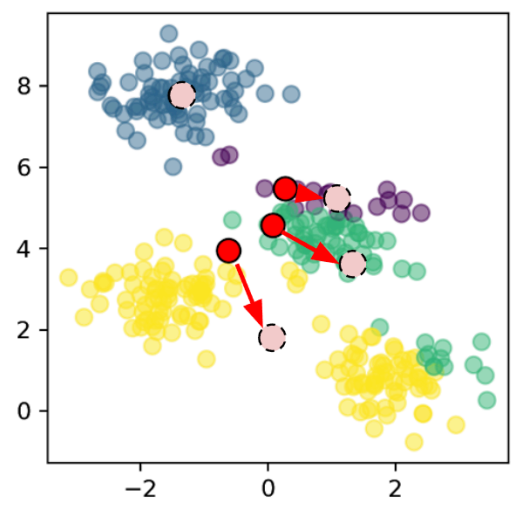
K methods

Week2

Some of the slides are adopted from Prof Enric JDF's ML course

Recap – K-means clustering



Unsupervised Learning

Key Equation:

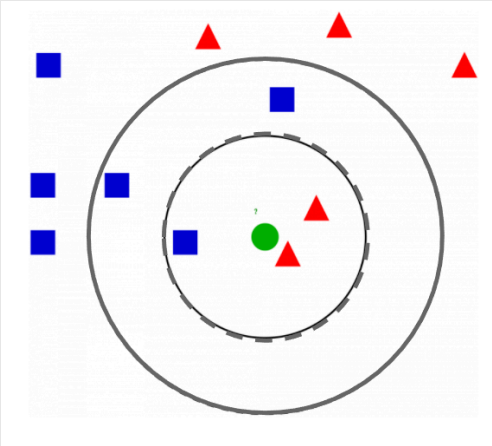
Minimize the intra-class variance : $\sum_{i=1}^{k} \sum_{x \in S_i} \|x - c_i\|^2$

Description:

Iteratively choose a cluster center (mean) and assigns points to the nearest cluster mean

Need to tune the hyper parameter k using elbow method

Recap – K-NN classification



Supervised Learning

Key Equation:

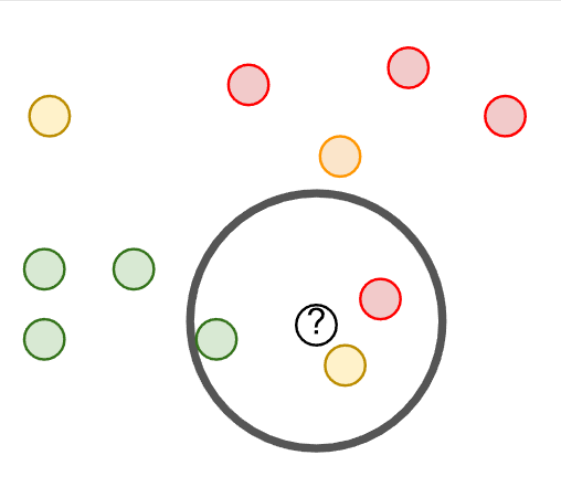
$$f(x) = \arg\max_C \#\{i | y_i = C\}_{i \in NN(x)}$$

Description:

Remembers all the input data and returns the most frequently occurring class in the set of k nearest neighbors.

Need to tune the hyper parameter k

Recap – K-NN Regression



Supervised Learning

Key Equation:

$$f(x) = \frac{1}{k}\sum_{i \in NN(x)} y_i$$

Description:

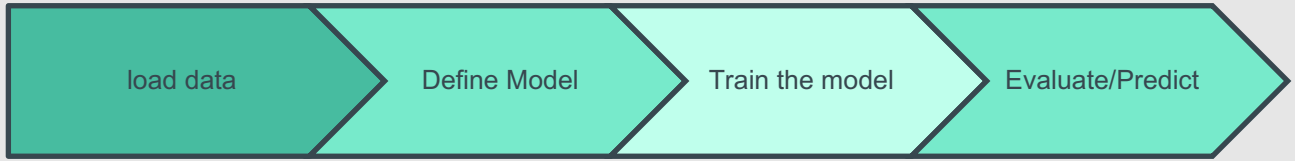
Remembers all the input data and returns the average value of k nearest neighbors.

Need to tune the hyper parameter k

Sci-kit learn library

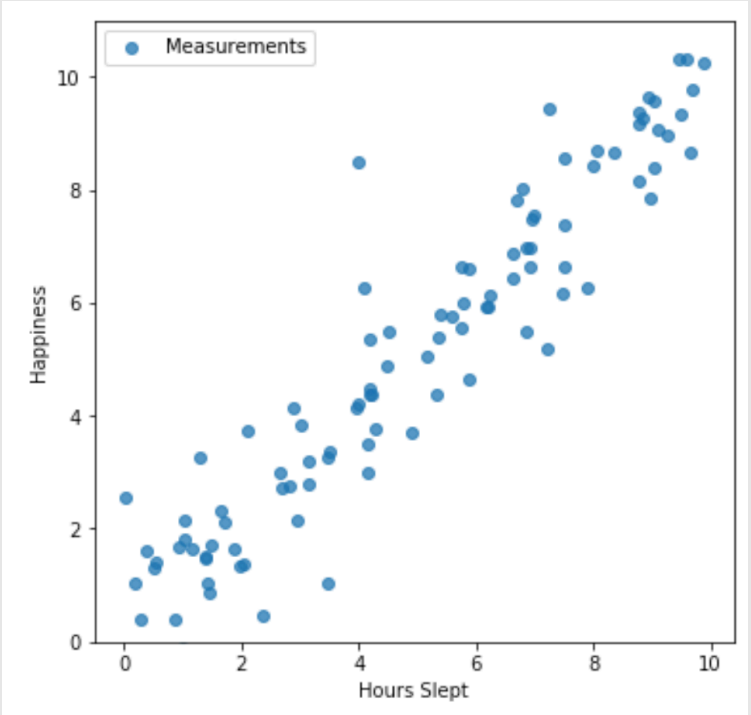
- Open-source python library
- Efficient implementation of most ML algorithms
- Easy to extend and apply to most the dataset in the wild

Workflow in sklearn



Example: 1D regression problem

Hours Slept	Happiness
8.5	4
5.0	4.7
5.46	7.5
0.7	1.0
4.5	3.7



Example: 1D regression problem

```
# import the k-NN regression class
from sklearn.neighbors import KNeighborsRegressor

# initialize the model class
model = KNeighborsRegressor(n_neighbors=k)

# train the model using the dataset
model = model.fit(x,y)

# make a prediction using the model
y_predicted = model.predict(x)
```

Array-like: NumPy array, DataFrame, list.

