18-661 Introduction to Machine Learning

Nearest Neighbors

Spring 2020

ECE – Carnegie Mellon University

Midterm Information

Midterm will be on Wednesday, 2/26. SV and Pittsburgh students will take the midterm in class (the usual room and time). Kigali students will take the midterm at 6:00pm local time.

- Closed-book except for one double-sided letter-size handwritten page of notes.
- We will provide formulas for relevant probability distributions.
- You will not need a calculator. Only pens/pencils, erasers, and scratch paper are allowed.

Will cover all topics presented through Wednesday in class.

- (1) point estimation/MLE/MAP, (2) linear regression, (3) naive Bayes, (4) logistic regression, and (5) SVMs.
- Understand all homework questions and derivations in lecture/recitation, as well as practice exam questions.

Midterm: Concepts That You Should Know

This is a quick overview of the most important concepts/methods/models that you should expect to see on the midterm.

- MLE/MAP: how to find the likelihood of one or more observations given a system model, how to incorporate knowledge of a prior distribution, how to optimize the likelihood, loss functions
- Linear regression: how to formulate the linear regression optimization problem, how it relates to MLE/MAP, ridge regression, overfitting and regularization, gradient descent, bias-variance trade-off
- Naive Bayes: Bayes' rule, naive classification rule, why it is naive
- Logistic regression: how to formulate logistic regression, how it relates to MLE, comparison to naive Bayes, sigmoid function, softmax function, cross-entropy function
- SVMs: hinge loss formulation, max-margin formulation, dual of the SVM problem, kernel functions

Outline

1. Review of Kernel SVMs
2. Nearest Neighbor Classifier
3. Practical Aspects of NN

Review of Kernel SVMs

Primal and Dual SVM Formulations: Kernel Versions

Primal formulation

$$\begin{aligned} \min_{\boldsymbol{w}, b, \boldsymbol{\xi}} \quad & \frac{1}{2}\|\boldsymbol{w}\|_2^2 + C\sum_n \xi_n \\ \text{s.t.} \quad & y_n[\boldsymbol{w}^\top \phi(\boldsymbol{x}_n) + b] \geq 1 - \xi_n, \quad \forall \ n \\ & \xi_n \geq 0, \quad \forall \ n \end{aligned}$$

Dual formulation

$$\begin{aligned} \max_{\boldsymbol{\alpha}} \sum_n \alpha_n - \frac{1}{2}\sum_{m,n} y_m y_n \alpha_m \alpha_n \phi(\boldsymbol{x}_m)^\top \phi(\boldsymbol{x}_n) \\ \text{s.t.} \quad 0 \leq \alpha_n \leq C, \quad \forall \ n \\ \sum_n \alpha_n y_n = 0 \end{aligned}$$

- $\phi(\mathbf{x})$ is the feature vector for the data $\mathbf{x}$;
- In the dual problem, we only need to know $\phi(\boldsymbol{x}_m)^\top \phi(\boldsymbol{x}_n)$.

Dual Kernel SVM

We replace the inner products $\phi(\boldsymbol{x}_m)^\top \phi(\boldsymbol{x}_n)$ with a kernel function

$$\begin{aligned}\max_{\boldsymbol{\alpha}} \quad & \sum_n \alpha_n - \frac{1}{2}\sum_{m,n} y_m y_n \alpha_m \alpha_n k(\boldsymbol{x}_m, \boldsymbol{x}_n) \\ \text{s.t.} \quad & 0 \leq \alpha_n \leq C, \quad \forall\ n \\ & \sum_n \alpha_n y_n = 0\end{aligned}$$

- $k(\mathbf{x}_m, \mathbf{x}_n)$ is a scalar and it is independent of the dimension of the feature vector $\phi(\mathbf{x})$.
- $k(\mathbf{x}_m, \mathbf{x}_n)$ roughly measures the similarity of $\mathbf{x}_m$ and $\mathbf{x}_n$.
- $k(\mathbf{x}_m, \mathbf{x}_n)$ is a kernel function if it is symmetric and positive-definite ($k(\mathbf{x}, \mathbf{x}) > 0$ for all $\mathbf{x} > 0$).

