CS 453X: Class 15

Jacob Whitehill

Exercise

Exercise

- For a soft-margin SVM with cost $C > 0$, where the SVM is trained on *linearly separable* data with a linear kernel:
- Minimize: $\frac{1}{2}\mathbf{w}^\top \mathbf{w} + C \sum_{i=1}^{n} \xi^{(i)}$
- Subject to: $y^{(i)} \left({\mathbf{x}^{(i)}}^\top \mathbf{w} + b \right) \geq 1 - \xi^{(i)}$
- Can it ever occur that the optimal hyperplane will *not* perfectly separate the data?

Gaussian RBF SVM

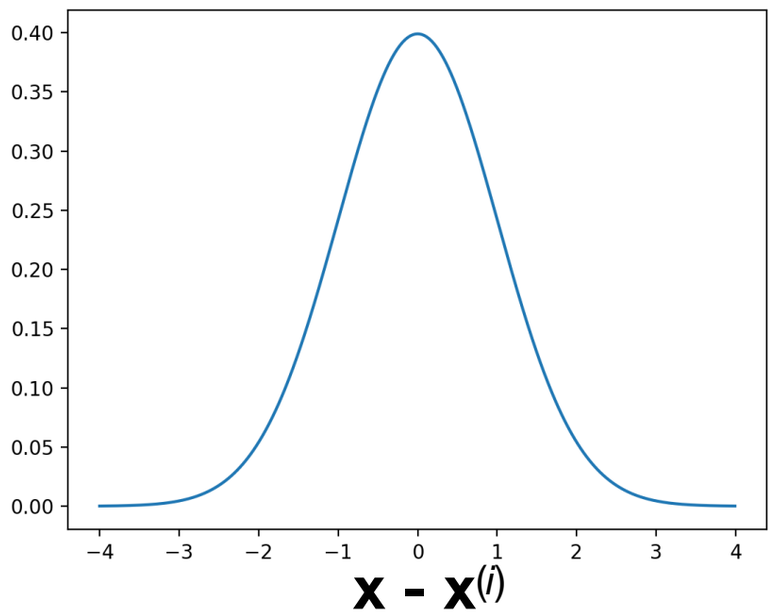
RBF SVM at test time

- An RBF-SVM's output on some example **x** will be:

$$\begin{aligned} g(\mathbf{x}) &= \phi(\mathbf{x})^\top \mathbf{w} + b \\ &= \sum_{i=1}^{n} \alpha^{(i)} y^{(i)} \phi(\mathbf{x})^\top \phi(\mathbf{x}^{(i)}) + b \\ &= \sum_{i=1}^{n} \alpha^{(i)} y^{(i)} k(\mathbf{x}, \mathbf{x}^{(i)}) + b \end{aligned}$$

- where:

$$k(\mathbf{x}^{(i)}, \mathbf{x}^{(j)}) = \exp\left(-\gamma \left(\mathbf{x}^{(i)} - \mathbf{x}^{(j)}\right)^2\right)$$



RBF SVM at test time

- An RBF-SVM's output on some example **x** will be:

$$
\begin{aligned}
g(\mathbf{x}) &= \phi(\mathbf{x})^\top \mathbf{w} + b \\
&= \sum_{i=1}^{n} \alpha^{(i)} y^{(i)} \phi(\mathbf{x})^\top \phi(\mathbf{x}^{(i)}) + b \\
&= \sum_{i=1}^{n} \alpha^{(i)} y^{(i)} k(\mathbf{x}, \mathbf{x}^{(i)}) + b
\end{aligned}
$$

- Procedure:
  - Compute the kernel response of the example **x** with each of our support vectors $\mathbf{x}^{(i)}$.

RBF SVM at test time

- An RBF-SVM's output on some example **x** will be:

$$
\begin{aligned}
g(\mathbf{x}) &= \phi(\mathbf{x})^\top \mathbf{w} + b \\
&= \sum_{i=1}^{n} \alpha^{(i)} y^{(i)} \phi(\mathbf{x})^\top \phi(\mathbf{x}^{(i)}) + b \\
&= \sum_{i=1}^{n} \alpha^{(i)} y^{(i)} k(\mathbf{x}, \mathbf{x}^{(i)}) + b
\end{aligned}
$$

- Procedure:
  - Compute the kernel response of the example **x** with each of our support vectors $\mathbf{x}^{(i)}$.
  - Multiply the kernel response by *i*'s label $y^{(i)}$ and the dual variable $\alpha^{(i)}$.

