CSC 411: Lecture 3 - Linear Classification

Ethan Fetaya, James Lucas and Emad Andrews

This lecture:

- Linear classification (binary).
- First order optimization.
- Key concepts:
  - Decision boundaries.
  - Loss functions.
  - metrics to evaluate classification.
  - Stochastic gradient descent.

Last week: Mapping $\mathbf{x} \in \mathbb{R}^d$ into $y \in \mathbb{R}$.

This week: Mapping $\mathbf{x} \in \mathbb{R}^d$ into categorical y (in a finite set S).
Usually use $S = \{1, .., k\}$, $S = \{0, 1\}$ or $S = \{-1, 1\}$ (our focus now).

Linear model: $\hat{y} = f(\mathbf{x}, \mathbf{w}) = \mathbf{w}^T \mathbf{x}$ outputs a real score.
How do we turn it into a binary decision? Threshold -

$$\hat{y} = f(\mathbf{x}, \mathbf{w}) = \text{sign}(\mathbf{w}^T \mathbf{x}) = \begin{cases} 1 & \text{if } \mathbf{w}^T \mathbf{x} \geq 0 \\ -1 & \text{if } \mathbf{w}^T \mathbf{x} < 0 \end{cases}$$

Decision boundary is the hyperspace defined by $\mathbf{w}$.

$\mathbf{w}^T\mathbf{x} = 0$ is a hyperplane (line in $d = 2$) passing though the origin and orthogonal to $\mathbf{w}$. $\mathbf{w}^T\mathbf{x} + w_0 = 0$ shifts it by w_0.

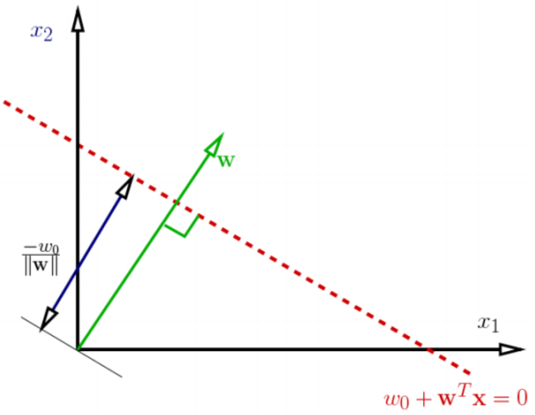

x_2
$\mathbf{w}$
$\frac{-w_0}{\|\mathbf{w}\|}$
x_1
$w_0 + \mathbf{w}^T\mathbf{x} = 0$

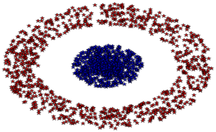
Figure from G. Shakhnarovich

Decision boundary is invariant to scaling.

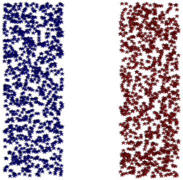
If we can separate the classes by a hyperplane, the problem is linearly separable

Causes of non perfect separation:

- Model is too simple.
- Noise (optimal classifier might not be perfect).
- Errors in data targets (miss labelings).
- Simple features that do not account for all variations.
- Need different feature parametrization.



X,Y coordinates



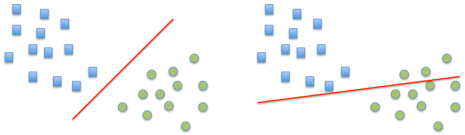
Polar coordinates

Should we make the model complex enough to have perfect separation in the training data?

Learning consists of finding a good decision boundary.

We need to find **w** (direction) and w_0 (location) of the boundary.

What does "good" mean? Is this boundary good?



We need a criteria that tell us how to select the parameters.

A natural loss function: zero-one loss. $\ell_{0-1}(\hat{y}, y) = \begin{cases} 1 & \text{if } y \neq \hat{y} \\ 0 & \text{if } y = \hat{y} \end{cases}$

Is this minimization easy to do? Why?

Asymmetric Binary Loss: Should we treat both types of mistakes equally? $\ell_{ABL}(\hat{y}, y) = \begin{cases} \alpha & \text{if } y = 0 \wedge \hat{y} = 1 \\ \beta & \text{if } y = 1 \wedge \hat{y} = 0 \\ 0 & \text{if } y = \hat{y} \end{cases}$

When is this important?

Goal: Optimizing ℓ_{0-1} (or ℓ_{ABL}).

Problem: (NP)hard, piecewise constant.

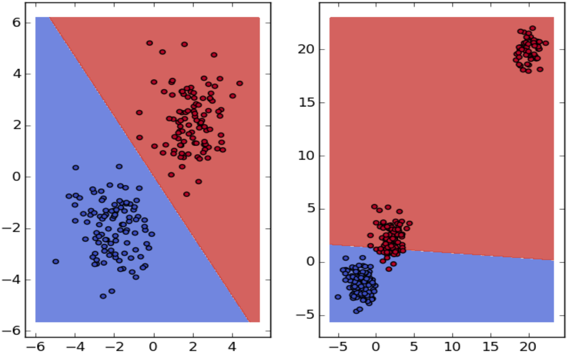
Approach: use a surrogate loss $\hat{\ell}$ to optimize instead.

What makes a good surrogate loss?

- Easy to optimize
  - (Piecewise) Smooth.
  - Convex.
- Representative - low surrogate loss means low original loss.
  - Upper bound $\forall y \forall \hat{y} \ell(y, \hat{y}) \leq \hat{\ell}(y, \hat{y})$.