Test prediction

Learning $\boldsymbol{w}$ and b:

$$\boldsymbol{w} = \sum_n \alpha_n y_n \phi(\boldsymbol{x}_n)$$

$$b = y_n - \boldsymbol{w}^\top \phi(\boldsymbol{x}_n) = y_n - \sum_m \alpha_m y_m k(\boldsymbol{x}_m, \boldsymbol{x}_n)$$

But for test prediction on a new point $\mathbf{x}$, do we need the form of $\phi(\mathbf{x})$ in order to find the sign of $\boldsymbol{w}^\top \phi(\boldsymbol{x}) + b$? Fortunately, no!

Test Prediction:

$$h(\boldsymbol{x}) = \textsc{sign}(\sum_n y_n \alpha_n k(\boldsymbol{x}_n, \boldsymbol{x}) + b)$$

At test time it suffices to know the kernel function! So we really do not need to know ϕ.

Example of Kernel SVM

Given a dataset $\{(\boldsymbol{x}_n, y_n)$ for $n = 1, 2, \ldots, N\}$, how do you classify it using kernel SVM ?

The linear decision boundary is pretty bad

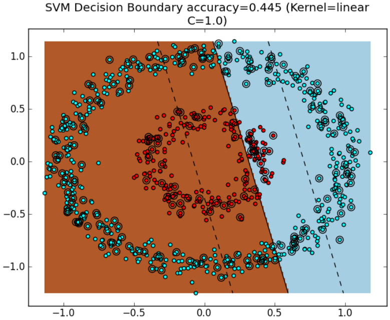
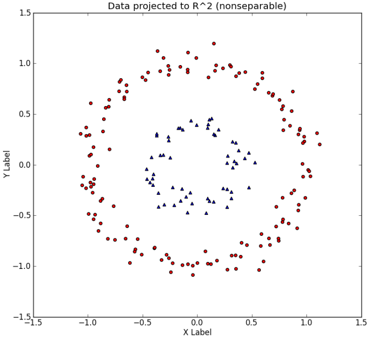


Image Source: `https://www.eric-kim.net/eric-kim-net/posts/1/kernel_trick.html`

Example of Kernel SVM

Given a dataset $\{(\boldsymbol{x}_n, y_n)$ for $n = 1, 2, \ldots, N\}$, how do you classify it using kernel SVM ?

Use kernel $\phi(x) = [x_1, x_2, x_1^2 + x_2^2]$ to transform the data in a 3D space



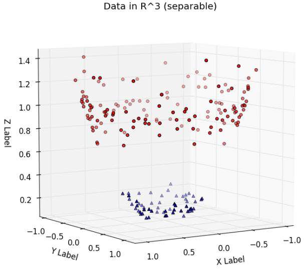


Image Source: `https://www.eric-kim.net/eric-kim-net/posts/1/kernel_trick.html`

Example of Kernel SVM

Given a dataset $\{(\boldsymbol{x}_n, y_n)$ for $n = 1, 2, \ldots, N\}$, how do you classify it using kernel SVM ?

Then find the decision boundary. How? Solve the Dual problem

$$\max_{\boldsymbol{\alpha}} \sum_n \alpha_n - \frac{1}{2} \sum_{m,n} y_m y_n \alpha_m \alpha_n \boldsymbol{\phi}(\boldsymbol{x}_m)^\top \boldsymbol{\phi}(\boldsymbol{x}_n)$$

$$\text{s.t.} \quad 0 \le \alpha_n \le C, \quad \forall\ n$$

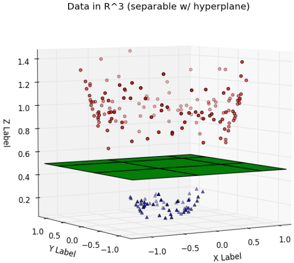
$$\sum_n \alpha_n y_n = 0$$

Then find $\mathbf{w}$ and b. Predict $y = \text{sign}(\mathbf{w}^T \phi(\mathbf{x}) + b)$.

Example of Kernel SVM

Given a dataset $\{(\boldsymbol{x}_n, y_n)$ for $n = 1, 2, \ldots, N\}$, how do you classify it using kernel SVM ?

