbol{x}} f(\boldsymbol{x})$ must also be orthogonal to the constraint surface (or else moving in the direction of the derivative would increase the value).
- Therefore, there must exist λ such that $\nabla_{\boldsymbol{x}} f = \lambda \nabla_{\boldsymbol{x}} g$.

Consequently, the sought minimum either fulfills $\nabla_{\boldsymbol{x}} f - \lambda \nabla_{\boldsymbol{x}} g = 0$ for some λ, or it is an unconstrained minimum – in that case, the equation also holds with $\lambda = 0$.



https://upload.wikimedia.org/wikipedia/commons/e/ed/L

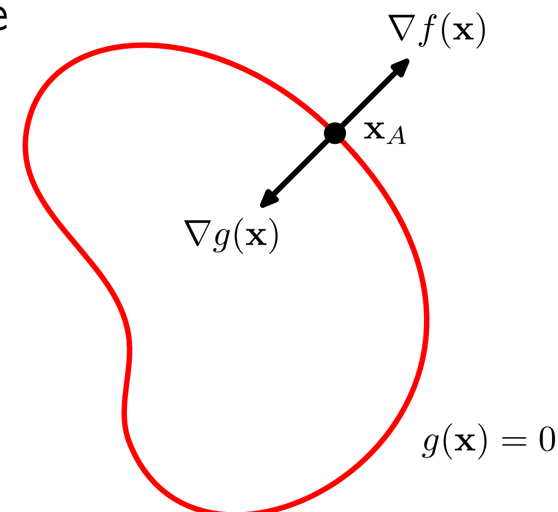


Figure E.1 of Pattern Recognition and Machine Learning.

Let $f(\boldsymbol{x}) : \mathbb{R}^D \to \mathbb{R}$ be a function that has a minimum (or a maximum) in $\boldsymbol{x}$ subject to equality constraint $g(\boldsymbol{x}) = 0$. Assume that both f and g have continuous partial derivatives and that $\nabla_{\boldsymbol{x}} g(\boldsymbol{x}) \neq 0$.

Then there exists a $\lambda \in \mathbb{R}$, such that the **Lagrangian function**

$$\mathcal{L}(\boldsymbol{x}, \lambda) \stackrel{\text{def}}{=} f(\boldsymbol{x}) - \lambda g(\boldsymbol{x})$$

has a zero gradient in both $\boldsymbol{x}$ and λ.

In detail,

- $\frac{\partial \mathcal{L}}{\partial \lambda} = 0$ leads to $g(\boldsymbol{x}) = 0$;
- $\nabla_{\boldsymbol{x}} \mathcal{L} = 0$ is the previously derived $\nabla_{\boldsymbol{x}} f - \lambda \nabla_{\boldsymbol{x}} g = 0$.

Minimization – Multiple Equality Constraints

ÚFAL

We can use induction if there are multiple equality constraints, resulting in the following generalization.

Let $f(\boldsymbol{x}) : \mathbb{R}^D \rightarrow \mathbb{R}$ be a function that has a minimum (or a maximum) in $\boldsymbol{x}$ subject to equality constraints $g_1(\boldsymbol{x}) = 0, \ldots, g_m(\boldsymbol{x}) = 0$. Assume that $f, g_1, \ldots, g_m$ have continuous partial derivatives and that the gradients $\nabla_{\boldsymbol{x}} g_1(\boldsymbol{x}), \ldots, \nabla_{\boldsymbol{x}} g_m(\boldsymbol{x})$ are linearly independent.

Then there exist $\lambda_1 \in \mathbb{R}, \ldots, \lambda_m \in \mathbb{R}$, such that the **Lagrangian function**

$$\mathcal{L}(\boldsymbol{x}, \boldsymbol{\lambda}) \stackrel{\text{def}}{=} f(\boldsymbol{x}) - \sum_{i=1}^{m} \lambda_i g_i(\boldsymbol{x})$$

has a zero gradient in both $\boldsymbol{x}$ and $\boldsymbol{\lambda}$.

This strategy of finding constrained minima is known as the **method of Lagrange multipliers**.

Example of Minimization with Equality Constraint

Assume we want to find a categorical distribution $\boldsymbol{p} = (p_1, \ldots, p_n)$ with maximum entropy.

Then we want to minimize $-H(\boldsymbol{p})$ under the constraints

- $p_i \geq 0$ for all i,
- $\sum_{i=1}^n p_i = 1$.

