DSC 140B

Representation Learning

Lecture 09 | Part 1

Recap

Last Time

- We saw how to **embed** a similarity graph into $\mathbb{R}^k$.
- **Goal**: similar nodes are embedded nearby.
- **Approach**:
  1. Construct the **graph Laplacian** L.
  2. Compute the bottom k eigenvectors of L whose eigenvalues are > zero.
  3. The k new features for the ith node are given by the ith components of these eigenvectors.

Now

- What if we aren't given a similarity graph, but points in $\mathbb{R}^d$?

DSC 140B

Representation Learning

Lecture 09 | Part 2

Laplacian Eigenmaps

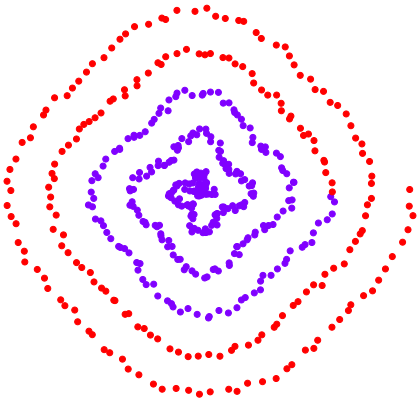
Problem: Non-linear Dimensionality Reduction

- **Given**: points in $\mathbb{R}^d$, target dimension k
- **Goal**: **embed** the points in $\mathbb{R}^k$ so that points that were close in geodesic distance are close after

Idea

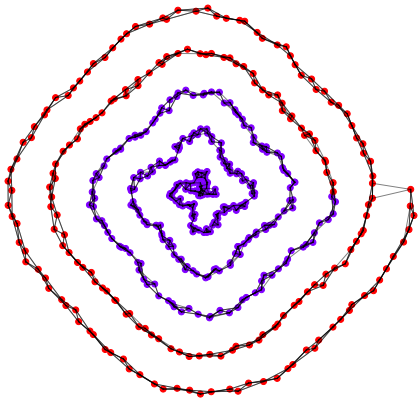
- Build a similarity graph from points in $\mathbb{R}^d$
  - epsilon neighbors, k-neighbors, or fully connected
- Embed the similarity graph in $\mathbb{R}^k$ using eigenvectors of graph Laplacian

Example 1: Spiral



Example 1: Spiral

- Build a k-neighbors graph.
- Note: follows the 1-d shape of the data.



Example 1: Spectral Embedding

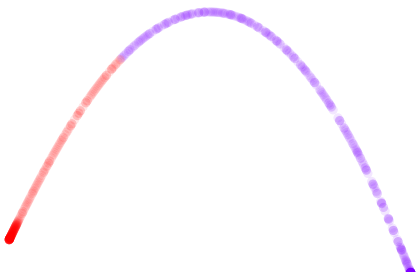
- Let W be the weight matrix (k-neighbor adjacency matrix)
- Compute $L = D - W$
- Compute bottom k non-zero eigenvectors of L, use as embedding

Example 1: Spiral

- Embedding into $\mathbb{R}^1$

Example 1: Spiral

- Embedding into $\mathbb{R}^2$



Example 1: Spiral

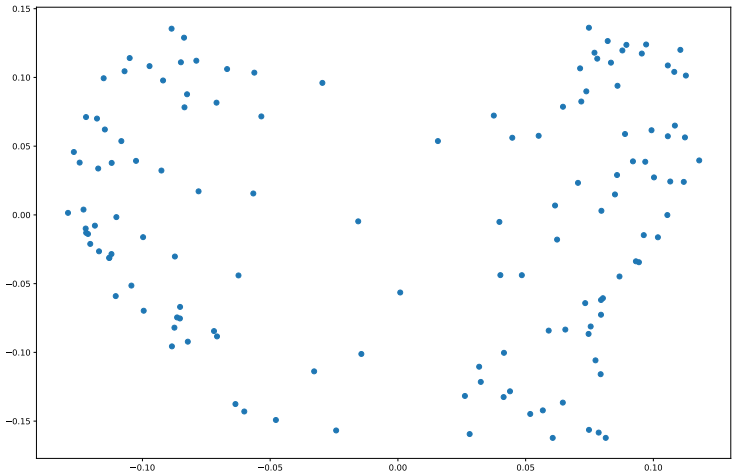
```
import sklearn.neighbors
import sklearn.manifold
W = sklearn.neighbors.kneighbors_graph(
        X, n_neighbors=4
)
embedding = sklearn.manifold.spectral_embedding(
        W, n_components=2
)
```