Here is the resulting decision boundary


Data in R^3 (separable w/ hyperplane)

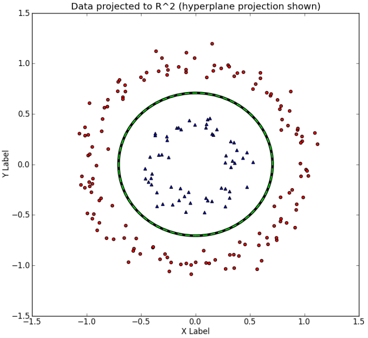

Data projected to R^2 (hyperplane projection shown)

Image Source: `https://www.eric-kim.net/eric-kim-net/posts/1/kernel_trick.html`

Example of Kernel SVM

Given a dataset $\{(\boldsymbol{x}_n, y_n) \text{ for } n = 1, 2, \ldots, N\}$, how do you classify it using kernel SVM ?

In general, you don't need to concretely define $\phi(\mathbf{x})$. In the dual problem we can just use the kernel function $k(\mathbf{x}_m, \mathbf{x}_n)$. For cases where $\phi(\mathbf{x})$ is concretely defined, $k(\mathbf{x}_m, \mathbf{x}_n) = \phi(\mathbf{x}_m)^T \phi(\mathbf{x}_n)$.

$$\max_{\boldsymbol{\alpha}} \sum_n \alpha_n - \frac{1}{2} \sum_{m,n} y_m y_n \alpha_m \alpha_n \boldsymbol{\phi}(\boldsymbol{x}_m)^\top \boldsymbol{\phi}(\boldsymbol{x}_n)$$

$$\text{s.t.} \quad 0 \leq \alpha_n \leq C, \quad \forall\ n$$

$$\sum_n \alpha_n y_n = 0$$

Advantages of SVM

SVM

1. Is less sensitive to outliers.
2. Maximizes distance of training data from the boundary
3. Generalizes well to many nonlinear models.
4. Only requires a subset of the training points.
5. Scales better with high-dimensional data.

Outline

1. Review of Kernel SVMs

2. Nearest Neighbor Classifier

3. Practical Aspects of NN

Parametric vs. Nonparametric

- So far, we've discussed parametric machine learning models:
    - Linear regression
    - Naive Bayes
    - Logistic regression
    - Linear SVMs
- Now we will discuss two *nonparametric* models:
    - Nearest neighbors
    - Decision trees

Parametric vs. Nonparametric

Key difference:

- Parametric models assume that the data can be characterized via some fixed set of parameters θ. Given this set of parameters, our future predictions are independent of the data $\mathcal{D}$, i.e., $P(x|\theta, \mathcal{D}) = P(x|\theta)$.
    - Often simpler and faster to learn, but can sometimes be a poor fit
- Nonparametric models instead assume that the model features depend on the data $\mathcal{D}$. The number of features tends to grow with the size of the dataset.
    - More complex and expensive, but can learn more flexible patterns
- Both parametric and non-parametric methods can be used for either regression or classification.

Outline

1. Review of Kernel SVMs

2. Nearest Neighbor Classifier

3. Practical Aspects of NN

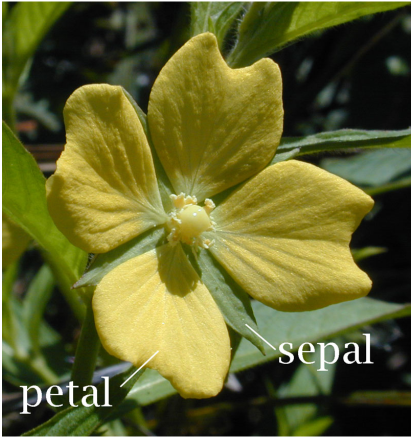
Nearest Neighbor Classifier

Recognizing flowers

Types of Iris: setosa, versicolor, and virginica

Measuring the properties of the flowers

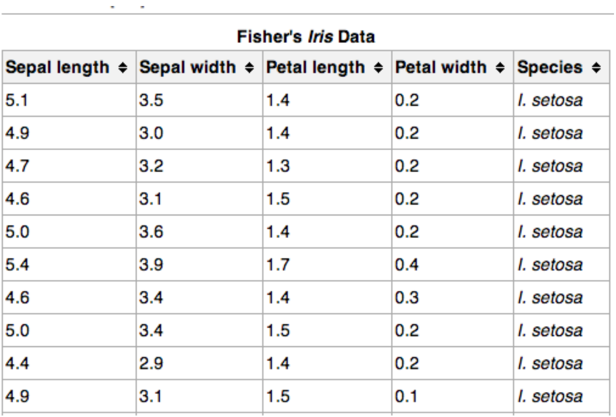
Features: the widths and lengths of sepal and petal



