Language Technologies Institute

Carnegie Mellon University

Algorithms for NLP

CS 11-711 · Fall 2020

Lecture 3: Nonlinear text classification

Emma Strubell

Announcements

- **Project 1: Text classification** will be available after class today, **due Friday September 25**
- Han will lead recitation this Friday to introduce Project 1, and cover environment setup (Python, Jupyter, NumPy, PyTorch).

Recap

Representing text as a bag of words

- Given a text $\boldsymbol{w} = (w_1, w_2, \ldots, w_T) \in \mathcal{V}^*$
- Choose a label $y \in \mathcal{Y}$
- The **bag-of-words** is a fixed-length vector of word counts:

$\boldsymbol{w}$ = The drinks were strong but the fish tacos were bland

$\boldsymbol{x}$ =

0	…	1	1	…	1	…	0	1	2	1	…	2	…	0
aardvark	…	bland	but	…	fish	…	taco	tacos	the	strong	…	were	…	zyther

- Length of $\boldsymbol{x}$ is equal to the size of the vocabulary, V

Recap

Linear classification on bag-of-words

- Let $\psi(\boldsymbol{x}, y)$ score the compatibility of bag-of-words $\boldsymbol{x}$ and label y. Then:

$$\hat{y} = \underset{y}{\text{argmax}}\, \psi(\boldsymbol{x}, y).$$

- In a **linear classifier** this scoring function has the simple form:

$$\psi(\boldsymbol{x}, y) = \boldsymbol{\theta} \cdot \boldsymbol{f}(\boldsymbol{x}, y) = \sum_{j=1} \theta_j \times f_j(\boldsymbol{x}, y),$$

where $\boldsymbol{\theta}$ is a vector of weights, and $\boldsymbol{f}$ is a **feature function**

Recap

Feature functions

- In classification, the feature function is usually a simple combination of $\boldsymbol{x}$ and y, such as:

$$f_j(\boldsymbol{x}, y) = \begin{cases} x_{\text{bland}} & \text{if } y = \text{negative} \\ 0 & \text{otherwise} \end{cases}$$

- If we have K labels, this corresponds to column vectors that look like:

$\boldsymbol{f}(\boldsymbol{x}, y = 1) =$

x_0	x_1	...	$x_{\lvert V\rvert}$	0	0	0	0	0	0	0	0	0	...	0

T

$(K-1) \times V$

$\boldsymbol{f}(\boldsymbol{x}, y = 2) =$

0	0	0	...	0	x_0	x_1	...	$x_{\lvert V\rvert}$	0	0	0	0	...	0

T

V $\quad$ $(K-2) \times V$

$\boldsymbol{f}(\boldsymbol{x}, y = K) =$

0	0	0	0	0	0	0	0	0	...	0	x_0	x_1	...	$x_{\lvert V\rvert}$

T

$(K-1) \times V$

How to obtain θ ?

How to obtain θ ?

- The **learning** problem is to find the right weights θ.

How to obtain θ ?

- The **learning** problem is to find the right weights θ.
- Naïve Bayes: set θ equal to the empirical frequencies:

$$\hat{\mu}_y = p(y) = \frac{\text{count}(y)}{\sum_{y'} \text{count}(y')} \qquad \hat{\phi}_{y,j} = p(\mathbf{x}_j \mid y) = \frac{\text{count}(y, j)}{\sum_{j'=1}^{V} \text{count}(y, j')}$$

How to obtain θ ?

- The **learning** problem is to find the right weights θ.
- Naïve Bayes: set θ equal to the empirical frequencies:

$$\hat{\mu}_y = p(y) = \frac{\text{count}(y)}{\sum_{y'} \text{count}(y')} \qquad \hat{\phi}_{y,j} = p(\mathbf{x}_j \mid y) = \frac{\text{count}(y, j)}{\sum_{j'=1}^{V} \text{count}(y, j')}$$

- Perceptron update:

$$\boldsymbol{\theta}^{(t+1)} \leftarrow \boldsymbol{\theta}^{(t)} + \boldsymbol{f}(\boldsymbol{x}^{(i)}, y^{(i)}) - \boldsymbol{f}(\boldsymbol{x}^{(i)}, \hat{y})$$

How to obtain θ ?

- The **learning** problem is to find the right weights θ.
- Naïve Bayes: set θ equal to the empirical frequencies:

$$\hat{\mu}_y = p(y) = \frac{\text{count}(y)}{\sum_{y'} \text{count}(y')} \qquad \hat{\phi}_{y,j} = p(\mathbf{x}_j \mid y) = \frac{\text{count}(y, j)}{\sum_{j'=1}^{V} \text{count}(y, j')}$$

- Perceptron update:

$$\boldsymbol{\theta}^{(t+1)} \leftarrow \boldsymbol{\theta}^{(t)} + \boldsymbol{f}(\boldsymbol{x}^{(i)}, y^{(i)}) - \boldsymbol{f}(\boldsymbol{x}^{(i)}, \hat{y})$$

- Large-margin update:

$$\boldsymbol{\theta}^{(t+1)} \leftarrow \boldsymbol{\theta}^{(t)} + \mathbf{f}(\mathbf{x}^{(i)}, y^{(i)}) - \mathbf{f}(\mathbf{x}^{(i)}, \hat{y}) \quad \text{with} \quad \hat{y} = \underset{y \in \mathcal{Y}}{\text{argmax}}\, \boldsymbol{\theta} \cdot \mathbf{f}(\mathbf{x}^{(i)}, y) + c(y^{(i)}, y)$$

How to obtain θ ?

- The **learning** problem is to find the right weights θ.
- Naïve Bayes: set θ equal to the empirical frequencies:

$$\hat{\mu}_y = p(y) = \frac{\text{count}(y)}{\sum_{y'} \text{count}(y')} \qquad \hat{\phi}_{y,j} = p(\mathbf{x}_j \mid y) = \frac{\text{count}(y, j)}{\sum_{j'=1}^{V} \text{count}(y, j')}$$

- Perceptron update:

$$\boldsymbol{\theta}^{(t+1)} \leftarrow \boldsymbol{\theta}^{(t)} + \boldsymbol{f}(\boldsymbol{x}^{(i)}, y^{(i)}) - \boldsymbol{f}(\boldsymbol{x}^{(i)}, \hat{y})$$

- Large-margin update:

$$\boldsymbol{\theta}^{(t+1)} \leftarrow \boldsymbol{\theta}^{(t)} + \mathbf{f}(\mathbf{x}^{(i)}, y^{(i)}) - \mathbf{f}(\mathbf{x}^{(i)}, \hat{y}) \quad \text{with} \quad \hat{y} = \underset{y \in \mathcal{Y}}{\text{argmax}}\, \boldsymbol{\theta} \cdot \mathbf{f}(\mathbf{x}^{(i)}, y) + c(y^{(i)}, y)$$

- Logistic regression update:

$$\boldsymbol{\theta}^{(t+1)} \leftarrow \boldsymbol{\theta}^{(t)} + \mathbf{f}(\mathbf{x}^{(i)}, y^{(i)}) - E_{y|\mathbf{x}}\left[\mathbf{f}(\mathbf{x}^{(i)}, y)\right]$$

How to obtain θ ?

- The **learning** problem is to find the right weights θ.
- Naïve Bayes: set θ equal to the empirical frequencies:

$$\hat{\mu}_y = p(y) = \frac{\text{count}(y)}{\sum_{y'} \text{count}(y')} \qquad \hat{\phi}_{y,j} = p(\mathbf{x}_j \mid y) = \frac{\text{count}(y,j)}{\sum_{j'=1}^{V} \text{count}(y,j')}$$

- Perceptron update:

$$\boldsymbol{\theta}^{(t+1)} \leftarrow \boldsymbol{\theta}^{(t)} + \boldsymbol{f}(\boldsymbol{x}^{(i)}, y^{(i)}) - \boldsymbol{f}(\boldsymbol{x}^{(i)}, \hat{y})$$

- Large-margin update:

$$\boldsymbol{\theta}^{(t+1)} \leftarrow \boldsymbol{\theta}^{(t)} + \mathbf{f}(\mathbf{x}^{(i)}, y^{(i)}) - \mathbf{f}(\mathbf{x}^{(i)}, \hat{y}) \quad \text{with} \quad \hat{y} = \underset{y \in \mathcal{Y}}{\text{argmax}}\, \boldsymbol{\theta} \cdot \mathbf{f}(\mathbf{x}^{(i)}, y) + c(y^{(i)}, y)$$

- Logistic regression update:

$$\boldsymbol{\theta}^{(t+1)} \leftarrow \boldsymbol{\theta}^{(t)} + \mathbf{f}(\mathbf{x}^{(i)}, y^{(i)}) - E_{y|\mathbf{x}}\left[\mathbf{f}(\mathbf{x}^{(i)}, y)\right]$$

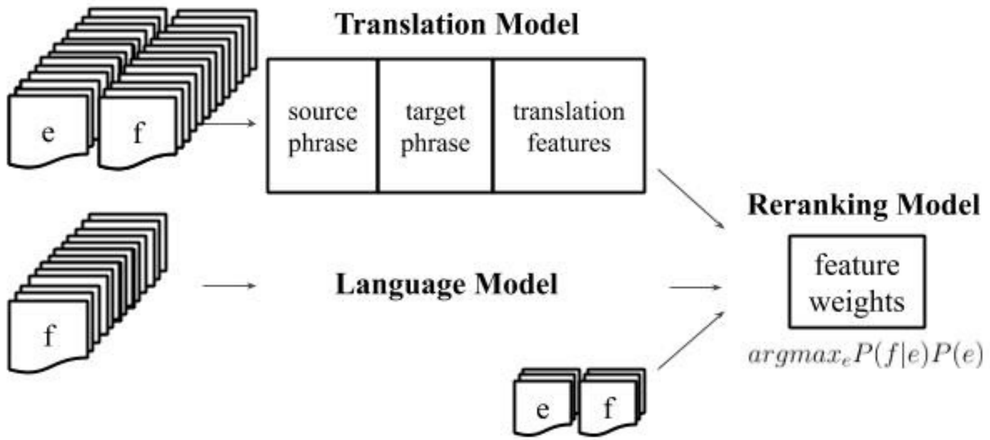
- All these methods for **supervised learning** assume a labeled dataset of N examples:

$$\{(\boldsymbol{x}^{(i)}, y^{(i)})\}_{i=1}^{N}$$

Today

Nonlinear classification & evaluating classifiers

Engineered features

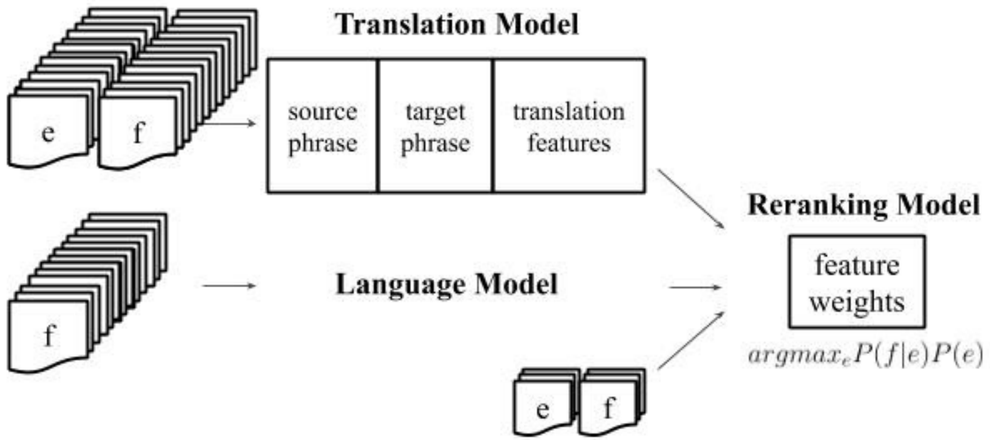


Today

Nonlinear classification & evaluating classifiers

Engineered features

linear classification

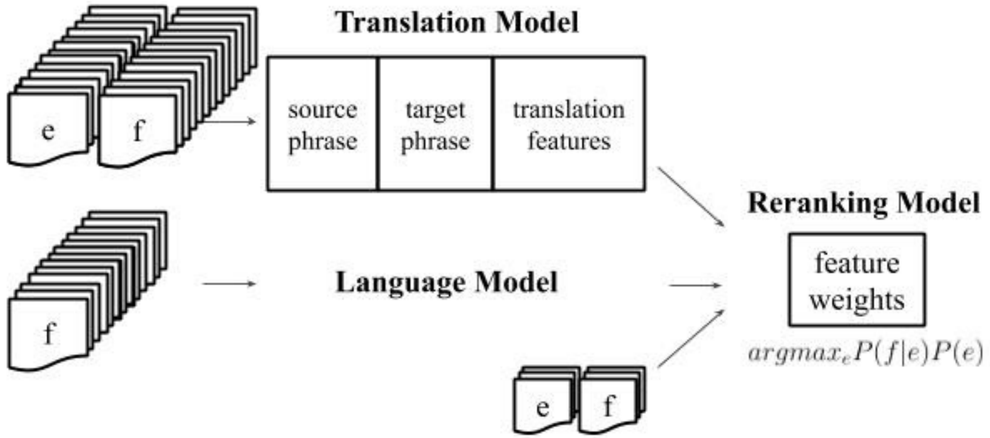


Today

Nonlinear classification & evaluating classifiers

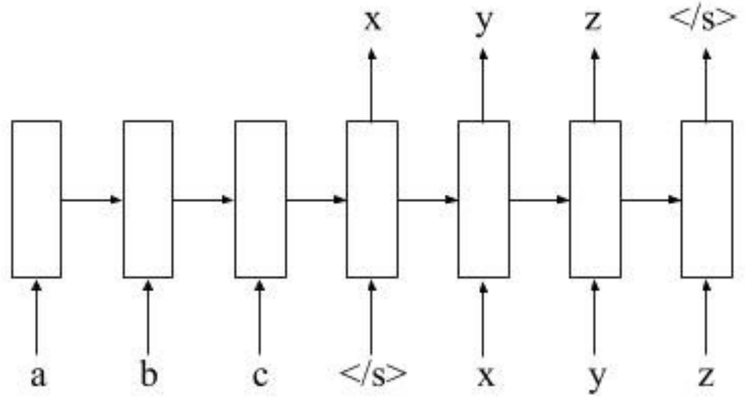
Engineered features

linear classification



~mid 2010s

Learned features

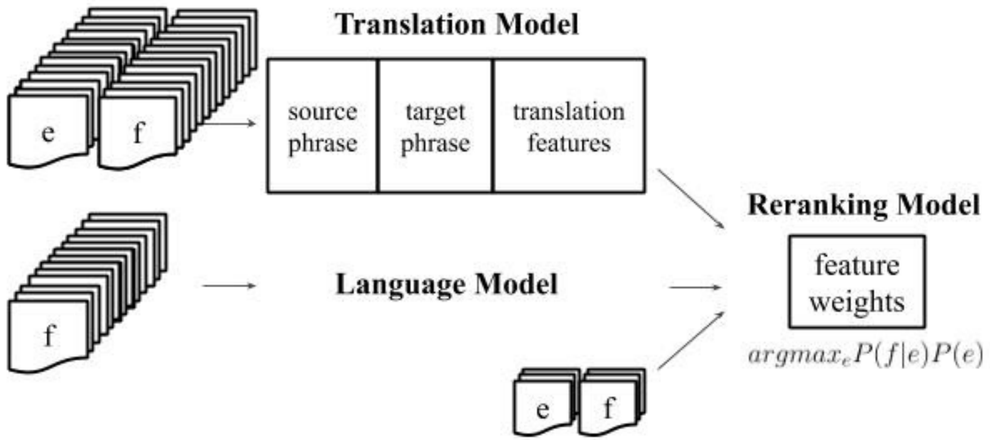


Today

Nonlinear classification & evaluating classifiers

Engineered features

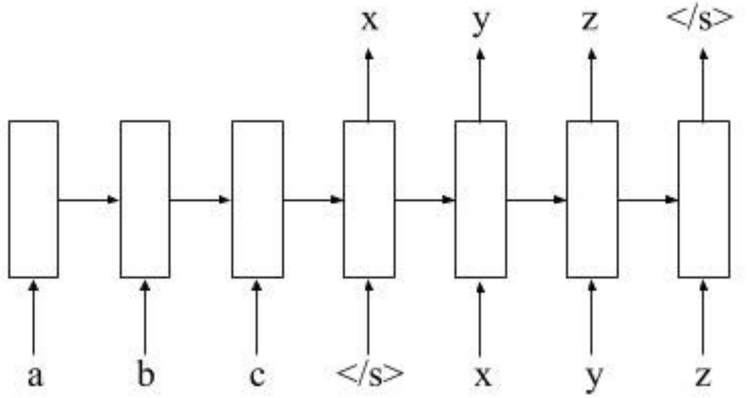
linear classification



~mid 2010s

Learned features

nonlinear classification



A simple feed-forward architecture

A simple feed-forward architecture

- Suppose we want to label stories as $\mathcal{Y} = \{\textsc{Good}, \textsc{Bad}, \textsc{Okay}\}$

A simple feed-forward architecture

- Suppose we want to label stories as $\mathcal{Y} = \{\textsc{Good}, \textsc{Bad}, \textsc{Okay}\}$
- What makes a good story?

A simple feed-forward architecture

- Suppose we want to label stories as $\mathcal{Y} = \{\textsc{Good}, \textsc{Bad}, \textsc{Okay}\}$
- What makes a good story?
    - Exciting plot, compelling characters, interesting setting…

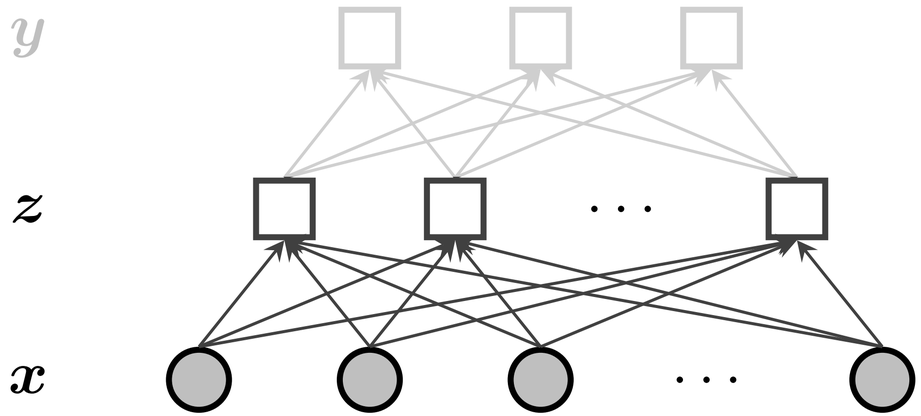
A simple feed-forward architecture

- Suppose we want to label stories as $\mathcal{Y} = \{\text{Good}, \text{Bad}, \text{Okay}\}$
- What makes a good story?
  - Exciting plot, compelling characters, interesting setting…
- Let's call this vector of features $\boldsymbol{z}$.

