Nearest Neighbors (continued), Perceptron

Sriram Sankararaman

The instructor gratefully acknowledges Fei Sha, Ameet Talwalkar, Eric Eaton, and Jessica Wu whose slides are heavily used, and the many others who made their course material freely available online.

Announcements

- If you attempted mini-quiz and problem set 0, you should have received a PTE.
- If not, talk to one of the TAs at the end of lecture.

Outline

Nearest neighbor classification (NNC)

Training

- Store the entire training set.

Nearest neighbor classification (NNC)

Testing

$$\boldsymbol{x}(1) = \boldsymbol{x}_{\mathsf{nn}(\boldsymbol{x})}$$

where $\mathsf{nn}(\boldsymbol{x}) \in [\mathsf{N}] = \{1, 2, \cdots, \mathsf{N}\}$, i.e., the index to one of the training instances

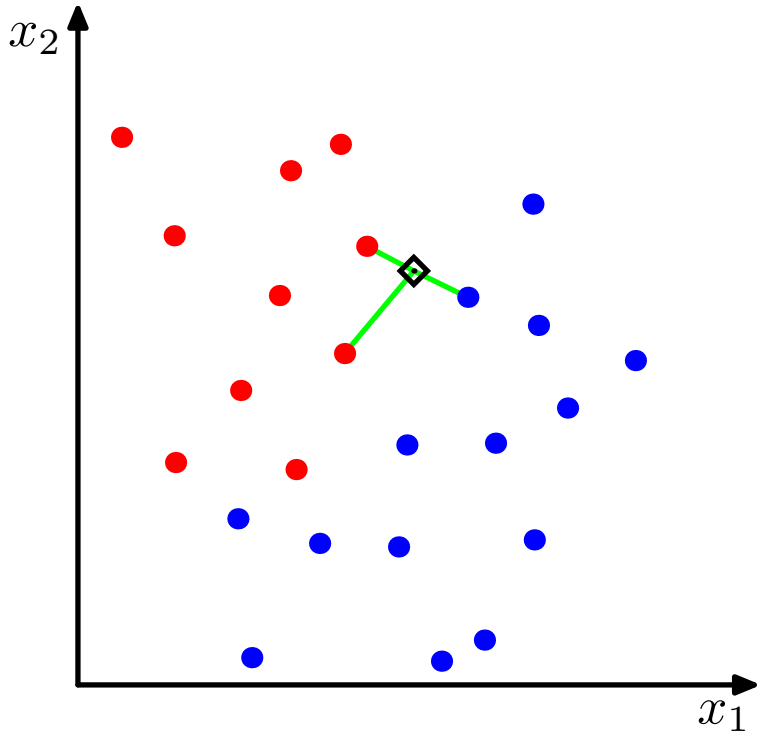
$$\mathsf{nn}(\boldsymbol{x}) = \arg\min_{n \in [\mathsf{N}]} \|\boldsymbol{x} - \boldsymbol{x}_n\|_2^2 = \arg\min_{n \in [\mathsf{N}]} \sum_{d=1}^{\mathsf{D}} (x_d - x_{nd})^2$$

Classification rule

$$y = h(\boldsymbol{x}) = y_{\mathsf{nn}(\boldsymbol{x})}$$

Visual example

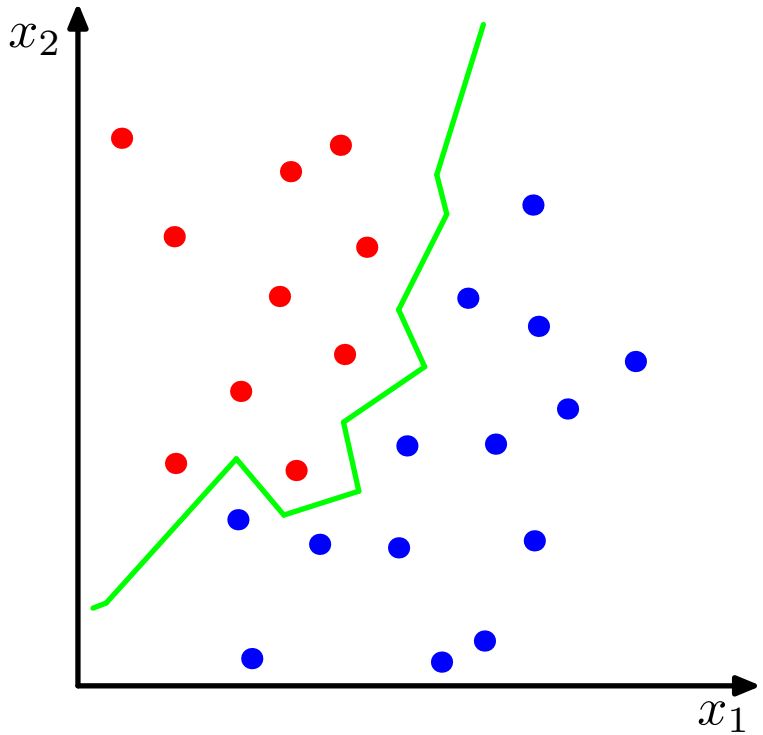
In this 2-dimensional example, the nearest point to $\boldsymbol{x}$ is a red training instance, thus, $\boldsymbol{x}$ will be labeled as red.



