18-661 Introduction to Machine Learning

Nearest Neighbors

Spring 2019

Prof. Gauri Joshi
Course Logistics

Important Dates

- Homework 4: Due today
- Recitation tomorrow: Mock mid-term discussion and extra office hours
- Midterm exam: In-class next Tuesday, 3/5.
  - Topics: MLE/MAP, Linear Regression, Logistic Regression, and SVM
  - Multiple-choice questions
  - One handwritten US-letter size cheat-sheet (two-sided) allowed – printing notes written on an IPad/equivalent is NOT permitted.
  - No need for a calculator. Only pen/pencil and scratch paper are allowed.
Important Dates

- **Homework 4**: Due today
- **Recitation tomorrow**: Mock mid-term discussion and extra office hours
Important Dates

• **Homework 4:** Due today

• **Recitation tomorrow:** Mock mid-term discussion and extra office hours

• **Midterm exam:** In-class next Tuesday, 3/5.
Important Dates

- **Homework 4**: Due today
- **Recitation tomorrow**: Mock mid-term discussion and extra office hours
- **Midterm exam**: In-class next Tuesday, 3/5.
  - Topics: MLE/MAP, Linear Regression, Logistic Regression, and SVM
Important Dates

- **Homework 4:** Due today
- **Recitation tomorrow:** Mock mid-term discussion and extra office hours
- **Midterm exam:** In-class next Tuesday, 3/5.
  - Topics: MLE/MAP, Linear Regression, Logistic Regression, and SVM
  - Multiple-choice questions
Important Dates

• **Homework 4**: Due today

• **Recitation tomorrow**: Mock mid-term discussion and extra office hours

• **Midterm exam**: In-class next Tuesday, 3/5.
  - Topics: MLE/MAP, Linear Regression, Logistic Regression, and SVM
  - Multiple-choice questions
  - One handwritten US-letter size cheat-sheet (two-sided) allowed – printing notes written on an IPad/equivalent is NOT permitted.
Course Logistics

Important Dates

• **Homework 4:** Due today

• **Recitation tomorrow:** Mock mid-term discussion and extra office hours

• **Midterm exam:** In-class next Tuesday, 3/5.
  - Topics: MLE/MAP, Linear Regression, Logistic Regression, and SVM
  - Multiple-choice questions
  - One handwritten US-letter size cheat-sheet (two-sided) allowed – printing notes written on an IPad/equivalent is NOT permitted.
  - No need for a calculator. Only pen/pencil and scratch paper are allowed.
Review of SVMs
Logistic Regression Loss: Illustration

\[ \mathcal{L}(w) = -\sum_n \{y_n \log \sigma(w^\top x_n) + (1 - y_n) \log[1 - \sigma(w^\top x_n)] \} \]

- Loss grows approx. linearly as we move away from the boundary
- Alternative: **Hinge Loss Function**
Hinge Loss: Illustration

\[ \mathcal{L}(\mathbf{w}) = -\sum_n \{y_n \log \sigma(\mathbf{w}^\top \mathbf{x}_n) + (1 - y_n) \log[1 - \sigma(\mathbf{w}^\top \mathbf{x}_n)]\} \]

- Loss grows linearly as we move away from the boundary
- No penalty if a point is more than 1 unit from the boundary
- Makes the search for the boundary easier (as we will see later)
Hinge Loss: Mathematical Expression

\[ \mathcal{L}(w) = - \sum_{n} \max(0, 1 - y_n (w^T x_n + b)) \]

- Change of notation \( y = 0 \rightarrow y = -1 \)
- Separate the bias term \( b \) from \( w \)
- Makes the mathematical expression more compact
Optimization Problem of support vector machines (SVM)

Minimizing the total hinge loss on all the training data

\[
\min_{w,b} \sum_n \max(0, 1 - y_n[w^\top x_n + b]) + \frac{\lambda}{2} \|w\|^2
\]

Analogous to regularized least squares, as we balance between two terms (the loss and the regularizer).

- Can solve using gradient descent to get the optimal \(w\) and \(b\)
- Gradient of the first term will be either 0, \(x_n\) or \(-x_n\) depending on \(y_n\) and \(w^\top x_n + b\)
- Much easier to compute than in logistic regression, where we need to compute the sigmoid function \(\sigma(w^\top x_n + b)\) in each iteration
Max Margin Intuition: Where to put the decision boundary?

