10-301/601: Introduction to Machine Learning Lecture 5 – KNNs & Model Selection

Henry Chai & Matt Gormley

9/14/22

Front Matter

- Announcements:
  - HW2 released 9/7, due 9/19 at 11:59 PM
  - HW1 exit poll released 9/8, you must respond by 9/15 in order to receive full credit
  - HW1 grades released 9/13
    - If you received 66 points or more on the written portion (~90%), you will receive full credit for the written portion
    - If you received less than 66 points, you have until **Wednesday, September 21st at 11:59 PM** (one week from today) to resubmit the written portion
    - **Please only resubmit if you received less than 66 points**

Q & A:

Man, I've really been struggling with HW2, especially the programming...

- … should I violate academic integrity and use someone else's code?
- **NO, ABSOLUTELY NOT.**
  - Doing so carries serious penalties, both in this class and at the university level.
- … But really, how would you all know?

MOSS

- Measure Of Software Similarity (MOSS) is an automatic system for determining the similarity of programs.
- One application of MOSS is detecting plagiarism in programming classes.
- Moss reports:
  - The Andrew IDs associated with the file submissions
  - The number of lines matched
  - The percent lines matched
  - Color coded submissions where similarities are found

MOSS

- At first glance, these submissions may look different…

```python
# Python program to find ordered words
import requests

# Scrapes the words from the URL below and stores
# them in a list
def getWords():

# contains about 2500 words
    url = "http://www.puzzlers.org/pub/wordlists/unixdict.txt"
    fetchData = requests.get(url)

# extracts the content of the webpage
    wordList = fetchData.content

# decodes the UTF-8 encoded text and splits the
# string to turn it into a list of words
    wordList = wordList.decode("utf-8").split()

    return wordList


# function to determine whether a word is ordered or not
def isOrdered():

# fetching the wordList
    collection = getWords()

# since the first few of the elements of the
# dictionary are numbers, getting rid of those
# numbers by slicing off the first 17 elements
    collection = collection[16:]
    word = ''

    for word in collection:
        result = 'Word is ordered'
        i = 0
        l = len(word) - 1

        if (len(word) < 3): # skips the 1 and 2 lettered strings
            continue

        # traverses through all characters of the word in pairs
        while i < l:
            if (ord(word[i]) > ord(word[i+1])):
                result = 'Word is not ordered'
                break
            else:
                i += 1

    # only printing the ordered words
        if (result == 'Word is ordered'):
            print(word,': ',result)


# execute isOrdered() function
if __name__ == '__main__':
    isOrdered()
```

```python
import requests

def Ordered():
    coll = getWs()
    coll = coll[16:]
    word = ''
    for word in coll:
        r = 'Word is ordered'
        a = 0
        length = len(word) - 1
        if (len(word) < 3):
            continue
        while a < length:
            if (ord(word[a]) > ord(word[a+1])):
                r = 'Word is not ordered'
                break
            else:
                a += 1
        if (r == 'Word is ordered'):
            print(word,': ',r)

def getWs():
    url = "http://www.puzzlers.org/pub/wordlists/unixdict.txt"
    fetch = requests.get(url)
    words = fetch.content
    words = words.decode("utf-8").split()
    return words

if __name__ == '__main__':
    Ordered()
```