Ignoring the first constraint for the time being, we form a Lagrangian

$$\mathcal{L} = \left(\sum_i p_i \log p_i\right) - \lambda\left(\sum_i p_i - 1\right).$$

Computing the derivative with respect to p_i and setting it equal to zero, we get

$$0 = \frac{\partial \mathcal{L}}{\partial p_i} = 1 \cdot \log(p_i) + p_i \cdot \frac{1}{p_i} - \lambda = \log(p_i) + 1 - \lambda.$$

Therefore, all $p_i = e^{\lambda - 1}$ must be the same, and the constraint $\sum_{i=1}^n p_i = 1$ yields $p_i = \frac{1}{n}$.

Softmax as Maximum Entropy Classifier

Derivation of Softmax using Maximum Entropy

Let $\mathbb{X} = \{(\boldsymbol{x}_1, t_1), (\boldsymbol{x}_2, t_2), \ldots, (\boldsymbol{x}_N, t_N)\}$ be training data of a K-class classification, with $\boldsymbol{x}_i \in \mathbb{R}^D$ and $t_i \in \{1, 2, \ldots, K\}$.

We want to model it using a function $\pi : \mathbb{R}^D \to \mathbb{R}^K$ so that $\pi(\boldsymbol{x})$ gives a distribution of classes for input $\boldsymbol{x}$.

We impose the following conditions on π:

- $$\text{for } 1 \leq k \leq K: \; \pi(\boldsymbol{x})_k \geq 0,$$
- $$\sum_{k=1}^{K} \pi(\boldsymbol{x})_k = 1,$$
- $$\text{for } 1 \leq k \leq K: \; \sum_{i=1}^{N} \pi(\boldsymbol{x}_i)_k \boldsymbol{x}_i = \sum_{i=1}^{N} \Big[t_i = k\Big] \boldsymbol{x}_i.$$

Derivation of Softmax using Maximum Entropy

There are many such π, one particularly bad is

$$\pi(\boldsymbol{x}) = \begin{cases} \mathbf{1}_{t_i} & \text{if there exists } i : \boldsymbol{x}_i = \boldsymbol{x}, \\ \mathbf{1}_1 & \text{otherwise,} \end{cases}$$

where $\mathbf{1}_i$ is a one-hot encoding of i (vector of zeros, except for position i, which is equal to 1).

Therefore, we want to find a more **general** π – consequently, we turn to the principle of maximum entropy and search for π with maximum entropy.

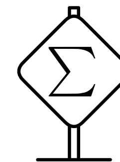
Derivation of Softmax using Maximum Entropy

We want to minimize $-\sum_{i=1}^N H(\pi(\boldsymbol{x}_i))$ given

- for $1 \le i \le N$, $1 \le k \le K$: $\pi(\boldsymbol{x}_i)_k \ge 0$,
- for $1 \le i \le N$: $\sum_{k=1}^K \pi(\boldsymbol{x}_i)_k = 1$,
- for $1 \le j \le D$, $1 \le k \le K$: $\sum_{i=1}^N \pi(\boldsymbol{x}_i)_k x_{i,j} = \sum_{i=1}^N \big[t_i = k\big] x_{i,j}$.

We therefore form a Lagrangian (ignoring the first inequality constraint):

$$\begin{aligned}\mathcal{L} = & \sum_{i=1}^N \sum_{k=1}^K \pi(\boldsymbol{x}_i)_k \log(\pi(\boldsymbol{x}_i)_k) \\ & - \sum_{j=1}^D \sum_{k=1}^K \lambda_{j,k} \Big(\sum_{i=1}^N \pi(\boldsymbol{x}_i)_k x_{i,j} - \big[t_i = k\big] x_{i,j} \Big) \\ & - \sum_{i=1}^N \beta_i \Big(\sum_{k=1}^K \pi(\boldsymbol{x}_i)_k - 1 \Big).\end{aligned}$$



We now compute partial derivatives of the Lagrangian, notably the values

$$\frac{\partial}{\partial \pi(\boldsymbol{x}_i)_k} \mathcal{L}.$$

We arrive at

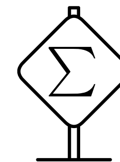
$$\frac{\partial}{\partial \pi(\boldsymbol{x}_i)_k} \mathcal{L} = \log(\pi(\boldsymbol{x}_i)_k) + 1 - \boldsymbol{x}_i^T \boldsymbol{\lambda}_{*,k} - \beta_i.$$

Setting the derivative of the Lagrangian to zero, we obtain

$$\pi(\boldsymbol{x}_i)_k = e^{\boldsymbol{x}_i^T \boldsymbol{\lambda}_{*,k} + \beta_i - 1}.$$

Such a form guarantees $\pi(\boldsymbol{x}_i)_k > 0$, which we did not include in the conditions.