Example 2: Face Pose

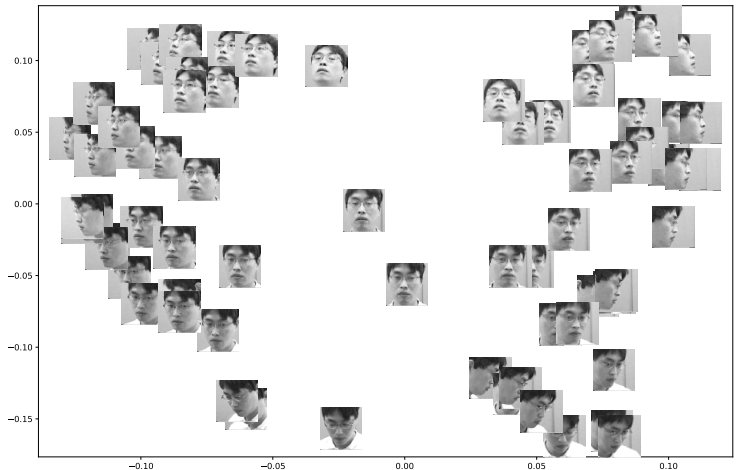
Example 2: Face Pose

- Construct fully-connected similarity graph with Gaussian similarity
- Embed with Laplacian eigenmaps

Example 2: Face Pose



Example 2: Face Pose



DSC 140B - Part 2

Deep Learning

So Far

- We have focused on **dimensionally reduction**.
- Useful for visualization and EDA, but also for making better predictions.

Now

- Let's shift focus to **prediction**.
- We're not stuck with the original features: we can transform the data to a new representation.
- **Goal**: find a representation that makes prediction **easier**.

DSC 140B

Representation Learning

Lecture 09 | Part 3

Linear Predictors

Prediction

- Prediction is maybe *the* core ML task.

- **Regression**: predict a continuous value.
  - Predict house price given size, location, etc.
  - Predict future salary given GPA, experience, etc.

- **Classification**: predict a discrete label.
  - Predict species of bird in a photo (**multiclass**).
  - Predict if parking is available on campus based on time of day, day of week, etc. (**binary**).

Prediction Models

- There are many ways to approach prediction.
  - We talk about them in DSC 140A.
- For now, we will focus on **linear predictors**.

Review: Linear Predictors

- **Linear predictors** make predictions using a **linear combination** of the input features:

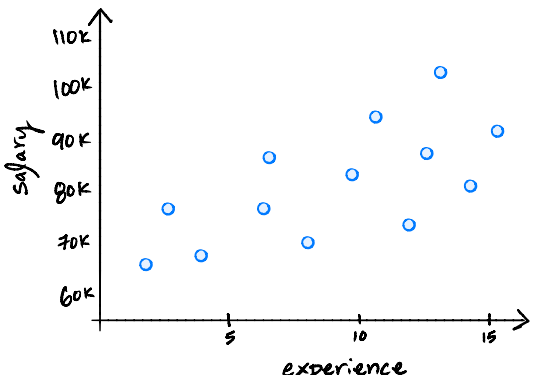
$$H(\vec{x}) = w_0 + w_1 x_1 + w_2 x_2 + \cdots + w_d x_d$$

- Here:
  - H is the **prediction function**.
  - $\vec{x} = (x_1, x_2, \ldots, x_d)$ are the input **features**.
  - $w_0, w_1, \ldots, w_d$ are the **model parameters** (weights).

Example

- A linear prediction function for salary.

$$H_1(\vec{x}) = \$50{,}000 + (\text{experience}) \times \$8{,}000$$



Weight Vectors

- We package the weights into a **parameter vector**:

$$\vec{w} = (w_0, w_1, w_2, \ldots, w_d)$$

- Then:

$$\begin{aligned} H(\vec{x}) &= w_0 + w_1 x_1 + w_2 x_2 + \cdots + w_d x_d \\ &= \vec{w} \cdot \text{Aug}(\vec{x}) \end{aligned}$$

where $\text{Aug}(\vec{x}) = (1, x_1, x_2, \ldots, x_d)$.

Learning the Weights

- The weights totally determine the model.
- But how do we find the weights?
- DSC 40A: **empirical risk minimization** (ERM).

Empirical Risk Minimization (ERM)

- Step 1: choose a **hypothesis class**
  - We've chosen linear predictors: $H(\vec{x}) = \vec{w} \cdot \text{Aug}(\vec{x})$
- Step 2: choose a **loss function**
- Step 3: minimize **expected loss (empirical risk)**

Loss Function

- A **loss function** quantifies how wrong a single prediction is.

$$L(H(\vec{x}^{(i)}),\ y_i)$$

L(prediction for example i, correct answer for example i)

Empirical Risk

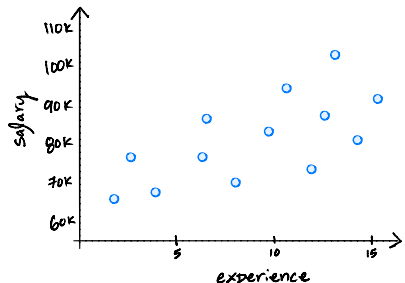
- A good H is good *on average* over entire data set.
- The **expected loss** (or **empirical risk**) is one way of measuring this:

$$R(H) = \frac{1}{n} \sum_{i=1}^{n} L(H(\vec{x}^{(i)}), y_i)$$

- Note: depends on H and the data!