(a)

Decision boundary

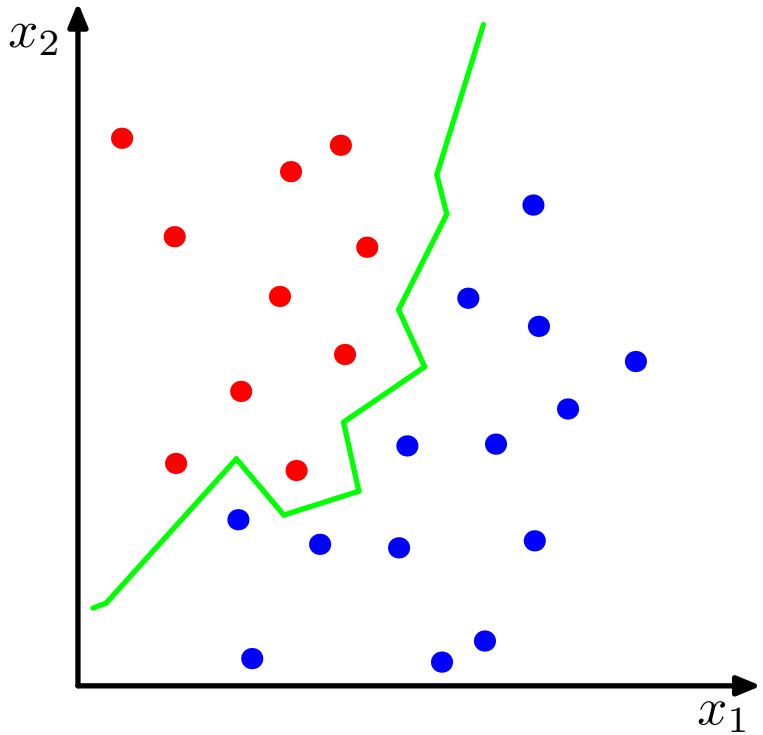
For every point in the space, determine its label using the NNC rule. Gives rise to a *decision boundary* that partitions the space into different regions.



(b)

Decision boundary

For every point in the space, determine its label using the NNC rule. Gives rise to a *decision boundary* that partitions the space into different regions.



(b)

Compare to decision boundary of decision trees

K-nearest neighbor (KNN) classification

Increase the number of nearest neighbors to use?

- 1-nearest neighbor: $\text{nn}_1(\boldsymbol{x}) = \arg\min_{n \in [\text{N}]} \|\boldsymbol{x} - \boldsymbol{x}_n\|_2^2$
- 2nd-nearest neighbor: $\text{nn}_2(\boldsymbol{x}) = \arg\min_{n \in [\text{N}] - \text{nn}_1(\boldsymbol{x})} \|\boldsymbol{x} - \boldsymbol{x}_n\|_2^2$
- 3rd-nearest neighbor: $\text{nn}_3(\boldsymbol{x}) = \arg\min_{n \in [\text{N}] - \text{nn}_1(\boldsymbol{x}) - \text{nn}_2(\boldsymbol{x})} \|\boldsymbol{x} - \boldsymbol{x}_n\|_2^2$

K-nearest neighbor (KNN) classification

Increase the number of nearest neighbors to use?

- 1-nearest neighbor: $\text{nn}_1(\boldsymbol{x}) = \arg\min_{n \in [\text{N}]} \|\boldsymbol{x} - \boldsymbol{x}_n\|_2^2$
- 2nd-nearest neighbor: $\text{nn}_2(\boldsymbol{x}) = \arg\min_{n \in [\text{N}] - \text{nn}_1(\boldsymbol{x})} \|\boldsymbol{x} - \boldsymbol{x}_n\|_2^2$
- 3rd-nearest neighbor: $\text{nn}_3(\boldsymbol{x}) = \arg\min_{n \in [\text{N}] - \text{nn}_1(\boldsymbol{x}) - \text{nn}_2(\boldsymbol{x})} \|\boldsymbol{x} - \boldsymbol{x}_n\|_2^2$

The set of K-nearest neighbors

$$\text{knn}(\boldsymbol{x}) = \{\text{nn}_1(\boldsymbol{x}), \text{nn}_2(\boldsymbol{x}), \cdots, \text{nn}_K(\boldsymbol{x})\}$$

Let $\boldsymbol{x}(k) = \boldsymbol{x}_{\text{nn}_k(\boldsymbol{x})}$, then

$$\|\boldsymbol{x} - \boldsymbol{x}(1)\|_2^2 \leq \|\boldsymbol{x} - \boldsymbol{x}(2)\|_2^2 \cdots \leq \|\boldsymbol{x} - \boldsymbol{x}(K)\|_2^2$$

How to classify with K neighbors?

