CSCI567 Machine Learning (Spring 2021)

Sirisha Rambhatla

University of Southern California

Jan 20, 2021

Outline

Outline

Homeworks

- HW 0 is due on Friday. It will not be graded, only to get everyone familiar with the submission mechanism.
- HW 1 will be released on Friday (01/22/2021). Starting HW 1 assignments will be graded.

Outline

Last Class: Foundations of ML

- We discussed different flavors of learning problems
- Tools from probability, information theory, and optimization
- **Today:** We will start our journey of Supervised learning starting with classification.

Outline

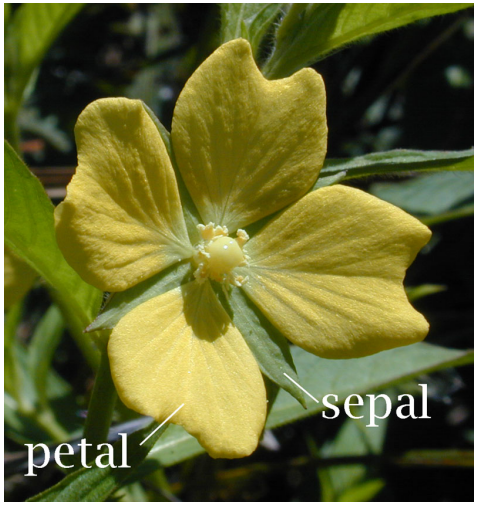
Recognizing flowers

Types of Iris: setosa, versicolor, and virginica



Measuring the properties of the flowers

Features and attributes: the widths and lengths of sepal and petal



Often, data is conveniently organized as a table

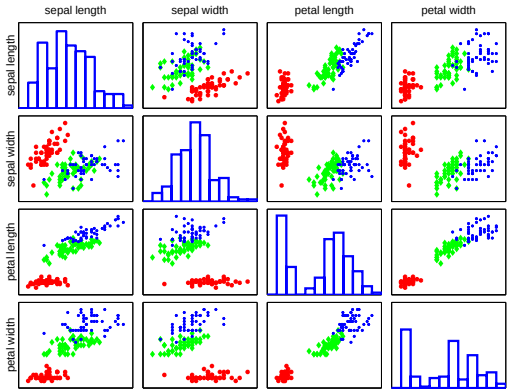
Fisher's *Iris* Data

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

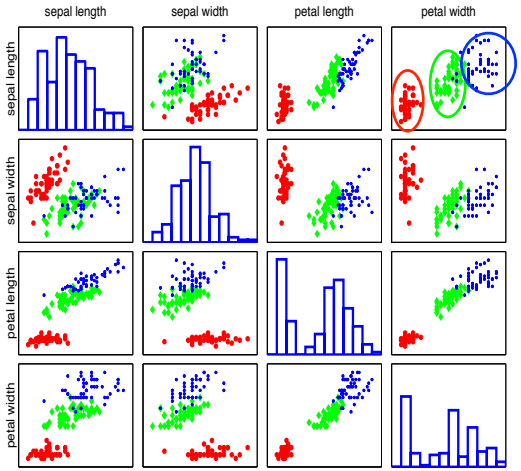
Visualization of data helps identify the right learning model to use

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



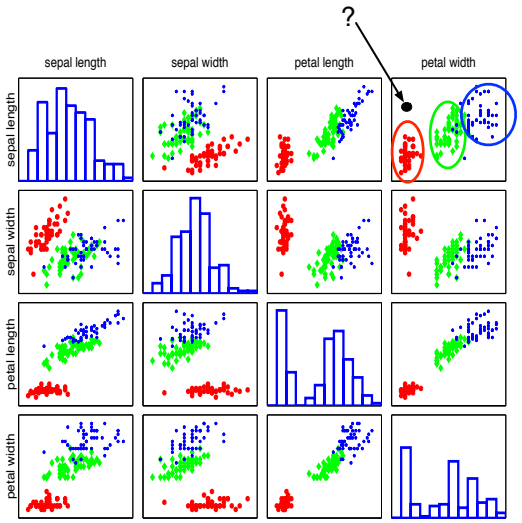
Different types seem well-clustered and separable