Idea: Find a decision boundary in the 'middle' of the two classes that:

- Perfectly classifies the training data
- Is as far away from every training point as possible

Let us apply this intuition to build a classifier that MAXIMIZES THE MARGIN between training points and the decision boundary.
Distance from a Hyperplane

How to find the distance of a point \( a \) to the hyperplane?

- If we define point \( a_0 \) on the line, then this distance corresponds to length of \( a - a_0 \) in direction of \( w^* = \frac{w}{||w||} \), which equals \( w^T (a - a_0) \).
- We know \( w^T a_0 = -b \) since \( w^T a_0 + b = 0 \).
- Then the distance equals \( \frac{1}{||w||} (w^T a + b) \).
The *unsigned* distance from a point $x$ to decision boundary (hyperplane) $\mathcal{H}$ is

$$d_{\mathcal{H}}(x) = \frac{|\mathbf{w}^\top x + b|}{\|\mathbf{w}\|_2}$$
Distance from a point to decision boundary

The *unsigned* distance from a point $x$ to decision boundary (hyperplane) $\mathcal{H}$ is

$$d_{\mathcal{H}}(x) = \frac{|\mathbf{w}^\top x + b|}{\|\mathbf{w}\|_2}$$

We can remove the absolute value $|\cdot|$ by exploiting the fact that the decision boundary classifies every point in the training dataset correctly.

Namely, $(\mathbf{w}^\top x + b)$ and $x$’s label $y$ must have the same sign, so:

$$d_{\mathcal{H}}(x) = \frac{y[\mathbf{w}^\top x + b]}{\|\mathbf{w}\|_2}$$
Defining the Margin

Margin
Smallest distance between the hyperplane and all training points

\[ \text{MARGIN}(\mathbf{w}, b) = \min_n y_n [\mathbf{w}^\top \mathbf{x}_n + b] / \| \mathbf{w} \|_2 \]

\( \mathcal{H} : \mathbf{w}^\top \phi(x) + b = 0 \)

\[ \frac{|\mathbf{w}^\top \phi(x) + b|}{\| \mathbf{w} \|_2} \]
How should we pick \((w, b)\) based on its margin?

We want a decision boundary that is as far away from all training points as possible, so we to \textit{maximize} the margin!

\[
\max_{w, b} \left( \min_n y_n \frac{w^T x_n + b}{\|w\|_2} \right) = \max_{w, b} \left( \frac{1}{\|w\|_2} \min_n y_n [w^T x_n + b] \right)
\]

\textbf{Only involves points near the boundary} (more on this later).
We can further constrain the problem by scaling \((\mathbf{w}, b)\) such that

\[
\min_n y_n [\mathbf{w}^\top \mathbf{x}_n + b] = 1
\]

We’ve fixed the numerator in the \text{MARGIN}(\mathbf{w}, b) equation, and we have:

\[
\text{MARGIN}(\mathbf{w}, b) = \frac{\min_n y_n [\mathbf{w}^\top \mathbf{x}_n + b]}{\|\mathbf{w}\|_2} = \frac{1}{\|\mathbf{w}\|_2}
\]

Hence the points closest to the decision boundary are at distance \(\frac{1}{\|\mathbf{w}\|_2}\)!
SVM: max margin formulation for separable data

Assuming separable training data, we thus want to solve:

$$\max_{w,b} \frac{1}{\|w\|_2^2} \text{ margin such that } y_n[w^T x_n + b] \geq 1, \forall n$$

This is equivalent to

$$\min_{w,b} \frac{1}{2} \|w\|_2^2 \text{ scaling of } w, b$$

s.t. \( y_n[w^T x_n + b] \geq 1, \forall n \)

Given our geometric intuition, SVM is called a max margin (or large margin) classifier. The constraints are called large margin constraints.
SVM for non-separable data

Constraints in separable setting

\[ y_n [\mathbf{w}^\top \mathbf{x}_n + b] \geq 1, \quad \forall \ n \]

Constraints in non-separable setting

Idea: modify our constraints to account for non-separability! Specifically, we introduce slack variables \( \xi_n \geq 0 \):

\[ y_n [\mathbf{w}^\top \mathbf{x}_n + b] \geq 1 - \xi_n, \quad \forall \ n \]

- For “hard” training points, we can increase \( \xi_n \) until the above inequalities are met
- What does it mean when \( \xi_n \) is very large?
We do not want $\xi_n$ to grow too large, and we can control their size by incorporating them into our optimization problem:

$$\min_{w, b, \xi} \quad \frac{1}{2} \|w\|_2^2 + C \sum_n \xi_n$$

s.t. \quad y_n [w^\top x_n + b] \geq 1 - \xi_n, \quad \forall \ n

$$\xi_n \geq 0, \quad \forall \ n$$

What is the role of $C$?