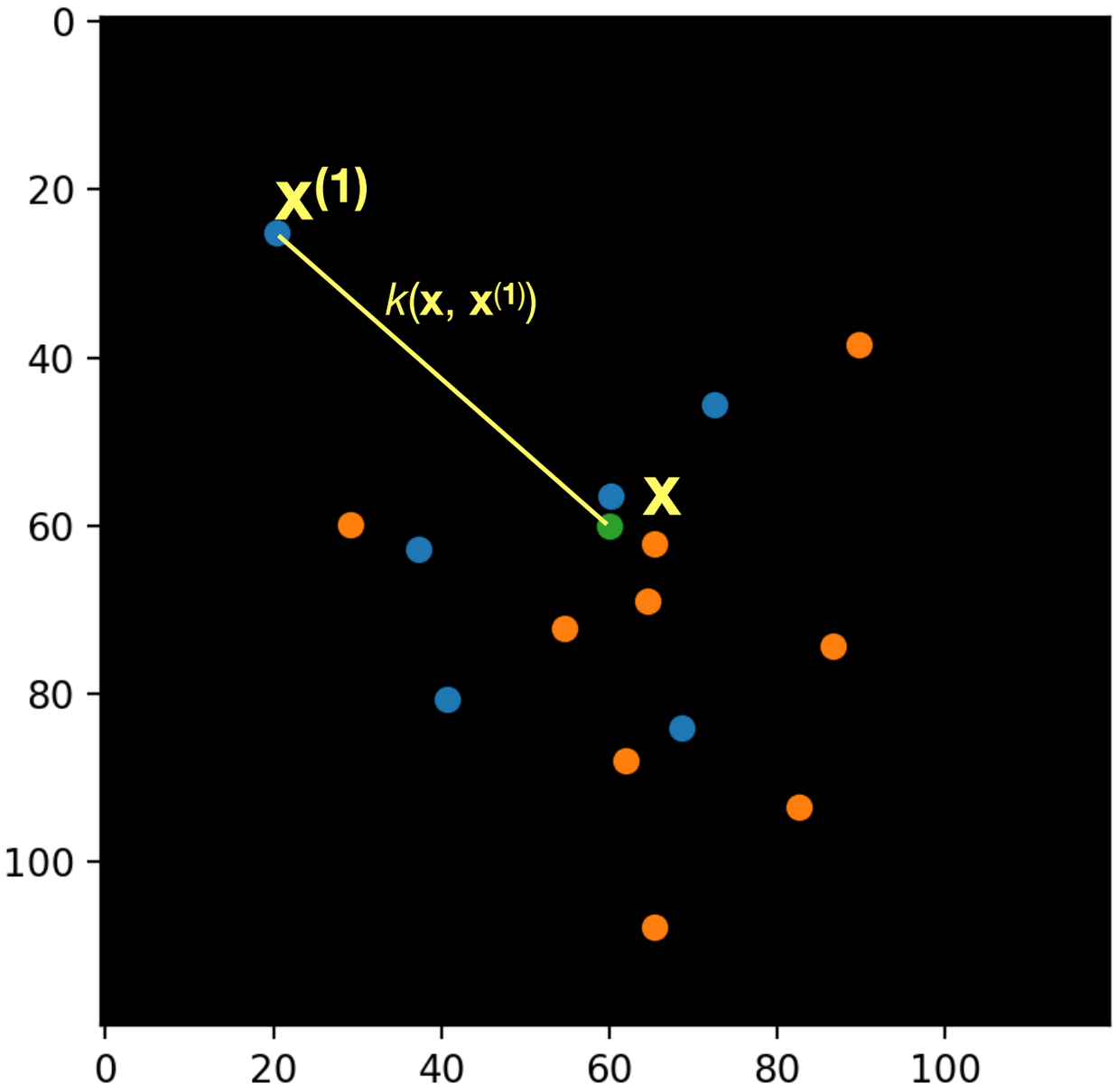
RBF SVM at test time

- An RBF-SVM's output on some example **x** will be:

$$
\begin{aligned}
g(\mathbf{x}) &= \phi(\mathbf{x})^\top \mathbf{w} + b \\
&= \sum_{i=1}^{n} \alpha^{(i)} y^{(i)} \phi(\mathbf{x})^\top \phi(\mathbf{x}^{(i)}) + b \\
&= \sum_{i=1}^{n} \alpha^{(i)} y^{(i)} k(\mathbf{x}, \mathbf{x}^{(i)}) + b
\end{aligned}
$$

- Procedure:
  - Compute the kernel response of the example **x** with each of our support vectors $\mathbf{x}^{(i)}$.
  - Multiply the kernel response by *i*'s label $y^{(i)}$ and the dual variable $\alpha^{(i)}$.
  - Sum across all support vectors.