How to classify with K neighbors?

Classification rule

- Every neighbor votes: suppose y_n (the label) for $\boldsymbol{x}_n$ is c, then
  - vote for c is 1
  - vote for $c' \neq c$ is 0

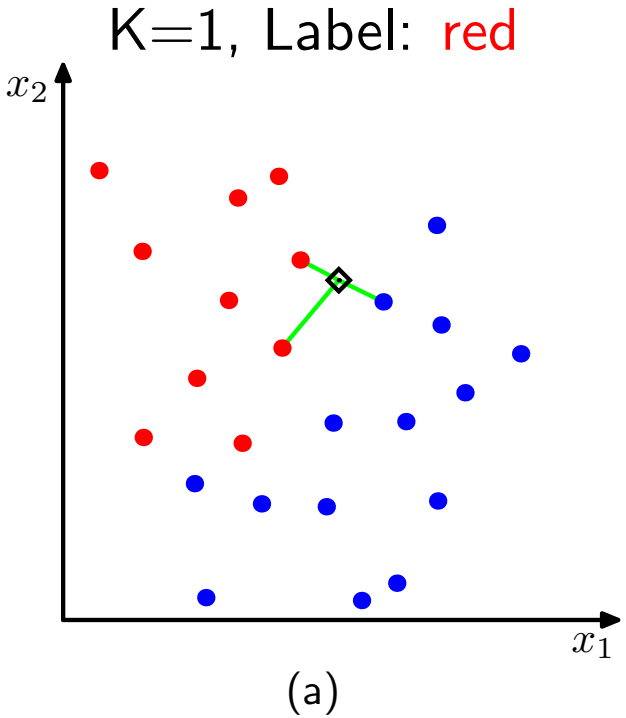
  We use the *indicator function* $\mathbb{I}(y_n == c)$ to represent.
- Aggregate everyone's vote

$$v_c = \sum_{n \in \mathsf{knn}(\boldsymbol{x})} \mathbb{I}(y_n == c), \quad \forall \quad c \in [\mathsf{C}]$$

- Label with the majority

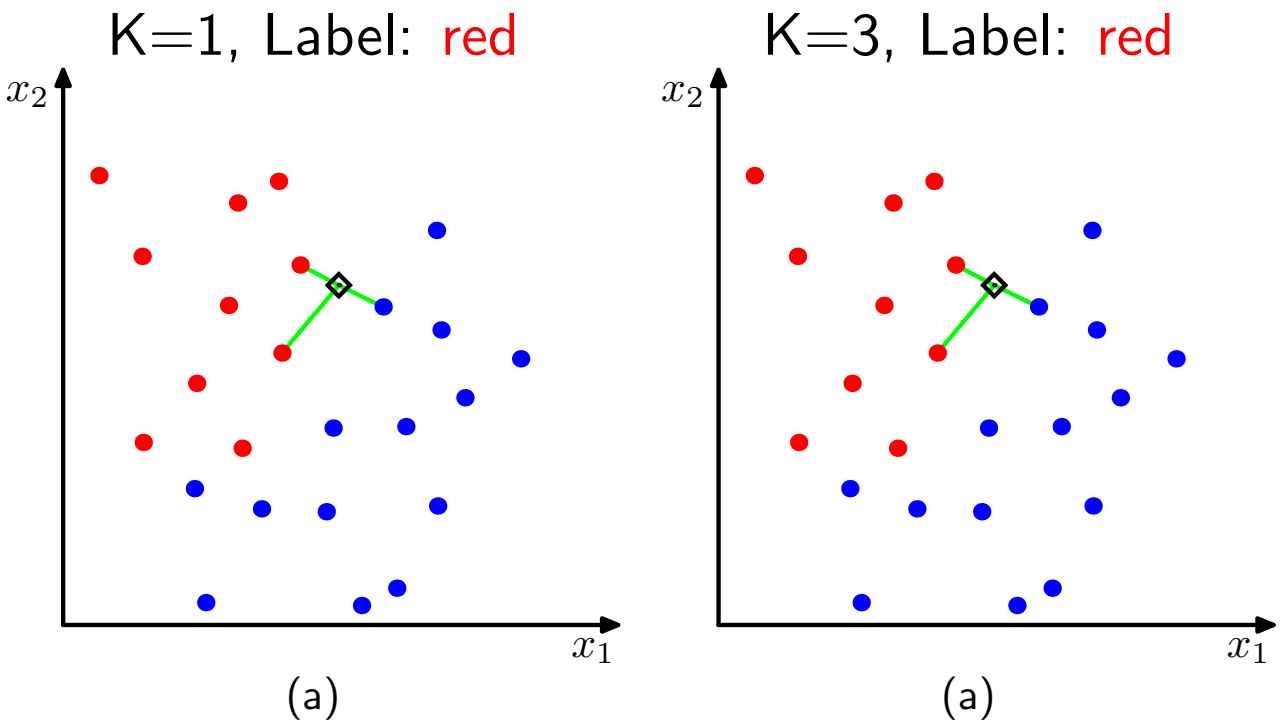
$$y = h(\boldsymbol{x}) = \arg\max_{c \in [\mathsf{C}]} v_c$$