Derivation of Softmax using Maximum Entropy



In order to find out the β_i values, we turn to the constraint

$$\sum_k \pi(\boldsymbol{x}_i)_k = \sum_k e^{\boldsymbol{x}_i^T \boldsymbol{\lambda}_{*,k} + \beta_i - 1} = 1,$$

from which we get

$$e^{\beta_i} = \frac{1}{\sum_k e^{\boldsymbol{x}_i^T \boldsymbol{\lambda}_{*,k} - 1}},$$

yielding

$$\pi(\boldsymbol{x}_i)_k = e^{\boldsymbol{x}_i^T \boldsymbol{\lambda}_{*,k} + \beta_i - 1} = \frac{e^{\boldsymbol{x}_i^T \boldsymbol{\lambda}_{*,k}}}{\sum_{k'} e^{\boldsymbol{x}_i^T \boldsymbol{\lambda}_{*,k'}}} = \operatorname{softmax}(\boldsymbol{x}_i^T \boldsymbol{\lambda})_k.$$

F-Score

When evaluating binary classification, we have used **accuracy** so far.

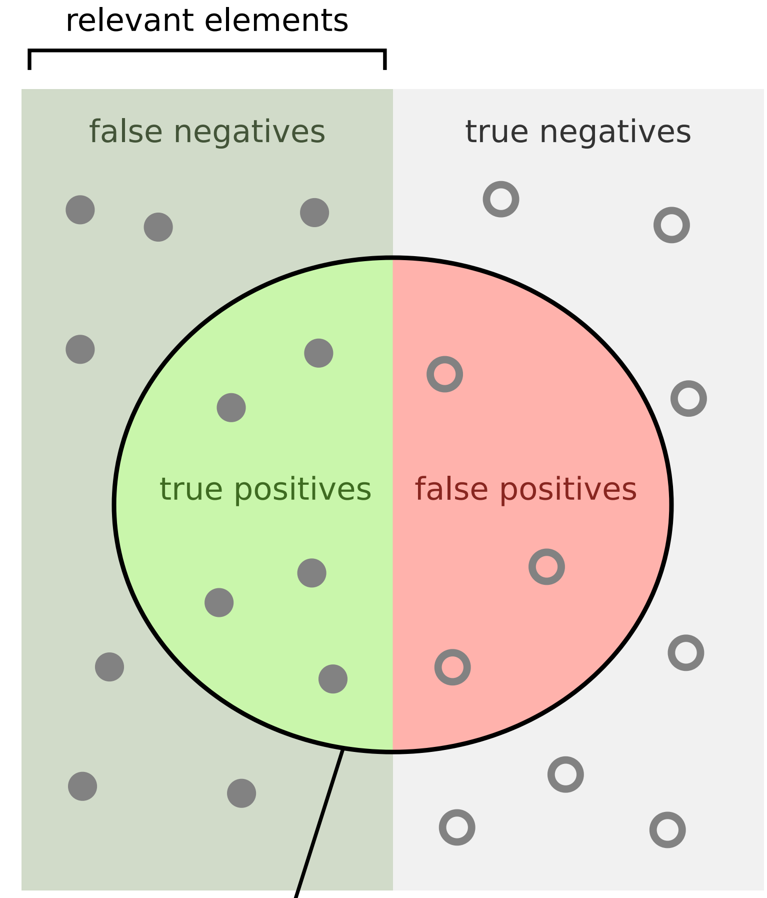
However, there are other metrics we might want to consider. One of them is F_1-score.

Consider the following **confusion matrix**:

	Target positive	**Target negative**
Predicted positive	**True Positive (TP)**	**False Positive (FP)**
Predicted negative	**False Negative (FN)**	**True Negative (TN)**

Accuracy can be computed as

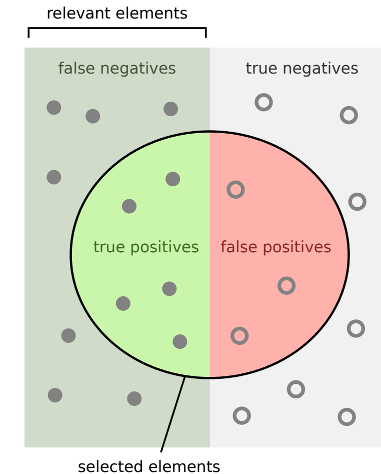
$$\text{accuracy} = \frac{TP + TN}{TP + TN + FP + FN}.$$



https://upload.wikimedia.org/wikipedia/commons/2/26/Precisionrecall.svg

F_1-score

	Target positive	**Target negative**
Predicted positive	**True Positive (TP)**	**False Positive (FP)**
Predicted negative	**False Negative (FN)**	**True Negative (TN)**



https://upload.wikimedia.org/wikipedia/commons/2/26/Precisionrecall.svg

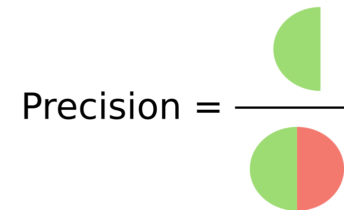
In some cases, we are mostly interested in positive examples.