You try it: applying k-NN

- Train a k-NN model with k=1 on the dataset
- Predict how happy you will be after sleeping 1, 2, ...10 hours

Example

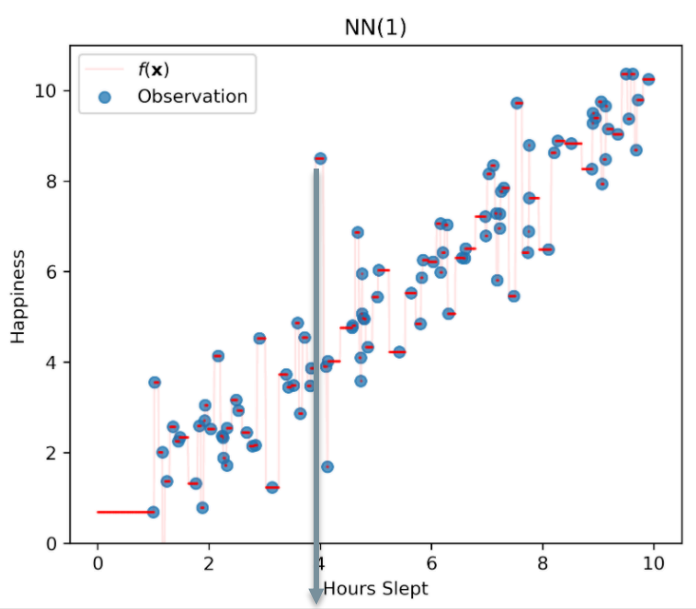
```
# same as before ...
from sklearn.neighbors import KNeighborsRegressor
model = KNeighborsRegressor(n_neighbors=1)
model = model.fit(x,y)

# using a loop
for hours_slept in range(1,11):
    happiness   = model.predict([[hours_slept]])
    print("If you sleep %.0f hours, you will be %.1f happy!" %
                                                    (hours_slept,happiness))
# without for loop
hours_slept = np.arange(1,11,1).reshape(10,1)
happiness   = model.predict(hours_slept)
np.round(happiness,1)
```

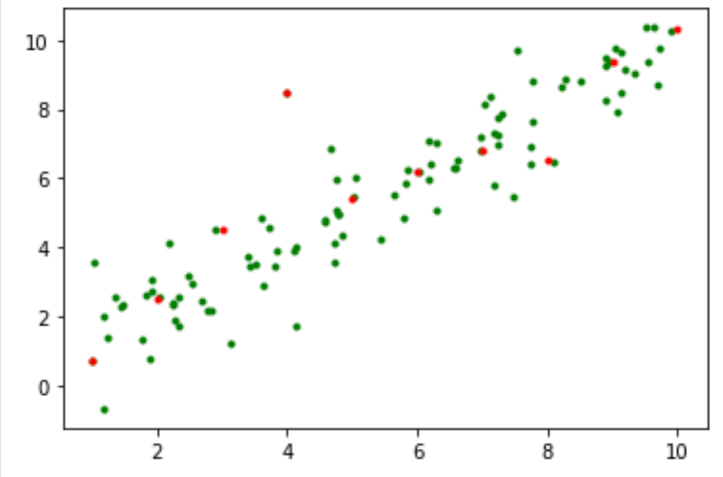
```
If you sleep 1 hours, you will be 0.7 happy!
If you sleep 2 hours, you will be 2.5 happy!
If you sleep 3 hours, you will be 4.5 happy!
If you sleep 4 hours, you will be 8.5 happy!
If you sleep 5 hours, you will be 5.4 happy!
If you sleep 6 hours, you will be 6.2 happy!
If you sleep 7 hours, you will be 6.8 happy!
If you sleep 8 hours, you will be 6.5 happy!
If you sleep 9 hours, you will be 9.4 happy!
If you sleep 10 hours, you will be 10.3 happy!
```

Finding k

Plot the prediction on the whole range

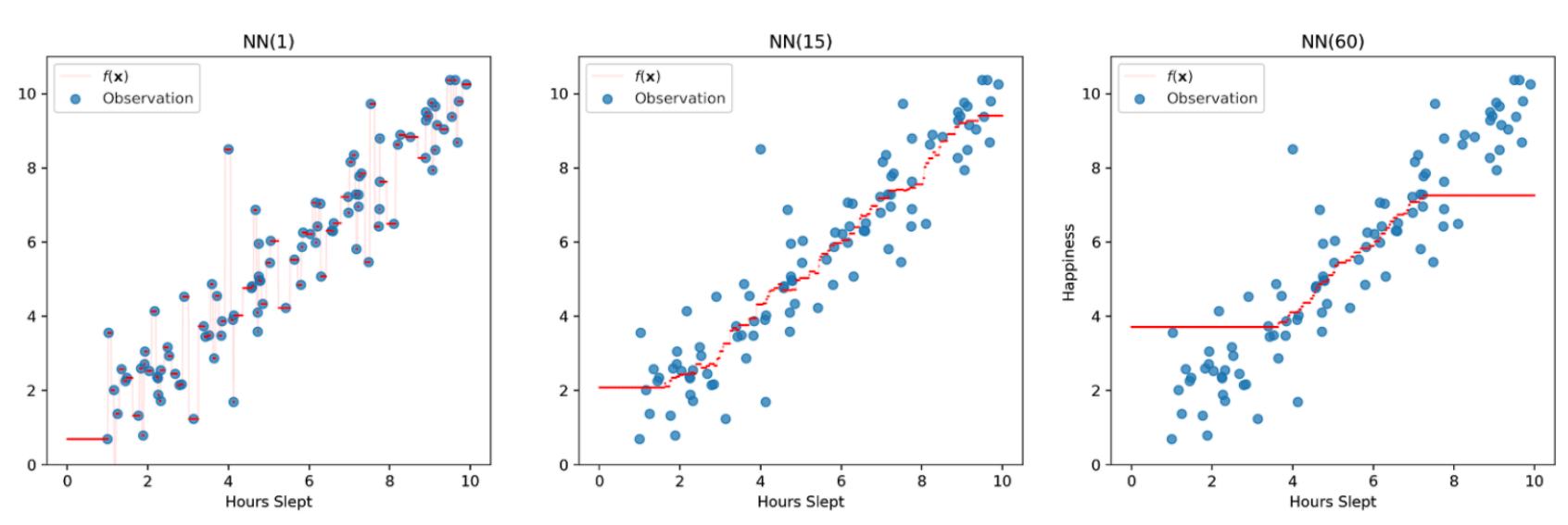
Do you see any problems with this plot? How would you resolve it?