Example

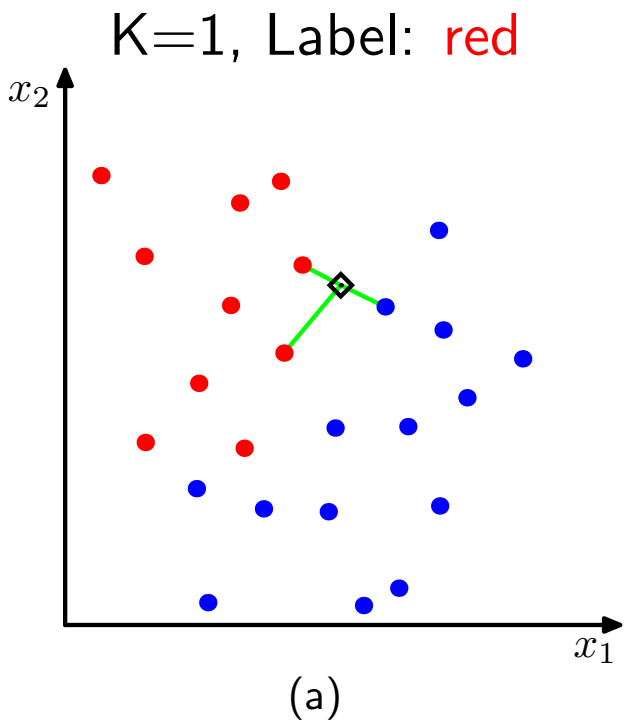


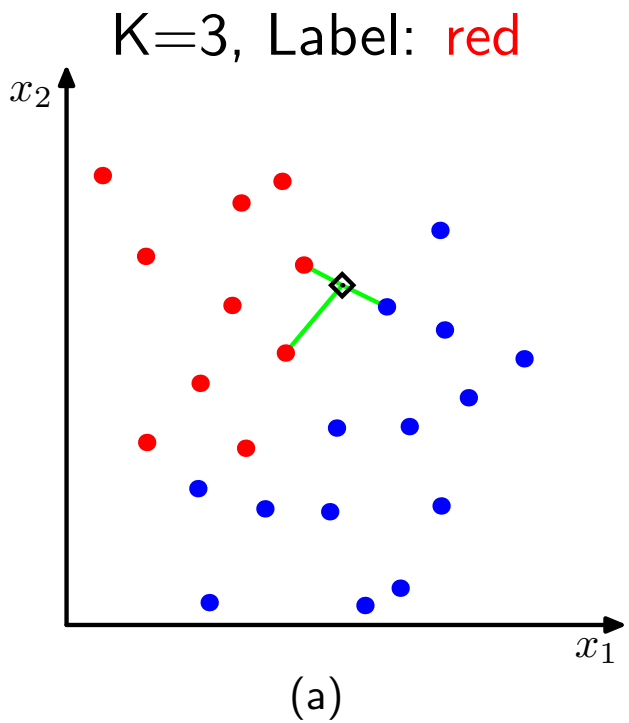
(a)