Loss Functions for Regression

- We want $H(\vec{x}^{(i)}) \approx y_i$.
- **Absolute loss**: $|H(\vec{x}^{(i)}) - y_i|$
- **Square loss**: $(H(\vec{x}^{(i)}) - y_i)^2$



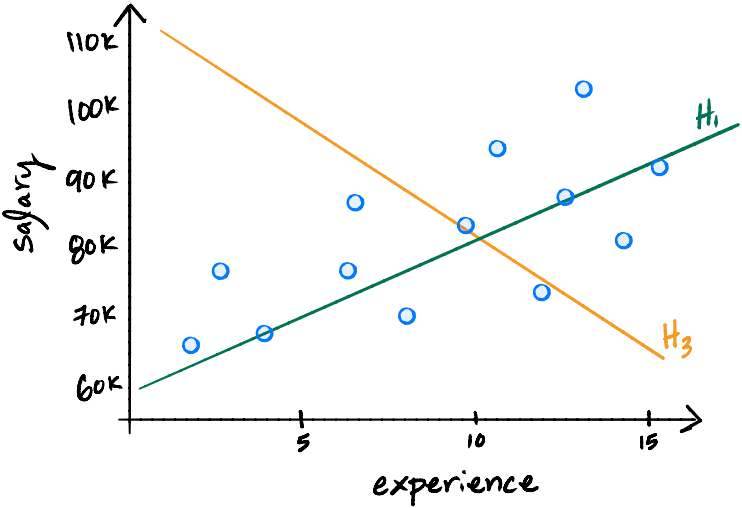
Mean Squared Error

- **Expected square loss** (mean squared error):

$$R_{\text{sq}}(H) = \frac{1}{n}\sum_{i=1}^{n}(H(\vec{x}^{(i)}) - y_i)^2$$

- This is the empirical risk for the square loss.

- **Goal:** find H minimizing MSE.


110K
100K
90K
80K
70K
60K
salary
5
10
15
experience
H_1
H_3

Solution

- In DSC 40A we used calculus to find the solution.
- We derived the **normal equations**:

$$\vec{w} = (X^T X)^{-1} X^T \vec{y}$$

  where X is the $n \times d$ **design matrix** and $\vec{y}$ is the vector of target values.

- This is called **least squares regression**.

DSC 140B

Representation Learning

Lecture 09 | Part 4

Least Squares Classifiers

Movie Ratings

- Five of your friends rate a movie from 0-10:
  - x_1: 9
  - x_2: 3
  - x_3: 7
  - x_4: 2
  - x_5: 8
- **Task**: Will you like the movie? (yes / no)

Classification

- Linear prediction functions can be used in classification, too.

$$H(\vec{x}) = w_0 + w_1 x_1 + w_2 x_2 + \ldots + w_d x_d$$

- Same ERM paradigm also useful.

A Classifier from a Regressor

- Binary classification can be thought of as regression where the targets are 1 and -1
  - (or 0 and 1, or ...)

- $H(\vec{x})$ outputs a real number. Use the **sign** function to turn it into –1, 1:

$$\text{sign}(z) = \begin{cases} 1 & z > 0 \\ -1 & z < 0 \\ 0 & \text{otherwise} \end{cases}$$

- Final prediction: $\text{sign}(H(\vec{x}))$

Exercise

Suppose H is a linear prediction function with parameter vector $\vec{w} = (2, -1, 3)^T$.

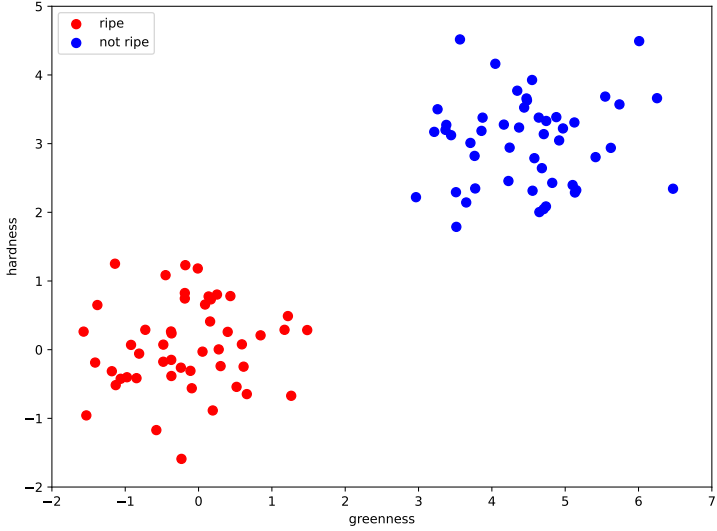
What is the predicted class label for the input $\vec{x} = (1, -4)^T$?

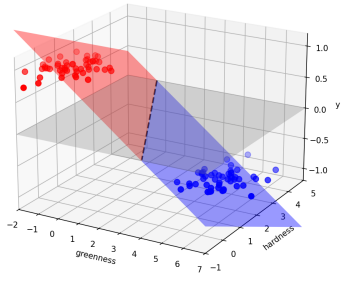
Example: Mango Ripeness

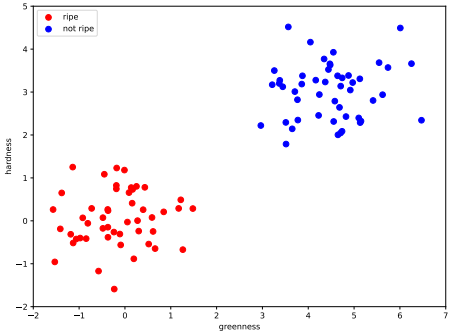
- Predict whether a mango is ripe given greenness and hardness.