An outlier: a point that is much bigger or smaller that it's neighbors

Result for different k

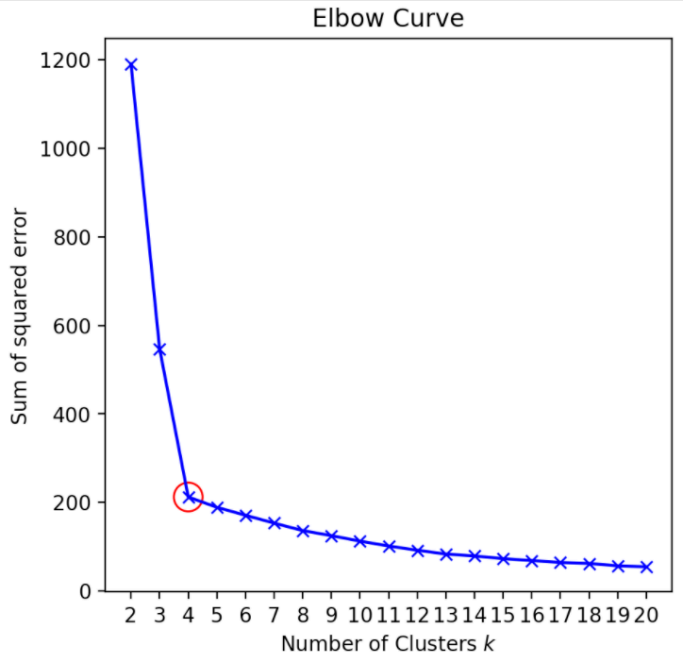


Higher k leads to increased amount of smoothing. Notice the edges for NN(60)

Last class: choosing the right value for k

- Try various k
- Evaluate sum of squared distance
  $\sum_{i=1}^{k} \sum_{x \in S_i} \|x - c_i\|^2$

- Look for and “elbow” point

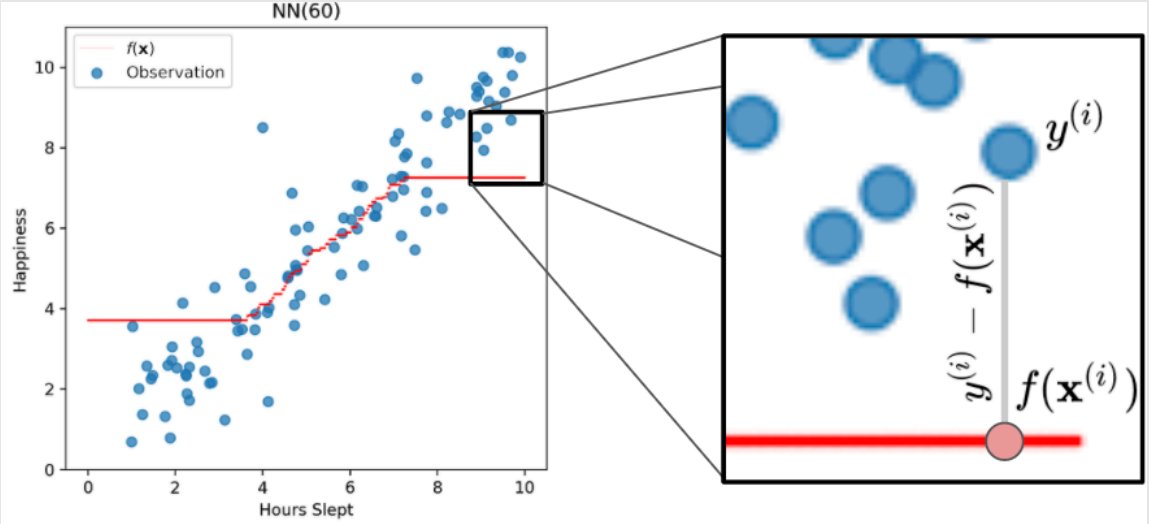


Measuring goodness of fit

Look at the prediction error for each data point and average out:

$$MSE = \frac{1}{N}\sum_{i=1}^{N} \left(y^{(i)} - f(x^{(i)})\right)^2$$

L2 loss



$x^{(i)}$ is the feature vector for observation i, it could be a multi dimensional vector.

You try it: Calculate the MSE for your predications

- Make a function “make_mse(model, x, y)” which calculates the MSE of a predictive model trained with sklearn.

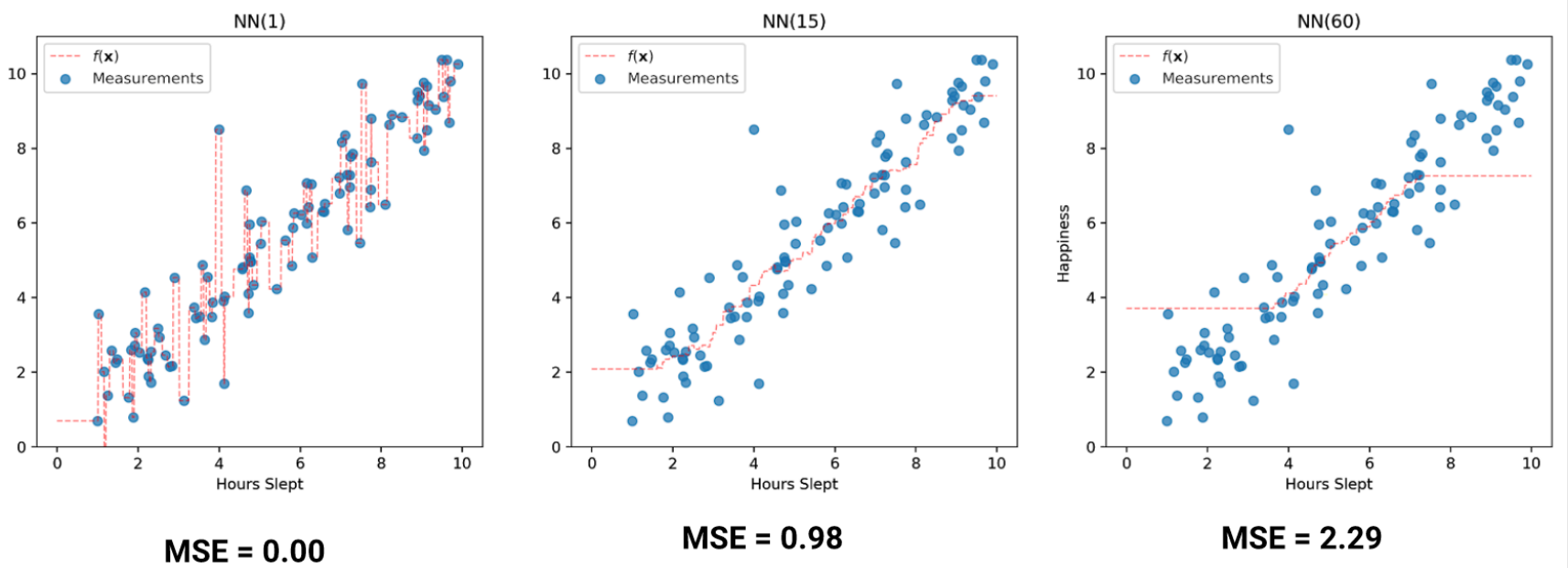
- Use the function to calculate the MSE on all data for the model with
  - K=1
  - K=15
  - K=60