Using two features: petal width and sepal length



Labeling an unknown flower type

Closer to red cluster: so labeling it as setosa



General setup for multi-class classification

Training data (set)

- N samples/instances: $\mathcal{D}^{\text{TRAIN}} = \{(\boldsymbol{x}_1, y_1), (\boldsymbol{x}_2, y_2), \cdots, (\boldsymbol{x}_{\mathsf{N}}, y_{\mathsf{N}})\}$
- Each $\boldsymbol{x}_n \in \mathbb{R}^{\mathsf{D}}$ is called a feature vector.
- Each $y_n \in [\mathsf{C}] = \{1, 2, \cdots, \mathsf{C}\}$ is called a label/class/category.
- They are used to learn a *classifier* $f : \mathbb{R}^{\mathsf{D}} \to [\mathsf{C}]$ for future prediction.

Special case: binary classification

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

Nearest neighbor classification (NNC)

The index of the **nearest neighbor** of a point $\boldsymbol{x}$ is

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

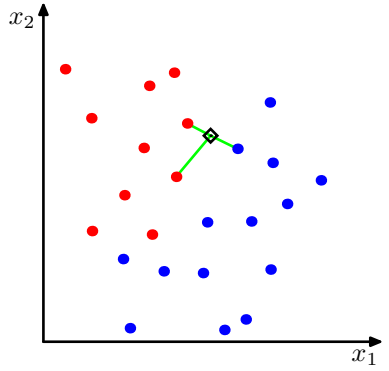
where $\|\cdot\|_2$ is the ℓ_2/Euclidean distance.

Classification rule

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

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)

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	setoas
2	1.4	7.0	versicolor
3	2.5	6.7	virginica
⋮	⋮	⋮	

Flower with unknown category

petal width $= 1.8$ and sepal length $= 6.4$ (i.e. $\boldsymbol{x} = (1.8, 6.4)$)

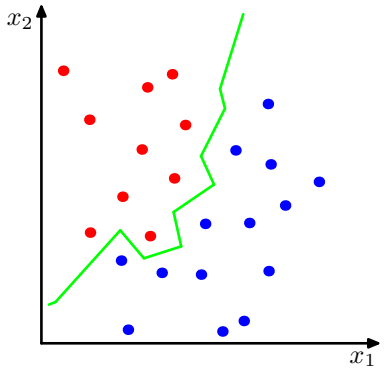
Calculating distance $\|\boldsymbol{x} - \boldsymbol{x}_n\|_2 = \sqrt{(x_1 - x_{n1})^2 + (x_2 - x_{n2})^2}$

ID	distance
1	1.75
2	0.72
3	0.76

Thus, the category is *versicolor*.

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)

Is NNC doing the right thing for us?

Intuition

We should compute accuracy (A) — the percentage of data points being correctly classified, or the error rate (ϵ)— the percentage of data points being incorrectly classified. (accuracy + error rate = 1)

Defined on the training data set

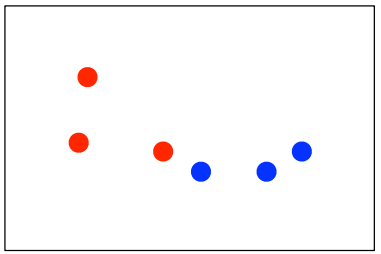
$$A^{\text{TRAIN}} = \frac{1}{\mathsf{N}} \sum_n \mathbb{I}[f(\boldsymbol{x}_n) == y_n], \quad \epsilon^{\text{TRAIN}} = \frac{1}{\mathsf{N}} \sum_n \mathbb{I}[f(\boldsymbol{x}_n) \neq y_n]$$

where $\mathbb{I}[\cdot]$ is the indicator function.

Is this the right measure?