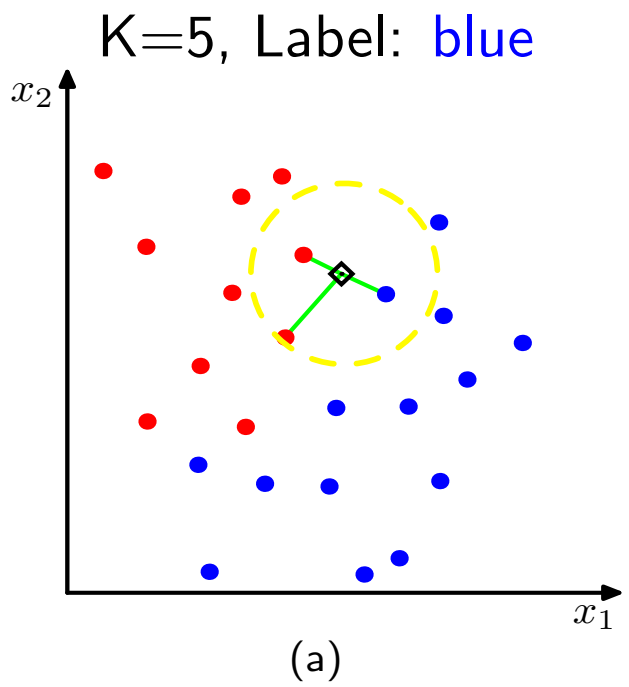
Example



Example

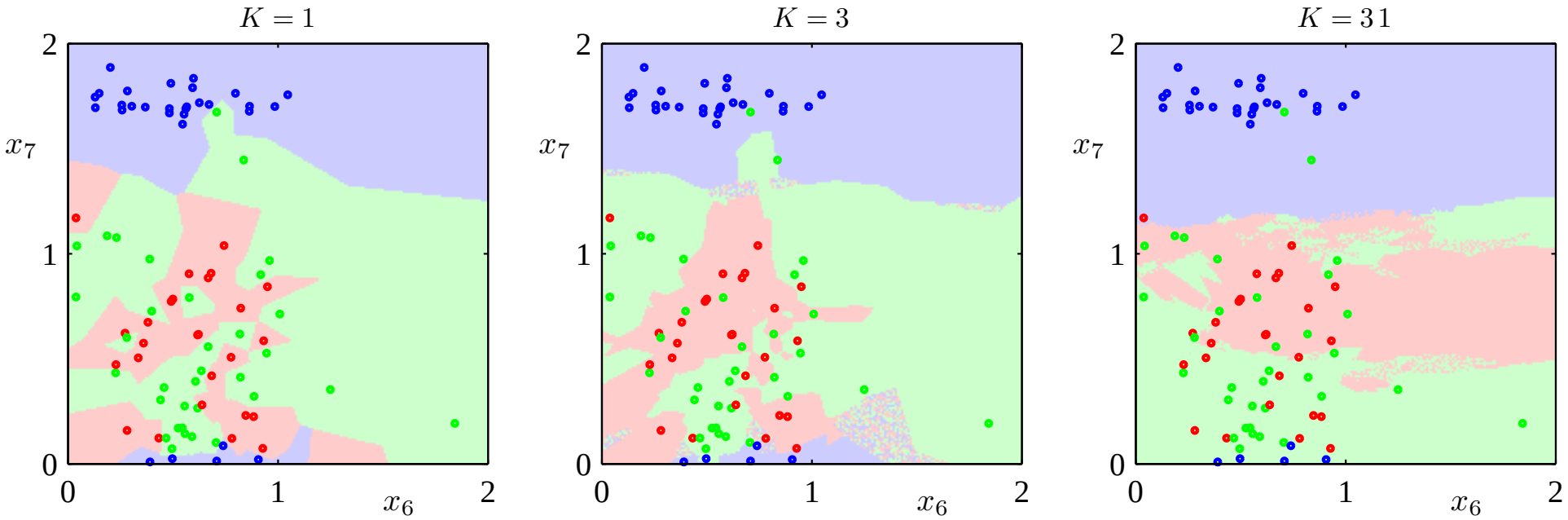






Decision boundary as a function of K

Decision boundary as a function of K



When K increases, the decision boundary becomes smooth.

Mini-summary

Advantages of NNC

- Computationally, simple and easy to implement – just computing the distance

Disadvantages of NNC

- Computationally intensive for large-scale problems: $O(\mathrm{N}D)$ for labeling a data point
- We need to "carry" the training data around. Without it, we cannot do classification. This type of method is called *nonparametric*.
- Choosing the right distance measure and K can be involved.

Hypeparameters in NNC

Two practical issues about NNC

- Choosing K, i.e., the number of nearest neighbors (default is 1)
- Choosing the right distance measure (default is Euclidean distance), for example, from the following generalized distance measure

$$\|\boldsymbol{x} - \boldsymbol{x}_n\|_p = \left(\sum_d |x_d - x_{nd}|^p \right)^{1/p}$$

for $p \geq 1$.

Those are not specified by the algorithm itself — resolving them requires empirical studies and are task/dataset-specific.

Tuning by using a validation dataset

Training data (set)

- N samples/instances: $\mathcal{D}^{\text{TRAIN}} = \{(\boldsymbol{x}_1, y_1), (\boldsymbol{x}_2, y_2), \cdots, (\boldsymbol{x}_{\text{N}}, y_{\text{N}})\}$
- They are used for learning $h(\cdot)$

Test (evaluation) data

- M samples/instances: $\mathcal{D}^{\text{TEST}} = \{(\boldsymbol{x}_1, y_1), (\boldsymbol{x}_2, y_2), \cdots, (\boldsymbol{x}_{\text{M}}, y_{\text{M}})\}$
- They are used for assessing how well $h(\cdot)$ will do in predicting an unseen $\boldsymbol{x} \notin \mathcal{D}^{\text{TRAIN}}$

Development (or validation) data

- L samples/instances: $\mathcal{D}^{\text{DEV}} = \{(\boldsymbol{x}_1, y_1), (\boldsymbol{x}_2, y_2), \cdots, (\boldsymbol{x}_{\text{L}}, y_{\text{L}})\}$
- They are used to optimize hyperparameter(s).