Is $\ell_2(y, \hat{y}) = (y - \hat{y})^2$ loss a good surrogate?

Easy to optimize? ✓ Representative? so-so



We will see better surrogates soon.

How to evaluate how good my classifier is? Metrics

- Metrics on a dataset is what we care about (performance).
- We typically cannot directly optimize for the metrics.
- Our loss function should reflect the problem we are solving. We then hope it will yield models that will do well on our dataset.

Accuracy: Percent of correct predictions, $1 - \ell_{0-1}(w)$.

Is it a good measure? Data balanced? Unbalanced?

Recall: The fraction of relevant instances that are retrieved.

$$R = \frac{TP}{TP + FN} = \frac{TP}{\text{all groundtruth instances}}$$

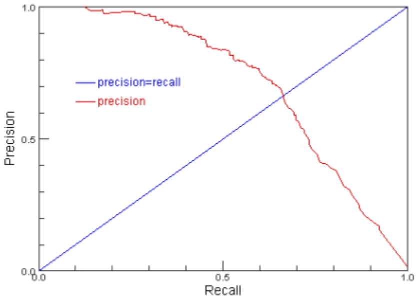
Precision: The fraction of retrieved instances that are correct.

$$P = \frac{TP}{TP + FP} = \frac{TP}{\text{all positive predictions}}$$

F1 score: Harmonic mean of precision and recall.

$$F1 = 2\frac{P \cdot R}{P + R}$$

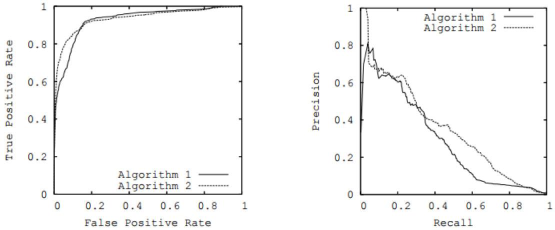
Precision-Recall curve: Trade-off between recall and precision using the decision threshold.



Average Precision (AP): area under the curve.

We might be interest in a single working point (recall or precision).

Receiver Operator Characteristic (ROC): Trade-off between false-positive-rate (FPR) and true-positive-rate (TPR) using the decision threshold.



(a) Comparison in ROC space

(b) Comparison in PR space

Better in ROC ⇒ better in PR (not always vice-versa).

Difference can be big with unbalanced data

[1] Figure from ”The Relationship Between Precision-Recall and ROC Curves”

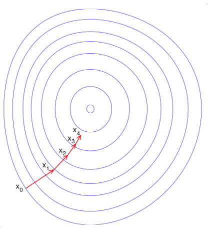
Gradient descent

Once we decide on a (smooth) loss ℓ - how do we find $\mathbf{w} = \arg\min L(\mathbf{w}) = \frac{1}{N}\sum_{i=1}^{N} \ell(y_i, f(x_i, \mathbf{w}))$?

One straightforward method: gradient descent

- initialize $\mathbf{w}_0$ (e.g., randomly)
- repeatedly update $\mathbf{w}$ based on the gradient

$$\mathbf{w}_{t+1} = \mathbf{w}_t - \lambda_t \nabla_{\mathbf{w}} L(\mathbf{w}_t)$$

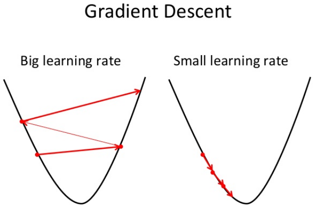


λ is the learning rate.

Update rule: $\mathbf{w}_{t+1} = \mathbf{w}_t - \lambda_t \nabla_{\mathbf{w}} L(\mathbf{w}_t)$

Finding a good learning rate is very important.

- Too large λ: unstable and can diverge.
- Too low λ: stable but very slow progress.
- Line search methods - usually too slow.
- Standard to decay λ as learning progresses.



Commonly found using simple grid search, some automatic tools exist.

[1] Image credit: https://www.slideshare.net/simaokasonse/learning-deep-learning.

What is the computational cost of computing $\nabla_{\mathbf{w}} L(\mathbf{w}_t)$?

$L(\mathbf{w}) = \frac{1}{N}\sum_{i=1}^{N} \ell(y_i, f(x_i, \mathbf{w}))$ - grows linearly in N (number of data points).

Huge (millions/billions) dataset ⇒ large cost for a tiny update!

Solution: Stochastic gradient descent. Instead of computing gradient $g_t = \nabla_{\mathbf{w}} L(\mathbf{w}_t) = \frac{1}{N}\sum_{i=1}^{N} \nabla \ell(y_i, f(x_i, \mathbf{w}))$, pick random datum j and compute $\hat{g}_t = \nabla \ell(y_j, f(x_j, \mathbf{w}))$

Will it work? Theoretically - yes (with the right learning rate decay). Practically - very noisy.

Better solution: Mini-batch. Middle-ground, average $1 < m << N$ gradients.

Mean is still g_t but variance is lower. Trade-off between accuracy (big batch) and runtime (small batch).

Algorithm 1 Mini-batch gradient descent epoch

```
Randomly shuffle examples in the training set
for $i = 0$ to $N/m$ do
    Update:
        $\mathbf{w} \leftarrow \mathbf{w} + \frac{1}{m} \sum_{j=0}^{m-1} \nabla \ell(y^{m \cdot i+j}, f(x^{m \cdot i+j}, \mathbf{w}))$
end for
```

This simple idea is a important component behind a lot of recent success.

People commonly use the term SGD for mini-batch optimization.