A simple feed-forward architecture

- Suppose we want to label stories as $\mathcal{Y} = \{\textsc{Good}, \textsc{Bad}, \textsc{Okay}\}$
- What makes a good story?
    - Exciting plot, compelling characters, interesting setting…
- Let's call this vector of features ***z***.
- If ***z*** is well-chosen, it will be easy to predict from ***x*** (the words), and it will make it easy to predict the label, *y*.

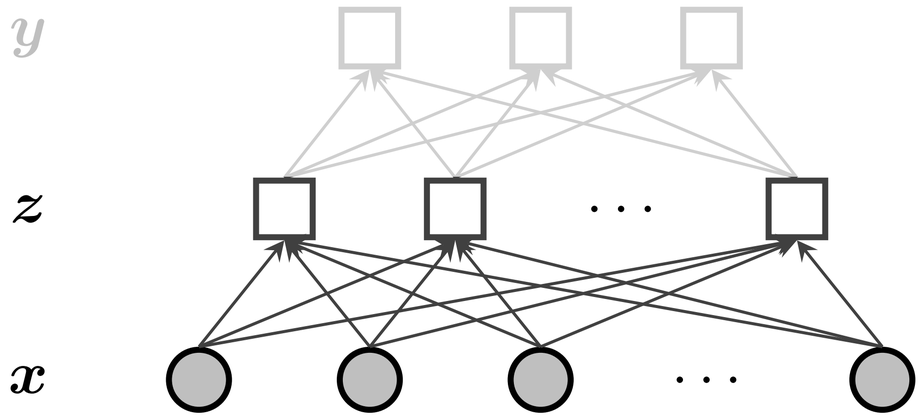
A simple feed-forward architecture



A simple feed-forward architecture

- Let's predict each z_k from $\boldsymbol{x}$ by binary logistic regression:

$$\Pr(z_k = 1 \mid \boldsymbol{x}) = \sigma(\boldsymbol{\theta}_k^{(x \to z)} \cdot \boldsymbol{x})$$



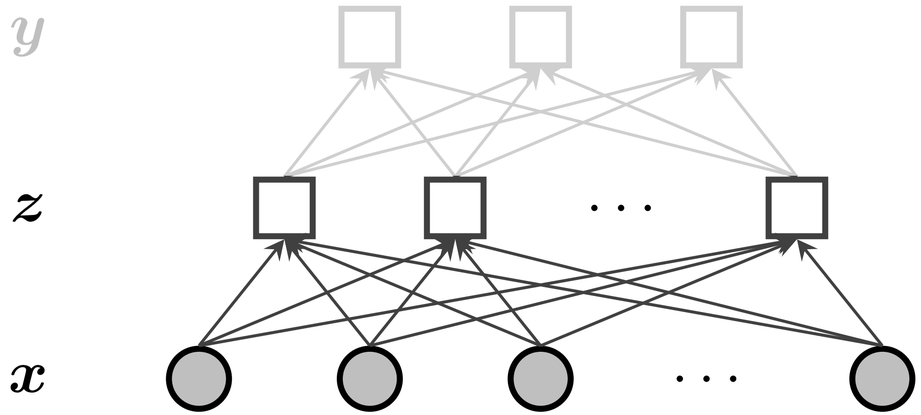
A simple feed-forward architecture

- Let's predict each z_k from $\boldsymbol{x}$ by binary logistic regression:

$$\Pr(z_k = 1 \mid \boldsymbol{x}) = \sigma(\boldsymbol{\theta}_k^{(x \to z)} \cdot \boldsymbol{x})$$

logistic fn aka sigmoid σ

$$= \frac{1}{1 + e^{-\theta_k^{(x \to z)} x}}$$



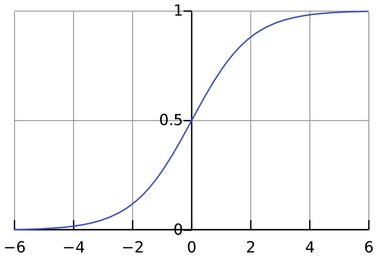
A simple feed-forward architecture

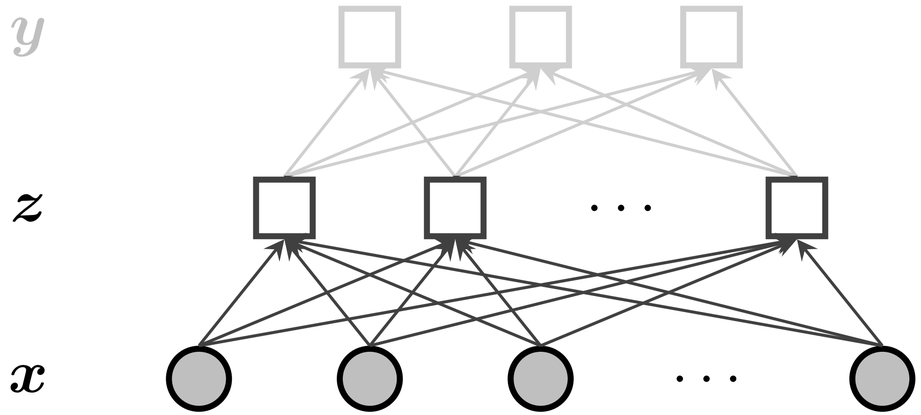
- Let's predict each z_k from $\boldsymbol{x}$ by binary logistic regression:

$$\Pr(z_k = 1 \mid \boldsymbol{x}) = \sigma(\boldsymbol{\theta}_k^{(x \to z)} \cdot \boldsymbol{x})$$

logistic fn aka sigmoid σ

$$= \frac{1}{1 + e^{-\theta_k^{(x \to z)} x}}$$





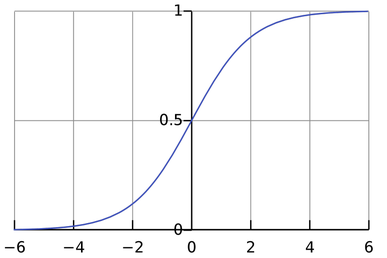
A simple feed-forward architecture

- Let's predict each z_k from $\boldsymbol{x}$ by binary logistic regression:

$$\Pr(z_k = 1 \mid \boldsymbol{x}) = \sigma(\boldsymbol{\theta}_k^{(x \to z)} \cdot \boldsymbol{x})$$

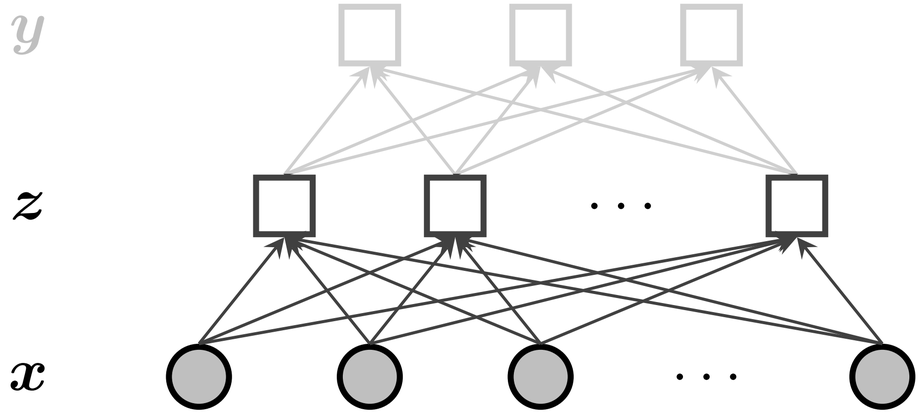
logistic fn aka sigmoid σ

$$= \frac{1}{1 + e^{-\theta_k^{(x \to z)} x}}$$



- The weights can be collected into a matrix,

$$\boldsymbol{\Theta}^{(x \to z)} = [\boldsymbol{\theta}_1^{(x \to z)}, \boldsymbol{\theta}_2^{(x \to z)}, \ldots, \boldsymbol{\theta}_{K_z}^{(x \to z)}]^\top$$



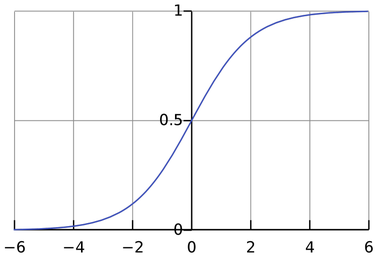
A simple feed-forward architecture

- Let's predict each z_k from $\boldsymbol{x}$ by binary logistic regression:

$$\Pr(z_k = 1 \mid \boldsymbol{x}) = \sigma(\boldsymbol{\theta}_k^{(x \to z)} \cdot \boldsymbol{x})$$

logistic fn aka sigmoid σ

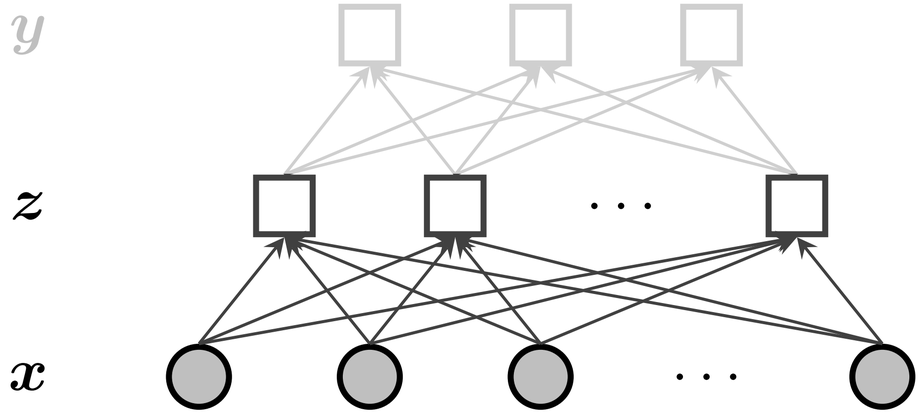
$$= \frac{1}{1 + e^{-\theta_k^{(x \to z)} x}}$$



- The weights can be collected into a matrix,

$$\boldsymbol{\Theta}^{(x \to z)} = [\boldsymbol{\theta}_1^{(x \to z)}, \boldsymbol{\theta}_2^{(x \to z)}, \ldots, \boldsymbol{\theta}_{K_z}^{(x \to z)}]^\top$$

- so that E[**z**] = $\sigma(\boldsymbol{\Theta}^{(x \to z)}\boldsymbol{x})$, where σ is applied element-wise.



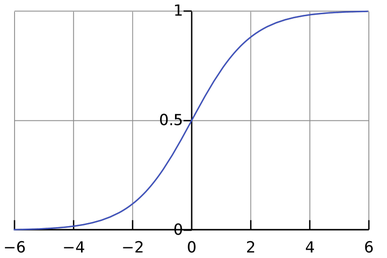
A simple feed-forward architecture

- Let's predict each z_k from $\boldsymbol{x}$ by binary logistic regression:

$$\Pr(z_k = 1 \mid \boldsymbol{x}) = \sigma(\boldsymbol{\theta}_k^{(x \to z)} \cdot \boldsymbol{x})$$

logistic fn aka sigmoid σ

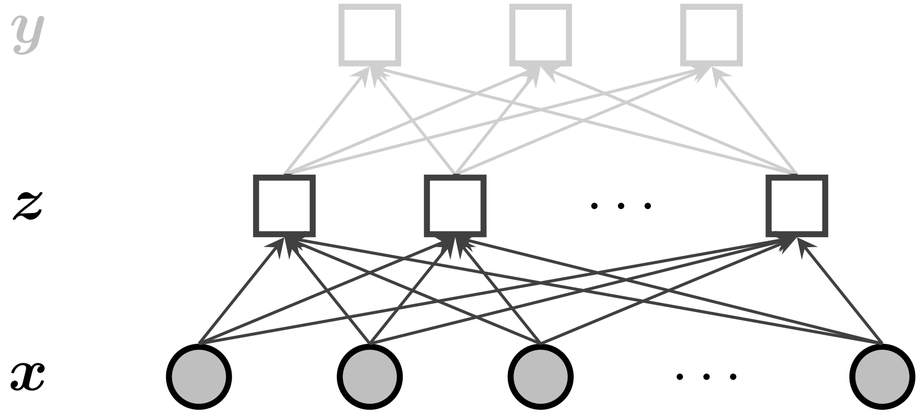
$$= \frac{1}{1 + e^{-\theta_k^{(x \to z)} x}}$$



- The weights can be collected into a matrix,

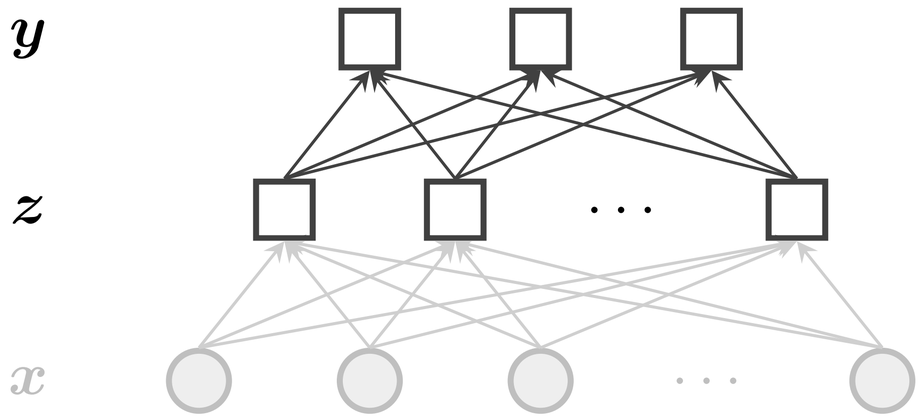
$$\boldsymbol{\Theta}^{(x \to z)} = [\boldsymbol{\theta}_1^{(x \to z)}, \boldsymbol{\theta}_2^{(x \to z)}, \ldots, \boldsymbol{\theta}_{K_z}^{(x \to z)}]^\top$$

- so that E[**z**] = $\sigma(\boldsymbol{\Theta}^{(x \to z)}\boldsymbol{x})$, where σ is applied element-wise.



matrix-vector product. dims: [k, V] * [V, 1] = [k, 1]

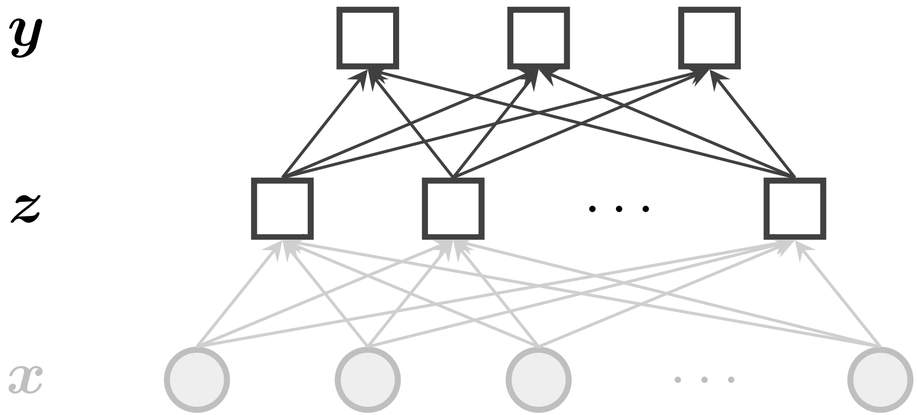
A simple feed-forward architecture



A simple feed-forward architecture

- Next we predict y from $\boldsymbol{z}$, again using logistic regression (multiclass):

$$\Pr(y = j \mid z) = \frac{\exp(\boldsymbol{\theta}_j^{(z \to y)} \cdot \boldsymbol{z} + b_j)}{\sum_{j' \in \mathcal{Y}} \exp(\boldsymbol{\theta}_{j'}^{(z \to y)} \cdot \boldsymbol{z} + b_{j'})}$$

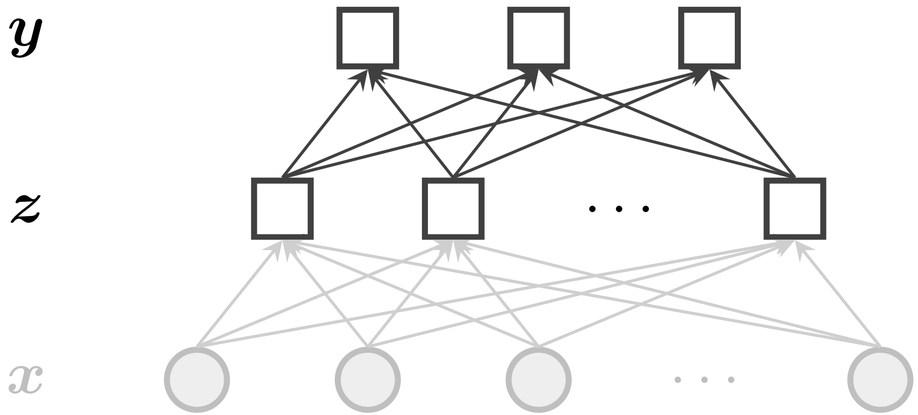


A simple feed-forward architecture

- Next we predict y from $\boldsymbol{z}$, again using logistic regression (multiclass):

$$\Pr(y = j \mid z) = \frac{\exp(\boldsymbol{\theta}_j^{(z \to y)} \cdot \boldsymbol{z} + b_j)}{\sum_{j' \in \mathcal{Y}} \exp(\boldsymbol{\theta}_{j'}^{(z \to y)} \cdot \boldsymbol{z} + b_{j'})}$$

additive bias/offset vector



A simple feed-forward architecture

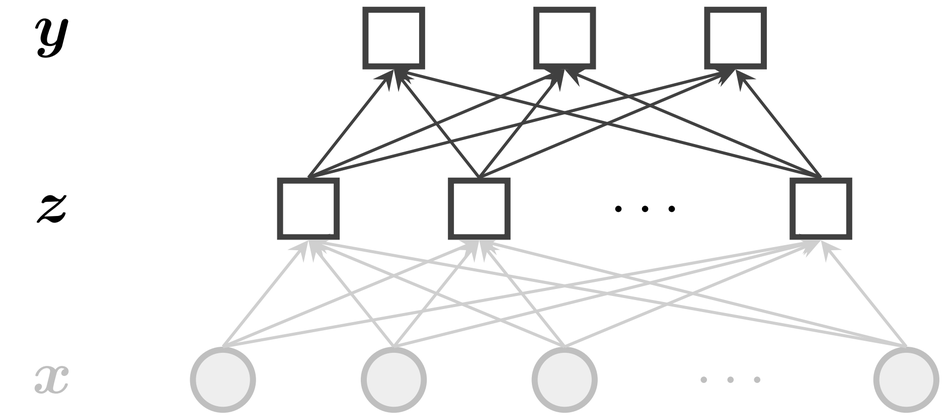
- Next we predict y from $\boldsymbol{z}$, again using logistic regression (multiclass):

$$\Pr(y = j \mid z) = \frac{\exp(\boldsymbol{\theta}_j^{(z \to y)} \cdot \boldsymbol{z} + b_j)}{\sum_{j' \in \mathcal{Y}} \exp(\boldsymbol{\theta}_{j'}^{(z \to y)} \cdot \boldsymbol{z} + b_{j'})}$$