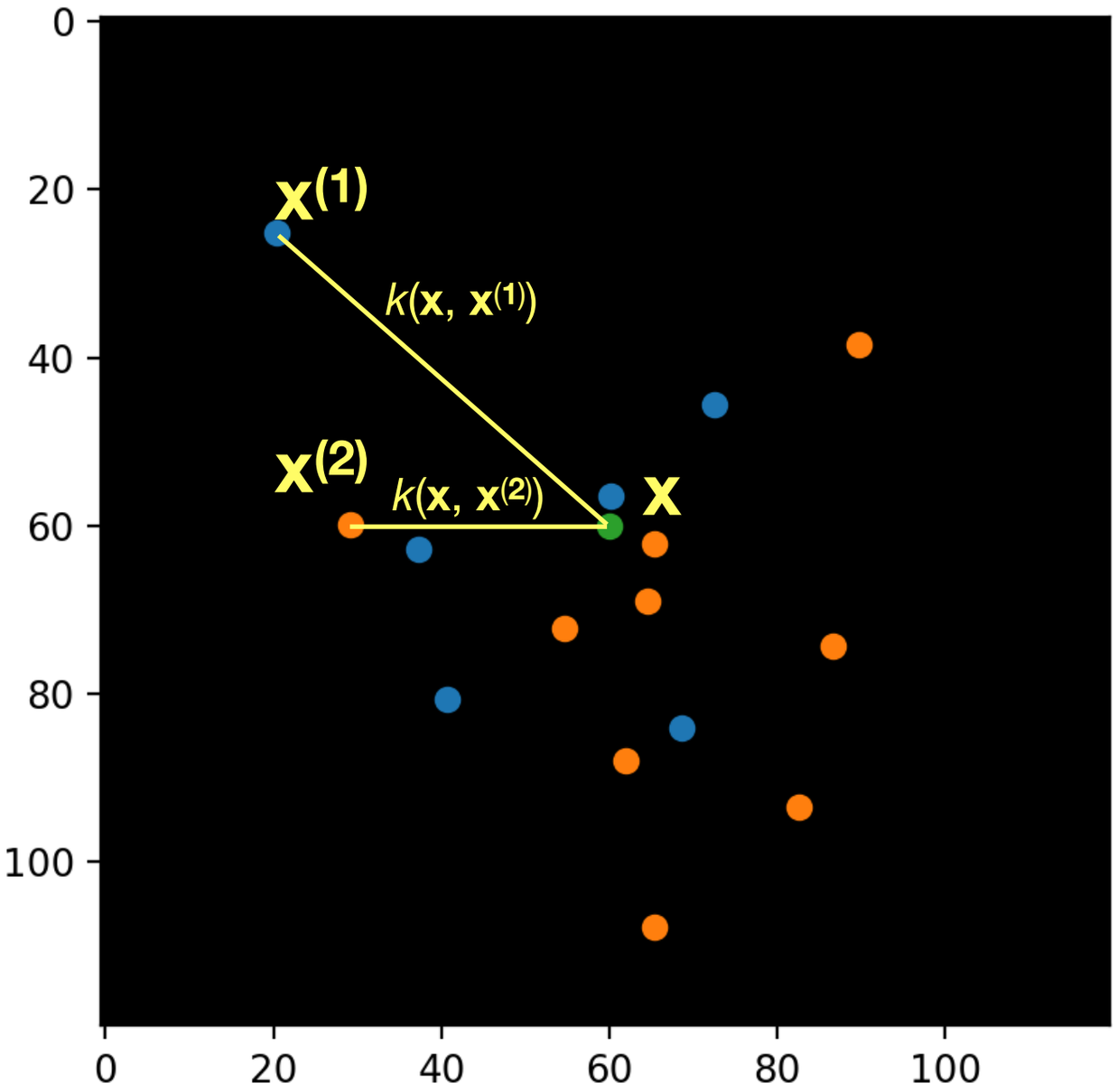
Example



$$\alpha^{(1)} y^{(1)} k(\mathbf{x}, \mathbf{x}^{(1)})$$

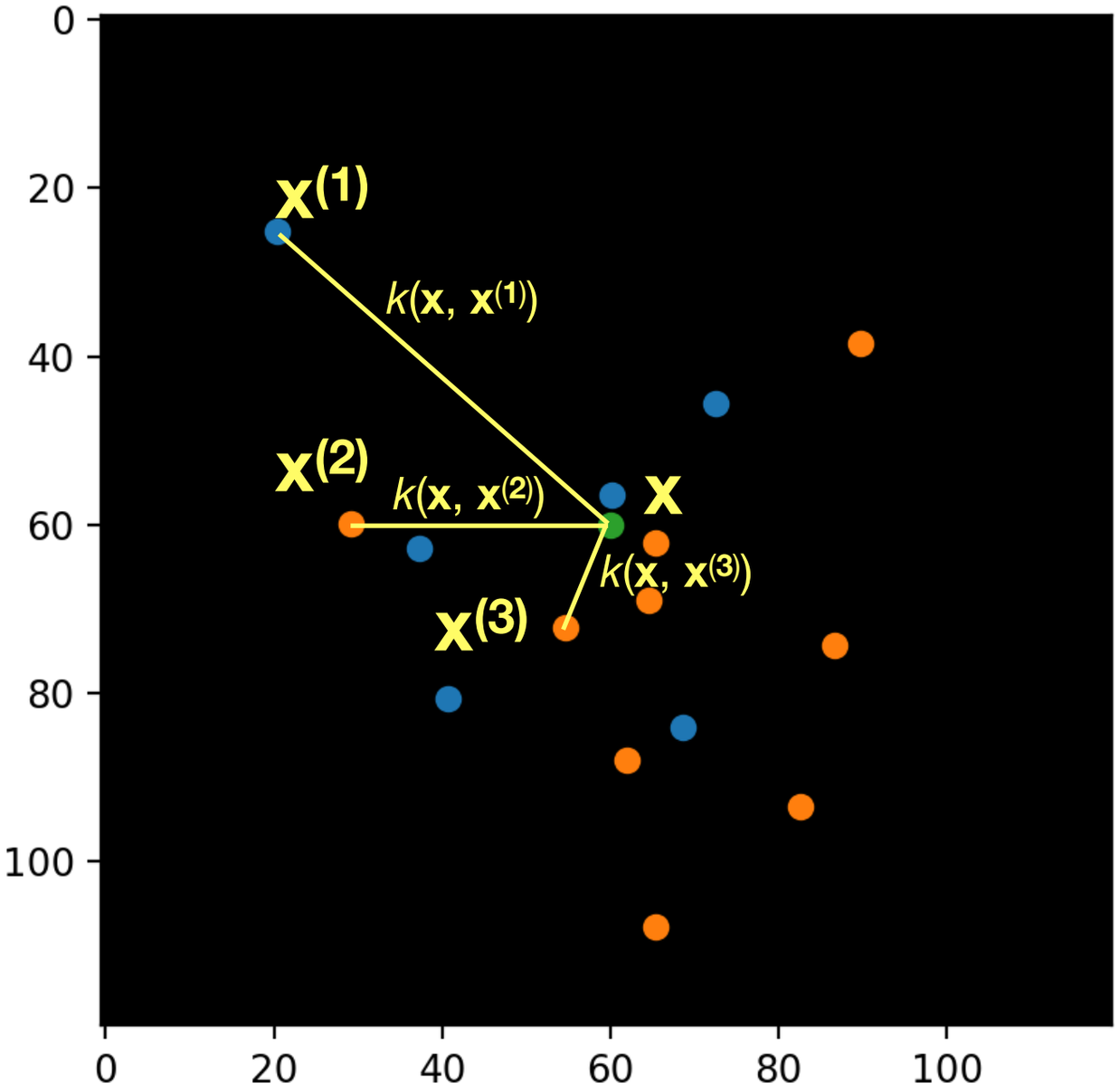
The value of *k* can be thought of as an inverse distance.