MOSS

- … when in fact, MOSS can easily identify the similarities

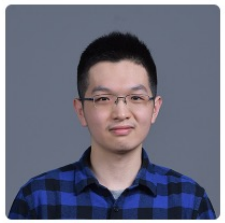
```
>>>> file: bedmunds@andrew.cmu.edu_1_handin.c
# Python program to find ordered words
import requests

# Scrapes the words from the URL below and stores
# them in a list

def getWords():

# contains about 2500 words
    url = "http://www.puzzlers.org/pub/wordlists/unixdict.txt"
    fetchData = requests.get(url)

# extracts the content of the webpage
    wordList = fetchData.content

# decodes the UTF-8 encoded text and splits the
# string to turn it into a list of words
    wordList = wordList.decode("utf-8").split()

    return wordList



# function to determine whether a word is ordered or not
def isOrdered():

# fetching the wordList
    collection = getWords()

# since the first few of the elements of the
# dictionary are numbers, getting rid of those
# numbers by slicing off the first 17 elements
    collection = collection[16:]
    word = ''

    for word in collection:
        result = 'Word is ordered'
        i = 0
        l = len(word) - 1

        if (len(word) < 3): # skips the 1 and 2 lettered strings
            continue

        # traverses through all characters of the word in pairs
        while i < l:
            if (ord(word[i]) > ord(word[i+1])):
                result = 'Word is not ordered'
                break
            else:
                i += 1

    # only printing the ordered words
        if (result == 'Word is ordered'):
            print(word,': ',result)


# execute isOrdered() function
if __name__ == '__main__':
    isOrdered()
```

```
>>>> file: dpbird@andrew.cmu.edu_1_handin.c


import requests

def Ordered():
    coll = getWs()
    coll = coll[16:]
    word = ''
    for word in coll:
        r = 'Word is ordered'
        a = 0
        length = len(word) - 1
        if (len(word) < 3):
            continue
        while a < length:
            if (ord(word[a]) > ord(word[a+1])):
                r = 'Word is not ordered'
                break
            else:
                a += 1
        if (r == 'Word is ordered'):
            print(word,': ',r)


def getWs():
    url = "http://www.puzzlers.org/pub/wordlists/unixdict.txt"
    fetch = requests.get(url)
    words = fetch.content
    words = words.decode("utf-8").split()
    return words

if __name__ == '__main__':
    Ordered()
```