- User-defined hyperparameter
- Trades off between the two terms in our objective
- Same idea as the regularization term in ridge regression, i.e., $C = \frac{1}{\lambda}$
Dual formulation of SVM

Dual is also a convex quadratic program

\[
\begin{align*}
\max_{\alpha} & \quad \sum_n \alpha_n - \frac{1}{2} \sum_{m,n} y_my_n\alpha_m\alpha_n x_m^\top x_n \\
\text{s.t.} & \quad 0 \leq \alpha_n \leq C, \quad \forall \ n \\
& \quad \sum_n \alpha_n y_n = 0
\end{align*}
\]

- There are \( N \) dual variables \( \alpha_n \), one for each data point.
- Independent of the size \( d \) of \( x \): SVM scales better for high-dimensional feature.
- May seem like a lot of optimization variables when \( N \) is large, but many of the \( \alpha_n \)'s become zero. \( \alpha_n \) is non-zero only if the \( n^{th} \) point is a support vector.
Why do many $\alpha_n$’s become zero?

$$\max_{\alpha} \sum_n \alpha_n - \frac{1}{2} \sum_{m,n} y_m y_n \alpha_m \alpha_n x_m^\top x_n$$

s.t.  $0 \leq \alpha_n \leq C, \ \forall \ n$

$$\sum_n \alpha_n y_n = 0$$

- **KKT** complementary slackness conditions tell us:

  $$(1) \ \lambda_n \xi_n = 0 \quad (2) \ \alpha_n \{1 - \xi_n - y_n [w^\top x_n + b]\} = 0$$

- (2) tells us that $\alpha_n > 0$ iff $1 - \xi_n = y_n [w^\top x_n + b]$
  - If $\xi_n = 0$, then support vector is on the margin
  - Otherwise, $\xi_n > 0$ means that the point is an outlier
Visualizing the support vectors

\[ w^T \phi(x) + b = 0 \]

\[ w^T \phi(x) + b = 1 \]

\[ w^T \phi(x) + b = -1 \]

Support vectors \((\alpha_n > 0)\) are highlighted by the dotted orange lines.

- \(\xi_n = 0\) and \(0 < \alpha_n < C\) when \(y_n[w^T x_n + b] = 1\).
- \(\xi_n > 0\) and \(\alpha_n = C\) if \(y_n[w^T x_n + b] < 1\).
Once we solve for $\alpha_n$’s, how to get $w$ and $b$?

Recovering $w$

$$
\frac{\partial L}{\partial w} = 0 \rightarrow w = \sum_n \alpha_n y_n x_n
$$

Only depends on support vectors, i.e., points with $\alpha_n > 0$!
Once we solve for $\alpha_n$’s, how to get $w$ and $b$?

**Recovering $w$**

$$\frac{\partial L}{\partial w} = 0 \rightarrow w = \sum_{n} \alpha_n y_n x_n$$

Only depends on support vectors, i.e., points with $\alpha_n > 0$!

**Recovering $b$**

If $0 < \alpha_n < C$ and $y_n \in \{-1, 1\}$:

$$y_n[\mathbf{w}^\top \mathbf{x}_n + b] = 1$$

$$b = y_n - \mathbf{w}^\top \mathbf{x}_n$$

$$b = y_n - \sum_{m} \alpha_m y_m \mathbf{x}_m^\top \mathbf{x}_n$$
Non-linear basis functions in SVM

- What if the data is not linearly separable?
- We can transform the feature vector $\mathbf{x}$ using non-linear basis functions. For example,

$$
\phi(\mathbf{x}) = \begin{bmatrix}
1 \\
 x_1 \\
x_2 \\
x_1 x_2 \\
x_1^2 \\
x_2^2
\end{bmatrix}
$$

- Replace $\mathbf{x}$ by $\phi(\mathbf{x})$ in both the primal and dual SVM formulations.
Primal and Dual SVM Formulations: Kernel Versions