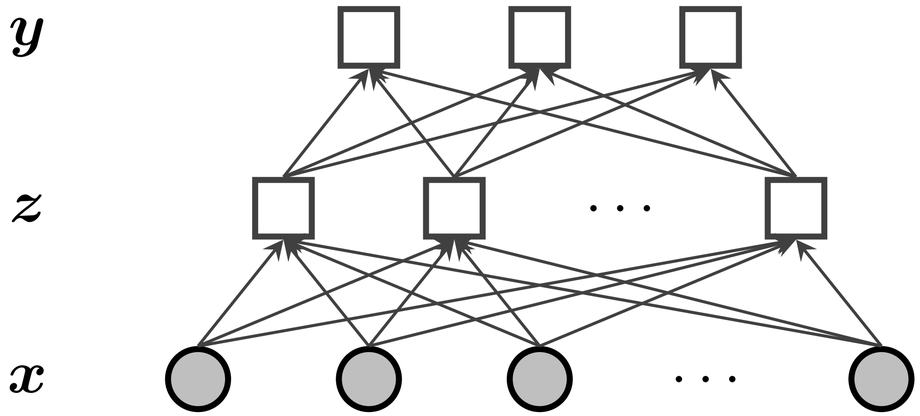
additive bias/offset vector

Vector of probabilities over each possible y is denoted:

$$p(\boldsymbol{y} \mid \boldsymbol{z}) = \text{SoftMax}(\boldsymbol{\Theta}^{(z \to y)} \boldsymbol{z} + \boldsymbol{b})$$

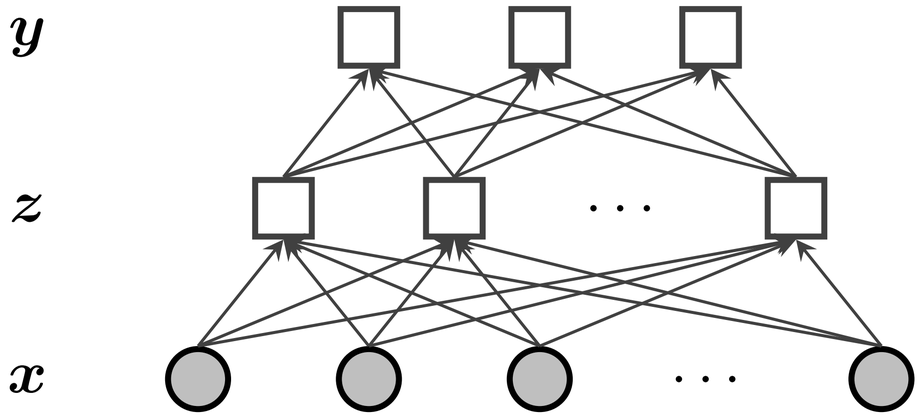


A simple feed-forward architecture



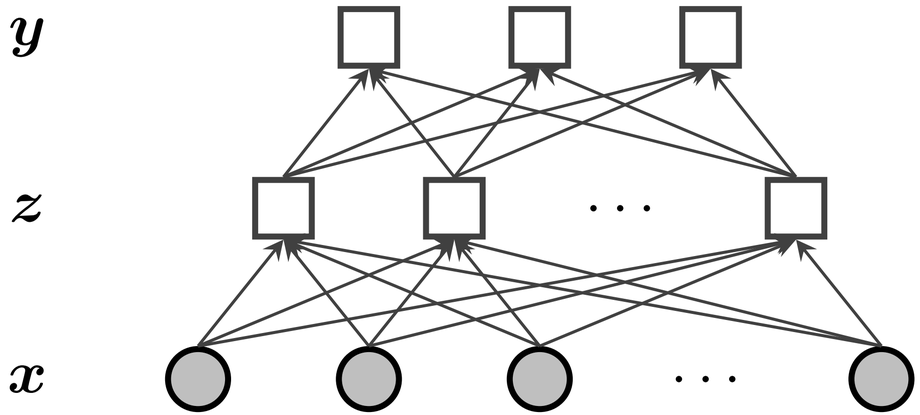
A simple feed-forward architecture

- In reality, we never observe $\boldsymbol{z}$, it is a **hidden layer**. We don't bother predicting 0/1 values for $\boldsymbol{z}$, we compute it directly from $\boldsymbol{x}$.



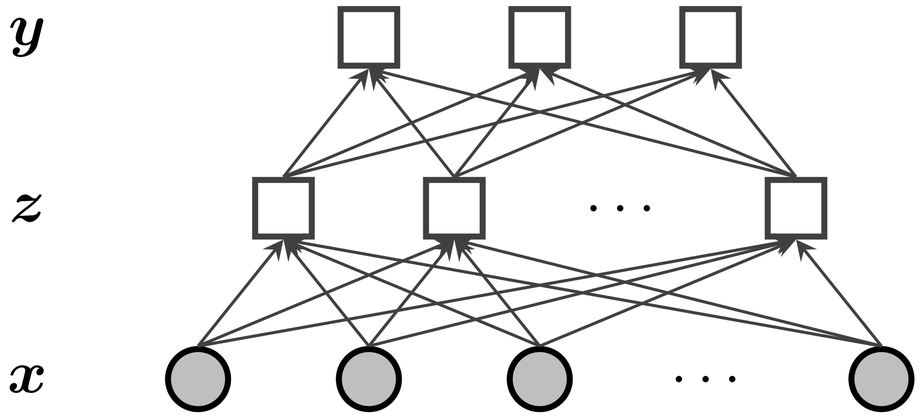
A simple feed-forward architecture

- In reality, we never observe $\boldsymbol{z}$, it is a **hidden layer**. We don't bother predicting 0/1 values for $\boldsymbol{z}$, we compute it directly from $\boldsymbol{x}$.
- This makes p(y | $\boldsymbol{x}$) a complex, nonlinear function of $\boldsymbol{x}$.



A simple feed-forward architecture

- In reality, we never observe $\boldsymbol{z}$, it is a **hidden layer**. We don't bother predicting 0/1 values for $\boldsymbol{z}$, we compute it directly from $\boldsymbol{x}$.
- This makes p(y | $\boldsymbol{x}$) a complex, nonlinear function of $\boldsymbol{x}$.
- We can have multiple hidden layers $\boldsymbol{z}^{(1)}$, $\boldsymbol{z}^{(2)}$, … adding even more expressiveness.



A simple feed-forward architecture

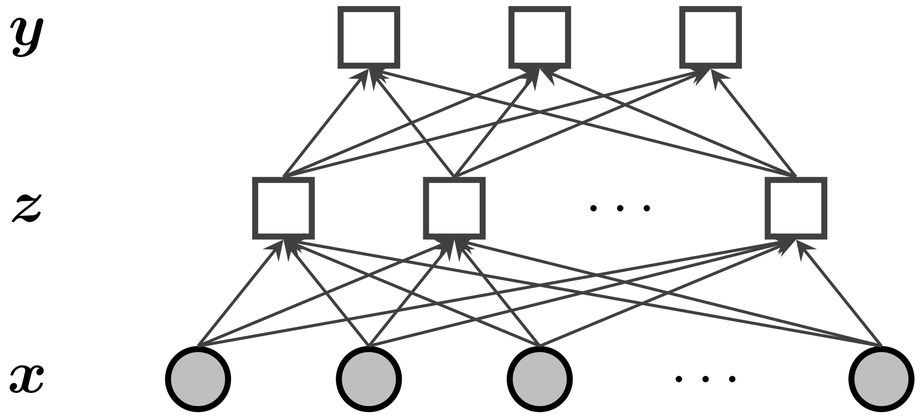
- In reality, we never observe $\boldsymbol{z}$, it is a **hidden layer**. We don't bother predicting 0/1 values for $\boldsymbol{z}$, we compute it directly from $\boldsymbol{x}$.
- This makes p($y \mid \boldsymbol{x}$) a complex, nonlinear function of $\boldsymbol{x}$.
- We can have multiple hidden layers $\boldsymbol{z}^{(1)}$, $\boldsymbol{z}^{(2)}$, … adding even more expressiveness.
- To summarize:

$$\boldsymbol{z} = \sigma(\boldsymbol{\Theta}^{(x \to z)} \boldsymbol{x})$$

$$p(\boldsymbol{y} \mid \boldsymbol{z}) = \text{SoftMax}(\boldsymbol{\Theta}^{(z \to y)} \boldsymbol{z} + \boldsymbol{b})$$

where

$$\sigma(\boldsymbol{\Theta}^{(x \to z)} \mathbf{x}) = \left[\sigma(\boldsymbol{\theta}_1^{(x \to z)} \cdot \mathbf{x}), \sigma(\boldsymbol{\theta}_2^{(x \to z)} \cdot \mathbf{x}), ..., \sigma(\boldsymbol{\theta}_{K_z}^{(x \to z)} \cdot \mathbf{x})\right]^T$$



Activation functions

Activation functions

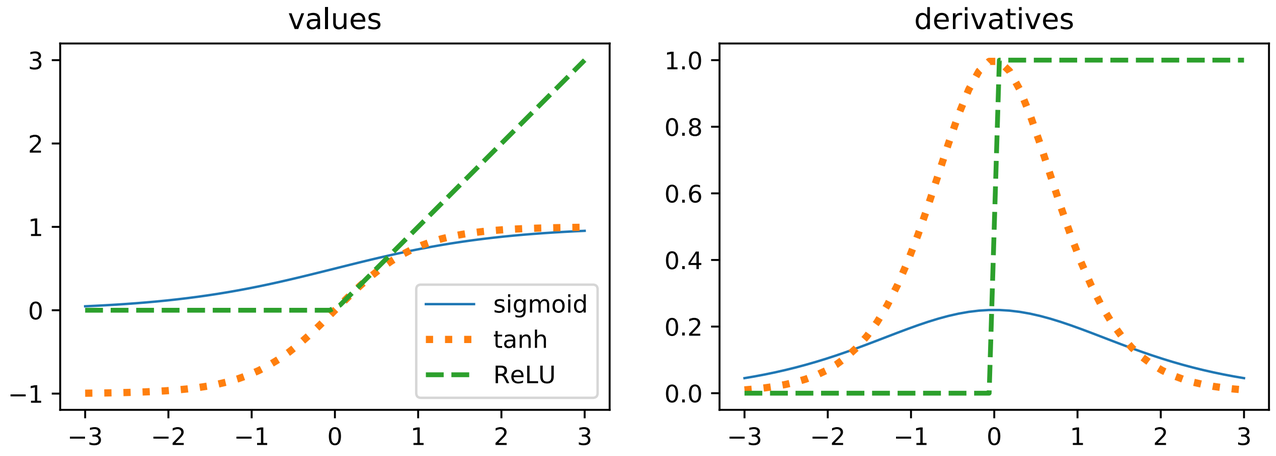
- The sigmoid in $\boldsymbol{z} = \sigma(\boldsymbol{\Theta}^{(x \to z)} \boldsymbol{x})$ is called an **activation function**.

Activation functions

- The sigmoid in $\boldsymbol{z} = \sigma(\boldsymbol{\Theta}^{(x \to z)} \boldsymbol{x})$ is called an **activation function**.
- In general, we can write $\boldsymbol{z} = f(\boldsymbol{\Theta}^{(x \to z)} \boldsymbol{x})$ to indicate an arbitrary activation function.

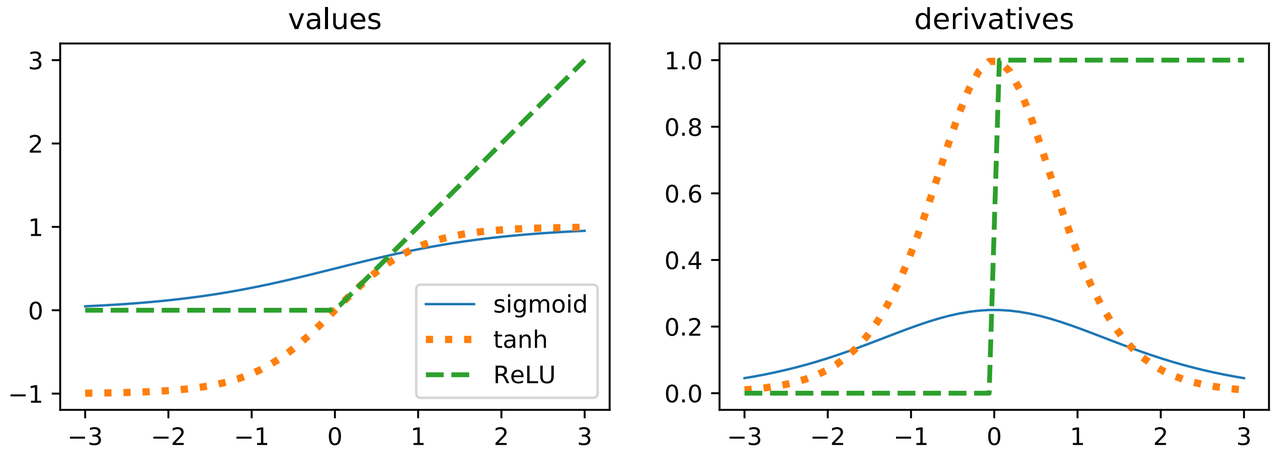
Activation functions

- The sigmoid in $\boldsymbol{z} = \sigma(\boldsymbol{\Theta}^{(x \to z)}\boldsymbol{x})$ is called an **activation function**.
- In general, we can write $\boldsymbol{z} = f(\boldsymbol{\Theta}^{(x \to z)}\boldsymbol{x})$ to indicate an arbitrary activation function.
- Other choices include:



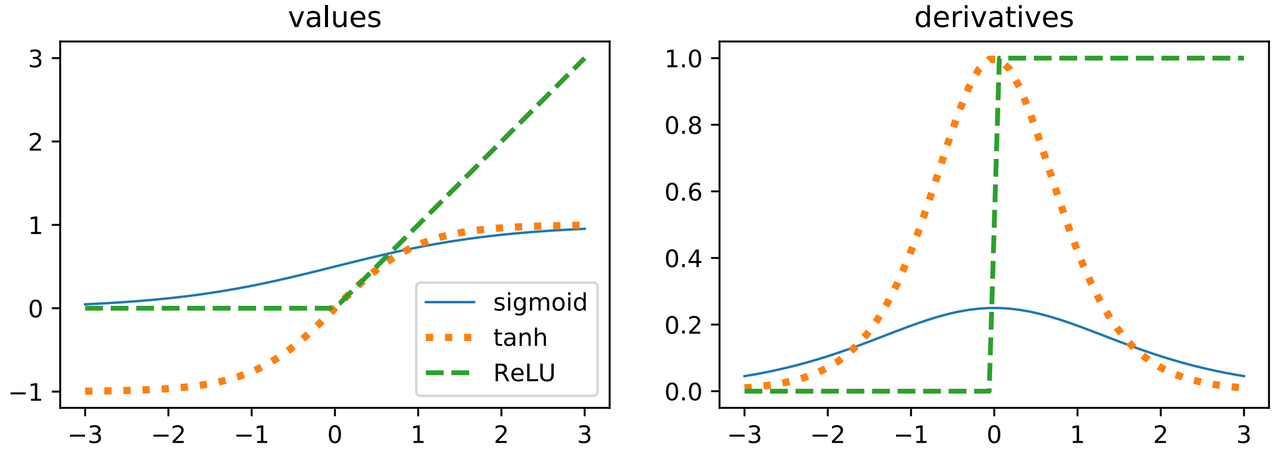
Activation functions

- The sigmoid in $\boldsymbol{z} = \sigma(\boldsymbol{\Theta}^{(x \to z)}\boldsymbol{x})$ is called an **activation function**.
- In general, we can write $\boldsymbol{z} = f(\boldsymbol{\Theta}^{(x \to z)}\boldsymbol{x})$ to indicate an arbitrary activation function.
- Other choices include:
  - **Hyperbolic tangent**: tanh, centered at 0, helps avoid saturation



Activation functions

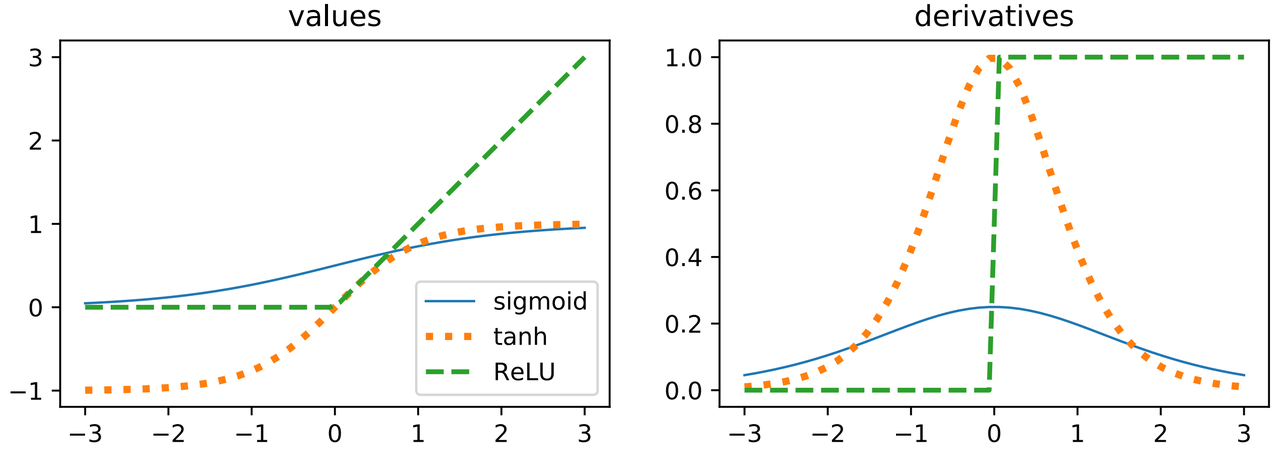
- The sigmoid in $\boldsymbol{z} = \sigma(\boldsymbol{\Theta}^{(x \to z)}\boldsymbol{x})$ is called an **activation function**.
- In general, we can write $\boldsymbol{z} = f(\boldsymbol{\Theta}^{(x \to z)}\boldsymbol{x})$ to indicate an arbitrary activation function.
- Other choices include:
  - **Hyperbolic tangent**: tanh, centered at 0, helps avoid saturation
  - **Rectified linear unit**: ReLU(*a*) = max(0, *a*), which is fast to evaluate, easy to analyze, even further avoids saturation.