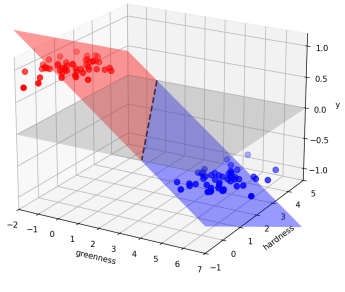
- Idea: gather a set of labeled **training data**.
  - Inputs along with correct output (i.e., "the answer").

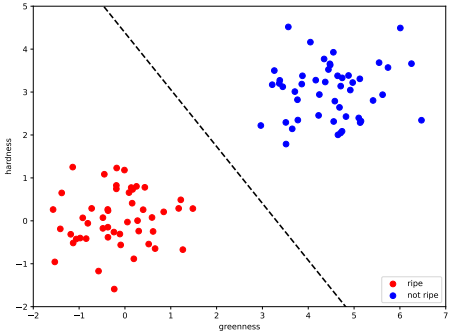
Greenness	Hardness	Ripe
0.7	0.9	1
0.2	0.5	-1
0.3	0.1	-1
⋮	⋮	⋮


ripe
not ripe
hardness
greenness
5
4
3
2
1
0
−1
−2
−2
−1
0
1
2
3
4
5
6
7


greenness
hardness
y


ripe
not ripe
hardness
greenness


greenness
hardness
y


greenness
hardness
ripe
not ripe

Decision Boundary

- The **decision boundary** is the place where the output of $H(x)$ switches from "yes" to "no".
  - If $H > 0 \mapsto$ "yes" and $H < 0 \mapsto$ "no", the decision boundary is where $H = 0$.

- If H is a linear predictor and[1]
  - $\vec{x} \in R^1$, then the decision boundary is just a number.
  - $\vec{x} \in \mathbb{R}^2$, the boundary is a straight line.
  - $\vec{x} \in \mathbb{R}^d$, the boundary is a $d - 1$ dimensional (hyper) plane.

[1] when plotted in the original feature coordinate space!

Empirical Risk Minimization

- Step 1: choose a **hypothesis class**
  - Let's assume we've chosen linear predictors
- Step 2: choose a **loss function**
- Step 3: minimize **expected loss (empirical risk)**

Square Loss for Classification

- We can use the square loss for classification.

$$L(H(\vec{x}^{(i)}), y_i) = (H(\vec{x}^{(i)}) - y_i)^2$$

- We get the **exact same** solution:

$$\vec{w} = (X^T X)^{-1} X^T \vec{y}$$

- This is called the **least squares classifier**.

Exercise

Can you think of a reason why the square loss might not be the best choice for classification?

Least Squares and Outliers

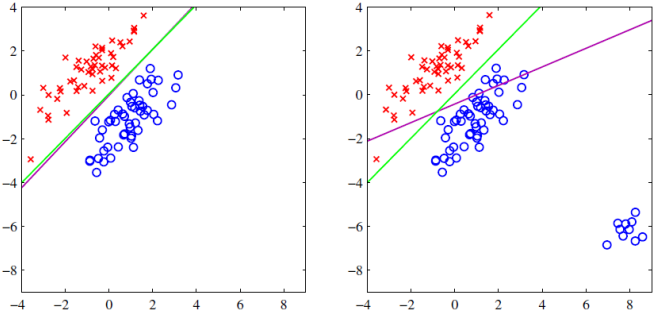


Figure 4.4 The left plot shows data from two classes, denoted by red crosses and blue circles, together with the decision boundary found by least squares (magenta curve) and also by the logistic regression model (green curve), which is discussed later in Section 4.3.2. The right-hand plot shows the corresponding results obtained when extra data points are added at the bottom left of the diagram, showing that least squares is highly sensitive to outliers, unlike logistic regression.

2

[2]Bishop, *Pattern Recognition and Machine Learning*

Square Loss for Classification

- We **can** use the square loss for classification
  - The “least squares classifier”
- However, the square loss penalizes being “too correct”
- **Example**: suppose the correct label is 1. What is the square loss of predicting 10? -9?