Often, data is conveniently organized as a table

Ex: Iris data (click here for all data)

- 4 features
- 3 classes



Fisher's *Iris* Data

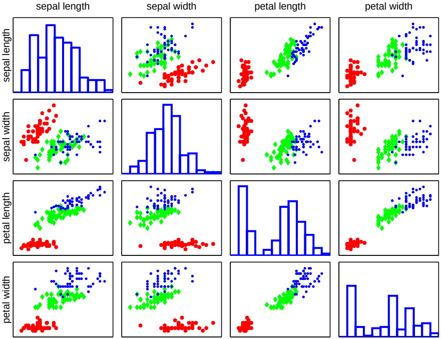
Sepal length ⬍	Sepal width ⬍	Petal length ⬍	Petal width ⬍	Species ⬍
5.1	3.5	1.4	0.2	*I. setosa*
4.9	3.0	1.4	0.2	*I. setosa*
4.7	3.2	1.3	0.2	*I. setosa*
4.6	3.1	1.5	0.2	*I. setosa*
5.0	3.6	1.4	0.2	*I. setosa*
5.4	3.9	1.7	0.4	*I. setosa*
4.6	3.4	1.4	0.3	*I. setosa*
5.0	3.4	1.5	0.2	*I. setosa*
4.4	2.9	1.4	0.2	*I. setosa*
4.9	3.1	1.5	0.1	*I. setosa*

Pairwise scatter plots of 131 flower specimens

Visualization of data helps to identify the right learning model

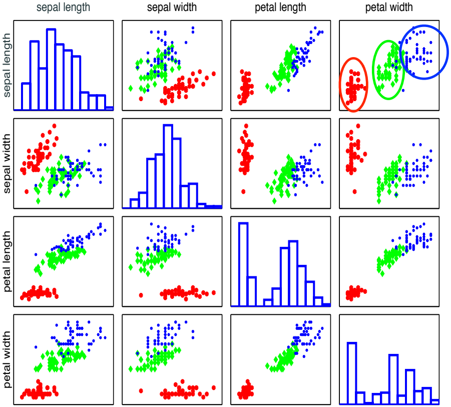
Which combination of features separates the three classes?

Figure 1: Each colored point is a flower specimen: setosa, versicolor, virginica

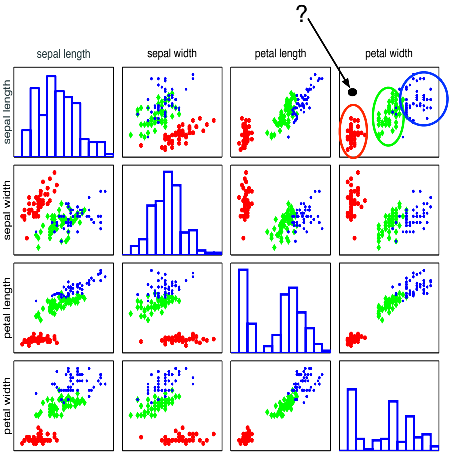


Different types seem well-clustered and separable

Using two features: petal width and sepal length



Labeling an unknown flower type



Closer to red cluster: so labeling it as setosa

Multi-class classification

Classify data into one of the multiple categories

- Input (feature vectors): $\boldsymbol{x} \in \mathbb{R}^{\mathsf{D}}$
- Output (label): $y \in [\mathsf{C}] = \{1, 2, \cdots, \mathsf{C}\}$
- Learning goal: $y = f(\boldsymbol{x})$

Recall special case: binary classification

- Number of classes: $\mathsf{C} = 2$
- Labels: $\{0, 1\}$ or $\{-1, +1\}$

More terminology

Training data (set)

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

Test (evaluation) data

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

Training data and test data should *not* overlap: $\mathcal{D}^{\text{TRAIN}} \cap \mathcal{D}^{\text{TEST}} = \emptyset$

Nearest neighbor classification (NNC)

Nearest neighbor of a (training or test) data point

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

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

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

Classification rule

$$y = f(\boldsymbol{x}) = y_{\mathrm{nn}(\boldsymbol{x})}$$

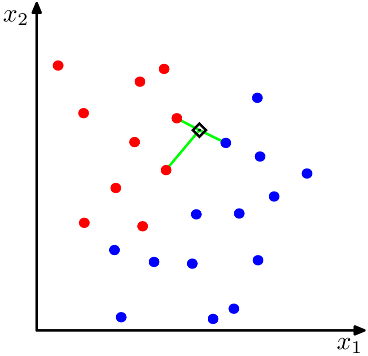
Example: if $\mathrm{nn}(\boldsymbol{x}) = 2$, then

$$y_{\mathrm{nn}(\boldsymbol{x})} = y_2,$$

which is the label of the 2nd data point.