$$MSE = \frac{1}{N}\sum_{i=1}^{N}\left(y^{(i)} - f(x^{(i)})\right)^2$$

Calculate MSE

```
# slow method
def model_mse(model, x, y):
    error_sum = 0
    for xi,yi in zip(x,y):
        error_sum += (yi - model.predict([[xi]]))**2
    return error_sum / len(x)

# faster method
def model_mse(model,x, y):
    predictions = model.predict(x)
    return np.mean(np.power(y-predictions,2))

# lazy method ;)
from sklearn.metrics import mean_squared_error
def model_mse(model,x, y):
    predictions = model.predict(x)
    return mean_squared_error(y, predictions)
```

MSE for different k



Goodness of fit accurately captures deviations of the prediction from the ground truth

How can it help us to find the best model?

Model Performance

All the data in the world on your problem

$$E[l(f(x), y)]_{(x,y)\sim P}$$

We usually do not know the true distribution

The data we have

$$\frac{1}{N}\sum_{i=1}^{N} l\left(y^{(i)}, f\left(x^{(i)}\right)\right)$$

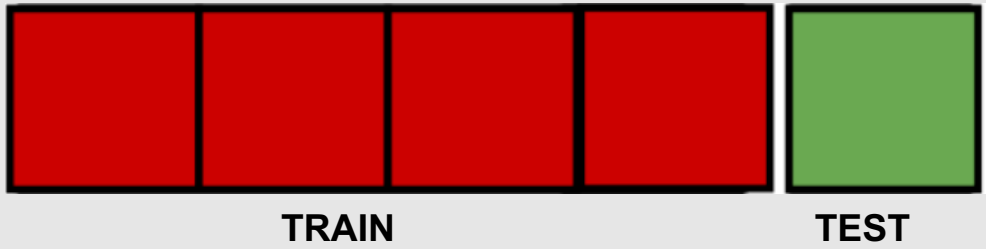
Weak law of large numbers

$$\frac{1}{|\mathcal{D}_{test}|}\sum_{(x^{(i)},y^{(i)})\in\mathcal{D}_{test}} l\left(y^{(i)}, f\left(x^{(i)}\right)\right)$$

Holdout test set

Cross validate on the testing data

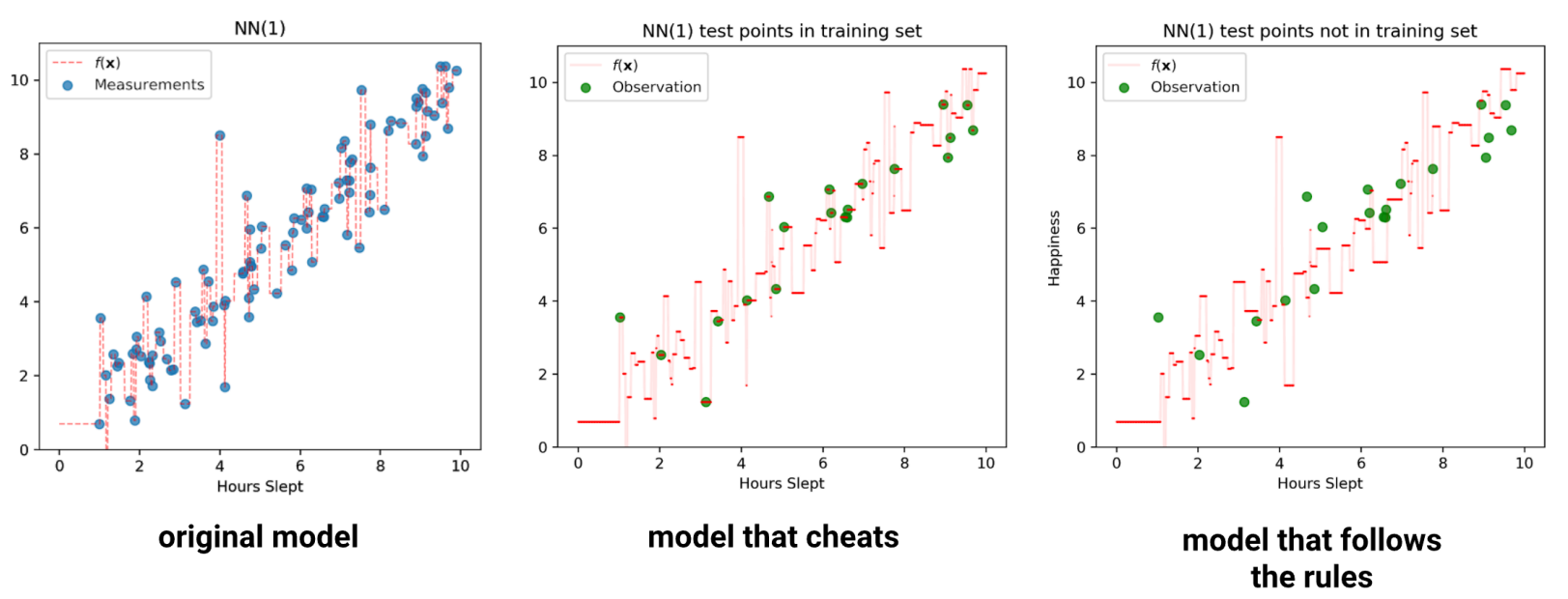
- Train the model on one part of the dataset (train set)
- Test the model the another (test set) and ideally only once



- This is the only way to truly test the predictive power.