Example

Training data



What are A^{TRAIN} and ϵ^{TRAIN}?

$$A^{\text{TRAIN}} = 100\%, \quad \epsilon^{\text{TRAIN}} = 0\%$$

For every training data point, its nearest neighbor is itself.

Test Error

Does it mean nearest neighbor is a very good algorithm?

Not really, having zero training error is simple!

We should care about accuracy when predicting unseen data

Test/Evaluation data

- $\mathcal{D}^{\text{TEST}} = \{(\boldsymbol{x}_1, y_1), (\boldsymbol{x}_2, y_2), \cdots, (\boldsymbol{x}_{\mathsf{M}}, y_{\mathsf{M}})\}$
- A fresh dataset, *not* overlap with training set.
- Test accuracy and test error

$$A^{\text{TEST}} = \frac{1}{\mathsf{M}} \sum_m \mathbb{I}[f(\boldsymbol{x}_m) == y_m], \quad \epsilon^{\text{TEST}} = \frac{1}{\mathsf{M}} \sum_M \mathbb{I}[f(\boldsymbol{x}_m) \neq y_m]$$

- Good measurement of a classifier's performance

Variant 1: measure nearness with other distances

Previously, we use the Euclidean distance

$$\mathsf{nn}(\boldsymbol{x}) = \operatorname*{argmin}_{n \in [\mathsf{N}]} \|\boldsymbol{x} - \boldsymbol{x}_n\|_2$$

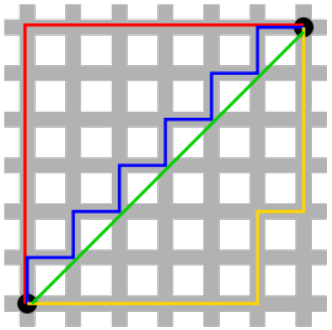
Many other alternative distances

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

$$\|\boldsymbol{x} - \boldsymbol{x}_n\|_1 = \sum_{d=1}^{\mathsf{D}} |x_d - x_{nd}|$$

More generally, L_p distance (for $p \geq 1$):

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



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

Variant 2: K-nearest neighbor (KNN)

Increase the number of nearest neighbors to use?

- 1-nearest neighbor: $\mathsf{nn}_1(\boldsymbol{x}) = \operatorname{argmin}_{n \in [\mathsf{N}]} \|\boldsymbol{x} - \boldsymbol{x}_n\|_2$
- 2-nearest neighbor: $\mathsf{nn}_2(\boldsymbol{x}) = \operatorname{argmin}_{n \in [\mathsf{N}] \setminus \mathsf{nn}_1(\boldsymbol{x})} \|\boldsymbol{x} - \boldsymbol{x}_n\|_2$
- 3-nearest neighbor: $\mathsf{nn}_3(\boldsymbol{x}) = \operatorname{argmin}_{n \in [\mathsf{N}] \setminus \{\mathsf{nn}_1(\boldsymbol{x}), \mathsf{nn}_2(\boldsymbol{x})\}} \|\boldsymbol{x} - \boldsymbol{x}_n\|_2$

The set of K-nearest neighbor

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

Note: we have

$$\|\boldsymbol{x} - \boldsymbol{x}_{\mathsf{nn}_1(\boldsymbol{x})}\|_2 \leq \|\boldsymbol{x} - \boldsymbol{x}_{\mathsf{nn}_2(\boldsymbol{x})}\|_2 \cdots \leq \|\boldsymbol{x} - \boldsymbol{x}_{\mathsf{nn}_K(\boldsymbol{x})}\|_2$$

How to classify with K neighbors?