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)

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 length $= 6.4$

Calculating distance from (x_1, x_2) to (x_{n1}, x_{n2}): $(x_1 - x_{n1})^2 + (x_2 - x_{n2})^2$

ID	distance
1	4.25
2	0.52
3	0.58

Thus, the predicted category is 2 (*versicolor*)

How to measure “nearness” with other distances?

Previously, we used the Euclidean distance

$$\mathrm{nn}(\boldsymbol{x}) = \mathrm{argmin}_{n\in[\mathrm{N}]} \|\boldsymbol{x} - \boldsymbol{x}_n\|_2^2$$

We can also use alternative distances

E.g., the following L_1 distance (i.e., city block distance, or Manhattan distance)

$$\begin{aligned}\mathrm{nn}(\boldsymbol{x}) &= \mathrm{argmin}_{n\in[\mathrm{N}]} \|\boldsymbol{x} - \boldsymbol{x}_n\|_1 \\ &= \mathrm{argmin}_{n\in[\mathrm{N}]} \sum_{d=1}^{\mathrm{D}} |x_d - x_{nd}|\end{aligned}$$

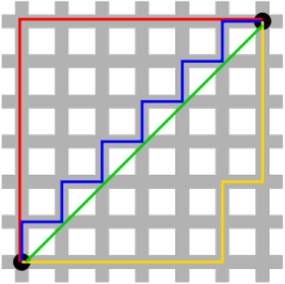
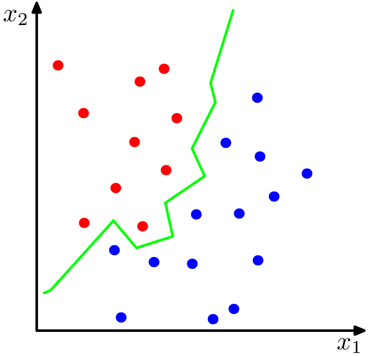


Figure 2: Green line is Euclidean distance. Red, Blue, and Yellow lines are L_1 distance

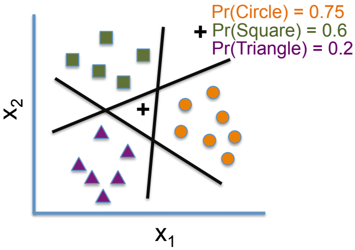
Decision boundary

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



(b)

Recall: Multi-class Classification



Previously, we learned a multi-class classifier by combining binary, linear decision boundaries to partition the feature space.

Parametric vs. Nonparametric, Revisited

Nonparametric models instead assume that the model features depend on the data $\mathcal{D}$. The number of features tends to grow with the size of the dataset.

- Parametric models are often simpler and faster to learn, but can sometimes be a poor fit
- Nonparametric models are more complex and expensive, but can learn more flexible patterns

How does this manifest for nearest neighbors?

- Nearest neighbors often learns a *highly nonlinear* decision boundary.
- But, we need to compare the test data point to *every sample in the training dataset*.

K-nearest neighbor (KNN) classification

Increase the number of nearest neighbors to use?

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

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?

Classification rule

- Every neighbor votes: suppose y_n (the true 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 the votes.
- Aggregate everyone's vote

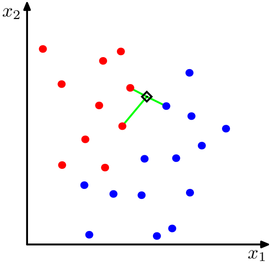
$$v_c = \sum_{n \in \text{knn}(\boldsymbol{x})} \mathbb{I}(y_n == c), \quad \forall \quad c \in [\text{C}]$$

- Label with the majority, breaking ties arbitrarily

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

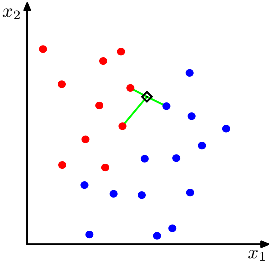
Example

K=1, Label: ??