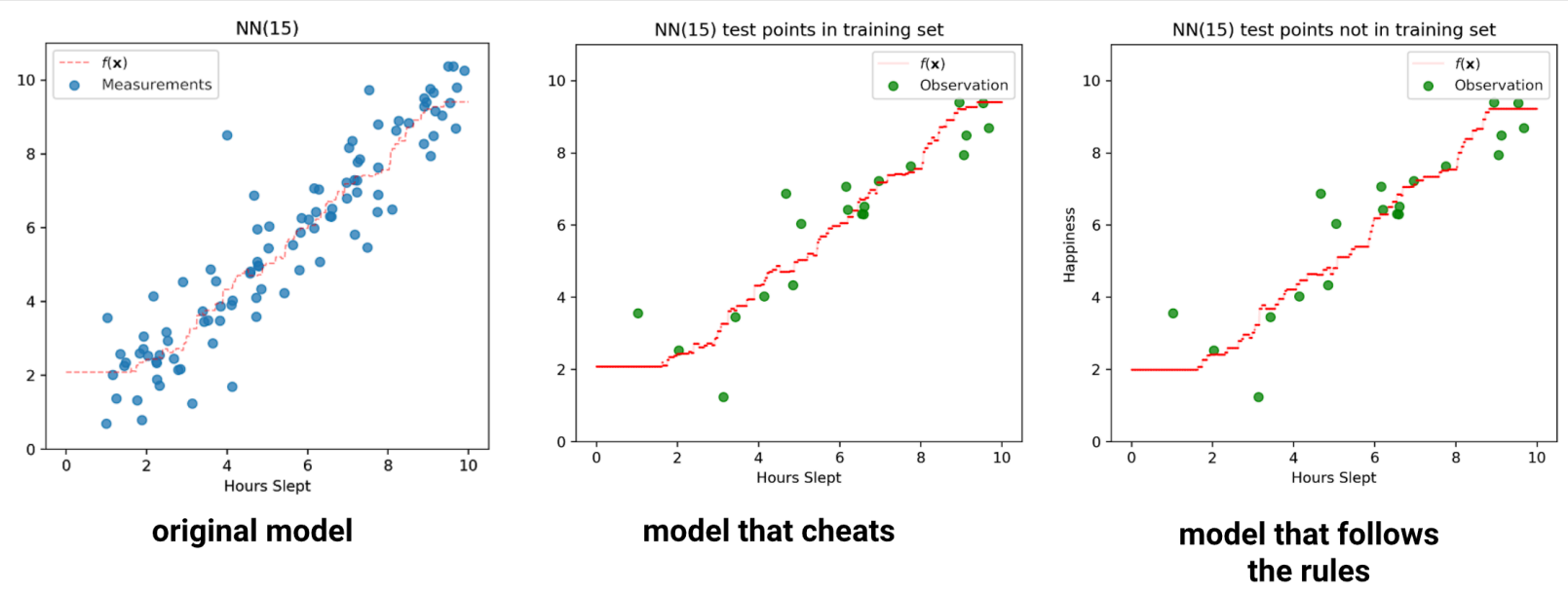
Do not cheat when testing a model:
The testing set should not ‘look at‘ the training data set.

Example: train and test NN(1)



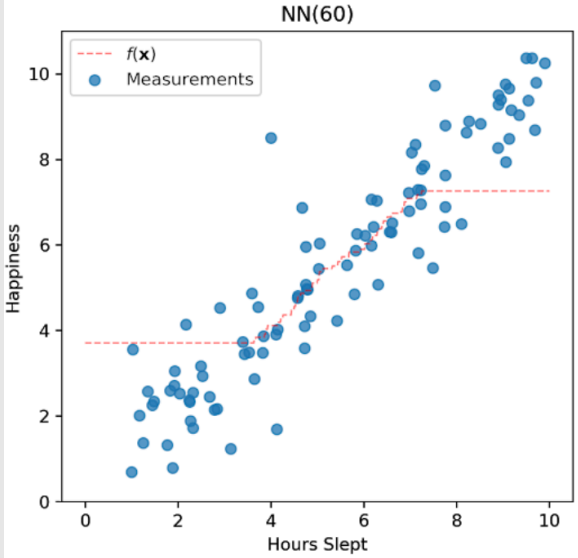
The NN(1) model train/test performance are not consistent

Example: train and test NN(15)

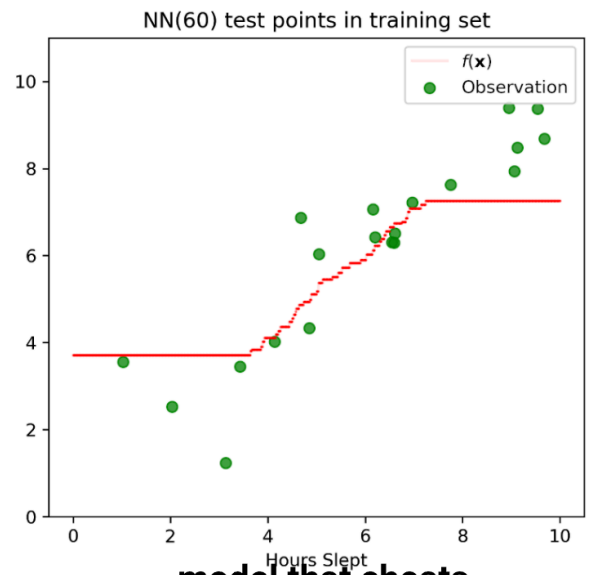


The NN(15) model train/test performance are consistent

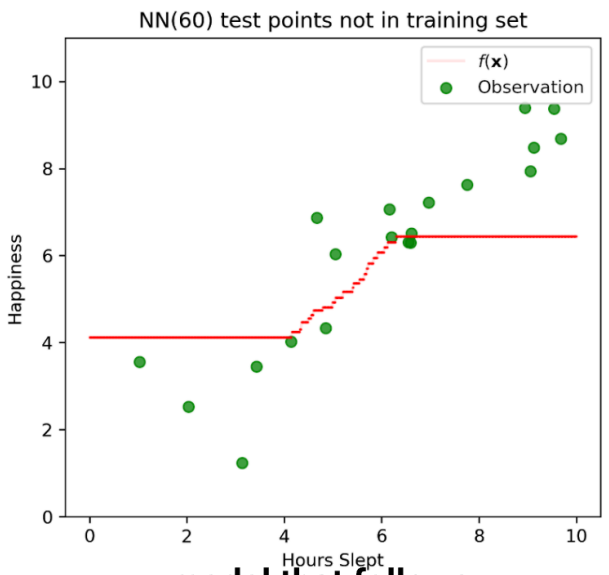
Example: train and test NN(60)



original model



model that cheats



model that follows the rules

The NN(60) model train/test performance are consistently bad.