Loss Functions

- There are many different loss functions for classification.
- Each leads to a different classifier:
  - Logistic Regression
  - Support Vector Machine
  - Perceptron
  - etc.
- But that's for another class... (DSC 140A)

DSC 140B

Representation Learning

Lecture 09 | Part 5

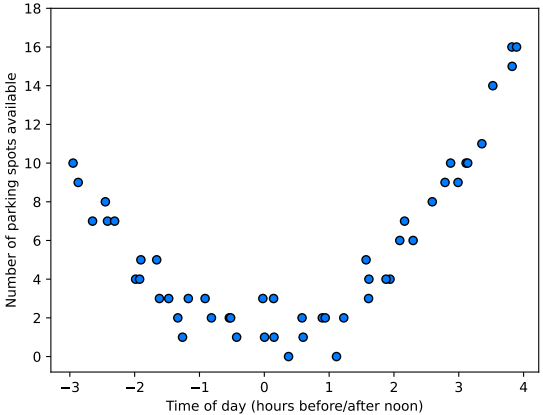
Feature Maps

Problem

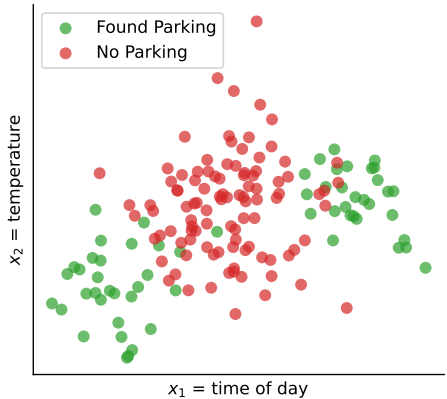
- Linear predictors only learn **linear** patterns.
  - Straight lines / planes.

- Patterns in real world data are often **non-linear**.

Example: Regression



Example: Classification



Idea

- The pattern is **non-linear** in its original representation.

- **Idea:**
  1. Change to a new representation where the pattern is **linear**.

  2. Train a linear predictor in the new representation.

Idea

- How do we change to a new representation?
- **Solution**: non-linear **feature maps**.
- Will allow us to:
  - fit complex, non-linear patterns;
  - while still using linear models

Feature Map

- A **feature map** $\vec{\phi} : \mathbb{R}^d \to \mathbb{R}^k$ is a function that takes in a d-dimensional vector and outputs a k-dimensional feature vector.

- I.e., it creates new features from the old ones.
  - Maybe in a non-linear way.

Example

- Define $\vec{\phi} : \mathbb{R}^2 \to \mathbb{R}^5$ as:

$$\vec{\phi}(x_1, x_2) = (x_1,\ x_2,\ x_1^2,\ x_2^2,\ x_1 x_2)^T$$

- If $\vec{x} = (2, 3)^T$, then:

$$\begin{aligned}\vec{\phi}(\vec{x}) &= (2,\ 3,\ 2^2,\ 3^2,\ 2 \times 3)^T \\ &= (2,\ 3,\ 4,\ 9,\ 6)^T\end{aligned}$$

Basis Functions

- A **basis function** is a function $\phi_j : \mathbb{R}^d \to \mathbb{R}$.
- It takes in an old feature vector and outputs a single new feature.
- We can think of a feature map $\vec{\phi} : \mathbb{R}^d \to \mathbb{R}^k$ as being made up of k basis functions.

$$\vec{\phi}(\vec{x}) = (\phi_1(\vec{x}), \phi_2(\vec{x}), \ldots, \phi_k(\vec{x}))^T$$

Example

- Let $\vec{\phi} : \mathbb{R}^2 \to \mathbb{R}^5$ be defined as:

$$\vec{\phi}(x_1, x_2) = (x_1, x_2, x_1^2, x_2^2, x_1 x_2)^T$$

- The corresponding basis functions are:

$$\phi_1(x_1, x_2) = x_1 \qquad \phi_2(x_1, x_2) = x_2$$
$$\phi_3(x_1, x_2) = x_1^2 \qquad \phi_4(x_1, x_2) = x_2^2$$
$$\phi_5(x_1, x_2) = x_1 x_2$$

A New Data Set

- Say we have a training set with d features:

$$(\vec{x}^{(1)}, y_1), \ldots, (\vec{x}^{(n)}, y_n)$$

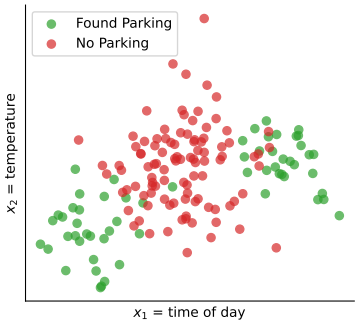
- A feature map $\vec{\phi} : \mathbb{R}^d \to \mathbb{R}^k$ gives us a **new** training set with k features:

$$(\vec{\phi}(\vec{x}^{(1)}), y_1), \ldots, (\vec{\phi}(\vec{x}^{(n)}), y_n)$$

Why?

- A (good) feature map can turn **non-linear** patterns in the old data into **linear patterns** in the new data.