Classification rule

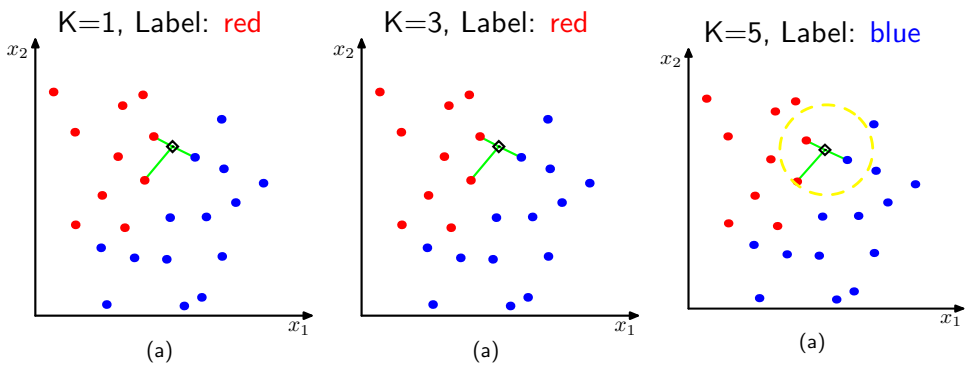
- Every neighbor votes: naturally $\boldsymbol{x}_n$ votes for its label y_n.
- Aggregate everyone's vote on a class label c

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

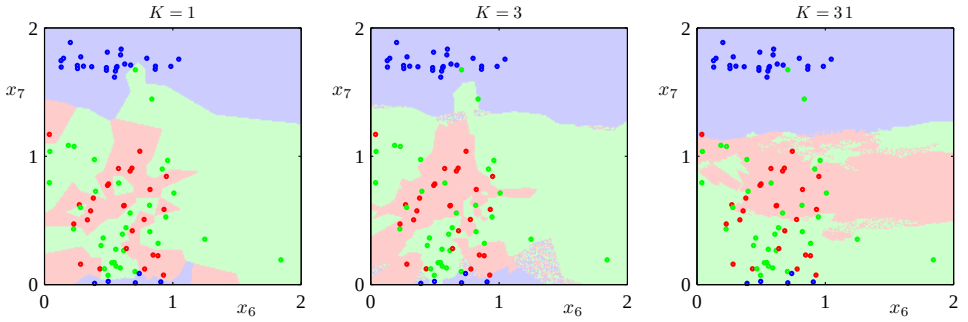
- Predict with the majority

$$f(\boldsymbol{x}) = \operatorname*{argmax}_{c \in [\mathsf{C}]} v_c$$

Example



Decision boundary



When K increases, the decision boundary becomes smoother.

What happens when $K = N$?

Variant 3: Preprocessing data

One issue of NNC: *distances depend on units of the features!*

One solution: preprocess data so it looks more "normalized".

Example:

- 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.

Which variants should we use?

Hyper-parameters in NNC

- The distance measure (e.g. the parameter p for L_p norm)
- K (i.e. how many nearest neighbor?)
- Different ways of preprocessing

Most algorithms have hyper-parameters. Tuning them is a significant part of applying an algorithm.

Tuning via a development dataset

Training data

- 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 to learn $f(\cdot)$

Test 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 to evaluate how well $f(\cdot)$ will do.

Development/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 hyper-parameter(s).

These three sets should *not* overlap!

Recipe

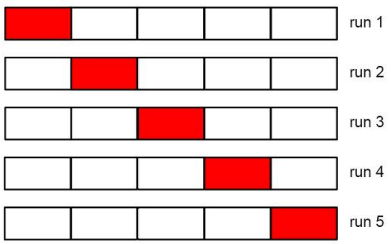
- For each possible value of the hyperparameter (e.g. $K = 1, 3, \cdots$)
  - Train a model using $\mathcal{D}^{\text{TRAIN}}$
  - Evaluate the performance of the model on $\mathcal{D}^{\text{DEV}}$
- Choose the model with the best performance on $\mathcal{D}^{\text{DEV}}$
- Evaluate the model on $\mathcal{D}^{\text{TEST}}$

S-fold Cross-validation

What if we do not have a development set?

- Split the training data into S equal parts.
- Use each part *in turn* as a development dataset and use the others as a training dataset.
- Choose the hyper-parameter leading to best *average* performance.

S = 5: 5-fold cross validation



Special case: S = N, called leave-one-out.