Training data, validation and test data should *not* overlap!

Recipe

- For each possible value of the hyperparameter (say $K = 1, 3, \cdots, 100$)
  - Train a model using $\mathcal{D}^{\text{TRAIN}}$
  - Evaluate the performance of the model on $\mathcal{D}^{\text{DEV}}$
- Choose the model with the best performance on $\mathcal{D}^{\text{DEV}}$
- Evaluate the model on $\mathcal{D}^{\text{TEST}}$

What if we do not have validation data?

- Use cross-validation.

Yet, another practical issue with NNC

Assumes all features are equally important!

- Distances depend on units of the features.

Example: classify Iris with two features

Training data

ID (n)	petal width (x_1)	sepal length (x_2)	category (y)
1	0.2	5.1	setosa
2	1.4	7.0	versicolor
3	2.5	6.7	virginica

Flower with unknown category

petal width $= 1.8$ and sepal width $= 6.4$

Calculating distance $= \sqrt{(x_1 - x_{n1})^2 + (x_2 - x_{n2})^2}$

ID	distance
1	1.75
2	0.72
3	0.76

Thus, the category is *versicolor* (the real category is *virginica*)

Change units of x_2 from cm to mm

Training data

ID (n)	petal width (x_1)	sepal length (x_2)	category (y)
1	0.2	51	setosa
2	1.4	70	versicolor
3	2.5	67	virginica

Change units of x_2 from cm to mm

Training data

ID (n)	petal width (x_1)	sepal length (x_2)	category (y)
1	0.2	51	setosa
2	1.4	70	versicolor
3	2.5	67	virginica

Flower with unknown category

petal width $= 1.8$ and sepal width $= 64$

Calculating distance $= \sqrt{(x_1 - x_{n1})^2 + (x_2 - x_{n2})^2}$

ID	distance
1	13
2	6
3	3

Thus, the category is *virginica* (the real category is *virginica*)

Preprocess data

Normalize data to have zero mean and unit standard deviation in each dimension

- Compute the means and standard deviations in each feature

$$\bar{x}_d = \frac{1}{\mathrm{N}} \sum_n x_{nd}, \qquad s_d^2 = \frac{1}{N-1} \sum_n (x_{nd} - \bar{x}_d)^2$$

- Scale the feature accordingly

$$x_{nd} \leftarrow \frac{x_{nd} - \bar{x}_d}{s_d}$$

Many other ways of normalizing data — you would need/want to try different ones and pick them using (cross)validation

Outline

1. Review of previous lecture
2. Perceptron
   - Setup for binary classification
   - Intuition
   - Algorithm
3. What we have learned

Perceptron learning

Special case: binary classification

- Instance (feature vectors): $\boldsymbol{x} \in \mathbb{R}^{\mathsf{D}}$
- Label: $y \in \{-1, +1\}$
- Model/Hypotheses:
  $H = \{h | h : \mathbb{X} \to \mathbb{Y}, h(\boldsymbol{x}) = sign(\sum_{d=1}^{D} w_d x_d + b)\}$.
- Learning goal: $\hat{y} = h(\boldsymbol{x})$

Perceptron learning

Special case: binary classification

- Instance (feature vectors): $\boldsymbol{x} \in \mathbb{R}^{\mathsf{D}}$
- Label: $y \in \{-1, +1\}$
- Model/Hypotheses:
  $H = \{h | h : \mathbb{X} \to \mathbb{Y}, h(\boldsymbol{x}) = sign(\sum_{d=1}^{D} w_d x_d + b)\}$.
- Learning goal: $\hat{y} = h(\boldsymbol{x})$
  - Learn $w_1, \ldots, w_D, b$.
  - Parameters: $w_1, \ldots, w_D, b$.
  - $\boldsymbol{w}$: weights, b: bias

Perceptron predict

- Input: $\boldsymbol{x} \in \mathbb{R}^{\mathsf{D}}$, $\boldsymbol{w} \in \mathbb{R}^{\mathsf{D}}$, $b \in \mathbb{R}$.

$$a = \sum_{d=1}^{D} w_d x_d + b = \boldsymbol{w}^{\mathrm{T}} \boldsymbol{x} + b$$

$$\hat{y} = sign(a)$$

- Output: $\hat{y}$.
- $\sum_{d=1}^{D} w_d x_d + b = \boldsymbol{w}^{\mathrm{T}} \boldsymbol{x} + b = 0$: hyperplane in D dimensions with parameters $(\boldsymbol{w}, b)$.
- $\boldsymbol{w}$: weights, b: bias
- a: activation