Example: Parking Classification



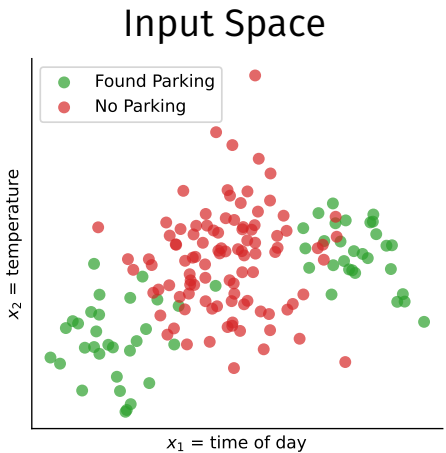
- Original features:

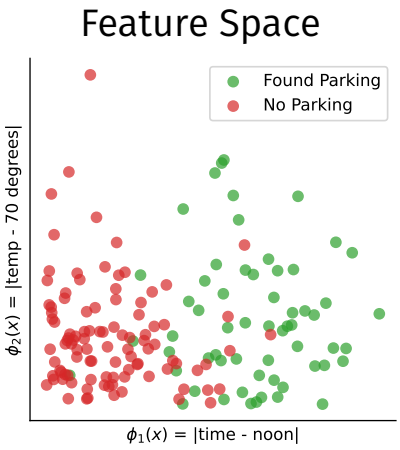
$$\vec{x} = (\text{time}, \text{temp.})^T$$

- Feature map:

$$\vec{\phi}(\vec{x}) = (|\text{time} - \text{Noon}|, |\text{temp.} - 70|)^T$$

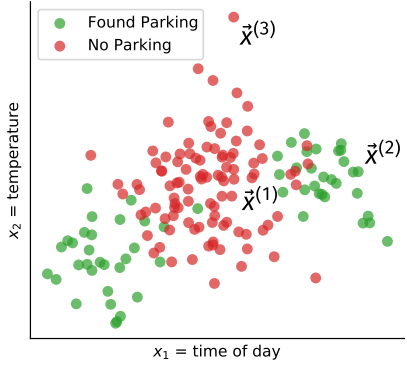
Example: Parking Classification

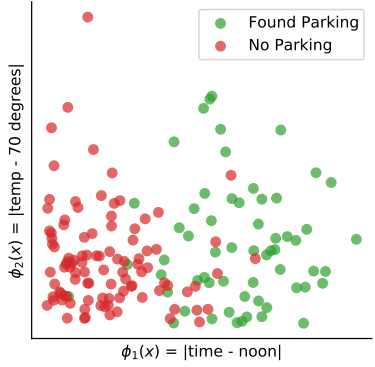




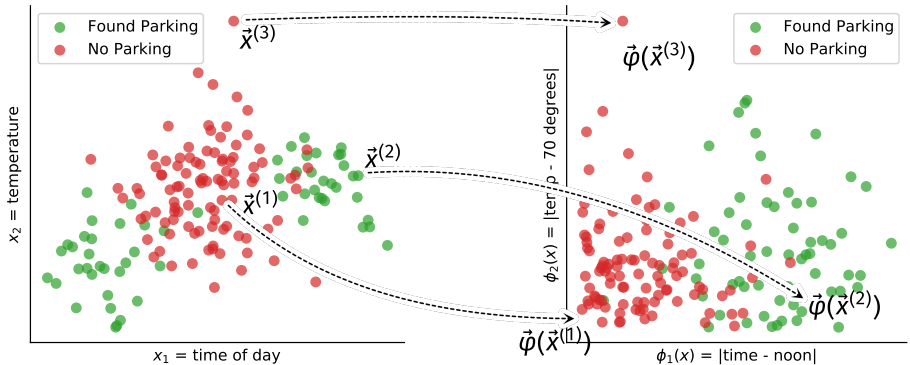
Exercise

(Approximately) where do $\vec{x}^{(1)}$, $\vec{x}^{(2)}$, and $\vec{x}^{(3)}$ get mapped to in feature space?





Solution


Found Parking
No Parking
$\vec{x}^{(3)}$
$\vec{x}^{(2)}$
$\vec{x}^{(1)}$
$\vec{\varphi}(\vec{x}^{(3)})$
$\vec{\varphi}(\vec{x}^{(2)})$
$\vec{\varphi}(\vec{x}^{(1)})$
x_2 = temperature
x_1 = time of day
$\phi_2(x)$ = |temp - 70 degrees|
$\phi_1(x)$ = |time - noon|
Found Parking
No Parking

Idea

- Feature maps turned **non-linear** patterns in input space into **linear** patterns in feature space.
- **Idea:** train a linear model in feature space.

Procedure: Learning with Feature Maps

- First, pick a feature map $\vec{\phi} : \mathbb{R}^d \to \mathbb{R}^k$.
- **To train**:
  - Given training set $(\vec{x}^{(1)}, y_1), \ldots, (\vec{x}^{(n)}, y_n)$.
  1. Map each $\vec{x}^{(i)}$ to feature space, creating a new data set $(\vec{\phi}(\vec{x}^{(1)}), y_1), \ldots, (\vec{\phi}(\vec{x}^{(n)}), y_n)$.
  2. Train linear model on the new data in feature space to get $\vec{w}^*$.
- **To predict:**
  - Given new input $\vec{x}$.
  1. Map $\vec{x}$ to feature space: $\vec{\phi}(\vec{x})$.
  2. Predict $H(\vec{x}; \vec{w}^*) = \vec{w}^* \cdot \text{Aug}(\vec{\phi}(\vec{x}))$.