Example



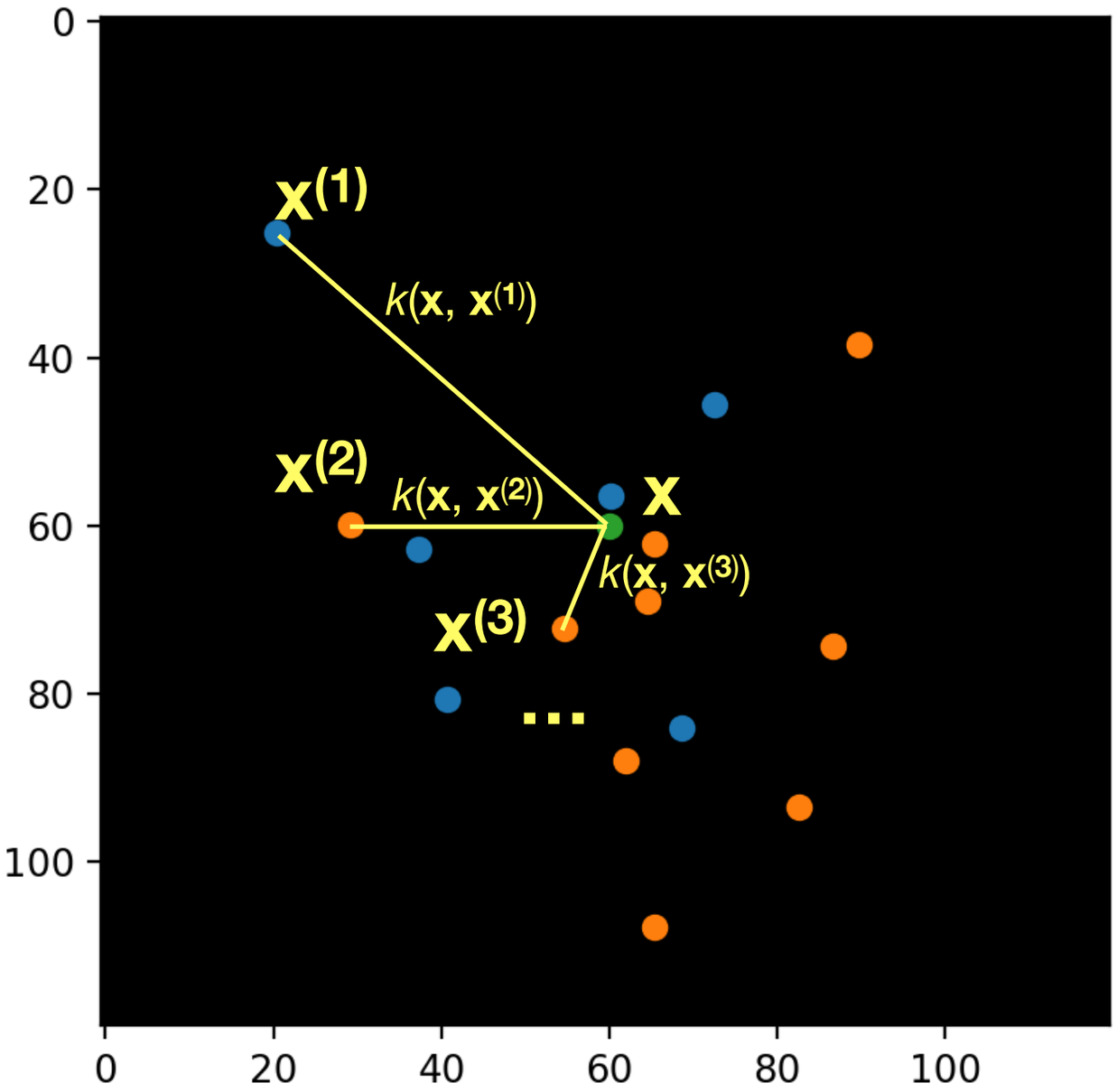
$$\alpha^{(2)} y^{(2)} k(\mathbf{x}, \mathbf{x}^{(2)})$$