Activation functions

- The sigmoid in $\boldsymbol{z} = \sigma(\boldsymbol{\Theta}^{(x \to z)}\boldsymbol{x})$ is called an **activation function**.
- In general, we can write $\boldsymbol{z} = f(\boldsymbol{\Theta}^{(x \to z)}\boldsymbol{x})$ to indicate an arbitrary activation function.
- Other choices include:
  - **Hyperbolic tangent**: tanh, centered at 0, helps avoid saturation
  - **Rectified linear unit**: ReLU(*a*) = max(0, *a*), which is fast to evaluate, easy to analyze, even further avoids saturation.
  - **Leaky ReLU**:

$$= \begin{cases} a, & a \geq 0 \\ .0001a, & \text{otherwise} \end{cases}$$



Training neural networks

Gradient descent

Training neural networks

Gradient descent

- In general, neural networks are learned by gradient descent, using minibatches:

$$\boldsymbol{\theta}_k^{(z \to y)} \leftarrow \boldsymbol{\theta}_k^{(z \to y)} - \eta^{(t)} \nabla_{\boldsymbol{\theta}_k^{(z \to y)}} \ell^{(i)}$$

where

Training neural networks

Gradient descent

- In general, neural networks are learned by gradient descent, using minibatches:

$$\boldsymbol{\theta}_k^{(z \to y)} \leftarrow \boldsymbol{\theta}_k^{(z \to y)} - \eta^{(t)} \nabla_{\boldsymbol{\theta}_k^{(z \to y)}} \ell^{(i)}$$

  where

  - $\eta^{(t)}$ is the learning rate at update t

Training neural networks

Gradient descent

- In general, neural networks are learned by gradient descent, using minibatches:

$$\boldsymbol{\theta}_k^{(z \to y)} \leftarrow \boldsymbol{\theta}_k^{(z \to y)} - \eta^{(t)} \nabla_{\boldsymbol{\theta}_k^{(z \to y)}} \ell^{(i)}$$

where

  - $\eta^{(t)}$ is the learning rate at update *t*
  - $\ell^{(i)}$ is the loss on instance (minibatch) *i*

Training neural networks

Gradient descent

- In general, neural networks are learned by gradient descent, using minibatches:

$$\boldsymbol{\theta}_k^{(z \to y)} \leftarrow \boldsymbol{\theta}_k^{(z \to y)} - \eta^{(t)} \nabla_{\boldsymbol{\theta}_k^{(z \to y)}} \ell^{(i)}$$

where

- $\eta^{(t)}$ is the learning rate at update t
- $\ell^{(i)}$ is the loss on instance (minibatch) i
- $\nabla_{\boldsymbol{\theta}_k^{(z \to y)}} \ell^{(i)}$ is the gradient of the loss with respect to the output weights $\boldsymbol{\theta}_k^{(z \to y)}$

$$\nabla_{\boldsymbol{\theta}_k^{(z \to y)}} \ell^{(i)} = \left[\frac{\partial \ell^{(i)}}{\partial \theta_{k,1}^{(z \to y)}}, \frac{\partial \ell^{(i)}}{\partial \theta_{k,2}^{(z \to y)}}, \ldots, \frac{\partial \ell^{(i)}}{\partial \theta_{k,K_y}^{(z \to y)}} \right]$$

Training neural networks

Backpropagation

Training neural networks

Backpropagation

- If we don't observe $\boldsymbol{z}$, how can we learn the weights $\boldsymbol{\Theta}^{(x \to z)}$?

Training neural networks

Backpropagation

- If we don't observe $\boldsymbol{z}$, how can we learn the weights $\boldsymbol{\Theta}^{(x \to z)}$?

- **Backpropagation**: compute a loss on $\boldsymbol{y}$, and apply the chain rule from calculus to compute a gradient on all parameters.

Training neural networks

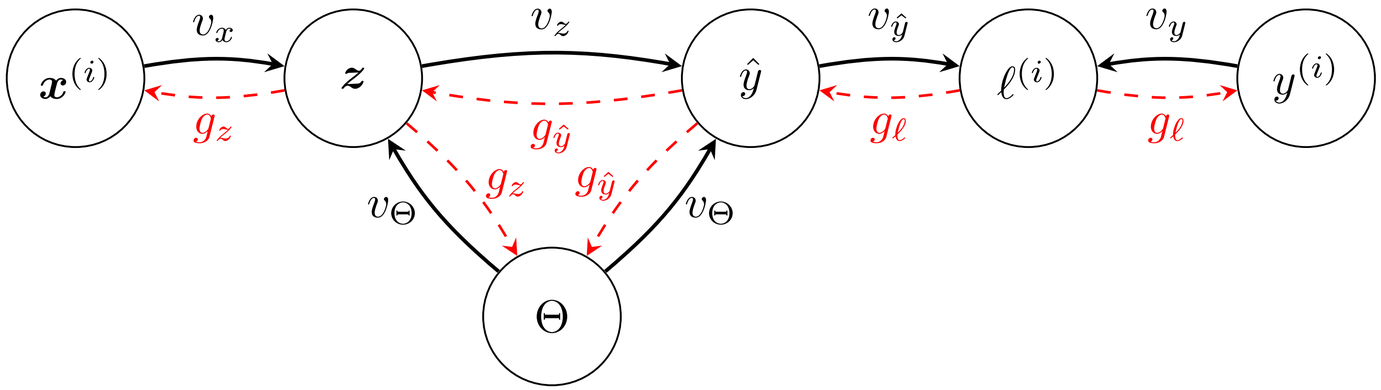
Backpropagation

- If we don't observe $\boldsymbol{z}$, how can we learn the weights $\boldsymbol{\Theta}^{(x \to z)}$?
- **Backpropagation**: compute a loss on $\boldsymbol{y}$, and apply the chain rule from calculus to compute a gradient on all parameters.
- Backpropagation as an algorithm: construct a directed acyclic **computation graph** with nodes for inputs, outputs, hidden layers, parameters.

Training neural networks

Backpropagation

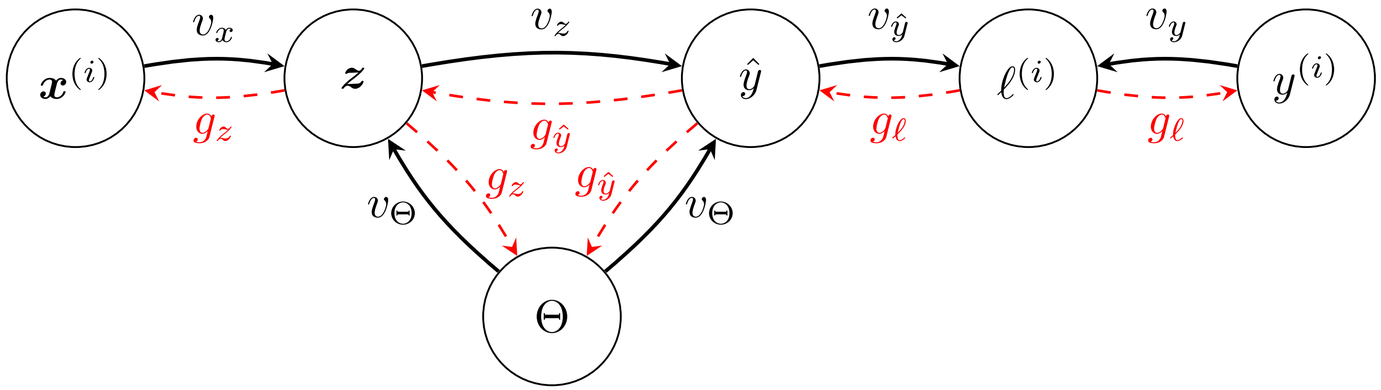
- Backpropagation as an algorithm: construct a directed acyclic **computation graph** with nodes for inputs, outputs, hidden layers, parameters.



Training neural networks

Backpropagation

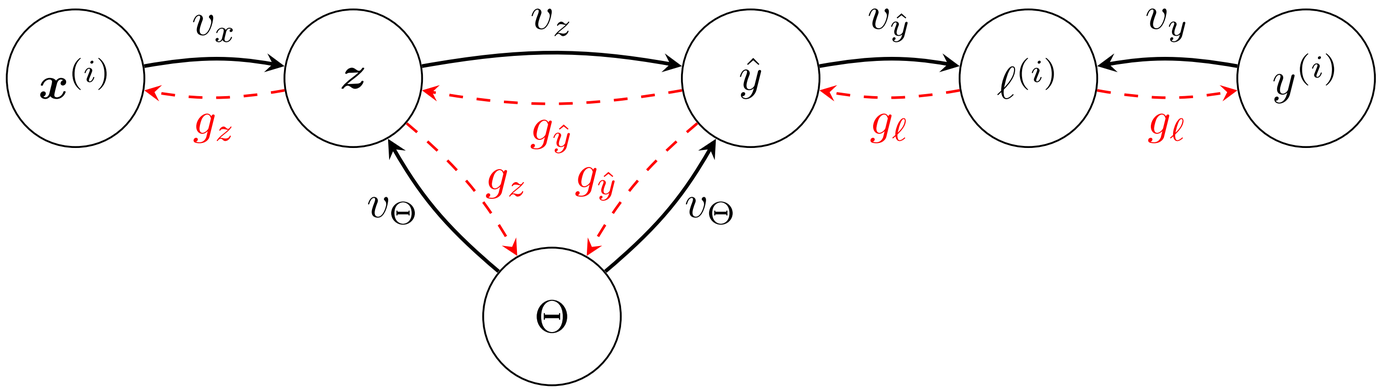
- Backpropagation as an algorithm: construct a directed acyclic **computation graph** with nodes for inputs, outputs, hidden layers, parameters.
  - **Forward pass**: values (e.g. v_x) go from parents to children



Training neural networks

Backpropagation

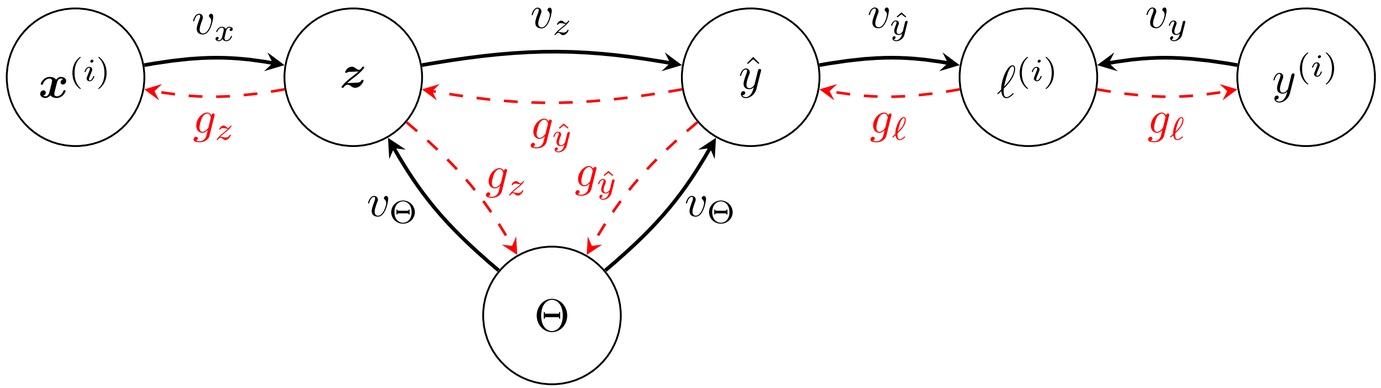
- Backpropagation as an algorithm: construct a directed acyclic **computation graph** with nodes for inputs, outputs, hidden layers, parameters.
  - **Forward pass**: values (e.g. v_x) go from parents to children
  - **Backward pass**: gradients (e.g. g_z) go from children to parents, implementing the chain rule



Training neural networks

Backpropagation

- Backpropagation as an algorithm: construct a directed acyclic **computation graph** with nodes for inputs, outputs, hidden layers, parameters.
  - **Forward pass**: values (e.g. v_x) go from parents to children
  - **Backward pass**: gradients (e.g. g_z) go from children to parents, implementing the chain rule
  - As long as the gradient is implemented for a layer/operation, you can add it to the graph, and let **automatic differentiation** compute updates for every layer.



How to represent text for classification?

Another choice of $\mathcal{R}$: word embeddings

How to represent text for classification?

Another choice of $\mathcal{R}$: word embeddings

- Text is naturally viewed as a sequence of tokens $w_1, w_2, \ldots, w_T$

How to represent text for classification?

Another choice of $\mathcal{R}$: word embeddings

- Text is naturally viewed as a sequence of tokens $w_1, w_2, \ldots, w_T$
- Context is lost when this sequence is converted to a bag-of-words.

How to represent text for classification?

Another choice of $\mathcal{R}$: word embeddings

- Text is naturally viewed as a sequence of tokens $w_1, w_2, \ldots, w_T$
- Context is lost when this sequence is converted to a bag-of-words.
- Instead, a **lookup layer** can compute **embeddings** (real-valued vectors) for each type, resulting in a matrix $\mathbf{X}^{(0)} \in \mathbb{R}^{K_e \times M}$

$\boldsymbol{w}$ =

-1.36	1.77	0.71	-0.25	0.11	-1.36	0.03	0.71	-0.45
-0.23	-0.58	1.43	-1.27	-0.71	-0.23	0.69	1.43	1.88
0.84	-0.33	0.11	1.36	-1.08	0.84	0.14	0.11	-0.18
…	…	…	…	…	…	…	…	…
-0.067	0.93	-5.6	0.74	-0.07	-0.067	-0.36	-5.6	-1.58
the	drinks	were	strong	but	the	tacos	were	bland

Evaluating classifiers

Evaluating your classifier

Evaluating your classifier

- Want to predict **future** performance, on unseen data.

Evaluating your classifier

- Want to predict **future** performance, on unseen data.
- It's hard to predict the future. Do not evaluate on data that was already used for:

Evaluating your classifier

- Want to predict **future** performance, on unseen data.
- It's hard to predict the future. Do not evaluate on data that was already used for:
  - training

Evaluating your classifier

- Want to predict **future** performance, on unseen data.
- It's hard to predict the future. Do not evaluate on data that was already used for:
  - training
  - hyperparameter selection

Evaluating your classifier

- Want to predict **future** performance, on unseen data.
- It's hard to predict the future. Do not evaluate on data that was already used for:
  - training
  - hyperparameter selection
  - selecting classification model, model structure

Evaluating your classifier

- Want to predict **future** performance, on unseen data.
- It's hard to predict the future. Do not evaluate on data that was already used for:
  - training
  - hyperparameter selection
  - selecting classification model, model structure
  - preprocessing decisions, such as vocabulary selection

Evaluating your classifier

- Want to predict **future** performance, on unseen data.
- It's hard to predict the future. Do not evaluate on data that was already used for:
  - training
  - hyperparameter selection
  - selecting classification model, model structure
  - preprocessing decisions, such as vocabulary selection
- Even if you follow all these rules, you will probably still over-estimate your classifier's performance, because real future data will differ from your test set in ways that you cannot anticipate.

Accuracy

Accuracy

- Most basic metric: **accuracy**. How often is the classifier right?

$$\text{acc}(\boldsymbol{y}, \hat{\boldsymbol{y}}) = \frac{1}{N} \sum_{i=1}^{N} \delta(y^{(i)} = \hat{y})$$

Accuracy

- Most basic metric: **accuracy**. How often is the classifier right?

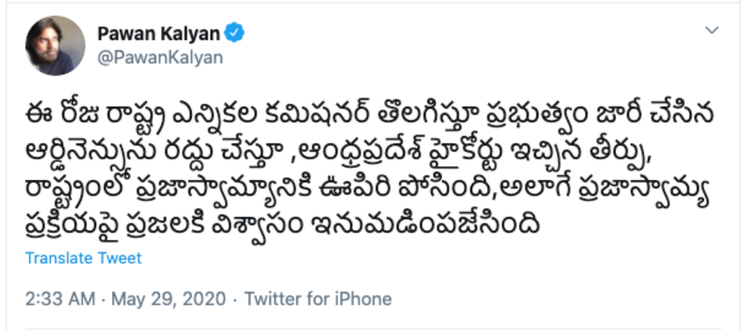
$$\text{acc}(\boldsymbol{y}, \hat{\boldsymbol{y}}) = \frac{1}{N} \sum_{i=1}^{N} \delta(y^{(i)} = \hat{y})$$

The problem with accuracy is **rare labels**, also known as **class imbalance**.

Accuracy

- Most basic metric: **accuracy**. How often is the classifier right?

$$\text{acc}(\boldsymbol{y}, \hat{\boldsymbol{y}}) = \frac{1}{N}\sum_{i=1}^{N} \delta(y^{(i)} = \hat{y})$$



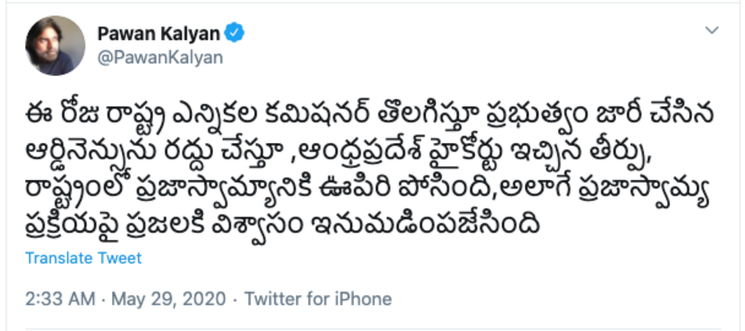
The problem with accuracy is **rare labels**, also known as **class imbalance**.