Exercise

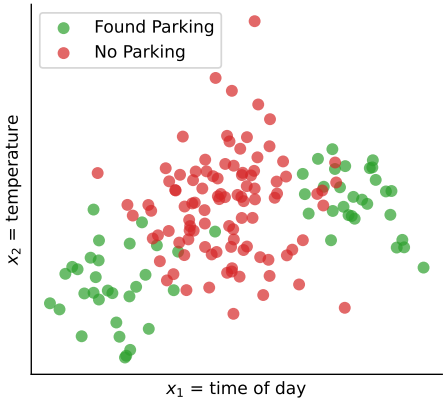
Suppose the original feature vectors are in $\mathbb{R}^2$ and the feature map is defined as

$$\vec{\phi}(x_1, x_2) = (x_1, x_2, x_1^2, x_2^2, x_1 x_2)^T$$

We train a least squares classifier in feature space. What is the dimensionality of $\vec{w}^*$?

Example: Least Squares

- Let's train a least squares classifier using a feature map.



Step 1: Pick a Feature Map

- In the input space, we have features (x_1, x_2).

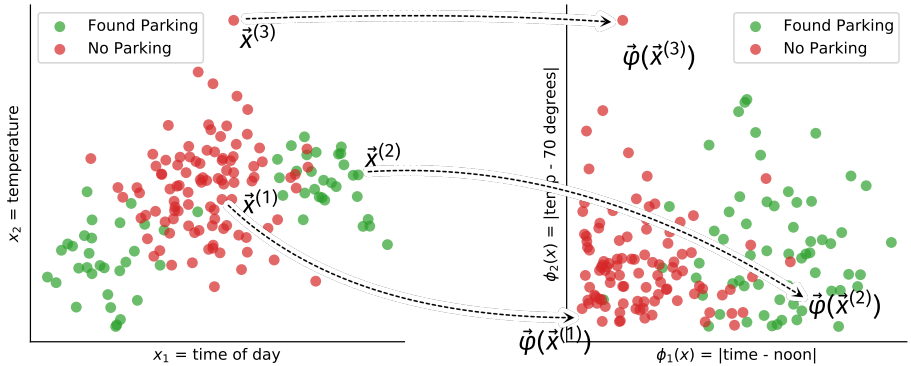
$$x_1 = \text{time}, \quad x_2 = \text{temperature}.$$

- We'll use the same feature map as before:

$$\vec{\phi}(x_1, x_2) = (|x_1 - 12|, |x_2 - 70|)^T$$

Step 2(a): Map to Feature Space

- Map every data point to feature space.



Step 2(b): Train in Feature Space

- Recall: we train a least squares classifier in **input space** by computing:

$$\vec{w}^* = (\vec{X}^T\vec{X})^{-1}\vec{X}^T\vec{y}$$

- Here, X is the (augmented) $(n \times d)$ design matrix:

$$X = \begin{pmatrix} \text{Aug}(\vec{x}^{(1)})^T \longrightarrow \\ \text{Aug}(\vec{x}^{(2)})^T \longrightarrow \\ \vdots \\ \text{Aug}(\vec{x}^{(n)})^T \longrightarrow \end{pmatrix} = \begin{pmatrix} 1 & x_1^{(1)} & x_2^{(1)} \\ 1 & x_1^{(2)} & x_2^{(2)} \\ \vdots & \vdots & \vdots \\ 1 & x_1^{(n)} & x_2^{(n)} \end{pmatrix}$$

Step 2(b): Train in Feature Space

- In feature space, our feature vectors are $\vec{\phi}(\vec{x}^{(1)}), \ldots, \vec{\phi}(\vec{x}^{(n)})$.

- So the design matrix becomes the $(n \times k)$ matrix:

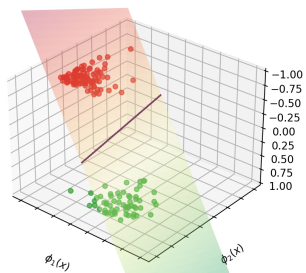
$$\Phi = \begin{pmatrix} \vec{\phi}(\vec{x}^{(1)})^T \longrightarrow \\ \vec{\phi}(\vec{x}^{(2)})^T \longrightarrow \\ \vdots \\ \vec{\phi}(\vec{x}^{(n)})^T \longrightarrow \end{pmatrix} = \begin{pmatrix} 1 & |x_1^{(1)} - 12| & |x_2^{(1)} - 70| \\ 1 & |x_1^{(2)} - 12| & |x_2^{(2)} - 70| \\ \vdots & \vdots & \vdots \\ 1 & |x_1^{(n)} - 12| & |x_2^{(n)} - 70| \end{pmatrix}$$