Example



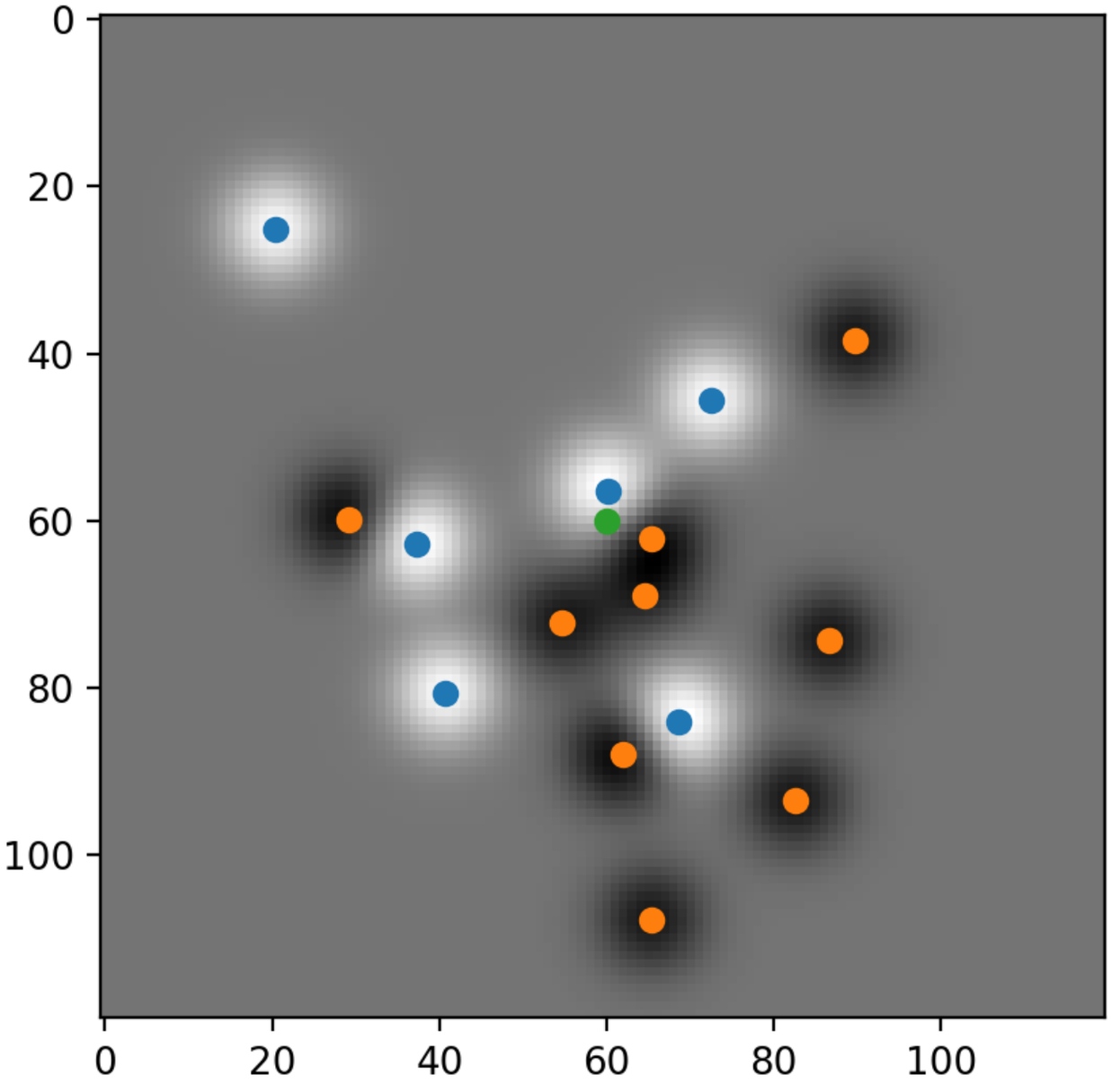
$$\alpha^{(3)} y^{(3)} k(\mathbf{x}, \mathbf{x}^{(3)})$$