- Consider a system for detecting whether a tweet is written in Telugu.

Accuracy

- Most basic metric: **accuracy**. How often is the classifier right?

$$\text{acc}(\boldsymbol{y}, \hat{\boldsymbol{y}}) = \frac{1}{N}\sum_{i=1}^{N} \delta(y^{(i)} = \hat{y})$$



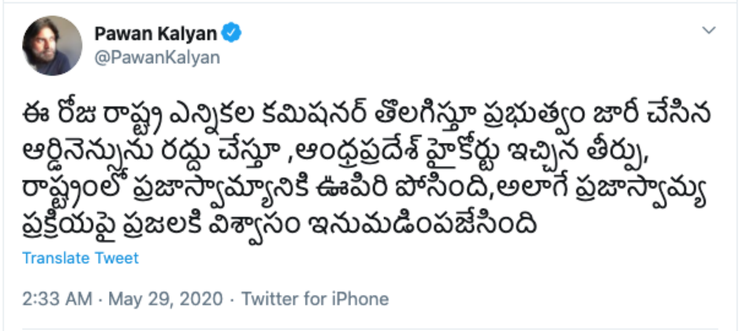
The problem with accuracy is **rare labels**, also known as **class imbalance**.

- Consider a system for detecting whether a tweet is written in Telugu.
- 0.3% of tweets are written in Telugu [Bergsma et al. 2012].

Accuracy

- Most basic metric: **accuracy**. How often is the classifier right?

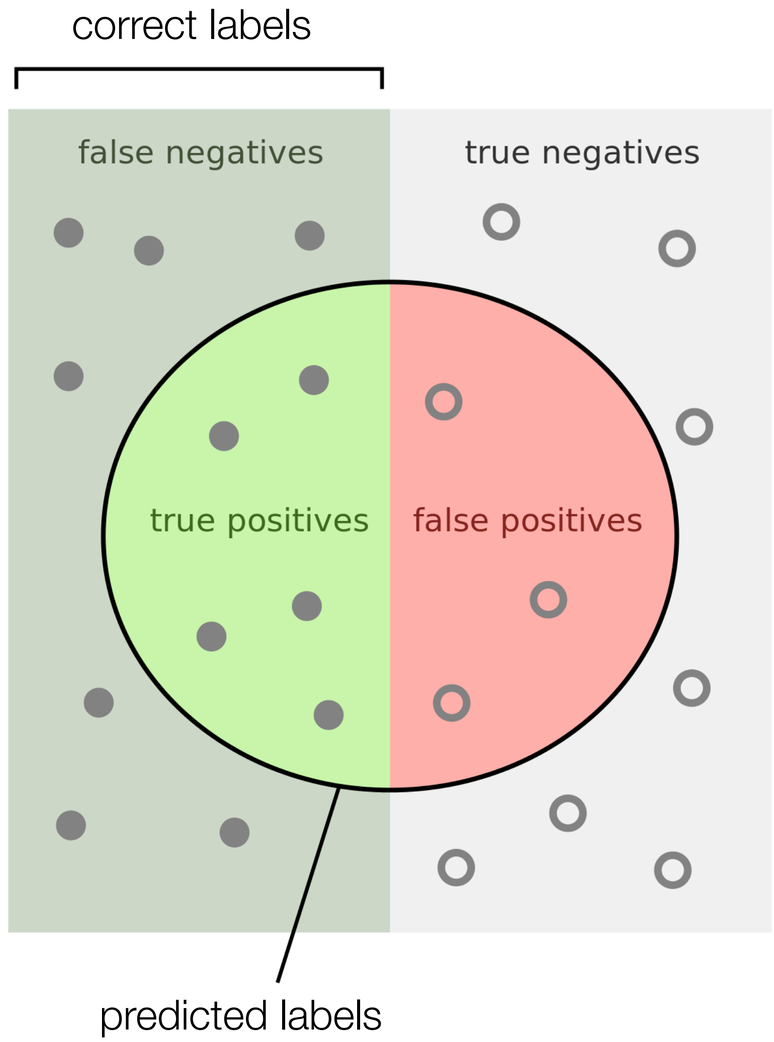
$$\text{acc}(\boldsymbol{y}, \hat{\boldsymbol{y}}) = \frac{1}{N}\sum_{i=1}^{N}\delta(y^{(i)} = \hat{y})$$



The problem with accuracy is **rare labels**, also known as **class imbalance**.

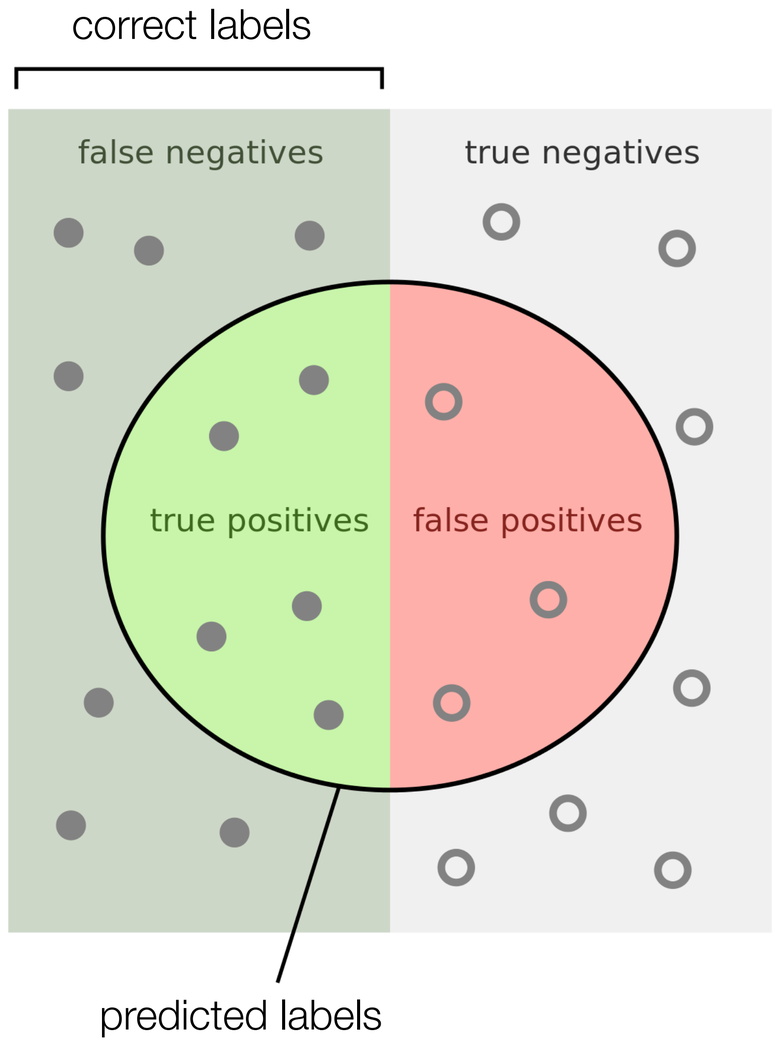
- Consider a system for detecting whether a tweet is written in Telugu.
- 0.3% of tweets are written in Telugu [Bergsma et al. 2012].
- A system that says $\hat{y}$ = `NotTelugu` 100% of the time is 99.7% accurate.

Beyond “right” and “wrong”



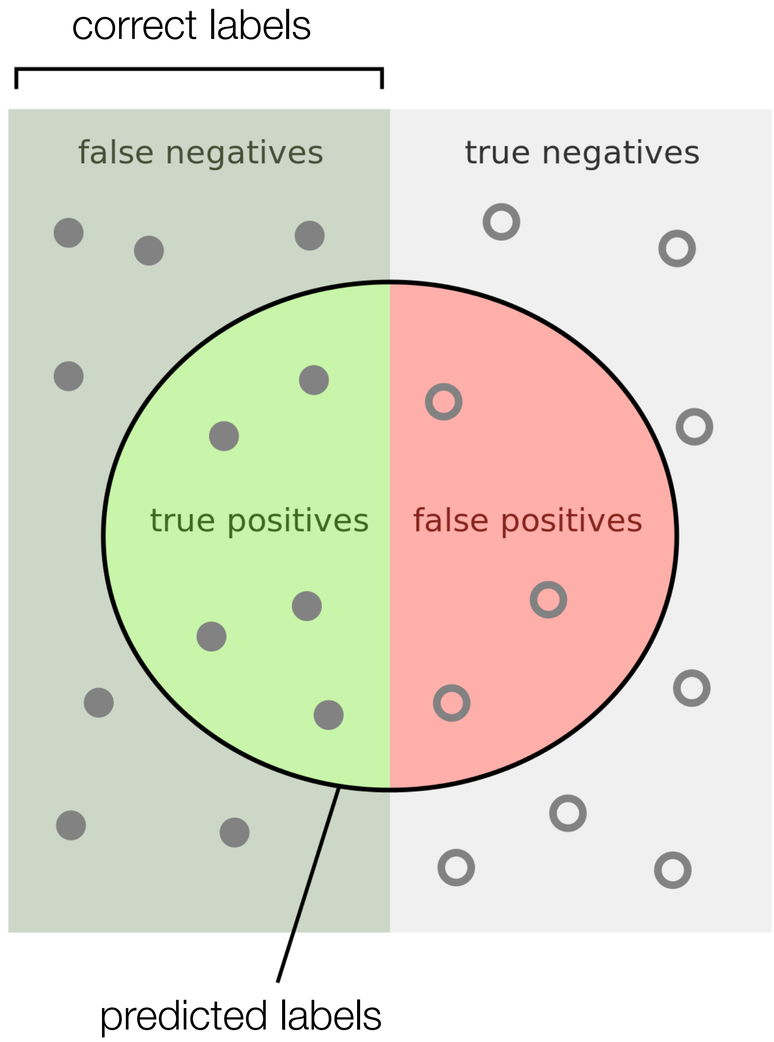
Beyond “right” and “wrong”

- For any label, there are two ways to be “wrong:”



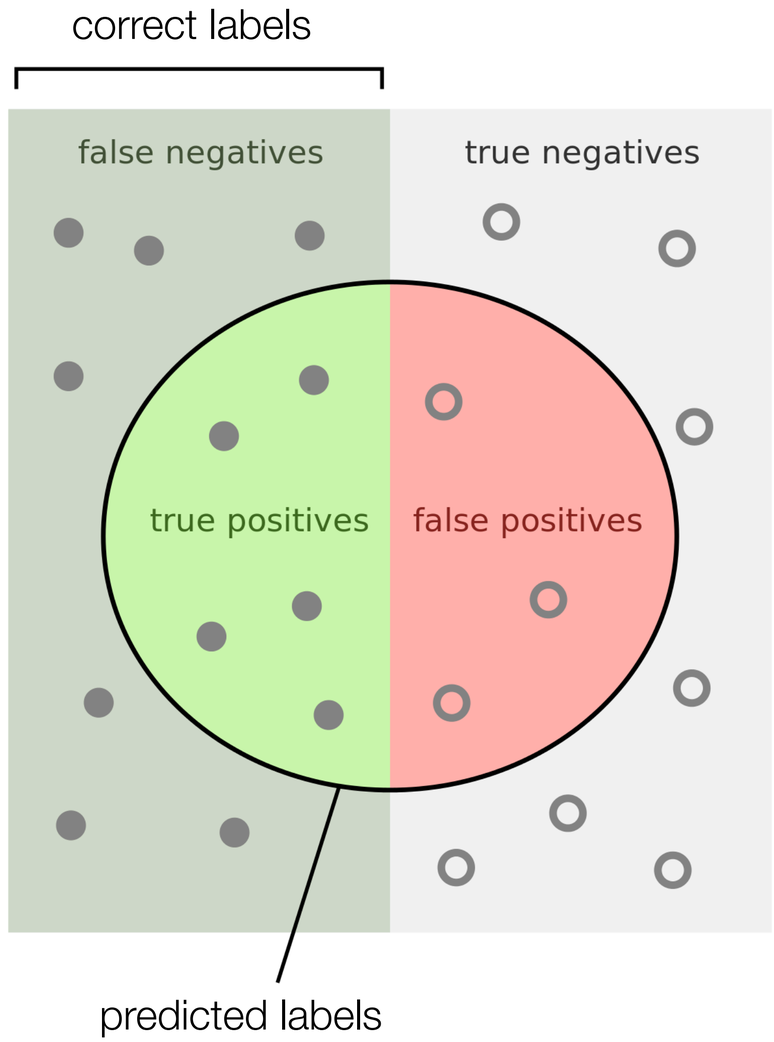
Beyond “right” and “wrong”

- For any label, there are two ways to be “wrong:”
  - **False positive**: the system incorrectly predicts the label.



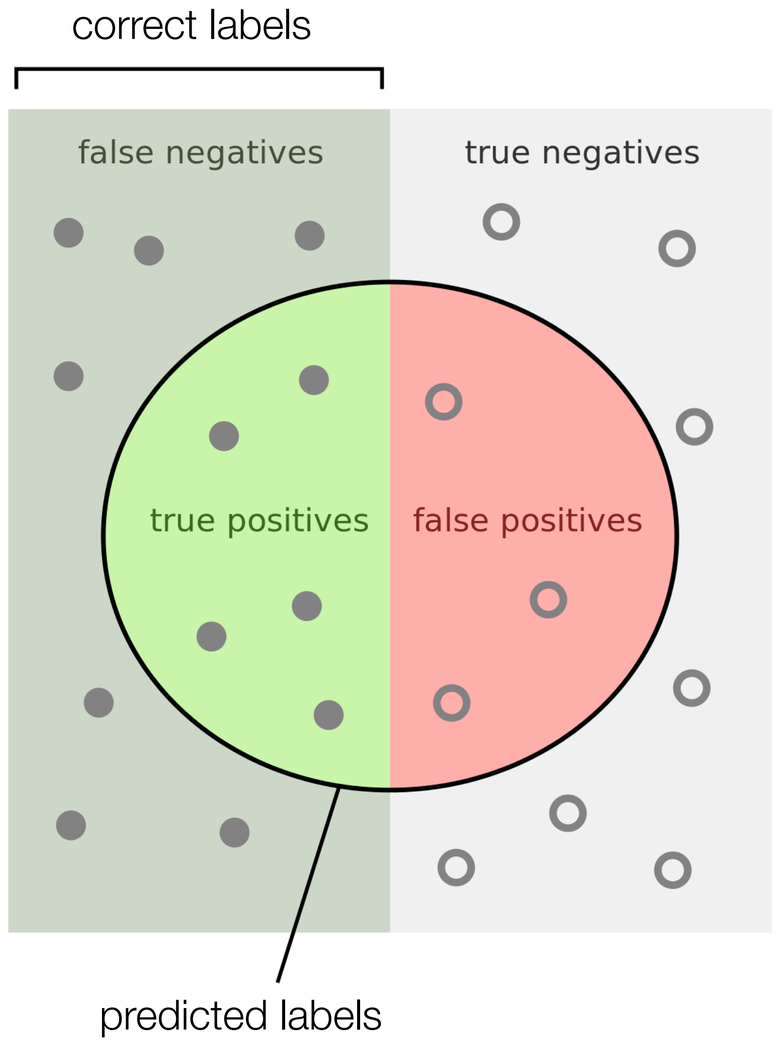
Beyond “right” and “wrong”

- For any label, there are two ways to be “wrong:”
  - **False positive**: the system incorrectly predicts the label.
  - **False negative**: the system incorrectly fails to predict the label.



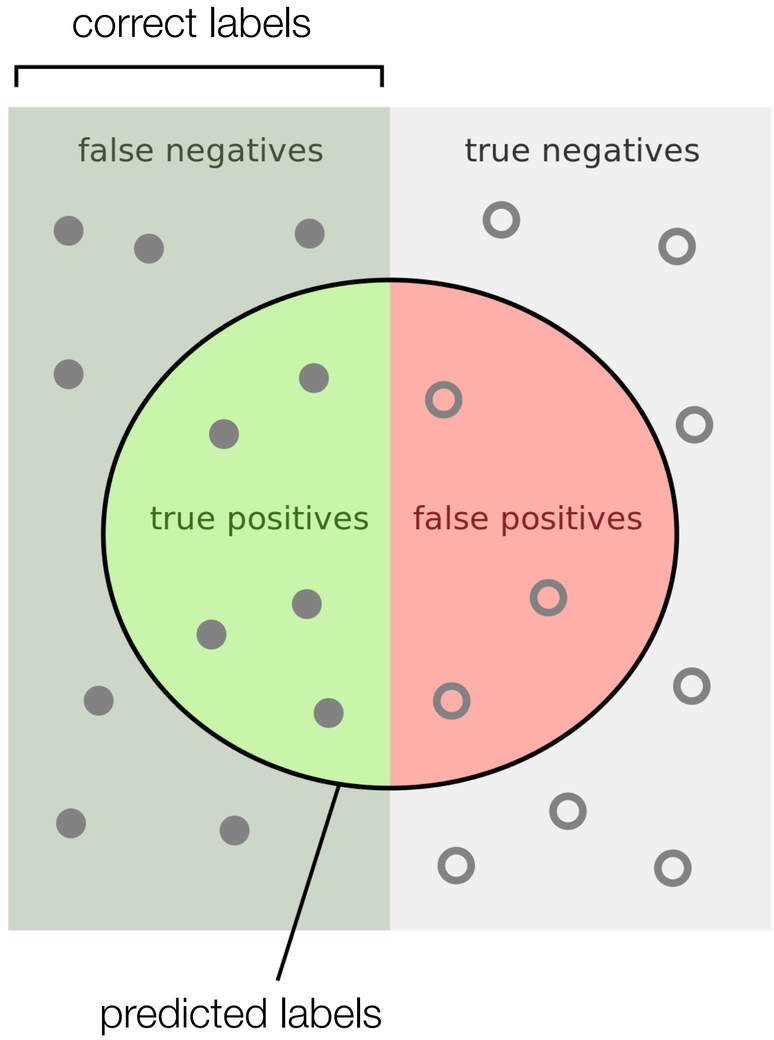
Beyond “right” and “wrong”

- For any label, there are two ways to be “wrong:”
  - **False positive**: the system incorrectly predicts the label.
  - **False negative**: the system incorrectly fails to predict the label.
- Similarly, there are two ways to be “right:”