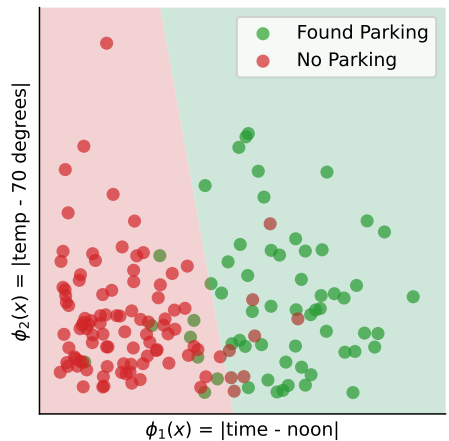
Step 2(b): Train in Feature Space

- The least squares solution in feature space is:

$$\vec{w}^* = (\Phi^T \Phi)^{-1} \Phi^T \vec{y}$$

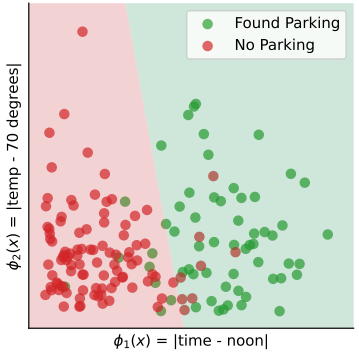
Solution in Feature Space





Step 3: Predict

- Given a new example $\vec{x}$ in input space:
  1. Map $\vec{x}$ to feature space: $\vec{\phi}(\vec{x})$.
  2. Predict sign($\vec{w}^* \cdot \text{Aug}(\vec{\phi}(\vec{x}))$).



Exercise

Let $\vec{\phi}(x_1, x_2) = (|x_1 - 12|, |x_2 - 70|)^T$. Suppose we train a least squares classifier in feature space and find $\vec{w}^* = (3, -1, 2)^T$.

Given a new point $\vec{x} = (10, 65)^T$ in input space, what is the prediction, $H(\vec{x})$?

The Prediction Function(s)

- There are, in a sense, **two** prediction functions to consider.

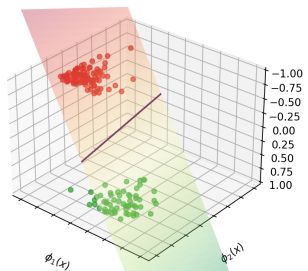
- First, the prediction function in feature space:

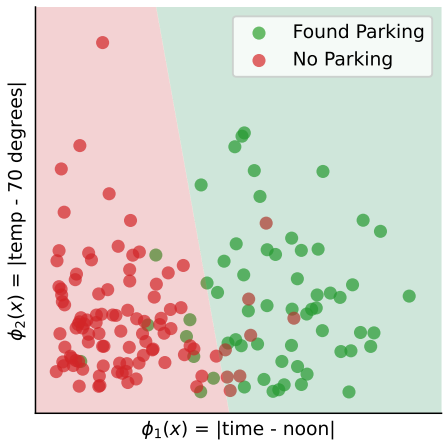
$$H_\phi(\vec{z}) = \vec{w} \cdot \text{Aug}(\vec{z})$$
$$= w_0 + w_1 z_1 + w_2 z_2 + \ldots + w_k z_k$$

- This function takes in a vector $\vec{z}$ that is already in feature space.

H_ϕ in Feature Space

$$H_\phi(\vec{z}) = w_0 + w_1 z_1 + w_2 z_2$$





The Prediction Function

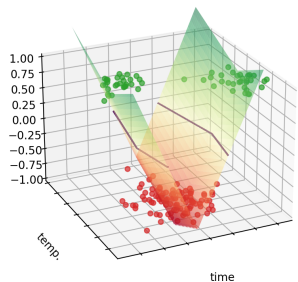
- There is also the prediction function $H(\vec{x})$ that takes in vectors in input space.

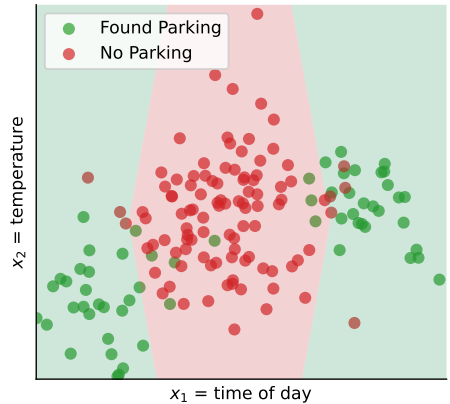
$$\begin{aligned} H(\vec{x}) &= H_{\phi}(\vec{\phi}(\vec{x})) \\ &= \vec{w} \cdot \text{Aug}(\vec{\phi}(\vec{x})) \\ &= w_0 + w_1\phi_1(\vec{x}) + w_2\phi_2(\vec{x}) + \ldots + w_k\phi_k(\vec{x}) \end{aligned}$$

- When plotted, this function will look **non-linear**.

H in Input Space

$$H(\vec{x}) = w_0 + w_1|x_1 - 12| + w_2|x_2 - 70|$$





Exercise

Let $\vec{\phi}(x_1, x_2) = (|x_1 - 12|, |x_2 - 70|)^T$. Suppose we train a least squares classifier in feature space and find $\vec{w}^* = (3, -1, 2)^T$.

Given a new point $\vec{x} = (10, 65)^T$ in input space, what is the prediction, $H(\vec{x})$? This time, compute the answer *without* explicitly computing $\vec{\phi}(\vec{x})$.

Problem

- Feature maps can help us learn non-linear patterns.
- But so far, it's very **manual**.
- Next time: how do we find a good feature map **automatically**?