**Primal**

\[
\begin{align*}
\min_{w, b, \xi} & \quad \frac{1}{2} \|w\|_2^2 + C \sum_n \xi_n \\
\text{s.t.} & \quad y_n [w^T \phi(x_n) + b] \geq 1 - \xi_n, \quad \forall \ n \\
& \quad \xi_n \geq 0, \quad \forall \ n
\end{align*}
\]

**Dual**

\[
\begin{align*}
\max_{\alpha} & \quad \sum_n \alpha_n - \frac{1}{2} \sum_{m,n} y_m y_n \alpha_m \alpha_n \phi(x_m)^T \phi(x_n) \\
\text{s.t.} & \quad \alpha_n \geq 0, \quad \forall \ n \\
& \quad 0 \leq \alpha_n \leq C, \quad \forall \ n \\
& \quad \sum_n \alpha_n y_n = 0
\end{align*}
\]
We replace the inner products $\phi(x_m)^\top \phi(x_n)$ with a kernel function

$$\begin{align*}
\max_{\alpha} & \quad \sum_n \alpha_n - \frac{1}{2} \sum_{m,n} y_m y_n \alpha_m \alpha_n k(x_m, x_n) \\
\text{s.t.} & \quad 0 \leq \alpha_n \leq C, \quad \forall \ n \\
& \quad \sum_n \alpha_n y_n = 0
\end{align*}$$
We replace the inner products $\phi(x_m)^\top \phi(x_n)$ with a kernel function

$$
\max_{\alpha} \sum_n \alpha_n - \frac{1}{2} \sum_{m,n} y_my_n \alpha_m \alpha_n k(x_m, x_n)
$$

s.t. $0 \leq \alpha_n \leq C, \forall n$

$$
\sum_n \alpha_n y_n = 0
$$

- $k(x_m, x_n)$ roughly measures the similarity of $x_m$ and $x_n$. If they are similar and signs of $y_m$ and $y_n$ match, they are likely to be away from the boundary. The dual problem will try to set of the corresponding $\alpha$’s to zero
- $k(x_m, x_n)$ is a kernel function if it is symmetric and positive-definite ($k(x, x) > 0$ for all $x > 0$).
Dual Kernel SVM

We replace the inner products $\phi(x_m)^\top \phi(x_n)$ with a kernel function

$$\max_{\alpha} \sum_n \alpha_n - \frac{1}{2} \sum_{m,n} y_m y_n \alpha_m \alpha_n k(x_m, x_n)$$

s.t. $0 \leq \alpha_n \leq C$, $\forall \ n$

$$\sum_n \alpha_n y_n = 0$$

• The dimension of $k(x_m, x_n)$ is 1 and it is independent of the dimension of the feature vector $\phi(x)$.

• We do not need to know the exact form of $\phi(x)$. E.g., if the kernel is the radial basis function $k(x_m, x_n) = \exp\left(-\|x_m - x_n\|^2\right)$, it’s not obvious what $\phi(x)$ should be.

• This lets us define much more flexible nonlinearities.
Test prediction

Learning \( w \) and \( b \):

\[
w = \sum_n \alpha_n y_n \phi(x_n)
\]

\[
b = y_n - w^\top \phi(x_n) = y_n - \sum_m \alpha_m y_m k(x_m, x_n)
\]

Test Prediction:

\[
h(x) = \text{SIGN} \left( \sum_n y_n \alpha_n k(x_n, x) + b \right)
\]

At test time it suffices to know the kernel function! So we really do not need to know \( \phi \).
1. Review of SVMs

2. Nearest neighbor classifier

3. Practical aspects of NN
Nearest neighbor classifier
• So far, we’ve discussed parametric machine learning models:
• So far, we’ve discussed parametric machine learning models:
  • Linear regression
• So far, we’ve discussed parametric machine learning models:
  • Linear regression
  • Naive Bayes
So far, we’ve discussed parametric machine learning models:

- Linear regression
- Naive Bayes
- Logistic regression
So far, we’ve discussed parametric machine learning models:

- Linear regression
- Naive Bayes
- Logistic regression
- Linear SVMs
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:
• 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
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
Key difference:

- **Parametric models** assume that the data can be characterized via some fixed set of features $\mathbf{x}$ and parameters $\mathbf{x}$. Given these parameters, our future predictions are independent of the data $\mathcal{D}$, i.e., $P(y_t|\mathbf{w}, \mathbf{x}_t, \mathcal{D}) = P(y|\mathbf{w}, \mathbf{x}_t)$.
Key difference:

- **Parametric models** assume that the data can be characterized via some fixed set of features $\mathbf{x}$ and parameters $\mathbf{x}$. Given these parameters, our future predictions are independent of the data $\mathcal{D}$, i.e., $P(y_t|\mathbf{w}, \mathbf{x}_t, \mathcal{D}) = P(y|\mathbf{w}, \mathbf{x}_t)$.
  - Often simpler and faster to learn, but can sometimes be a poor fit
Key difference:

- **Parametric models** assume that the data can be characterized via some fixed set of features $\mathbf{x}$ and parameters $\mathbf{x}$. Given these parameters, our future predictions are independent of the data $\mathcal{D}$, i.e., $P(y_t|\mathbf{w}, \mathbf{x}_t, \mathcal{D}) = P(y|\mathbf{w}, \mathbf{x}_t)$.
  - Often simpler and faster to learn, but can sometimes be a poor fit

- **Nonparametric models** instead assume that the model parameters $\mathbf{w}$ depend on the data $\mathcal{D}$. The number of parameters tend to grow with the size of the dataset.
Key difference:

- **Parametric models** assume that the data can be characterized via some fixed set of features $\mathbf{x}$ and parameters $\mathbf{x}$. Given these parameters, our future predictions are independent of the data $\mathcal{D}$, i.e., $P(y_t|\mathbf{w}, \mathbf{x}_t, \mathcal{D}) = P(y|\mathbf{w}, \mathbf{x}_t)$.
  - Often simpler and faster to learn, but can sometimes be a poor fit

- **Nonparametric models** instead assume that the model parameters $\mathbf{w}$ depend on the data $\mathcal{D}$. The number of parameters tend to grow with the size of the dataset.
  - More complex and expensive, but can learn more flexible patterns
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

<table>
<thead>
<tr>
<th>Sepal length</th>
<th>Sepal width</th>
<th>Petal length</th>
<th>Petal width</th>
<th>Species</th>
</tr>
</thead>
<tbody>
<tr>
<td>5.1</td>
<td>3.5</td>
<td>1.4</td>
<td>0.2</td>
<td><em>I. setosa</em></td>
</tr>
<tr>
<td>4.9</td>
<td>3.0</td>
<td>1.4</td>
<td>0.2</td>
<td><em>I. setosa</em></td>
</tr>
<tr>
<td>4.7</td>
<td>3.2</td>
<td>1.3</td>
<td>0.2</td>
<td><em>I. setosa</em></td>
</tr>
<tr>
<td>4.6</td>
<td>3.1</td>
<td>1.5</td>
<td>0.2</td>
<td><em>I. setosa</em></td>
</tr>
<tr>
<td>5.0</td>
<td>3.6</td>
<td>1.4</td>
<td>0.2</td>
<td><em>I. setosa</em></td>
</tr>
<tr>
<td>5.4</td>
<td>3.9</td>
<td>1.7</td>
<td>0.4</td>
<td><em>I. setosa</em></td>
</tr>
<tr>
<td>4.6</td>
<td>3.4</td>
<td>1.4</td>
<td>0.3</td>
<td><em>I. setosa</em></td>
</tr>
<tr>
<td>5.0</td>
<td>3.4</td>
<td>1.5</td>
<td>0.2</td>
<td><em>I. setosa</em></td>
</tr>
<tr>
<td>4.4</td>
<td>2.9</td>
<td>1.4</td>
<td>0.2</td>
<td><em>I. setosa</em></td>
</tr>
<tr>
<td>4.9</td>
<td>3.1</td>
<td>1.5</td>
<td>0.1</td>
<td><em>I. setosa</em></td>
</tr>
</tbody>
</table>
Pairwise scatter plots of 131 flower specimens

Visualization of data helps identify the right learning model to use

**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
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): \( x \in \mathbb{R}^D \)
- Output (label): \( y \in [C] = \{1, 2, \cdots, C\} \)
- Learning goal: \( y = f(x) \)

Recall special case: binary classification

- Number of classes: \( C = 2 \)
- Labels: \( \{0, 1\} \) or \( \{-1, +1\} \)
Training data (set)

- $N$ samples/instances: $\mathcal{D}^{\text{TRAIN}} = \{(x_1, y_1), (x_2, y_2), \cdots, (x_N, y_N)\}$
- They are used for learning $f(\cdot)$