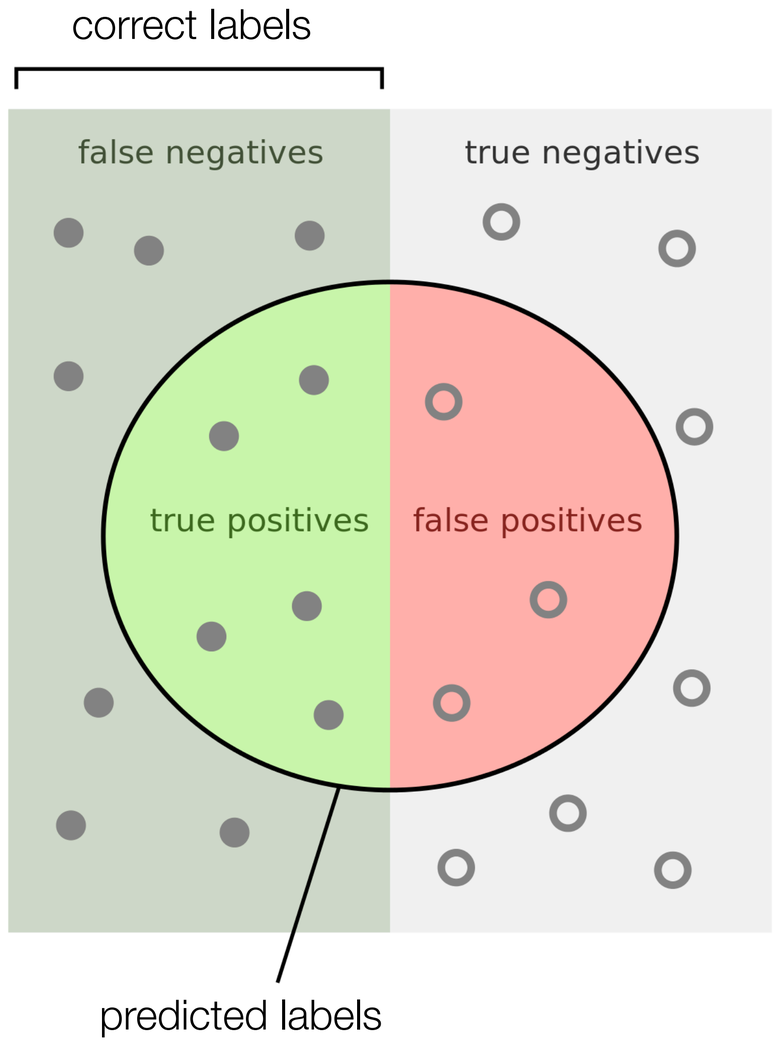
Beyond “right” and “wrong”

- For any label, there are two ways to be “wrong:”
  - **False positive**: the system incorrectly predicts the label.
  - **False negative**: the system incorrectly fails to predict the label.
- Similarly, there are two ways to be “right:”
  - **True positive**: the system correctly predicts the label.

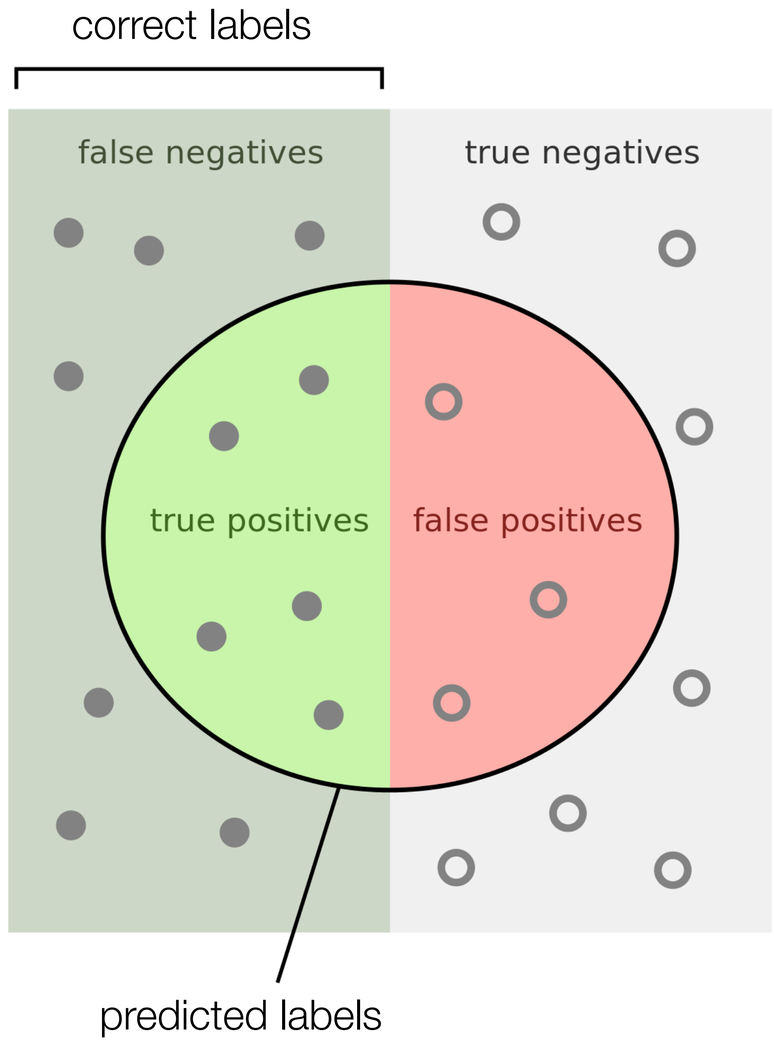


Beyond “right” and “wrong”

- For any label, there are two ways to be “wrong:”
  - **False positive**: the system incorrectly predicts the label.
  - **False negative**: the system incorrectly fails to predict the label.
- Similarly, there are two ways to be “right:”
  - **True positive**: the system correctly predicts the label.
  - **True negative**: the system correctly predicts that the label does not apply to this instance.



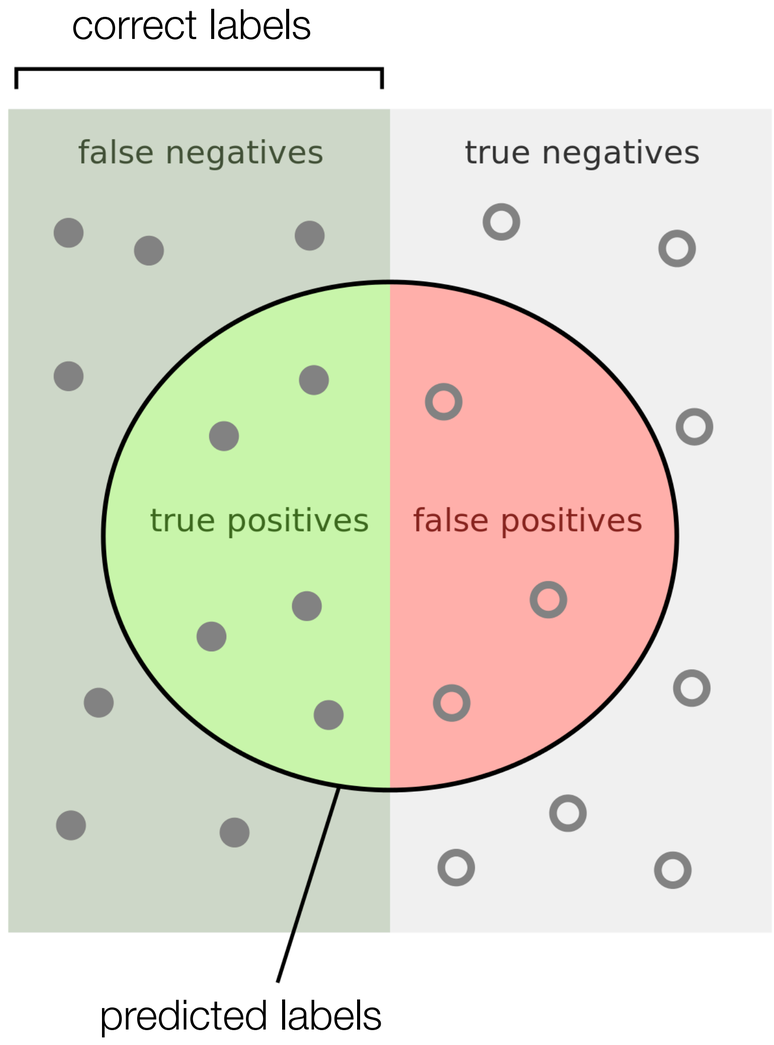
Precision and recall



Precision and recall

- **Recall**: fraction of positive instances that were correctly classified.

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

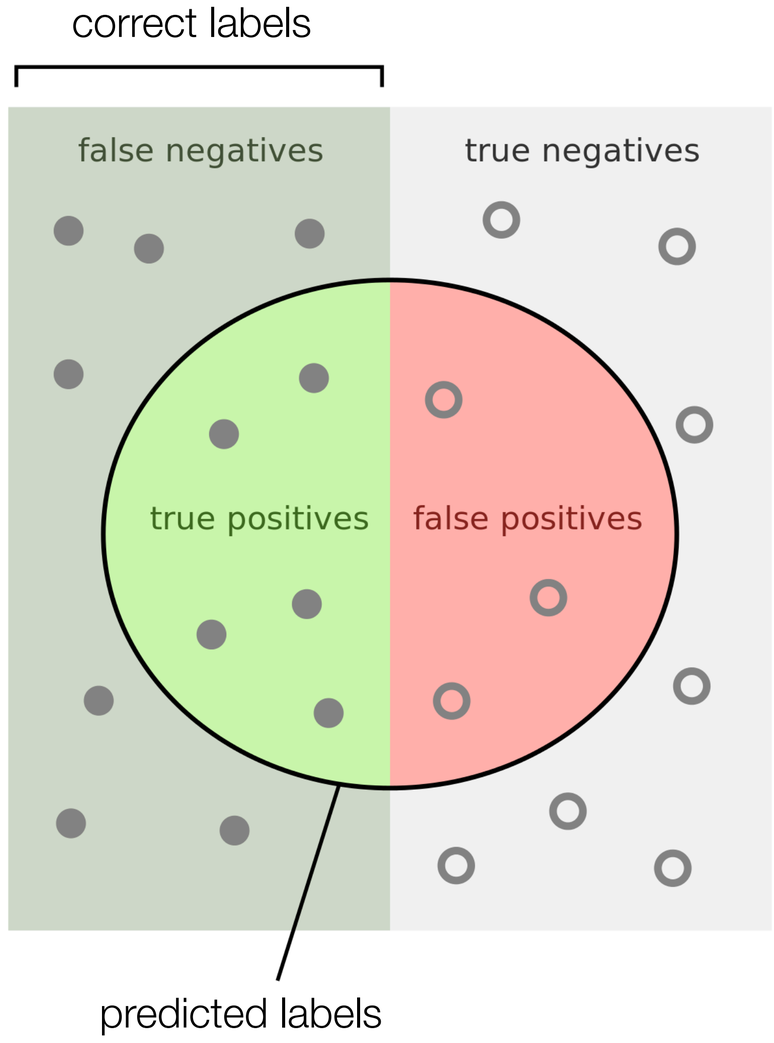


Precision and recall

- **Recall**: fraction of positive instances that were correctly classified.

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

  - The “never Telugu” classifier has 0 recall.

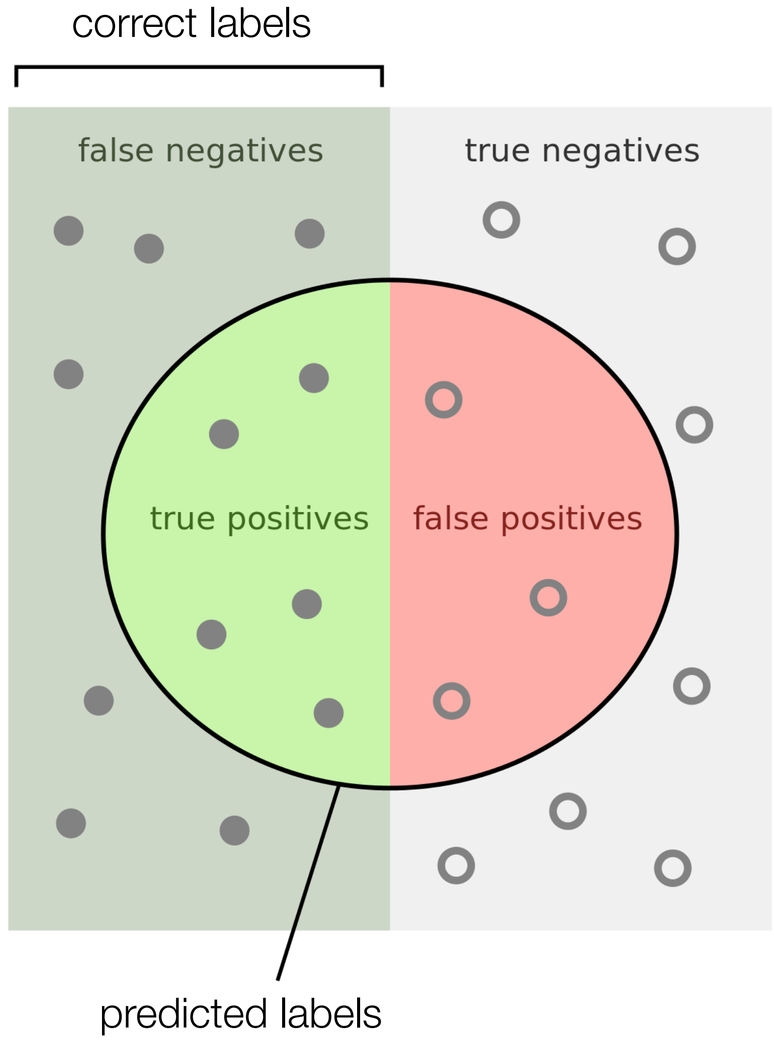


Precision and recall

- **Recall**: fraction of positive instances that were correctly classified.

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

  - The “never Telugu” classifier has 0 recall.
  - The “always Telugu” classifier has perfect recall.



Precision and recall

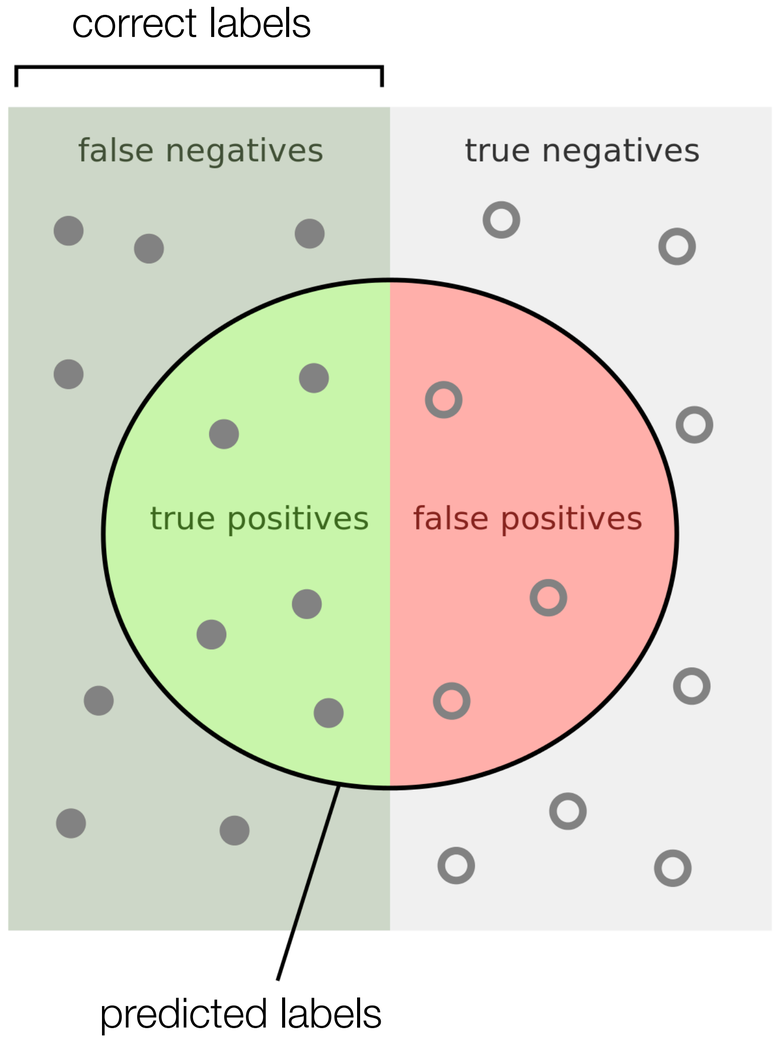
- **Recall**: fraction of positive instances that were correctly classified.

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

  - The “never Telugu” classifier has 0 recall.
  - The “always Telugu” classifier has perfect recall.

- **Precision**: fraction of positive *predictions* that were correct.

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



Precision and recall

- **Recall**: fraction of positive instances that were correctly classified.

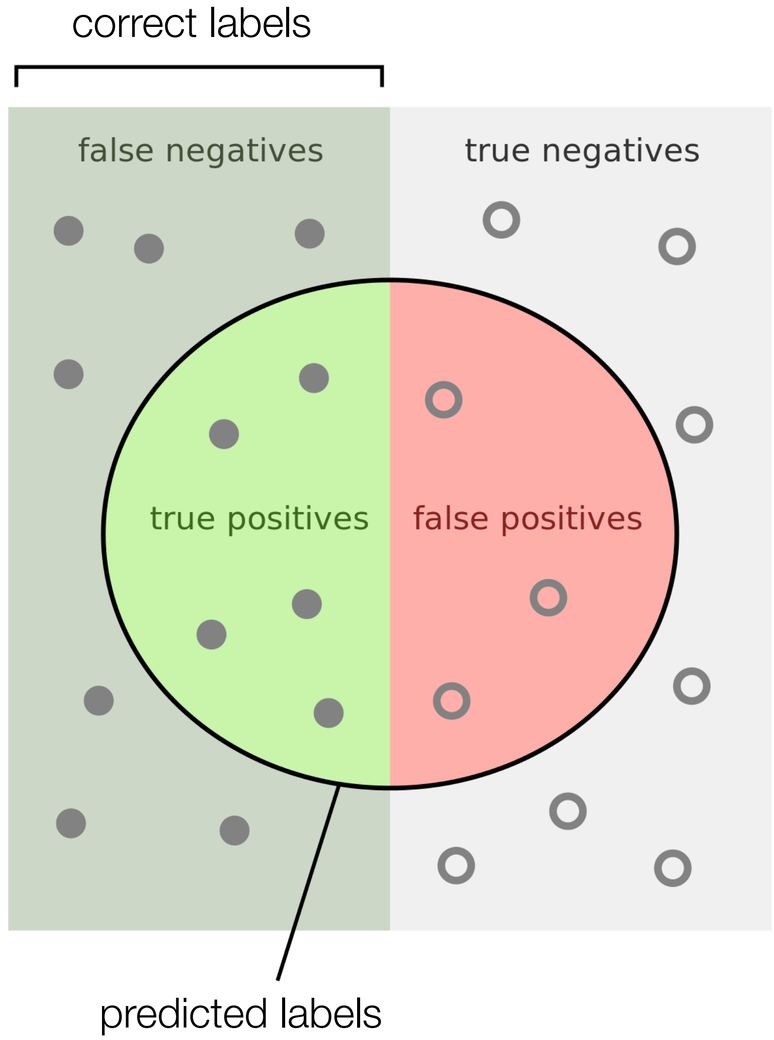
$$\text{recall} = \frac{TP}{TP + FN} =$$

  - The “never Telugu” classifier has 0 recall.
  - The “always Telugu” classifier has perfect recall.