Example: summary

Model	Train	Test
NN(1)	0.00	2.40
NN(15)	1.04	0.86
NN(60)	3.82	2.44

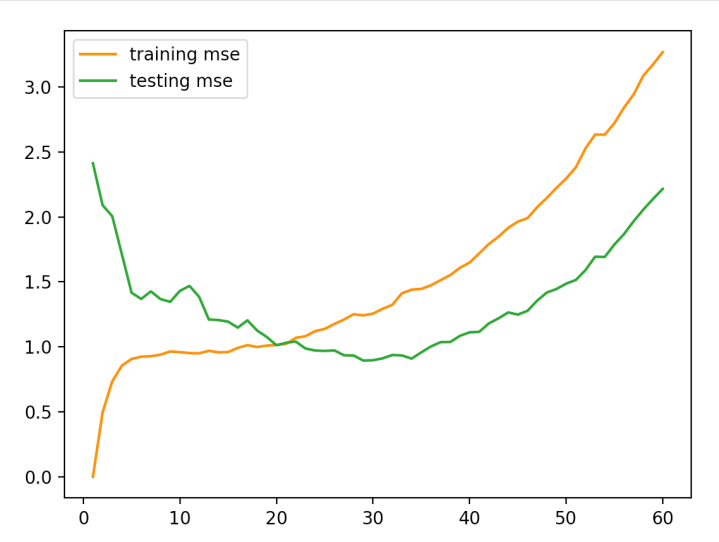
The best model is found be looking at the best performance on the testing set.

You try it

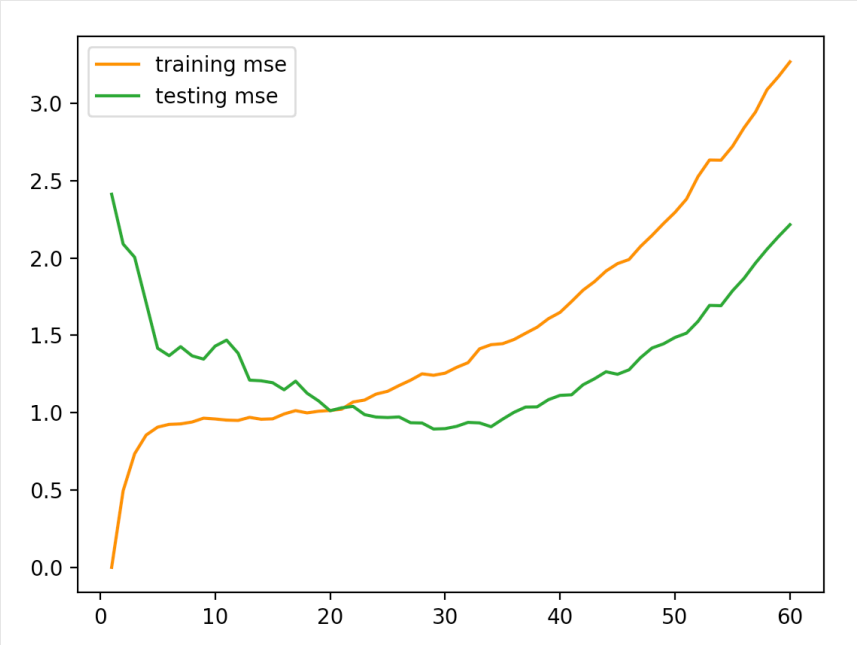
Use a for loop to find the best k value from 1 to 30

Plot the train and test MSE using plt.plot() from matplotlib

What is the best k?



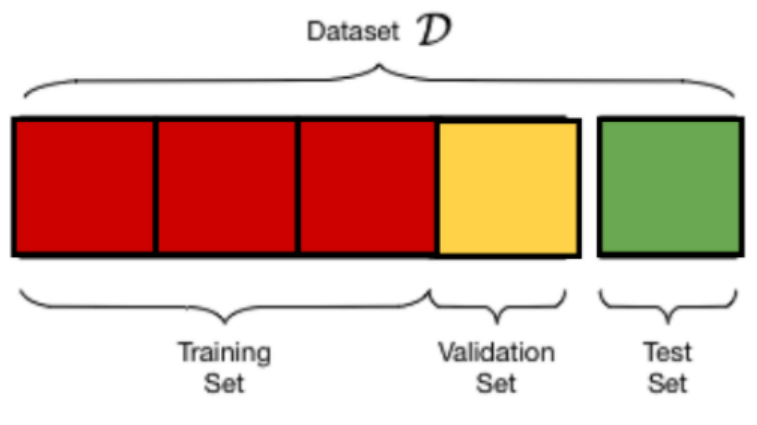
You try it



Optimal k is found where train/test performance are similarly good

One last problem

We are violating the rule of testing once!



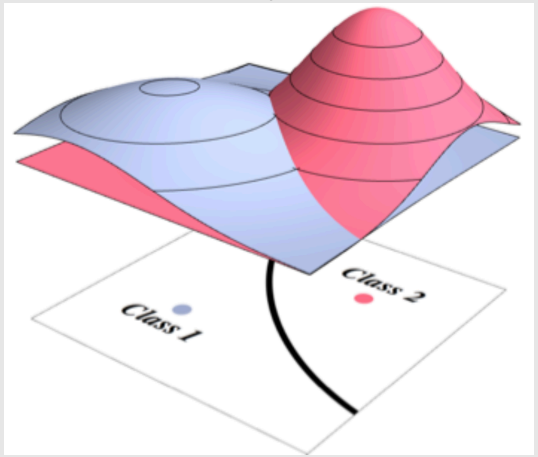
Solution: set aside another part of the data. The validation set is used to decide the hyperparameter k.

Homework 1

- Load the iris dataset and find the best value for k using prediction error and a train-test split as seen today but for classification.

Bayes error rate (binary classifier)

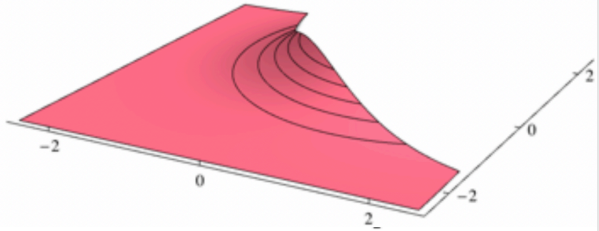
Assume we know $P(y|x)$, we can predict the label using $y^* = \underset{y}{\operatorname{argmax}}\, P(y|x)$



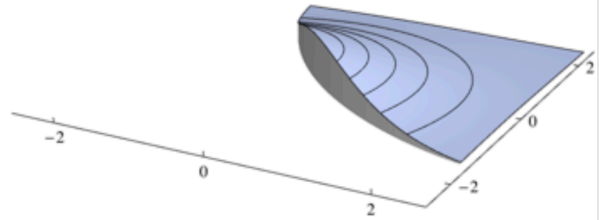
Of all classifiers, the Optimal Bayes classifier is the one with lowest misclassification rate