We define **precision** (percentage of correct predictions in predicted examples) and **recall** (percentage of correct predictions in the gold examples) as

$$
\begin{aligned}
\text{precision} &= \frac{TP}{TP + FP}, \\
\text{recall} &= \frac{TP}{TP + FN}.
\end{aligned}
$$

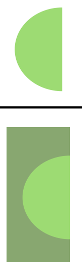
How many selected items are relevant?

Precision =



How many relevant items are selected?

Recall =



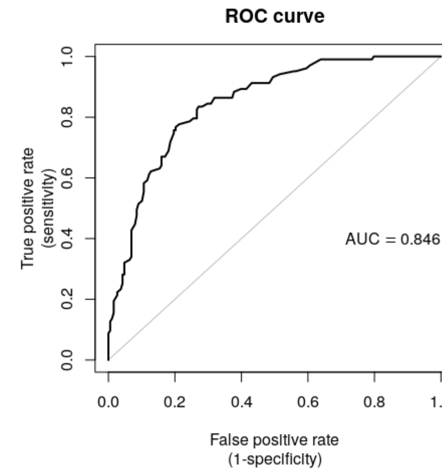
https://upload.wikimedia.org/wikipedia/commons/2/26/Precisionrecall.svg

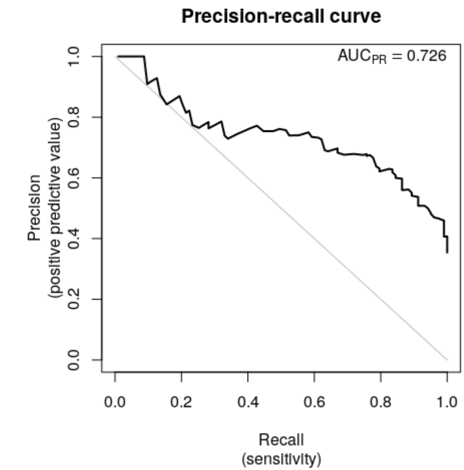
F_1-score

The precision and recall go "against each other": decreasing the classifier threshold usually increases recall and decreases precision, and vice versa.

We therefore define a single $\boldsymbol{F_1}$**-score** as a harmonic mean of precision and recall:

$$F_1 = \frac{2}{\text{precision}^{-1} + \text{recall}^{-1}} = \frac{2 \cdot \text{precision} \cdot \text{recall}}{\text{precision} + \text{recall}} = \frac{TP \quad + \quad TP}{TP + FP + TP + FN}.$$