- **Precision**: fraction of positive *predictions* that were correct.

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

  - The “never Telugu” classifier 0 precision.



Precision and recall

- **Recall**: fraction of positive instances that were correctly classified.

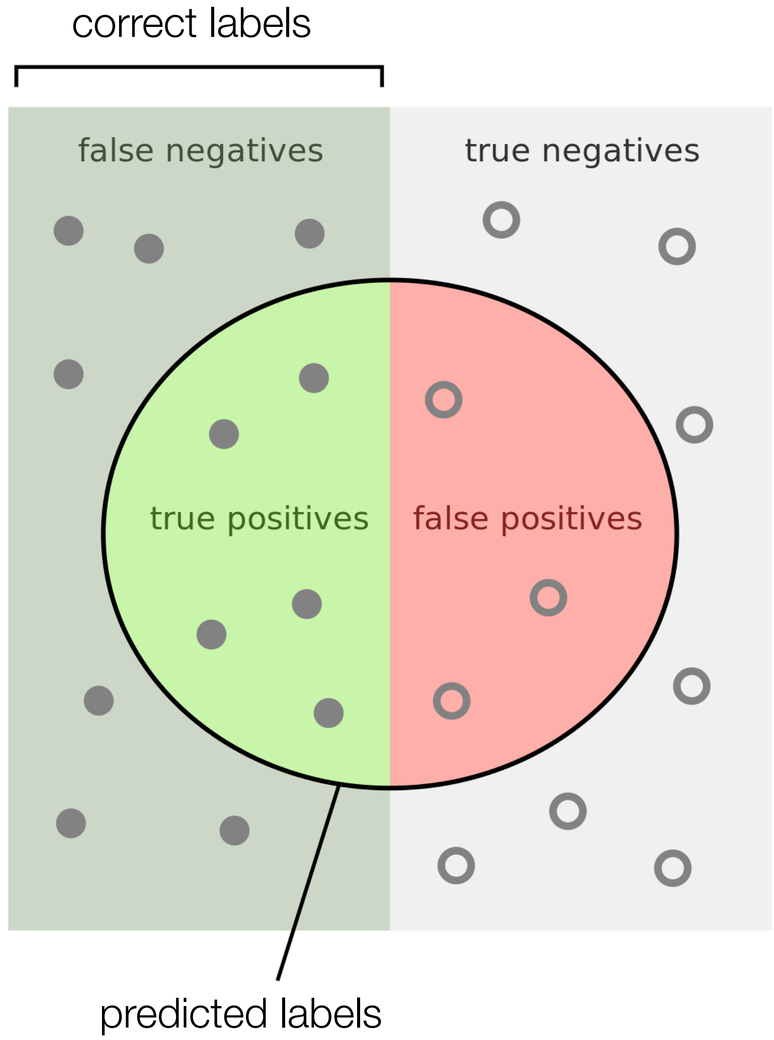
$$\text{recall} = \frac{TP}{TP + FN} =$$

  - The “never Telugu” classifier has 0 recall.
  - The “always Telugu” classifier has perfect recall.

- **Precision**: fraction of positive *predictions* that were correct.

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

  - The “never Telugu” classifier 0 precision.
  - The “always Telugu” classifier has 0.003 precision.



Combining precision and recall

Combining precision and recall

- Inherent tradeoff between precision and recall. Choice is problem-specific.

Combining precision and recall

- Inherent tradeoff between precision and recall. Choice is problem-specific.
  - For a preliminary medical diagnosis, we might prefer high recall. False positives can be screened out later.

Combining precision and recall

- Inherent tradeoff between precision and recall. Choice is problem-specific.
  - For a preliminary medical diagnosis, we might prefer high recall. False positives can be screened out later.
  - The “beyond reasonable doubt” standard of U.S. criminal law implies a preference for high precision.

Combining precision and recall

- Inherent tradeoff between precision and recall. Choice is problem-specific.
  - For a preliminary medical diagnosis, we might prefer high recall. False positives can be screened out later.
  - The “beyond reasonable doubt” standard of U.S. criminal law implies a preference for high precision.
- Most often, we weight them equally using **F_1 measure**: harmonic mean of precision and recall.

$$F_1 = \frac{2 \cdot \mathit{precision} \cdot \mathit{recall}}{\mathit{precision} + \mathit{recall}}$$

Combining precision and recall

- Inherent tradeoff between precision and recall. Choice is problem-specific.
  - For a preliminary medical diagnosis, we might prefer high recall. False positives can be screened out later.
  - The “beyond reasonable doubt” standard of U.S. criminal law implies a preference for high precision.
- Most often, we weight them equally using **F_1 measure**: harmonic mean of precision and recall.

$$F_1 = \frac{2 \cdot \mathit{precision} \cdot \mathit{recall}}{\mathit{precision} + \mathit{recall}} \qquad \min(\mathit{precision}, \mathit{recall}) \leq F_1 \leq 2 \cdot \min(\mathit{precision}, \mathit{recall})$$

Combining precision and recall

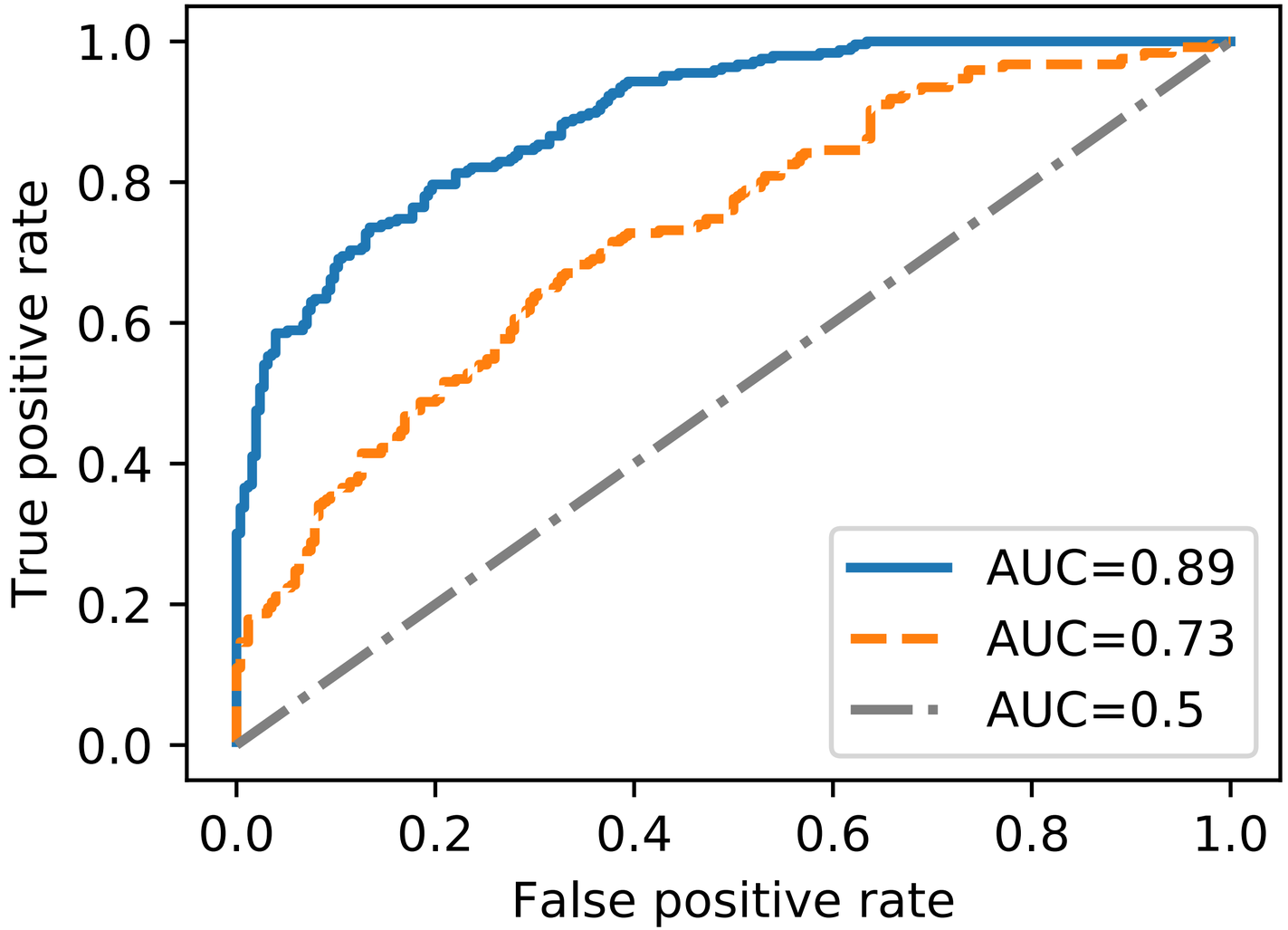
- Inherent tradeoff between precision and recall. Choice is problem-specific.
  - For a preliminary medical diagnosis, we might prefer high recall. False positives can be screened out later.
  - The “beyond reasonable doubt” standard of U.S. criminal law implies a preference for high precision.
- Most often, we weight them equally using **F_1 measure**: harmonic mean of precision and recall.

$$F_1 = \frac{2 \cdot \mathit{precision} \cdot \mathit{recall}}{\mathit{precision} + \mathit{recall}} \qquad \min(\mathit{precision}, \mathit{recall}) \leq F_1 \leq 2 \cdot \min(\mathit{precision}, \mathit{recall})$$

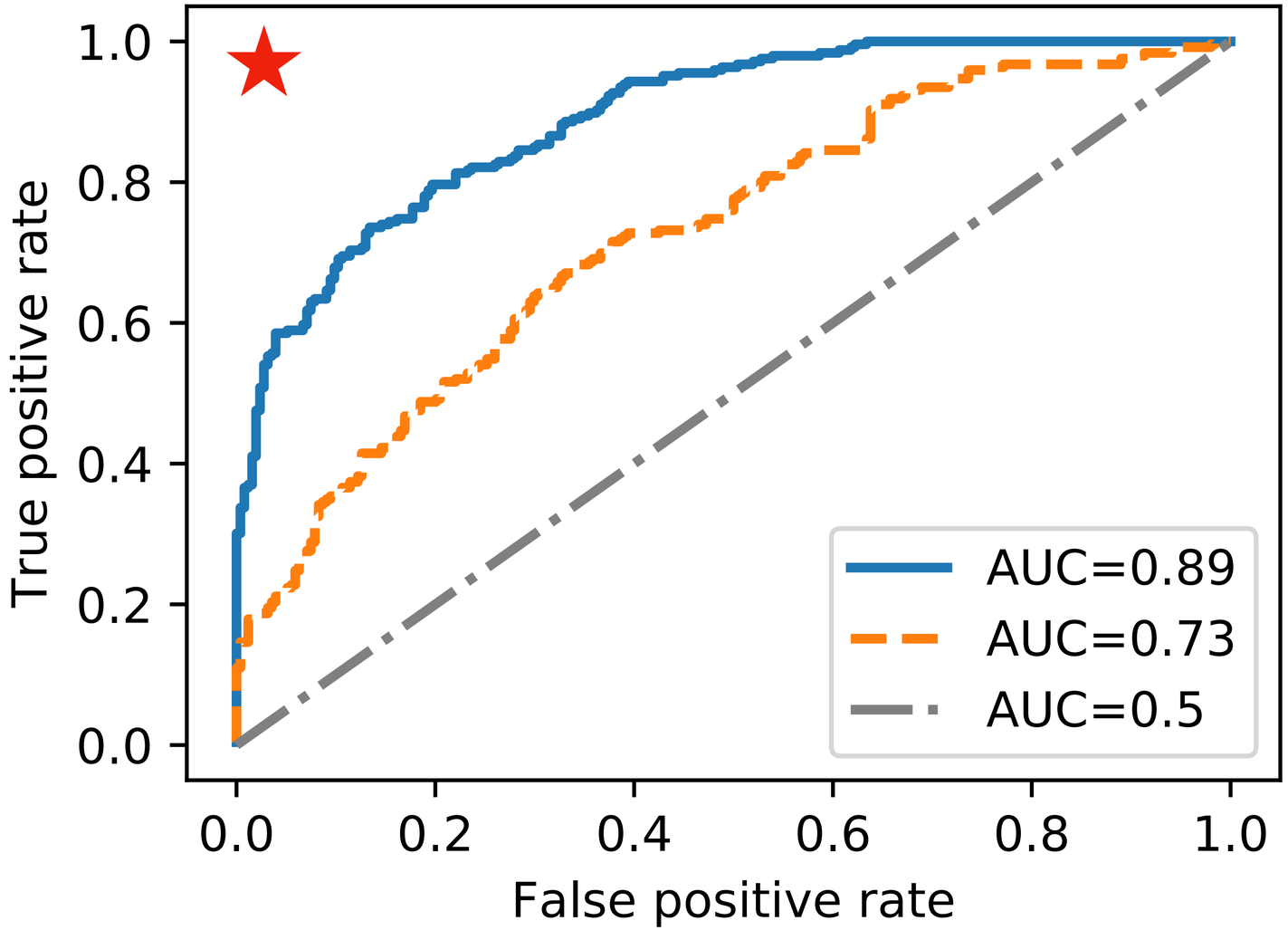
- Can generalize F-measure to adjust the tradeoff, such that recall is β-times as important as precision:

$$F_\beta = (1 + \beta^2)\frac{\mathit{precision} \cdot \mathit{recall}}{(\beta^2 \cdot \mathit{precision}) + \mathit{recall}}$$

Trading off precision and recall: ROC curve



Trading off precision and recall: ROC curve



Evaluating multi-class classification

Evaluating multi-class classification

- Recall and precision imply binary classification: each instance is either positive or negative.

Evaluating multi-class classification

- Recall and precision imply binary classification: each instance is either positive or negative.
- In multi-class classification, each instance is positive for one class, and negative for all other classes.

Evaluating multi-class classification

- Recall and precision imply binary classification: each instance is either positive or negative.
- In multi-class classification, each instance is positive for one class, and negative for all other classes.
- Two ways to combine performance across classes:

Evaluating multi-class classification

- Recall and precision imply binary classification: each instance is either positive or negative.
- In multi-class classification, each instance is positive for one class, and negative for all other classes.
- Two ways to combine performance across classes:
  - **Macro F_1**: Compute F_1 per class, then average across all classes.

Evaluating multi-class classification

- Recall and precision imply binary classification: each instance is either positive or negative.
- In multi-class classification, each instance is positive for one class, and negative for all other classes.
- Two ways to combine performance across classes:
  - **Macro F_1**: Compute F_1 per class, then average across all classes.
  - **Micro F_1**: Compute total number of true positives, false positives, false negatives pooled across all classes, and compute a single F_1.

Evaluating multi-class classification

- Recall and precision imply binary classification: each instance is either positive or negative.
- In multi-class classification, each instance is positive for one class, and negative for all other classes.
- Two ways to combine performance across classes:
  - **Macro F_1**: Compute F_1 per class, then average across all classes.

    → Weights all classes equally, regardless of frequency.
  - **Micro F_1**: Compute total number of true positives, false positives, false negatives pooled across all classes, and compute a single F_1.

Evaluating multi-class classification

- Recall and precision imply binary classification: each instance is either positive or negative.
- In multi-class classification, each instance is positive for one class, and negative for all other classes.
- Two ways to combine performance across classes:
  - **Macro F_1**: Compute F_1 per class, then average across all classes.

    → Weights all classes equally, regardless of frequency.
  - **Micro F_1**: Compute total number of true positives, false positives, false negatives pooled across all classes, and compute a single F_1.

    → Emphasizes performance on high frequency classes.

Comparing classifiers

Comparing classifiers

- Suppose two teams build classifiers to solve a problem:

Comparing classifiers

- Suppose two teams build classifiers to solve a problem:
  - C_1 gets 82% accuracy

Comparing classifiers

- Suppose two teams build classifiers to solve a problem:
  - C_1 gets 82% accuracy
  - C_2 gets 73% accuracy

Comparing classifiers

- Suppose two teams build classifiers to solve a problem:
  - C_1 gets 82% accuracy
  - C_2 gets 73% accuracy
- Will C_1 be more accurate in the future?

Comparing classifiers

- Suppose two teams build classifiers to solve a problem:
  - C_1 gets 82% accuracy
  - C_2 gets 73% accuracy
- Will C_1 be more accurate in the future?
  - What if the test set has 1000 examples?

Comparing classifiers

- Suppose two teams build classifiers to solve a problem:
  - C_1 gets 82% accuracy
  - C_2 gets 73% accuracy
- Will C_1 be more accurate in the future?
  - What if the test set has 1000 examples?
  - What if the test set has 11 examples?

Hypothesis testing

Hypothesis testing

- Consider two hypotheses that explain the observed data:

Hypothesis testing

- Consider two hypotheses that explain the observed data:
  - H_1: C_1 is more accurate than C_2 and therefore can be expected to be more accurate in the future (in the limit of an infinite number of independent evaluations).

Hypothesis testing

- Consider two hypotheses that explain the observed data:
  - H_1: C_1 is more accurate than C_2 and therefore can be expected to be more accurate in the future (in the limit of an infinite number of independent evaluations).
  - H_0: C_1 is not more accurate than C_2 , and its superior performance on the test set was due only to luck. This is the **null hypothesis**.