Test (evaluation) data

- $M$ samples/instances: $\mathcal{D}^{\text{TEST}} = \{(x_1, y_1), (x_2, y_2), \cdots, (x_M, y_M)\}$
- They are used for assessing how well $f(\cdot)$ will do in predicting an unseen $x \notin \mathcal{D}^{\text{TRAIN}}$

Training data and test data should *not* overlap: $\mathcal{D}^{\text{TRAIN}} \cap \mathcal{D}^{\text{TEST}} = \emptyset$
Nearest neighbor classification (NNC)

**Nearest neighbor**

\[ x(1) = x_{\text{nn}(x)} \]

where \( \text{nn}(x) \in \mathbb{N} = \{1, 2, \cdots, N\} \), i.e., the index to one of the training instances
Nearest neighbor classification (NNC)

**Nearest neighbor**

\[ x(1) = x_{nn}(x) \]

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

\[ nn(x) = \arg\min_{n \in [N]} \|x - x_n\|_2^2 = \arg\min_{n \in [N]} \sum_{d=1}^{D} (x_d - x_{nd})^2 \]
Nearest neighbor classification (NNC)

**Nearest neighbor**

\[ x(1) = x_{nn}(x) \]

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

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

**Classification rule**

\[ y = f(x) = y_{nn}(x) \]
In this 2-dimensional example, the nearest point to $x$ is a red training instance, thus, $x$ will be labeled as red.
Example: classify Iris with two features

<table>
<thead>
<tr>
<th>ID (n)</th>
<th>petal width ($x_1$)</th>
<th>sepal length ($x_2$)</th>
<th>category (y)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0.2</td>
<td>5.1</td>
<td>setosa</td>
</tr>
<tr>
<td>2</td>
<td>1.4</td>
<td>7.0</td>
<td>versicolor</td>
</tr>
<tr>
<td>3</td>
<td>2.5</td>
<td>6.7</td>
<td>virginica</td>
</tr>
</tbody>
</table>

Flower with unknown category
petal width = 1.8 and sepal length = 6.4

Calculating distance = \((x_1 - x_n)^2 + (x_2 - x_n)^2\)

<table>
<thead>
<tr>
<th>ID</th>
<th>distance</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1.6</td>
</tr>
<tr>
<td>2</td>
<td>1.3</td>
</tr>
<tr>
<td>3</td>
<td>0.52</td>
</tr>
<tr>
<td>3</td>
<td>0.58</td>
</tr>
</tbody>
</table>

Thus, the predicted category is versicolor
Example: classify Iris with two features

Training data

<table>
<thead>
<tr>
<th>ID (n)</th>
<th>petal width ($x_1$)</th>
<th>sepal length ($x_2$)</th>
<th>category (y)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0.2</td>
<td>5.1</td>
<td>setosa</td>
</tr>
<tr>
<td>2</td>
<td>1.4</td>
<td>7.0</td>
<td>versicolor</td>
</tr>
<tr>
<td>3</td>
<td>2.5</td>
<td>6.7</td>
<td>virginica</td>
</tr>
</tbody>
</table>

Flower with unknown category

petal width = 1.8 and sepal length = 6.4
Example: classify Iris with two features

Training data

<table>
<thead>
<tr>
<th>ID (n)</th>
<th>petal width ($x_1$)</th>
<th>sepal length ($x_2$)</th>
<th>category ($y$)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0.2</td>
<td>5.1</td>
<td>setosa</td>
</tr>
<tr>
<td>2</td>
<td>1.4</td>
<td>7.0</td>
<td>versicolor</td>
</tr>
<tr>
<td>3</td>
<td>2.5</td>
<td>6.7</td>
<td>virginica</td>
</tr>
</tbody>
</table>

Flower with unknown category

petal width = 1.8 and sepal length = 6.4
Calculating distance = $(x_1 - x_{n1})^2 + (x_2 - x_{n2})^2$

<table>
<thead>
<tr>
<th>ID</th>
<th>distance</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>$1.6^2 + 1.3^2 = 4.25$</td>
</tr>
<tr>
<td>2</td>
<td>$0.4^2 + 0.6^2 = 0.52$</td>
</tr>
<tr>
<td>3</td>
<td>$0.7^2 + 0.3^2 = 0.58$</td>
</tr>
</tbody>
</table>

Thus, the predicted category is *versicolor*
How to measure nearness with other distances?