Course Staff


Hayden Kim


Sami Kale


Chongling Zhu


Tara Lakdawala


Alex Xie


Lavanya Gupta


Aditi Sharma


Hang Shu


Brynn Edmunds


Abhi Vijayakumar


Yuchen Xu


Qiuyi Yin


Chu Weng

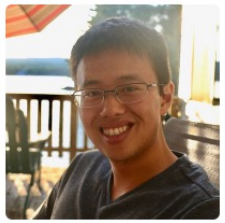

Brandon Wang


Shubham Virmani


Lulu Ricketts


Jack Lyu


Pranay Gundam


Henry Chai


Matt Gormley


Kalvin Chang


Monica Geng

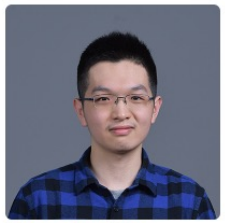

Jeneel Mashru

Course Staff

HW2/6


Hayden Kim


Sami Kale


Chongling Zhu


Tara Lakdawala


Alex Xie


Lavanya Gupta


Aditi Sharma


Hang Shu


Brynn Edmunds


Abhi Vijayakumar


Yuchen Xu


Qiuyi Yin


Chu Weng

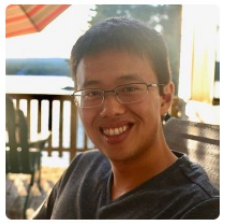

Brandon Wang


Shubham Virmani


Lulu Ricketts


Jack Lyu


Pranay Gundam


Henry Chai


Matt Gormley


Kalvin Chang

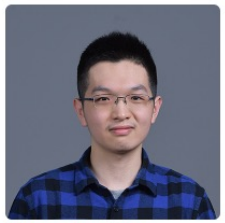

Monica Geng


Jeneel Mashru

Course Staff


Hayden Kim


Sami Kale


Chongling Zhu


Tara Lakdawala


Alex Xie


Lavanya Gupta


Aditi Sharma

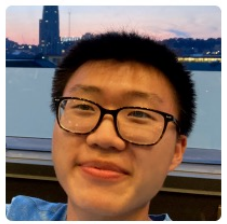

Hang Shu


Brynn Edmunds


Abhi Vijayakumar


Yuchen Xu


Qiuyi Yin


Chu Weng


Brandon Wang


Shubham Virmani


Lulu Ricketts


Jack Lyu


Pranay Gundam


Henry Chai


Matt Gormley


Kalvin Chang


Monica Geng

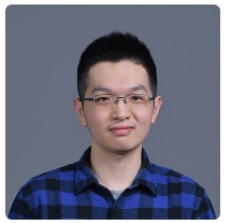

Jeneel Mashru

HW3/7

Course Staff


Hayden Kim


Sami Kale


Chongling Zhu


Tara Lakdawala

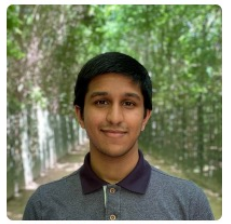

Alex Xie


Lavanya Gupta


Aditi Sharma


Hang Shu


Brynn Edmunds


Abhi Vijayakumar


Yuchen Xu


Qiuyi Yin


Chu Weng

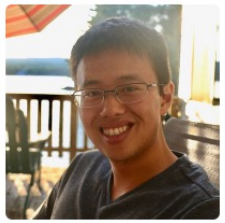

Brandon Wang


Shubham Virmani


Lulu Ricketts


Jack Lyu


Pranay Gundam


Henry Chai


Matt Gormley


Kalvin Chang


Monica Geng


Jeneel Mashru

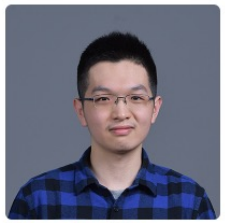
HW4/8

Course Staff

HW5/9


Hayden Kim


Sami Kale


Chongling Zhu


Tara Lakdawala

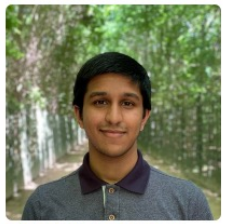

Alex Xie


Lavanya Gupta


Aditi Sharma

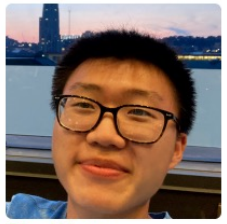

Hang Shu


Brynn Edmunds


Abhi Vijayakumar


Yuchen Xu


Qiuyi Yin


Chu Weng

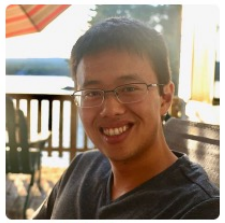

Brandon Wang


Shubham Virmani


Lulu Ricketts


Jack Lyu


Pranay Gundam


Henry Chai


Matt Gormley


Kalvin Chang


Monica Geng

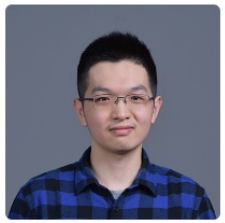

Jeneel Mashru

Q & A:

Who is the single most important person on our course staff?