With perfect knowledge of the input data, we may still make mistakes

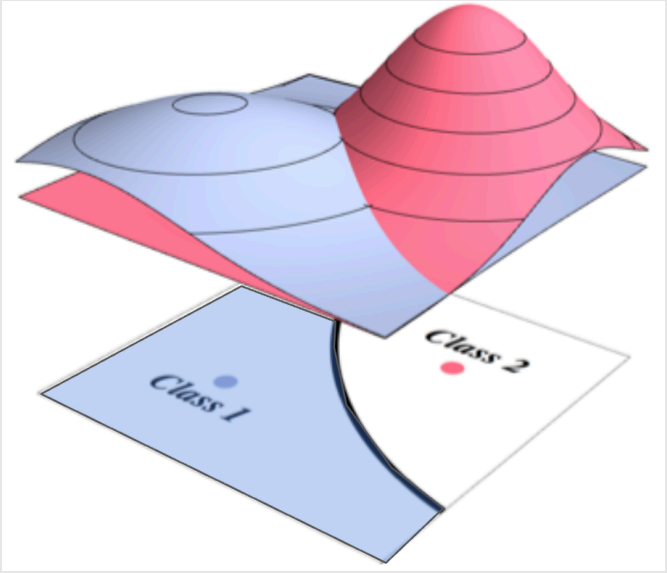
$P(y = 1 \mid \mathbf{x}) < 1$



$P(y = 2 \mid \mathbf{x}) < 1$



Bayes error rate (binary classifier)



Probability of correctly estimate y when x is in region for H_1

$$\epsilon_{Bayes} = 1 - \int_{x \in H_1} P(y = 1|x)p(x)dx - \int_{x \in H_2} P(y = 2|x)p(x)dx$$

Probability of correctly estimate y when x is in region for H_2

If the value of x is given, the probability of misclassification using Bayes optimal classifier is

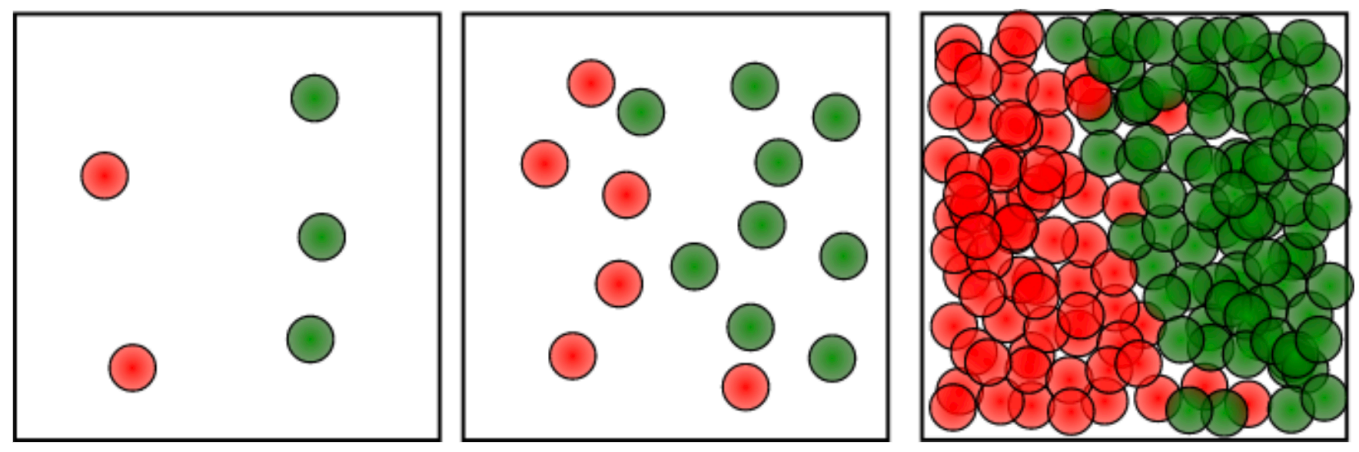
$$\epsilon_{Bayes} = 1 - P(y = y^*|x)$$

Bayes error rate (binary classifier)

Bayes optimal classifier provides a lower bound of the classification error rate. With the same feature representation, no classifier can obtain a lower error)

We can prove that as $n \to \infty$, the error of 1-NN classifier is no more than twice the error of the Bayes Optimal classifier.

Convergence of 1-NN



Suppose $x^{(NN)}$ is the nearest neighbor for test point $x^{(t)}$

When sample size M$\rightarrow \infty$, $dist\left(x^{(NN)}, x^{(t)}\right) \rightarrow 0$. We have $P\left(y|x^{(t)}\right) = P(y|x^{(NN)})$

Convergence of 1-NN

For any test point $x^{(t)}$, the best possible error rate is

$$P(mistake) = 1 - P(y^{(t)} = y^* | x^{(t)})$$

For 1-NN classifier,

$$\begin{aligned} P(mistake) &= P\left(y^{(t)} = y^* | x^{(t)}\right)\left(1 - P\left(y^{(NN)} = y^* | x^{(NN)}\right)\right) + P\left(y^{(NN)} = y^* | x^{(NN)}\right)\left(1 - P\left(y^{(t)} = y^* | x^{(t)}\right)\right) \\ &\leq \left(1 - P\left(y^{(NN)} = y^* | x^{(NN)}\right)\right) + \left(1 - P\left(y^{(t)} = y^* | x^{(t)}\right)\right) \\ &= 2\left(1 - P\left(y^{(t)} = y^* | x^{(t)}\right)\right) \end{aligned}$$

As n→∞, the 11-NN classifier is only a factor 2 worse than the best possible classifier