Cross-validation recipe

- Split the training data into S equal parts. Denote each part as $\mathcal{D}_s^{\text{TRAIN}}$.
- For each possible value of the hyper-parameter (e.g. $K = 1, 3, \cdots$)
  - For every $s \in [\text{S}]$
    - Train a model using $\mathcal{D}_{\setminus s}^{\text{TRAIN}} = \mathcal{D}^{\text{TRAIN}} - \mathcal{D}_s^{\text{TRAIN}}$
    - Evaluate the performance of the model on $\mathcal{D}_s^{\text{TRAIN}}$
  - Average the S performance metrics
- Choose the hyper-parameter with the best averaged performance
- **Use the best hyper-parameter to train a model using all $\mathcal{D}^{\text{train}}$**
- Evaluate the model on $\mathcal{D}^{\text{TEST}}$

Summary

Advantages of NNC

- Simple, easy to implement (widely used in practice)

Disadvantages of NNC

- Computationally intensive for large-scale problems: $O(ND)$ for each prediction *naively*. Here, N is the cardinality of the training set and D is the dimension of the training example.
- Need to *"carry"* the training data around. This type of method is called *nonparametric*.
- Choosing the right hyper-parameters can be involved.

Summary

Typical steps of developing a machine learning system:

- Collect data, split into training, development, and test sets.
- Train a model with a machine learning algorithm. Most often we apply cross-validation to tune hyper-parameters.
- Evaluate using the test data and report performance.
- Use the model to predict future/make decisions.

Outline

How good is NNC really?

To answer this question, we proceed in 3 steps

1. Define *more carefully* a performance metric for a classifier.
2. Hypothesize an ideal classifier - *the best possible one*.
3. Compare NNC to the ideal one.

Why does test error make sense?

Test error makes sense only when training set and test set are correlated.

Most standard assumption: every data point $(\boldsymbol{x}, y)$ (from $\mathcal{D}^{\text{TRAIN}}$, $\mathcal{D}^{\text{DEV}}$, or $\mathcal{D}^{\text{TEST}}$) is an *independent and identically distributed (i.i.d.)* sample of an unknown joint distribution $\mathcal{P}$.

- often written as $(\boldsymbol{x}, y) \overset{i.i.d.}{\sim} \mathcal{P}$

Test error of a fixed classifier is therefore a *random variable*.

Need a more "certain" measure of performance (so it's easy to compare different classifiers for example).

Expected error

What about the **expectation** of this random variable?

$$\mathbb{E}[\epsilon^{\text{TEST}}] = \frac{1}{M}\sum_{m=1}^{M} \mathbb{E}_{(\boldsymbol{x_m}, y_m)\sim\mathcal{P}}\mathbb{I}[f(\boldsymbol{x}_m) \neq y_m] = \mathbb{E}_{(\boldsymbol{x},y)\sim\mathcal{P}}\mathbb{I}[f(\boldsymbol{x}) \neq y]$$

- i.e. the expected error/mistake of f

Test error is a proxy of expected error. *The larger the test set, the better the approximation.*

What about the expectation of training error? Is training error a good proxy of expected error?

Expected risk

More generally, for a loss function $L(y', y)$,

- e.g. $L(y', y) = \mathbb{I}[y' \neq y]$, called *0-1 loss*. **Default**
- many more other losses as we will see.

the *expected risk* of f is defined as

$$R(f) = \mathbb{E}_{(\boldsymbol{x},y)\sim\mathcal{P}} L(f(\boldsymbol{x}), y).$$

For *0-1 loss* we have

$$R(f) = \mathbb{E}_{(\boldsymbol{x},y)\sim\mathcal{P}}\mathbb{I}[y' \neq y]$$

Bayes optimal classifier

What should we predict for $\boldsymbol{x}$, *knowing* $\mathcal{P}(y|\boldsymbol{x})$?

Bayes optimal classifier: $f^*(\boldsymbol{x}) = \text{argmax}_{c\in[C]}\, \mathcal{P}(c|\boldsymbol{x})$.