Hypothesis testing

- Consider two hypotheses that explain the observed data:
  - H_1: C_1 is more accurate than C_2 and therefore can be expected to be more accurate in the future (in the limit of an infinite number of independent evaluations).
  - H_0: C_1 is not more accurate than C_2 , and its superior performance on the test set was due only to luck. This is the **null hypothesis**.
- If the test set is small, H_0 might be true.

Hypothesis testing

- Consider two hypotheses that explain the observed data:
  - H_1: C_1 is more accurate than C_2 and therefore can be expected to be more accurate in the future (in the limit of an infinite number of independent evaluations).
  - H_0: C_1 is not more accurate than C_2 , and its superior performance on the test set was due only to luck. This is the **null hypothesis**.
- If the test set is small, H_0 might be true.
- If the test set is large, probability of observing a 9% difference in accuracy (73% → 82%) becomes vanishingly small unless C_1 really is more accurate.

Hypothesis testing

- Consider two hypotheses that explain the observed data:
  - H_1: C_1 is more accurate than C_2 and therefore can be expected to be more accurate in the future (in the limit of an infinite number of independent evaluations).
  - H_0: C_1 is not more accurate than C_2 , and its superior performance on the test set was due only to luck. This is the **null hypothesis**.
- If the test set is small, H_0 might be true.
- If the test set is large, probability of observing a 9% difference in accuracy (73% → 82%) becomes vanishingly small unless C_1 really is more accurate.
- These probabilities are quantified by hypothesis testing.

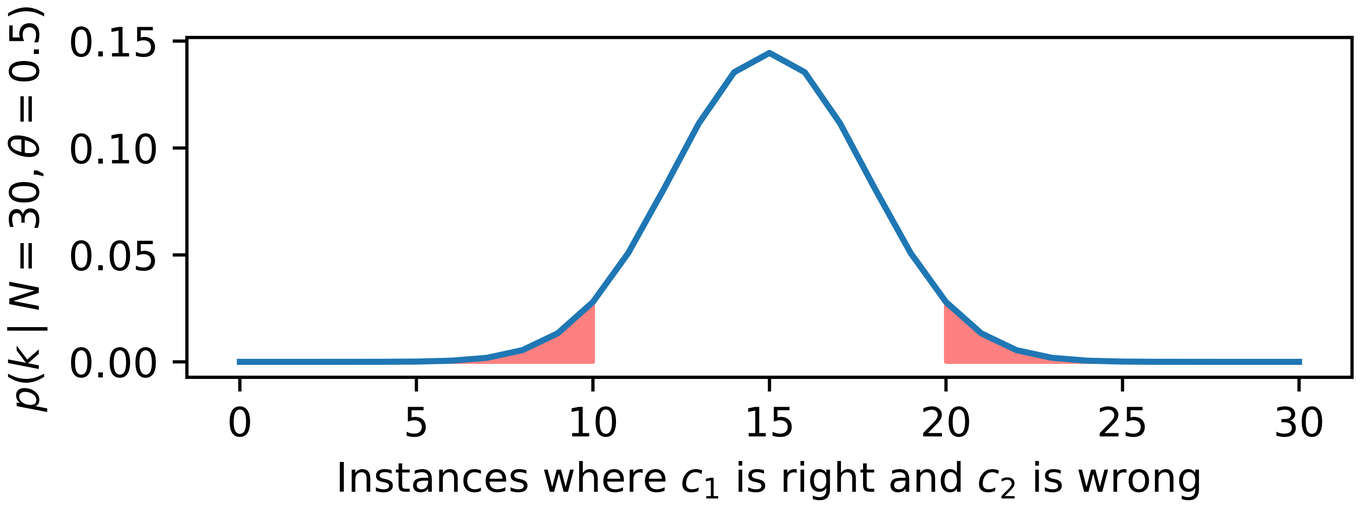
The binomial test

The binomial test

- If the two classifiers are equally accurate, then each time they disagree, they are equally likely to be correct.

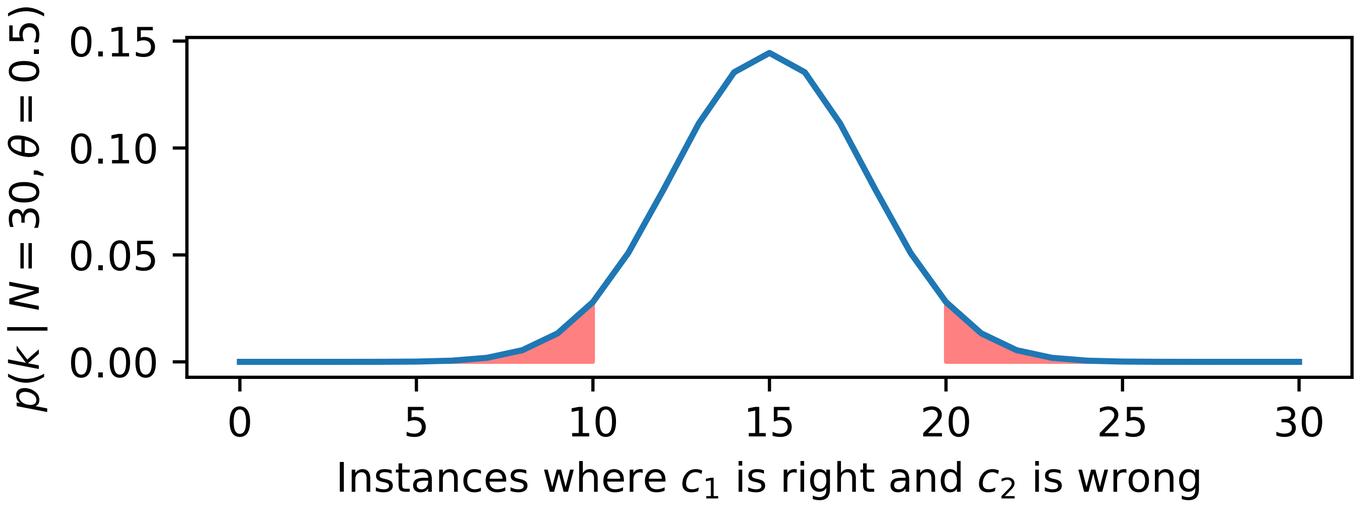
The binomial test

- If the two classifiers are equally accurate, then each time they disagree, they are equally likely to be correct.
- Over 30 such disagreements, each classifier will “win” roughly half the time.



The binomial test

- If the two classifiers are equally accurate, then each time they disagree, they are equally likely to be correct.
- Over 30 such disagreements, each classifier will “win” roughly half the time.



- The total probability mass in the pink region is less than 5%. If the data are in this region, we reject the null hypothesis with $p < 0.05$.

Other hypothesis tests

Other hypothesis tests

- The binomial test compares two classifiers in terms of accuracy. It can be computed in closed form using e.g. SciPy or R.

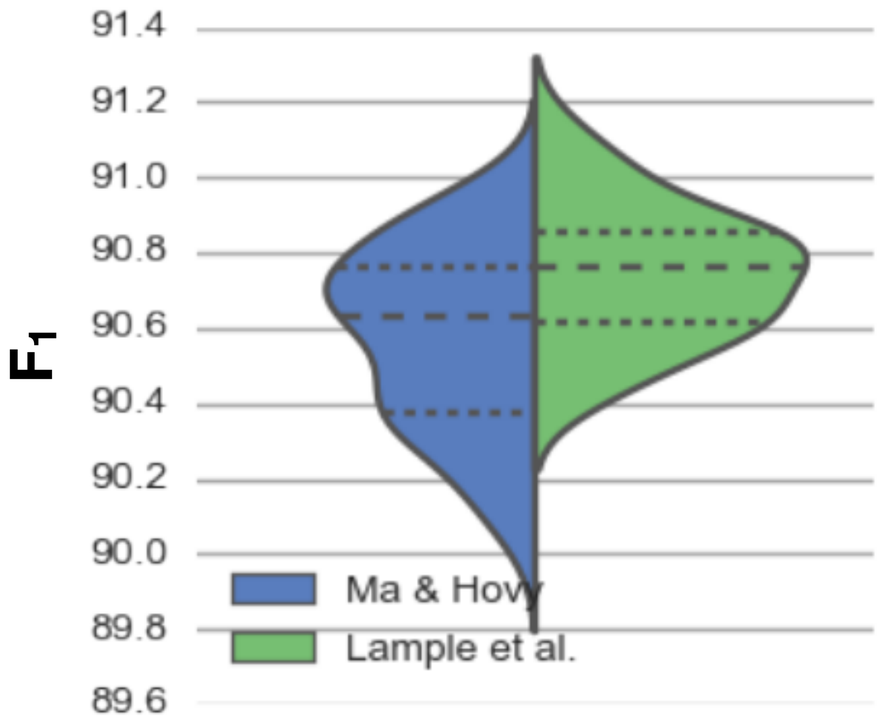
Other hypothesis tests

- The binomial test compares two classifiers in terms of accuracy. It can be computed in closed form using e.g. SciPy or R.
- Hypotheses about other metrics, such as F_1, cannot be tested in this way.

Other hypothesis tests

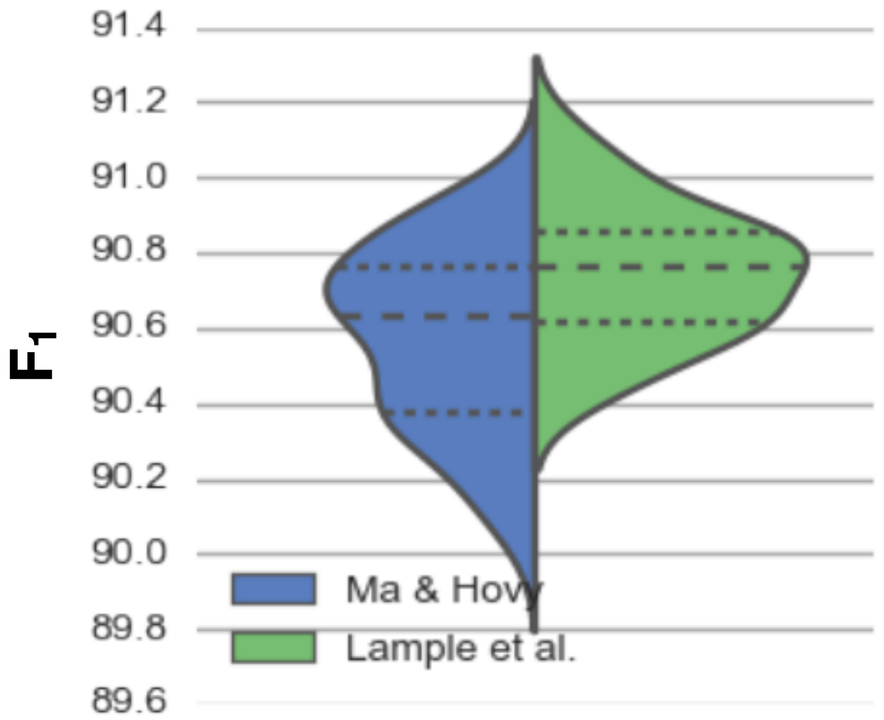
- The binomial test compares two classifiers in terms of accuracy. It can be computed in closed form using e.g. SciPy or R.
- Hypotheses about other metrics, such as F_1, cannot be tested in this way.
- For these hypotheses, best approach is randomization: randomly sample many test sets, and count how often each hypothesis holds.

Comparing classifiers



Comparing classifiers

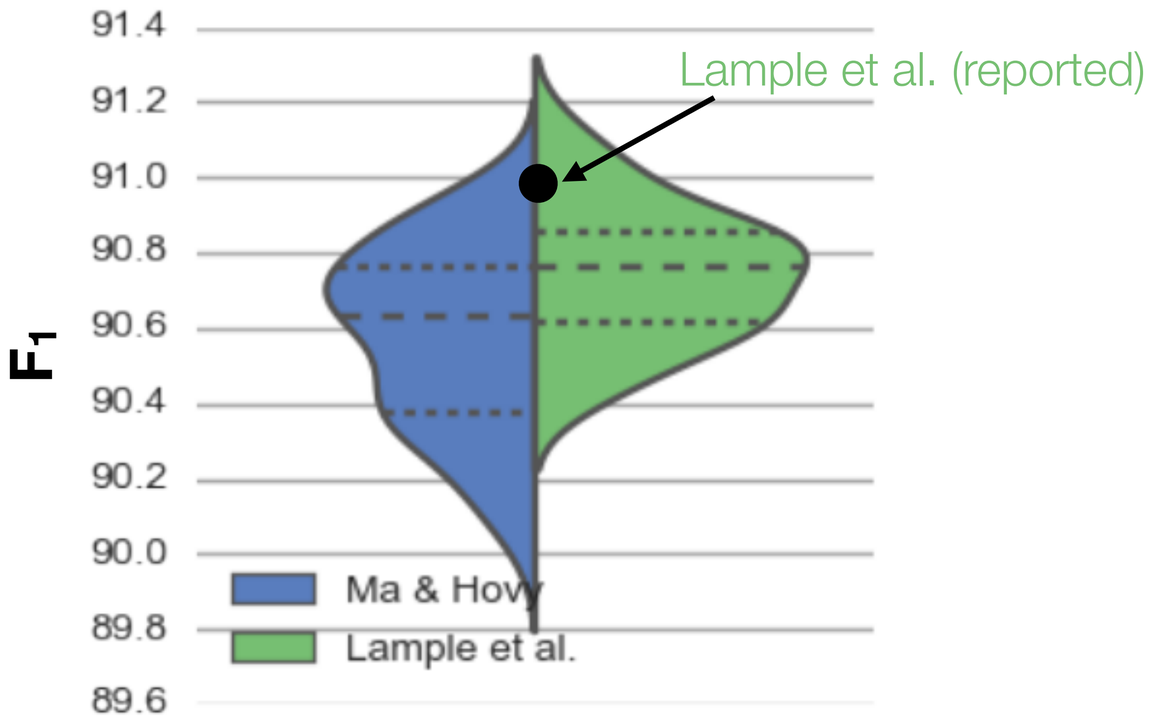
Two recent, state-of-the-art systems for NER are proposed by Ma and Hovy (2016)[5] and by Lample et al. (2016)[6]. Lample et al. report an F_1-score of 90.94% and Ma and Hovy report an F_1-score of 91.21%. Ma and Hovy draw the conclusion that their system achieves a significant improvement over the system by Lample et al.



Nils Reimers and Iryna Gurevych. Reporting score distributions makes a difference: Performance study of LSTM-networks for sequence tagging. EMNLP 2017.

Comparing classifiers

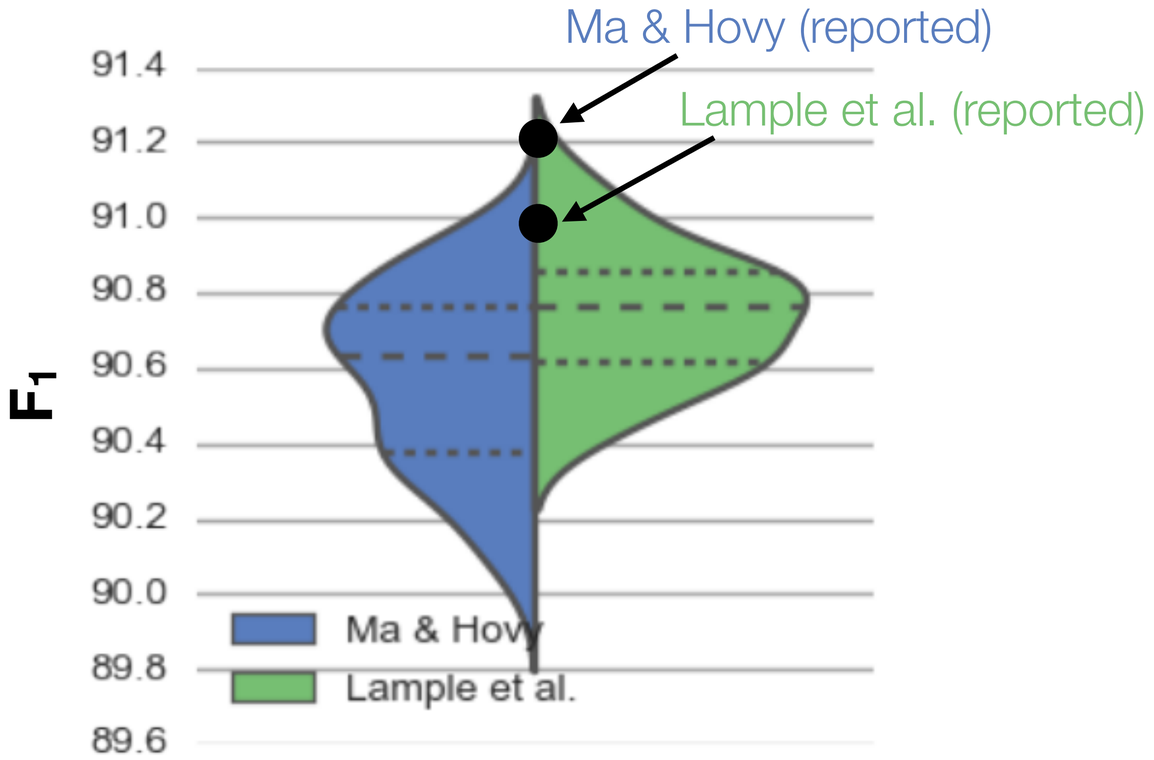
Two recent, state-of-the-art systems for NER are proposed by Ma and Hovy (2016)[5] and by Lample et al. (2016)[6]. Lample et al. report an F_1-score of 90.94% and Ma and Hovy report an F_1-score of 91.21%. Ma and Hovy draw the conclusion that their system achieves a significant improvement over the system by Lample et al.



Nils Reimers and Iryna Gurevych. Reporting score distributions makes a difference: Performance study of LSTM-networks for sequence tagging. EMNLP 2017.

Comparing classifiers

Two recent, state-of-the-art systems for NER are proposed by Ma and Hovy (2016)[5] and by Lample et al. (2016)[6]. Lample et al. report an F_1-score of 90.94% and Ma and Hovy report an F_1-score of 91.21%. Ma and Hovy draw the conclusion that their system achieves a significant improvement over the system by Lample et al.



Nils Reimers and Iryna Gurevych. Reporting score distributions makes a difference: Performance study of LSTM-networks for sequence tagging. EMNLP 2017.

Announcements

- **Project 1: Text classification** will be available after class today, **due Friday September 25**
- Han will lead recitation this Friday to introduce Project 1, and cover environment setup (Python, Jupyter, NumPy, PyTorch).