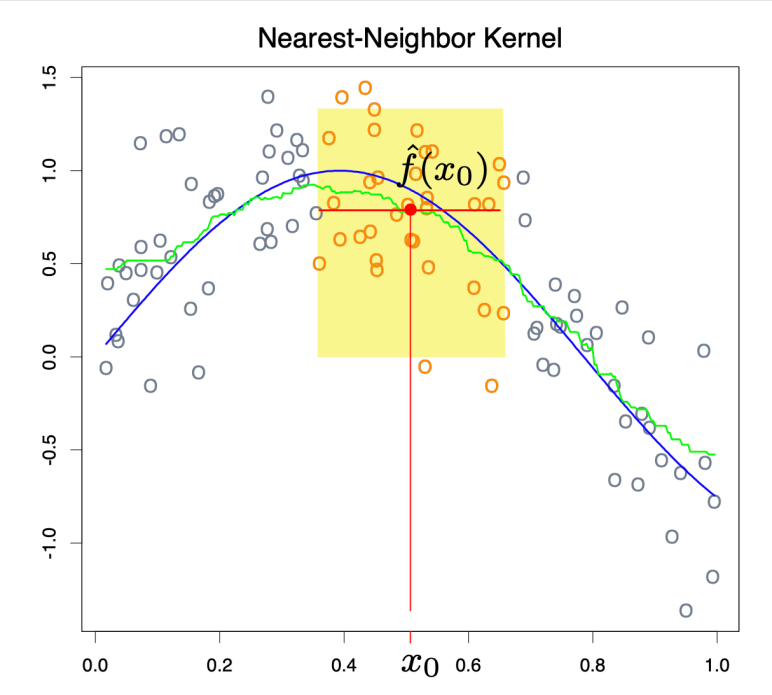
Make the KNN smooth

KNN regressor

$$f(x) = \frac{1}{k}\sum_{i \in NN(x)} y_i$$

Data generated from

$Y = \sin(4X) + \varepsilon, \varepsilon \sim N(0, 1/3)$



Make the KNN smooth

$$f(x) = \frac{1}{k}\sum_{i \in NN(x)} y_i$$

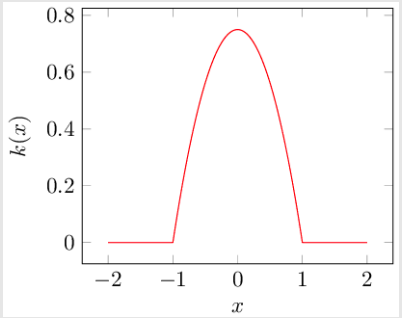
$$f(x) = \frac{\sum_{i=1}^{N} K_\lambda(x, x_i)\, y_i}{\sum_{i=1}^{N} K_\lambda(x, x_i)}$$

A special case with
$\sum_{i=1}^{N} K_\lambda(x, x_i) = 1$ for all $x_i \in NN(x)$

Epanechnikov Kernel

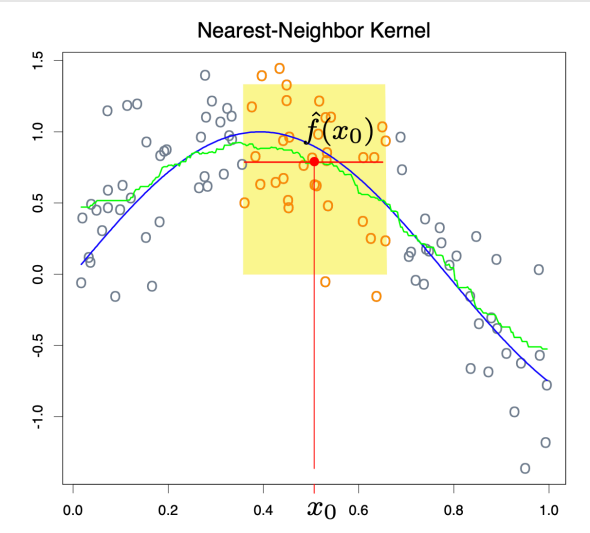
$$K_\lambda(x, x_i) = D\left(\frac{|x - x_i|}{\lambda}\right)$$

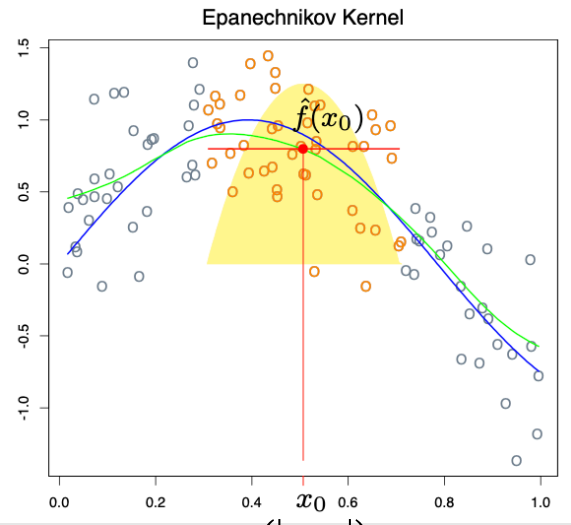
$$D(t) = \begin{cases} \frac{3}{4}(1 - \mathrm{t}^2) & \text{if } |t| \le 1 \\ 0 & \text{otherwise} \end{cases}$$



Window size parameter λ determines the width of the local neighborhood

Make the KNN smooth



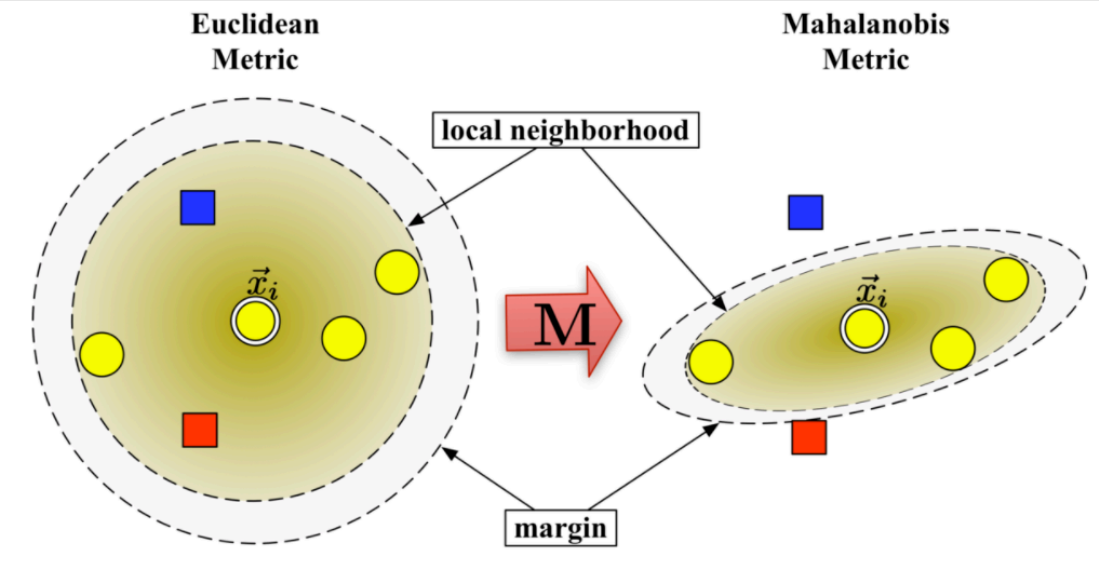


$$K_\lambda(x, x_i) = D\left(\frac{|x - x_i|}{\lambda}\right)$$

we can use **adaptive neighborhood size** to have a better estimate fit at boundary regions.

How to potentially improve performance of KNN

The k-nearest neighbor classifier fundamentally relies on a distance metric.
Learn a new distance metric under which the data instances are surrounded by instances with the same label



Pros and Cons of KNN Algorithm

- It is conceptually simple and does not require learning (term: memory-base).
- It can be used even with few examples.
- Even for moderate k: wonderful performer.
- It works very well in low dimensions for complex decision surfaces.

- With N observations and p predictors, KNN algorithm is *np* hard
- Require storage of the whole training data
- It suffers A LOT from the curse-of-dimensionality.