Hyperplanes through the origin

Consider $\boldsymbol{x}$ that satisfies $g(\boldsymbol{x}) = \boldsymbol{w}^T\boldsymbol{x} + b = 0$. These $\boldsymbol{x}$ define a hyperplane in D dimensions.
We can always write this as a hyperplane passing through the origin in $D + 1$ dimensions.

$$\tilde{\boldsymbol{x}} \equiv \begin{pmatrix} 1 \\ x_1 \\ \vdots \\ x_D \end{pmatrix} \tilde{\boldsymbol{w}} \equiv \begin{pmatrix} b \\ w_1 \\ \vdots \\ w_D \end{pmatrix}$$

$$\begin{aligned} \tilde{g}(\tilde{\boldsymbol{x}}) &= \tilde{\boldsymbol{w}}^T\tilde{\boldsymbol{x}} \\ &= \sum_{d=1}^{D} w_d x_d + b \\ &= g(\boldsymbol{x}) \end{aligned}$$

Perceptron learning

If we have only one training example $(\boldsymbol{x}_n, y_n)$.

Assume $b = 0$.

How can we change $\boldsymbol{w}$ such that

$$y_n = \text{sign}(\boldsymbol{w}^{\mathrm{T}} \boldsymbol{x}_n)$$

Two cases

- If $y_n = \text{sign}(\boldsymbol{w}^{\mathrm{T}} \boldsymbol{x}_n)$, do nothing.
- If $y_n \neq \text{sign}(\boldsymbol{w}^{\mathrm{T}} \boldsymbol{x}_n)$,

$$\boldsymbol{w}^{\mathrm{NEW}} \leftarrow \boldsymbol{w}^{\mathrm{OLD}} + y_n \boldsymbol{x}_n$$

Perceptron learning

If we have only one training example $(\boldsymbol{x}_n, y_n)$.

Assume $b = 0$.

How can we change $\boldsymbol{w}$ such that

$$y_n = \text{sign}(\boldsymbol{w}^{\mathrm{T}} \boldsymbol{x}_n)$$

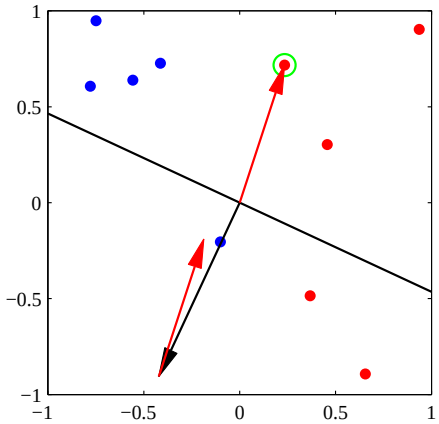
Another way of saying the same thing

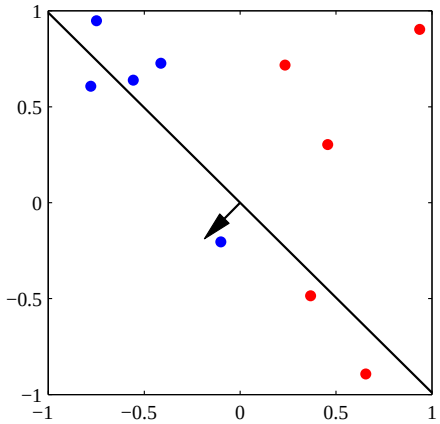
- $a = \boldsymbol{w}^{\mathrm{T}} \boldsymbol{x}_n$
- If $y_n a > 0$, do nothing.
- If $y_n a \leq 0$,

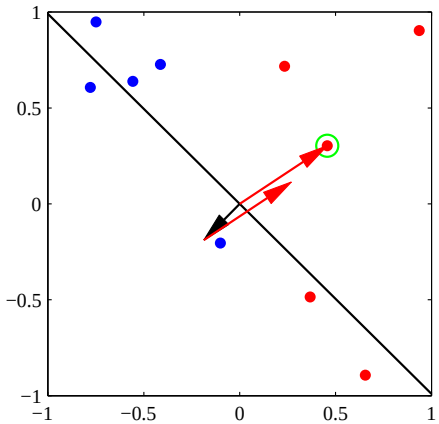
$$\boldsymbol{w}^{\mathrm{NEW}} \leftarrow \boldsymbol{w}^{\mathrm{OLD}} + y_n \boldsymbol{x}_n$$

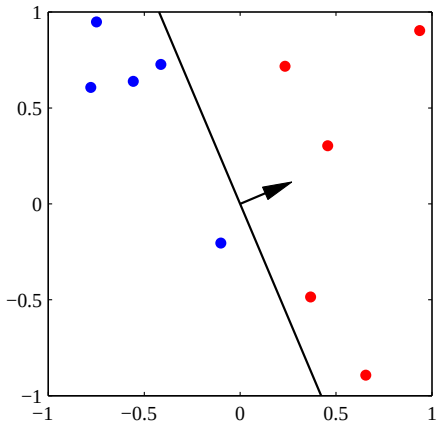
Example of perceptron update

Red is $+1$, Blue is -1









Why would it work?