https://modtools.files.wordpress.com/2020/01/roc_pr-1.png

Arithmetic mean of precision&recall is

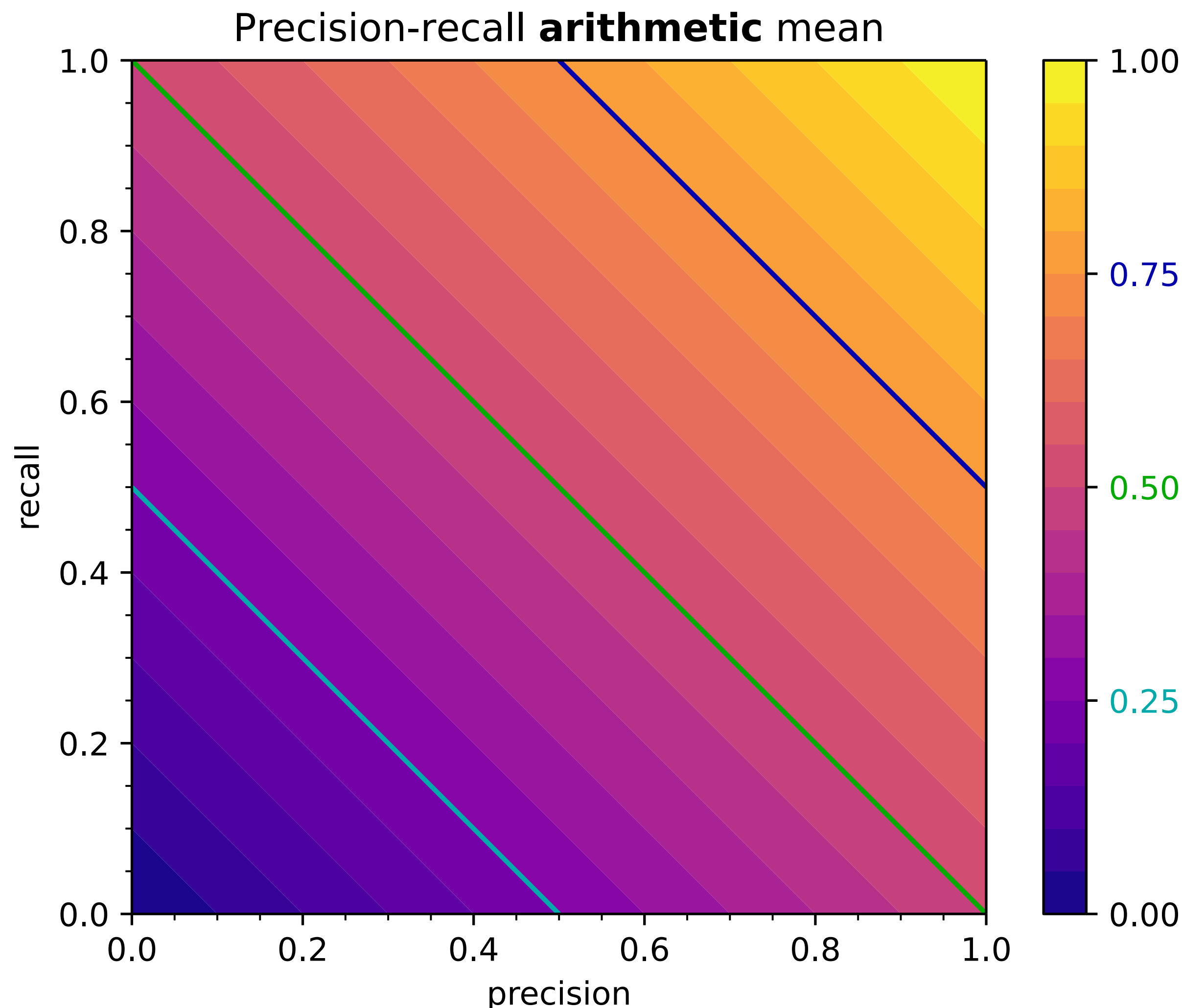
$$AM(p, r) \stackrel{\text{def}}{=} \frac{p + r}{2}.$$