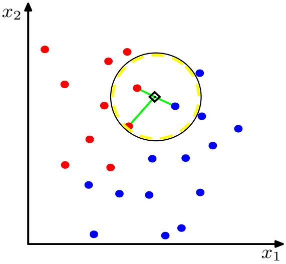
(a)

K=3, Label: ??



(a)

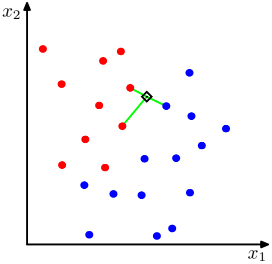
K=5, Label: ??



(a)

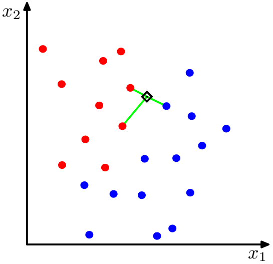
Example

K=1, Label: red



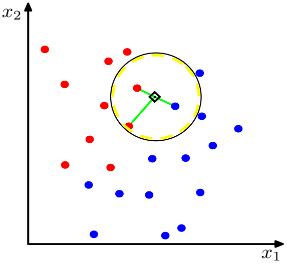
(a)

K=3, Label: red



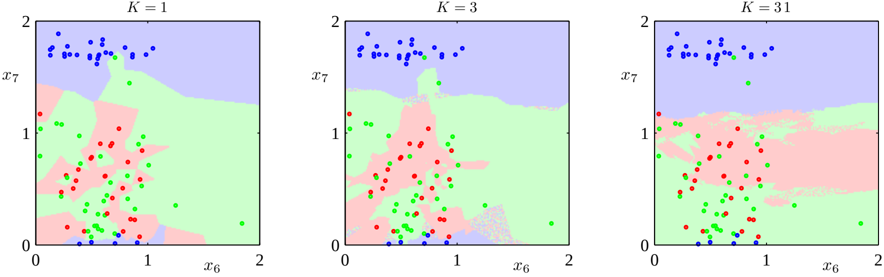
(a)

K=5, Label: blue



(a)

How to choose an optimal K?



When K increases, the decision boundary becomes smooth.

Why use nearest neighbors?

Advantages of NNC

- Computationally, simple and easy to implement – just compute distances, no optimization required
- Can learn complex decision boundaries

Disadvantages of NNC

- Computationally intensive for large-scale problems: $O(\mathsf{N}D)$ 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 difficult.

Practical Aspects of NN

Hyperparameters in NN

Two crucial choices for NN

- 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 $p \geq 1$.

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

Hyperparameter tuning on a validation dataset

Training data

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

Test data

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

Validation data

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

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

Recipe

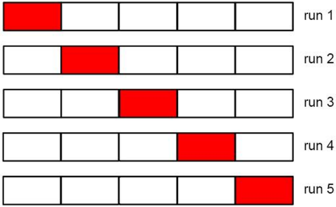
- For each possible value of the hyperparameter (say $K = 1, 3, \cdots, 100$)
    - Train a model using $\mathcal{D}^{\text{TRAIN}}$ (we don't need this step for NNC)
    - Evaluate the performance of the model on $\mathcal{D}^{\text{VAL}}$
- Choose the model with the best performance on $\mathcal{D}^{\text{VAL}}$
- Evaluate the model on $\mathcal{D}^{\text{TEST}}$

Cross-validation

What if we do not have validation data?

- We split the training data into S equal parts.
- We use each part *in turn* as a validation dataset and use the others as a training dataset.
- We choose the hyperparameter such that the model performs the best (based on average, variance, etc.)
- We re-train the model on the full training dataset with the best hyperparameter.

Figure 3: S = 5: 5-fold cross validation



Special case: when S = N, this will be leave-one-out.

But how do we choose the distances?

Distances depend on units of the features!

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 among them using (cross) validation

Summary so far

- Described a simple *nonparametric* learning algorithm
- Discussed a few practical aspects, such as tuning hyperparameters, with cross-validation – you will get experience with this in your homework!

Good luck with the midterm!