The optimal risk: $R(f^*) = \mathbb{E}_{\boldsymbol{x}\sim\mathcal{P}_{\boldsymbol{x}}}[1 - \max_{c\in[C]} \mathcal{P}(c|\boldsymbol{x})]$ where $\mathcal{P}_{\boldsymbol{x}}$ is the marginal distribution of $\boldsymbol{x}$.

That is we have $R(f^*) \leq R(f)$ for any f. **Verify!**

For special case $C = 2$, let $\eta(\boldsymbol{x}) = \mathcal{P}(0|\boldsymbol{x})$, then

$$\begin{aligned} R(f^*) &= \mathbb{E}_{\boldsymbol{x}\sim\mathcal{P}_{\boldsymbol{x}}}[\mathbb{E}_{\boldsymbol{y}|\boldsymbol{x}}[\mathbb{I}_{f^*(\boldsymbol{x})\neq y}]] \\ &= \mathbb{E}_{\boldsymbol{x}\sim\mathcal{P}_{\boldsymbol{x}}}[\eta(\boldsymbol{x})\mathbb{I}_{f^*(\boldsymbol{x})=1} + (1-\eta(\boldsymbol{x}))\mathbb{I}_{f^*(\boldsymbol{x})=0}] \\ &= \mathbb{E}_{\boldsymbol{x}\sim\mathcal{P}_{\boldsymbol{x}}}[\min\{\eta(\boldsymbol{x}), 1-\eta(\boldsymbol{x})\}], \end{aligned}$$

Comparing NNC to Bayes optimal classifier

Come back to the question: how good is NNC?

Theorem (Cover and Hart, 1967)

Let f_N be the 1-nearest neighbor binary classifier using N training data points, we have (under mild conditions)

$$R(f^*) \leq \lim_{N\to\infty} \mathbb{E}[R(f_N)] \leq 2R(f^*)$$

i.e., expected risk of NNC in the limit is at most twice of the best possible.

A pretty strong guarantee.
In particular, $R(f^*) = 0$ implies $\mathbb{E}[R(f_N)] \to 0$.

Proof sketch

Fact: $\boldsymbol{x}_{\mathsf{nn}_{(\boldsymbol{x})}} \to \boldsymbol{x}$ as $N \to \infty$ with probability 1

$$\begin{aligned}
\mathbb{E}[R(f_N)] &= \mathbb{E}[\mathbb{E}_{(\boldsymbol{x},y)\sim\mathcal{P}}\mathbb{I}[f_N(\boldsymbol{x}) \neq y]] \\
&\to \mathbb{E}_{\boldsymbol{x}\sim\mathcal{P}_{\boldsymbol{x}}}\mathbb{E}_{y,y'\overset{i.i.d.}{\sim}\mathcal{P}(\cdot|\boldsymbol{x})}[\mathbb{I}[y' \neq y]] \\
&= \mathbb{E}_{\boldsymbol{x}\sim\mathcal{P}_{\boldsymbol{x}}}\mathbb{E}_{y,y'\overset{i.i.d.}{\sim}\mathcal{P}(\cdot|\boldsymbol{x})}[\mathbb{I}[y' = 0 \text{ and } y = 1] + \mathbb{I}[y' = 1 \text{ and } y = 0]] \\
&= \mathbb{E}_{\boldsymbol{x}\sim\mathcal{P}_{\boldsymbol{x}}}[\eta(\boldsymbol{x})(1-\eta(\boldsymbol{x})) + (1-\eta(\boldsymbol{x}))\eta(\boldsymbol{x})] \\
&= 2\mathbb{E}_{\boldsymbol{x}\sim\mathcal{P}_{\boldsymbol{x}}}[\eta(\boldsymbol{x})(1-\eta(\boldsymbol{x}))] \\
&\leq 2\mathbb{E}_{\boldsymbol{x}\sim\mathcal{P}_{\boldsymbol{x}}}[\min\{\eta(\boldsymbol{x}), (1-\eta(\boldsymbol{x}))\}] \\
&= 2R(f^*)
\end{aligned}$$