As any mean, it is "between" the input values

$$\min(p, r) \le AM(p, r),$$
$$AM(p, r) \le \max(p, r).$$

However,

$$AM(1\%, 100\%) = 50.5\%.$$

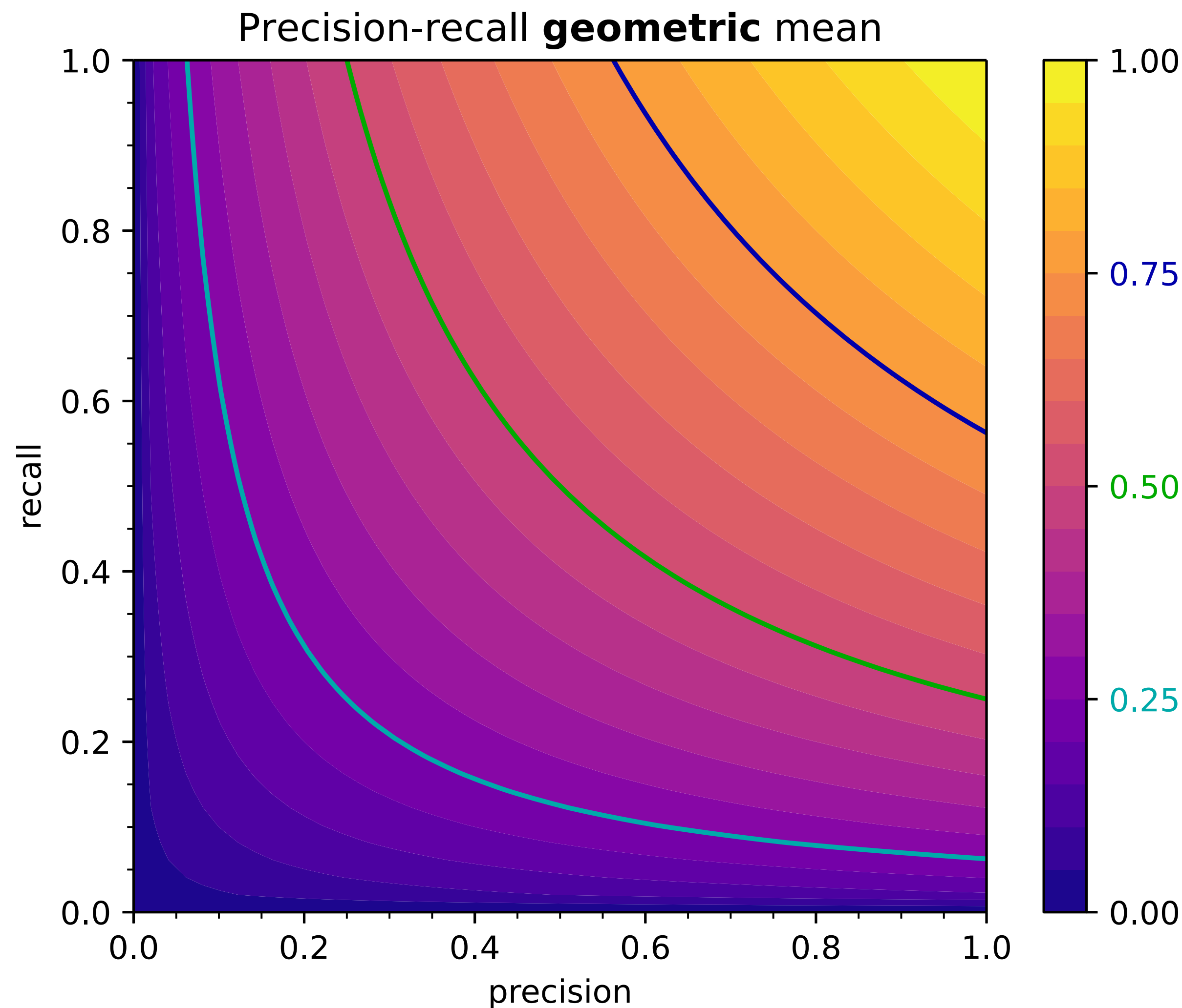


Geometric mean of precision&recall is

$$GM(p, r) \stackrel{\text{def}}{=} \sqrt{p \cdot r}.$$

It is better than the arithmetic mean, but still

$$GM(1\%, 100\%) = 10\%.$$



F_1-score and Other Means of Precision and Recall

Harmonic mean of precision&recall is

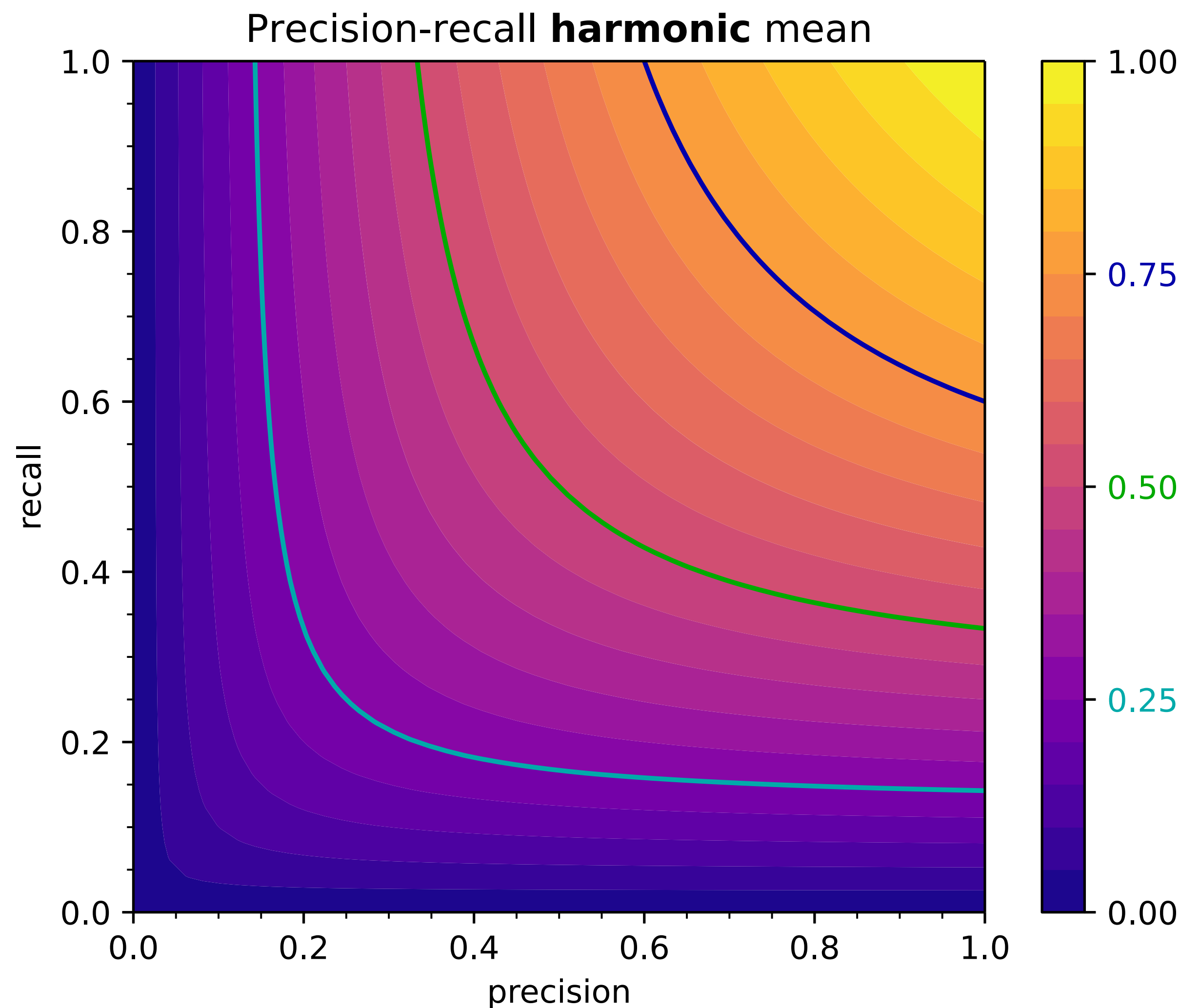
$$HM(p, r) \stackrel{\text{def}}{=} \frac{2}{\frac{1}{p} + \frac{1}{r}}.$$