If $y_n a \leq 0$, then

$$y_n(\boldsymbol{w}^{\mathrm{T}}\boldsymbol{x}_n) \leq 0$$

Why would it work?

If $y_n a \leq 0$, then

$$y_n(\boldsymbol{w}^{\mathrm{T}}\boldsymbol{x}_n) \leq 0$$

What would happen if we change to new $\boldsymbol{w}^{\mathrm{NEW}} = \boldsymbol{w} + y_n\boldsymbol{x}_n$?

$$y_n[(\boldsymbol{w} + y_n\boldsymbol{x}_n)^{\mathrm{T}}\boldsymbol{x}_n] = y_n\boldsymbol{w}^{\mathrm{T}}\boldsymbol{x}_n + y_n^2\boldsymbol{x}_n^{\mathrm{T}}\boldsymbol{x}_n$$

Why would it work?

If $y_n a \leq 0$, then

$$y_n(\boldsymbol{w}^{\mathrm{T}}\boldsymbol{x}_n) \leq 0$$

What would happen if we change to new $\boldsymbol{w}^{\mathrm{NEW}} = \boldsymbol{w} + y_n\boldsymbol{x}_n$?

$$y_n[(\boldsymbol{w} + y_n\boldsymbol{x}_n)^{\mathrm{T}}\boldsymbol{x}_n] = y_n\boldsymbol{w}^{\mathrm{T}}\boldsymbol{x}_n + y_n^2\boldsymbol{x}_n^{\mathrm{T}}\boldsymbol{x}_n$$

We are adding a positive number, so it is possible that

$$y_n(\boldsymbol{w}^{\mathrm{NEW}^{\mathrm{T}}}\boldsymbol{x}_n) > 0$$

i.e., we are more likely to classify correctly

Perceptron learning

Iteratively solving one case at a time

- REPEAT
- Pick a data point $\boldsymbol{x}_n$
- Compute $a = \boldsymbol{w}^{\mathrm{T}}\boldsymbol{x}_n$ using the *current* $\boldsymbol{w}$
- If $ay_n > 0$, do nothing. Else,

$$\boldsymbol{w} \leftarrow \boldsymbol{w} + y_n\boldsymbol{x}_n$$

- UNTIL converged.

Perceptron training/learning

$data$ = N **samples/instances:** $= \{(\boldsymbol{x}_1, y_1), \cdots, (\boldsymbol{x}_\mathsf{N}, y_\mathsf{N})\}$

Algorithm 1 PerceptronTrain $(data, maxIter)$

$w_d \leftarrow 0, d \in \{0, D\}$
for $iter = 1 \ldots MaxIter$ **do**
  for $(\boldsymbol{x}, y) \in data$ **do**
    $a \leftarrow \boldsymbol{w}^\mathrm{T} \boldsymbol{x}$
    if $ay \leq 0$ **then**
      $\boldsymbol{w} \leftarrow \boldsymbol{w} + y\boldsymbol{x}$
    end if
  end for
end for
return w

Design decisions

- $MaxIter$: Hyperparameter

Design decisions

- $MaxIter$: Hyperparameter
- How to loop over the data?
  - Constant.
  - Permuting once
  - Permuting in each iteration

Properties of perceptron learning

- This is an online algorithm – looks at one instance at a time.
- Does the algorithm terminate (convergence)?

Properties of perceptron learning

- This is an online algorithm – looks at one instance at a time.
- Does the algorithm terminate (convergence)?
  - If training data is not linearly separable, the algorithm does not converge.
  - If the training data is linearly separable, the algorithm stops in a finite number of steps (converges).

Properties of perceptron learning

- This is an online algorithm – looks at one instance at a time.
- Does the algorithm terminate (convergence)?
  - If training data is not linearly separable, the algorithm does not converge.
  - If the training data is linearly separable, the algorithm stops in a finite number of steps (converges).
- How long to convergence ?
  - Depends on the difficulty of the problem.

Perceptron

- Extensions
  - Voting
  - Averaging
- Limitations
  - Linear separability
- Interpreting the importance of features
  - The values of weight w_d tells us the importance of feature x_d.

Outline

Summary

- You should now be able to understand the differences between decision trees, perceptrons and nearest neighbors.
- Given data, use training, development and test splits (or cross-validation).
- Use training and development to tune hyperparameters that trades off overfitting and underfitting.
- Use test to get an estimate of generalization or accuracy on unseen data.