So far we use the Euclidean distance

$$nn(x) = \arg\min_{n \in [N]} \|x - x_n\|_2^2$$
How to measure nearness with other distances?

So far we use the Euclidean distance

$$nn(x) = \arg\min_{n \in [N]} \|x - x_n\|_2^2$$

We can also use alternative distances

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

$$nn(x) = \arg\min_{n \in [N]} \|x - x_n\|_1$$

$$= \arg\min_{n \in [N]} \sum_{d=1}^D |x_d - x_{nd}|$$
How to measure nearness with other distances?

So far we use the Euclidean distance

$$\text{nn}(x) = \arg\min_{n \in [N]} \| x - x_n \|_2^2$$

We can also use alternative distances
E.g., the following $L_1$ distance (i.e., city block distance, or Manhattan distance)

$$\text{nn}(x) = \arg\min_{n \in [N]} \| x - x_n \|_1$$

$$= \arg\min_{n \in [N]} \sum_{d=1}^{D} |x_d - x_{nd}|$$

**Figure 2:** Green line is Euclidean distance. Red, Blue, and Yellow lines are $L_1$ distance
Example: classify Iris with two features

Now use the Manhattan distance instead of Euclidean distance

<table>
<thead>
<tr>
<th>ID (n)</th>
<th>petal width ($x_1$)</th>
<th>sepal length ($x_2$)</th>
<th>category (y)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0.2</td>
<td>5.1</td>
<td>setosa</td>
</tr>
<tr>
<td>2</td>
<td>1.4</td>
<td>7.0</td>
<td>versicolor</td>
</tr>
<tr>
<td>3</td>
<td>2.5</td>
<td>6.7</td>
<td>virginica</td>
</tr>
</tbody>
</table>
Example: classify Iris with two features

Now use the Manhattan distance instead of Euclidean distance

<table>
<thead>
<tr>
<th>ID (n)</th>
<th>petal width ($x_1$)</th>
<th>sepal length ($x_2$)</th>
<th>category ($y$)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0.2</td>
<td>5.1</td>
<td>setosa</td>
</tr>
<tr>
<td>2</td>
<td>1.4</td>
<td>7.0</td>
<td>versicolor</td>
</tr>
<tr>
<td>3</td>
<td>2.5</td>
<td>6.7</td>
<td>virginica</td>
</tr>
</tbody>
</table>

Flower with unknown category
petal width = 1.8 and sepal length = 6.4
Example: classify Iris with two features

Now use the Manhattan distance instead of Euclidean distance

<table>
<thead>
<tr>
<th>ID (n)</th>
<th>petal width ($x_1$)</th>
<th>sepal length ($x_2$)</th>
<th>category ($y$)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0.2</td>
<td>5.1</td>
<td>setosa</td>
</tr>
<tr>
<td>2</td>
<td>1.4</td>
<td>7.0</td>
<td>versicolor</td>
</tr>
<tr>
<td>3</td>
<td>2.5</td>
<td>6.7</td>
<td>virginica</td>
</tr>
</tbody>
</table>

Flower with unknown category
petal width = 1.8 and sepal length = 6.4
Calculating distance = $|x_1 - x_{n1}| + |x_2 - x_{n2}|$

<table>
<thead>
<tr>
<th>ID</th>
<th>distance</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1.6 + 1.3 = 2.9</td>
</tr>
<tr>
<td>2</td>
<td>0.4 + 0.6 = 1.0</td>
</tr>
<tr>
<td>3</td>
<td>0.7 + 0.3 = 1.0</td>
</tr>
</tbody>
</table>

It is a tie between *versicolor* and *virginica* (the real category is *virginica*)
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.

![Decision boundary](image)
Decision boundary is sensitive to outliers

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.
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.

*Increase the number of nearest neighbors that we user?*
K-nearest neighbor (KNN) classification

Increase the number of nearest neighbors to use?

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

Increase the number of nearest neighbors to use?

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

The set of K-nearest neighbor

\[ \text{knn}(x) = \{ \text{nn}_1(x), \text{nn}_2(x), \ldots, \text{nn}_K(x) \} \]

Let \( x(k) = x_{\text{nn}_k(x)} \), then

\[ \|x - x(1)\|_2^2 \leq \|x - x(2)\|_2^2 \leq \cdots \leq \|x - x(K)\|_2^2 \]
How to classify with $K$ neighbors?