In addition to being bounded by the input values, HM is also dominated by the minimum of its input values:

$$\min(p, r) \leq HM(p, r),$$
$$HM(p, r) \leq \max(p, r),$$
$$HM(p, r) \leq 2 \min(p, r).$$

For example,

$$HM(1\%, 100\%) \approx 2\%.$$



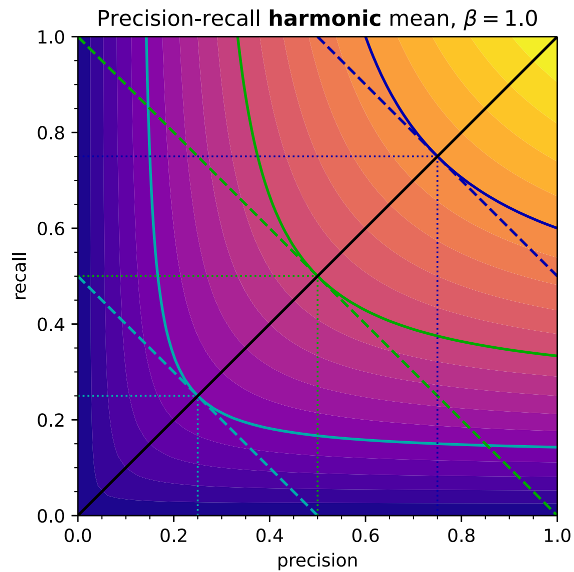
General F_β-score

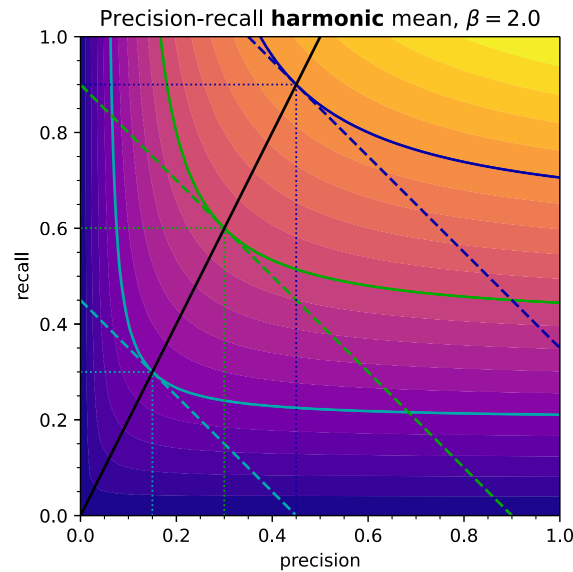
The F_1 score can be generalized to F_β score, which can be used as a metric when recall is β times more important than precision; F_2 favoring recall and $F_{0.5}$ favoring precision are commonly used.