Example



$$\sum_{i=1}^{n} \alpha^{(i)} y^{(i)} k(\mathbf{x}, \mathbf{x}^{(i)}) + b$$

Example

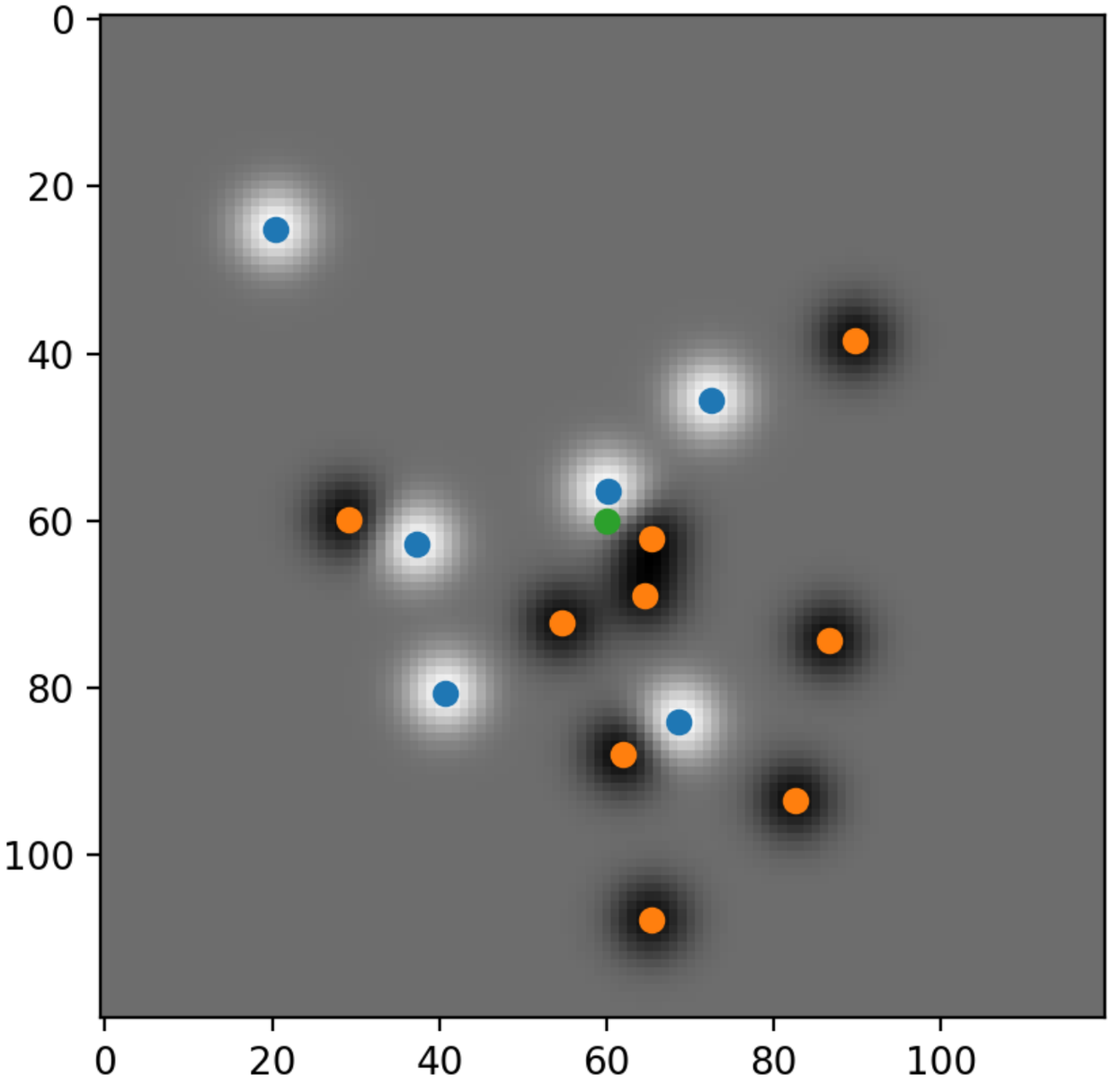


This graph shows, for each possible x, the value of:

$$\sum_{i=1}^{n} \alpha^{(i)} y^{(i)} k(\mathbf{x}, \mathbf{x}^{(i)}) + b$$

for $\gamma = 10^1$

Example

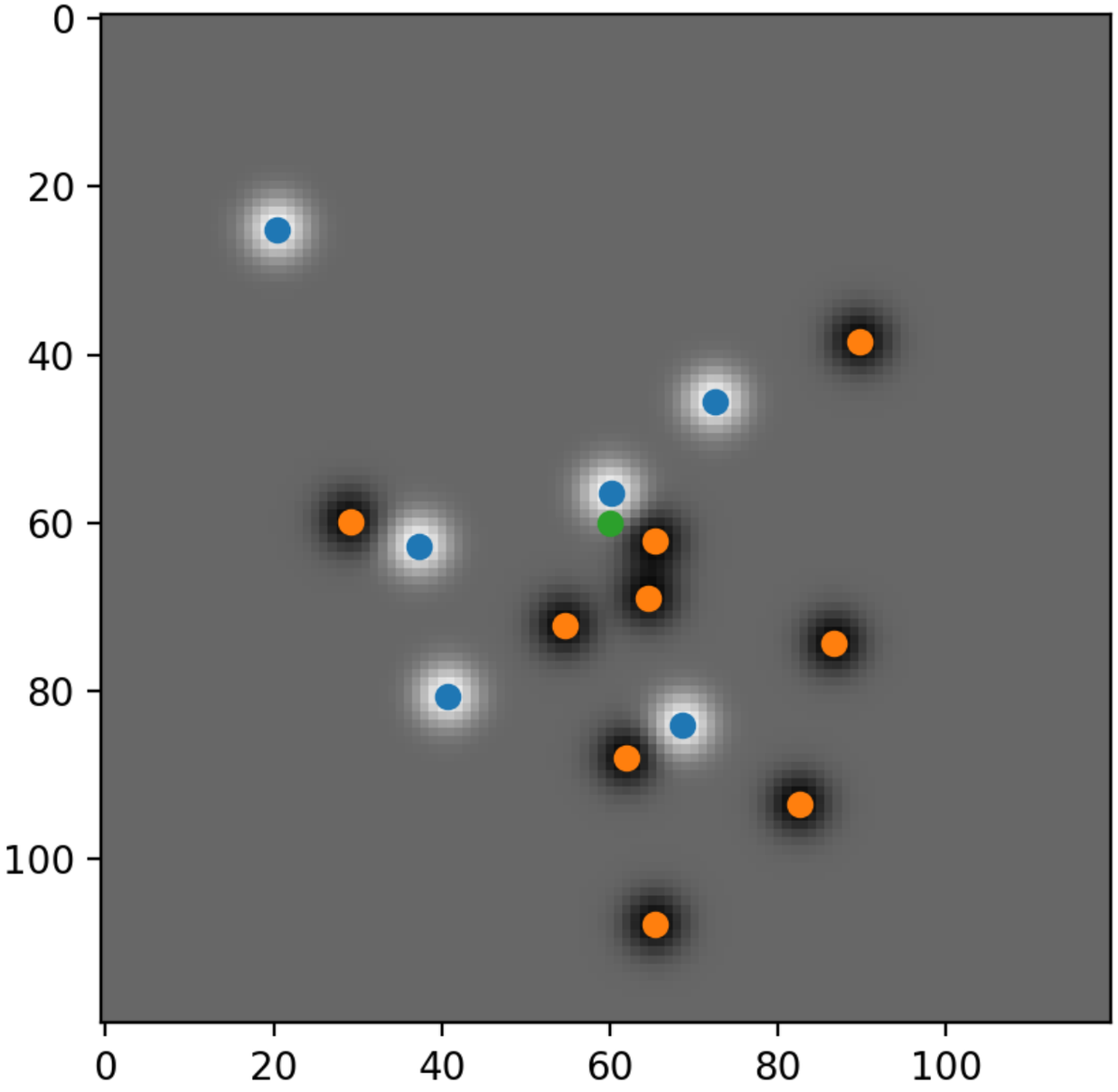


This graph shows, for each possible x, the value of:

$$\sum_{i=1}^{n} \alpha^{(i)} y^{(i)} k(\mathbf{x}, \mathbf{x}^{(i)}) + b$$

for $\gamma = 10^{1.25}$

Example



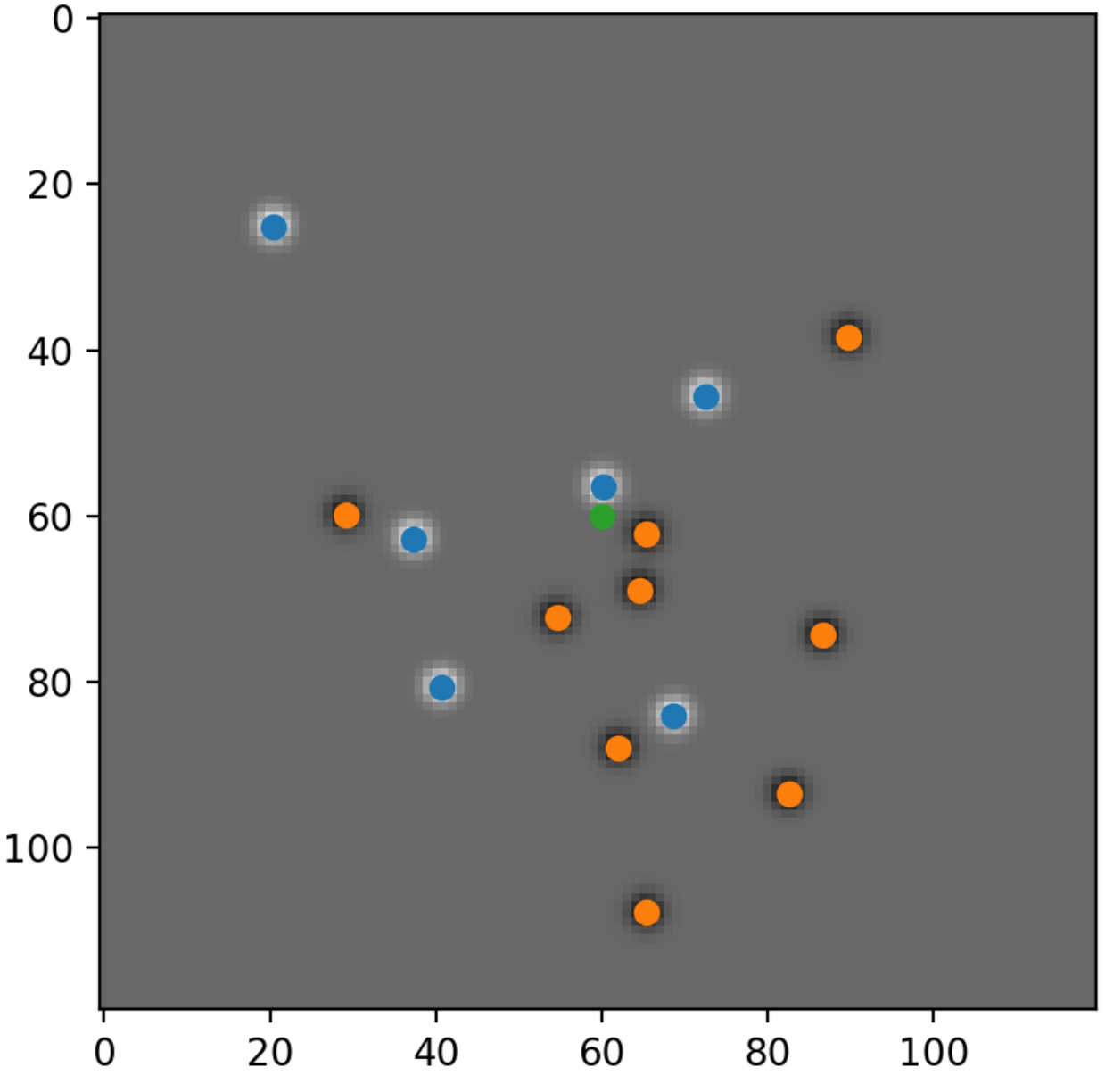
This graph shows, for each possible x, the value of:

$$\sum_{i=1}^{n} \alpha^{(i)} y^{(i)} k(\mathbf{x}, \mathbf{x}^{(i)}) + b$$

for $\gamma = 10^{1.5}$

As γ increases, the effect of more distance support vectors on $g(\mathbf{x})$ decreases.

Example



This graph shows, for each possible x, the value of:

$$\sum_{i=1}^{n} \alpha^{(i)} y^{(i)} k(\mathbf{x}, \mathbf{x}^{(i)}) + b$$

for $\gamma = 10^2$

For very large γ, only the nearest neighbor to x will impact $g(\mathbf{x})$.

Nearest neighbors

Nearest neighbor

- **Nearest neighbor** is both a classification and a regression method.

- It is one of the simplest ML models.

- Algorithm:

  - To predict the label of a new data point **x**, find the data point $\mathbf{x}^{(i^*)}$ in the training set closest to **x**:

$$\arg\min_i |\mathbf{x}^{(i)} - \mathbf{x}|$$

  - Return example i^*'s associated label $y^{(i^*)}$.

k nearest neighbors

- Instead of examining just the single data point closest to **x**, we can look at the *k* neighbors closest to **x**.

- To predict the label of **x**, we can either vote (for classification) or compute the average (for regression) of the *k* neighbors' labels.

k nearest neighbors

- In **`sklearn`**, use either:
  - **`sklearn.neighbors.KNeighborsClassifier(n_neighbors)`**
  - **`sklearn.neighbors.KNeighborsRegressor(n_neighbors)`**

k nearest neighbors

- While very simple, *k* nearest neighbors (kNN) has three significant drawbacks:

    1. The machine must always store the entire training set to make decisions (high storage costs).

$$\arg\min_i |\mathbf{x}^{(i)} - \mathbf{x}|$$

k nearest neighbors

- While very simple, *k* nearest neighbors (kNN) has three significant drawbacks:

    1. The machine must always store the entire training set to make decisions (high storage costs).

    2. The machine can be slow since the distance to *every* training example must be computed.

$$\arg\min_i |\mathbf{x}^{(i)} - \mathbf{x}|$$

k nearest neighbors

- While very simple, *k* nearest neighbors (kNN) has three significant drawbacks:

    1. The machine must always store the entire training set to make decisions (high storage costs).

    2. The machine can be slow since the distance to *every* training example must be computed.

        - There is substantial research on *approximate* nearest neighbors — with *high probability*, find a neighbor very close to **x**.

k nearest neighbors

- While very simple, *k* nearest neighbors (kNN) has three significant drawbacks:

    1. The machine must always store the entire training set to make decisions (high storage costs).

    2. The machine can be slow since the distance to *every* training example must be computed.

        - Note that RBF-SVMs are generally faster since *only the support vectors* need to be stored & compared.

k nearest neighbors

- While very simple, *k* nearest neighbors (kNN) has three significant drawbacks:

    1. The machine must always store the entire training set to make decisions (high storage costs).

    2. The machine can be slow since the distance to *every* training example must be computed.

    3. For high-dimensional inputs, many training examples are needed to “fill” the space.

Curse of dimensionality

- Suppose we want to have at least 10 training examples along each dimension of our input space.
  - 1 dimension ==> need 10 examples
  - 2 dimensions ==> need 10^2 examples

Curse of dimensionality

- Suppose we want to have at least 10 training examples along each dimension of our input space.
  - 1 dimension ==> need 10 examples
  - 2 dimensions ==> need 10^2 examples
  - …
  - *d* dimensions ==> need 10^d examples

Curse of dimensionality

- Suppose we want to have at least 10 training examples along each dimension of our input space.
  - 1 dimension ==> need 10 examples
  - 2 dimensions ==> need 10^2 examples
  - …
  - *d* dimensions ==> need 10^d examples
- Without good “coverage” of the input space, the kNN machine’s predictions may be very inaccurate.

Categorical variables

Categorical variables

- In many data-mining and machine learning contexts, the variables are categorical rather than numerical.
    - Example: https://www.kaggle.com/c/house-prices-advanced-regression-techniques/data

- A common strategy is to convert them to **dummy variables** using a 1-hot encoding.