**Classification rule**

- Every neighbor votes: suppose $y_n$ (the true label) for $x_n$ is $c$, then
  - vote for $c$ is 1
  - vote for $c'$ $\neq c$ is 0

We use the indicator function $I(y_n = c)$ to represent.

- Aggregate everyone’s vote $v_c = \sum_{n \in \text{knn}(x)} I(y_n = c)$, $\forall c \in [C]$

- Label with the majority $y = f(x) = \text{arg max}_{c \in [C]} v_c$
How to classify with $K$ neighbors?

Classification rule

- Every neighbor votes: suppose $y_n$ (the true label) for $x_n$ is $c$, then
  - vote for $c$ is $1$
  - vote for $c' \neq c$ is $0$

  We use the *indicator function* $\mathbb{1}(y_n == c)$ to represent.

- Aggregate everyone’s vote

  $$v_c = \sum_{n \in \text{knn}(x)} \mathbb{1}(y_n == c), \quad \forall \quad c \in [C]$$

- Label with the majority

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

K=1, Label: ??

K=3, Label: ??

K=5, Label: ??
Example

K=1, Label: red

K=3, Label: red

K=5, Label: blue
Larger $K$ fixes the previous 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.
How to choose an optimal $K$?

When $K$ increases, the decision boundary becomes smooth.
How to choose an optimal $K$?

When $K$ increases, the decision boundary becomes smooth.
Advantages of NNC

- Computationally, simple and easy to implement – just compute distances
- Can learn complex decision boundaries
Advantages of NNC

- Computationally, simple and easy to implement – just compute distances
- Can learn complex decision boundaries

Disadvantages of NNC

- Computationally intensive for large-scale problems: \( O(ND) \) 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 practical issues about 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

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

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

- N samples/instances: $D_{\text{TRAIN}} = \{(x_1, y_1), (x_2, y_2), \ldots, (x_N, y_N)\}$
- They are used for learning $f(\cdot)$

---

Test data

- M samples/instances: $D_{\text{TEST}} = \{(x_1, y_1), (x_2, y_2), \ldots, (x_M, y_M)\}$
- They are used for assessing how well $f(\cdot)$ will do in predicting an unseen $x / \in D_{\text{TRAIN}}$

Validation data

- L samples/instances: $D_{\text{VAL}} = \{(x_1, y_1), (x_2, y_2), \ldots, (x_L, y_L)\}$
- They are used to optimize hyperparameter(s).

Training data, validation and test data should not overlap!
Tuning by using a validation dataset

Training data

- N samples/instances: $D^{\text{TRAIN}} = \{(x_1, y_1), (x_2, y_2), \ldots, (x_N, y_N)\}$
- They are used for learning $f(\cdot)$

Test data

- M samples/instances: $D^{\text{TEST}} = \{(x_1, y_1), (x_2, y_2), \ldots, (x_M, y_M)\}$
- They are used for assessing how well $f(\cdot)$ will do in predicting an unseen $x \notin D^{\text{TRAIN}}$
Training data

- N samples/instances: $D^{\text{TRAIN}} = \{(x_1, y_1), (x_2, y_2), \cdots, (x_N, y_N)\}$
- They are used for learning $f(\cdot)$

Test data

- M samples/instances: $D^{\text{TEST}} = \{(x_1, y_1), (x_2, y_2), \cdots, (x_M, y_M)\}$
- They are used for assessing how well $f(\cdot)$ will do in predicting an unseen $x \notin D^{\text{TRAIN}}$

Validation data

- L samples/instances: $D^{\text{VAL}} = \{(x_1, y_1), (x_2, y_2), \cdots, (x_L, y_L)\}$
- 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 $K = 1, 3, \cdots, 100$)
Recipe

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

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

- Train a model using $D^{\text{TRAIN}}$
- Evaluate the performance of the model on $D^{\text{VAL}}$

Choose the model with the best performance on $D^{\text{VAL}}$
Recipe

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

**Figure 3: $S = 5$: 5-fold cross validation**
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.)

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

Figure 3: $S = 5$: 5-fold cross validation
Yet, another practical issue with NNC

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

\[
\bar{x}_d = \frac{1}{N} \sum_{n} x_{nd}, \quad s^2_d = \frac{1}{N-1} \sum_{n} (x_{nd} - \bar{x}_d)^2
\]

- 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

- Described a simple *nonparametric* learning algorithm
- Discussed practical aspects such as tuning hyperparameters using cross-validation
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, how to find the Lagrangian, kernel functions