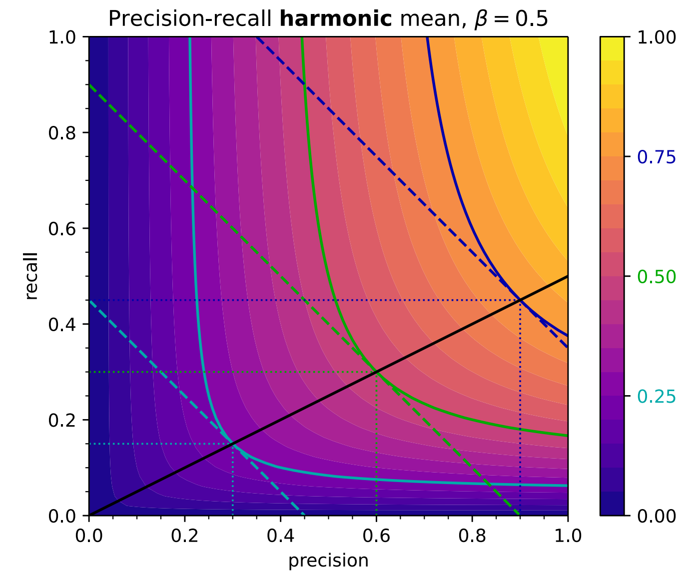
The formula for F_β is

$$\begin{aligned}
F_\beta &= \frac{1+\beta^2}{\mathrm{precision}^{-1} + \beta^2 \cdot \mathrm{recall}^{-1}} \\
&= \frac{(1+\beta^2) \cdot \mathrm{precision} \cdot \mathrm{recall}}{\beta^2 \cdot \mathrm{precision} + \mathrm{recall}} \\
&= \frac{\mathit{TP} \quad + \quad \beta^2 \cdot \mathit{TP}}{\mathit{TP} + \mathit{FP} + \beta^2 \cdot (\mathit{TP} + \mathit{FN})}.
\end{aligned}$$

F_1-score and Other Means of Precision and Recall



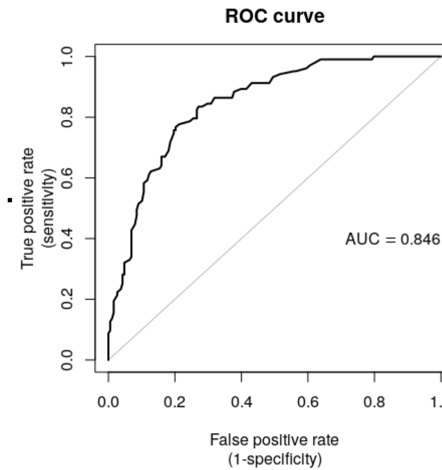


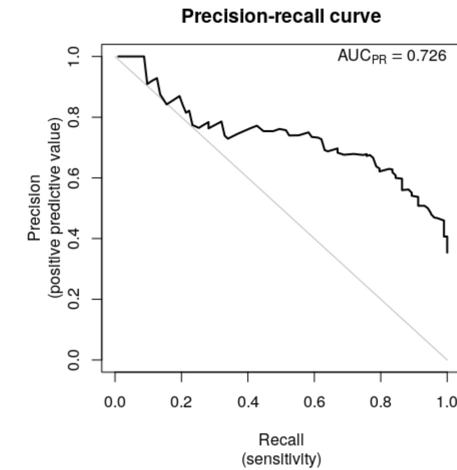


Precision-Recall Curve

Changing the threshold in logistic regression allows us to trade off precision for recall, and vice versa. Therefore, we can tune it on the development set to achieve the highest possible F_1 score, if required.

Also, if we want to evaluate F_1-score without considering a specific threshold, the **area under curve** (AUC) is sometimes used as a metric.





https://modtools.files.wordpress.com/2020/01/roc_pr-1.png

F_1-Score in Multiclass Classification

To extend F_1-score to multiclass classification, we expect one of the classes to be *negative* and the others to be *different kinds of positive*. For each of the positive classes, we compute the same confusion matrix as in the binary case (considering all other labels as negative ones), and then combine the results in one of the following ways:

- **micro-averaged** F_1 (or just **micro** F_1): we first sum all the TP, FP and FN of the individual binary classifications and compute the final F_1-score (this way, the frequency of the individual classes is taken into account);
- **macro-averaged** F_1 (or just **macro** F_1): we first compute the F_1-scores of the individual binary classifications and then compute an unweighted average (therefore, the frequency of the classes is more or less ignored).

Binary Confusion Metric Measures Overview

	Target positive	**Target negative**	
Predicted positive	**True Positive (TP)**	**False Positive (FP)** Type I Error	**precision** $\frac{TP}{TP+FP}$
Predicted negative	**False Negative (FN)** Type II Error	**True Negative (TN)**	
	true positive rate, recall, sensitivity $\frac{TP}{TP+FN}$	**false positive rate** $\frac{FP}{FP+TN}$ **specificity** $\frac{TN}{TN+FP}$	

- F_1-score $= \frac{2 \cdot \text{precision} \cdot \text{recall}}{\text{precision} + \text{recall}} = \frac{TP \quad + \quad TP}{TP+FP+TP+FN}$
- accuracy $= \frac{TP+TN}{TP+FP+FN+TN}$

Today's Lecture Objectives

After this lecture you should be able to

- Implement training of multi-layer perceptron using SGD
- Explain the theoretical foundation behind the softmax activation function (including the necessary math)
- Choose a suitable evaluation metric for various classification tasks