Hayden Kim


Sami Kale


Chongling Zhu


Tara Lakdawala


Alex Xie


Lavanya Gupta


Aditi Sharma


Hang Shu


Brynn Edmunds


Abhi Vijayakumar


Yuchen Xu


Qiuyi Yin


Chu Weng

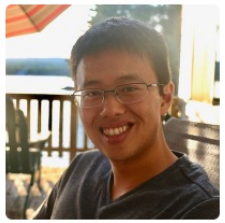

Brandon Wang


Shubham Virmani


Lulu Ricketts


Jack Lyu


Pranay Gundam


Henry Chai


Matt Gormley

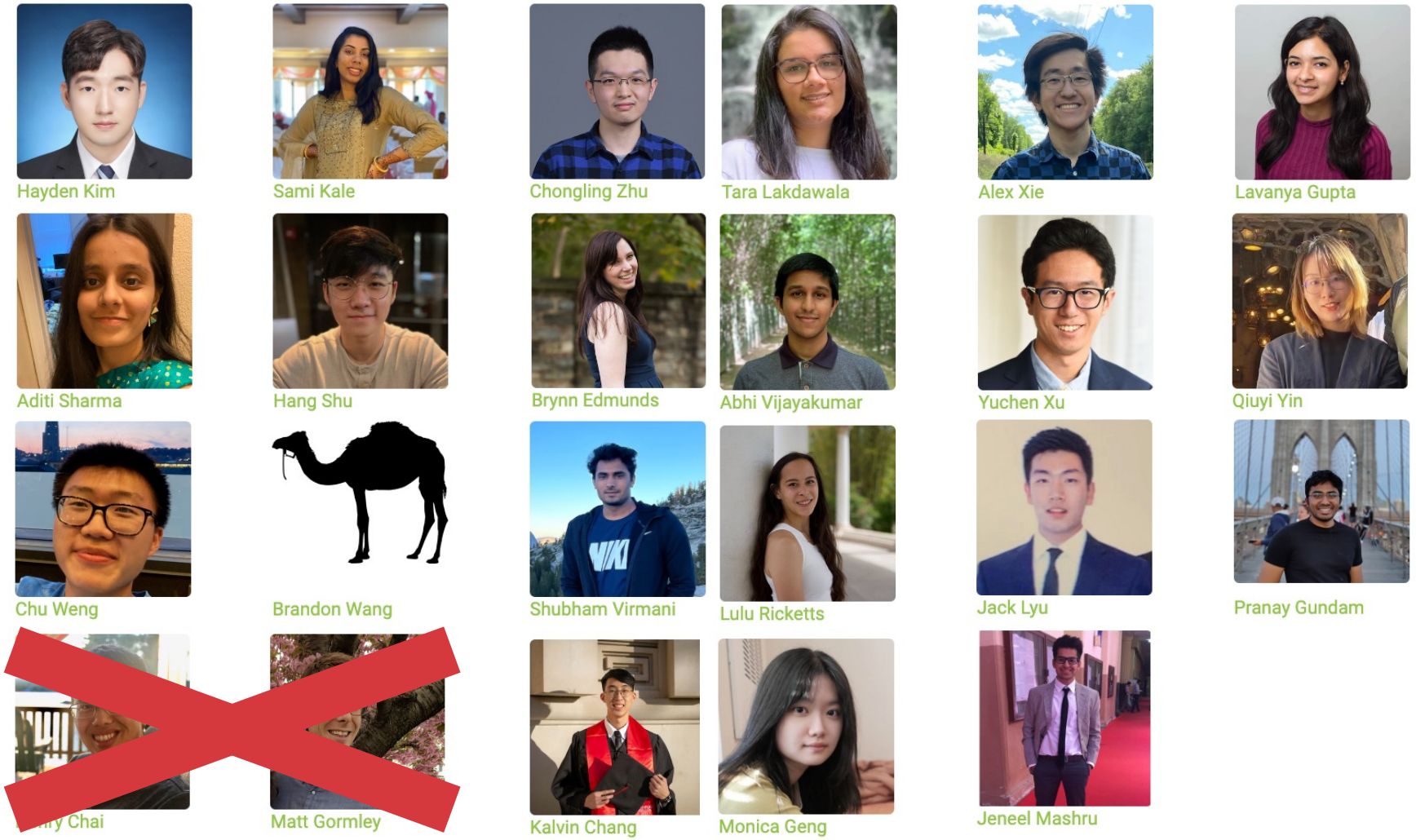

Kalvin Chang


Monica Geng


Jeneel Mashru

Q & A:

Who is the single most important person on our course staff?

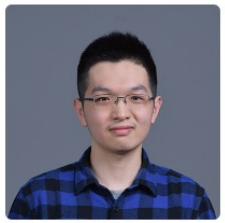


Brynn, Education Associate (EA)

eas-10-601 @cs.cmu.edu


Hayden Kim


Sami Kale


Chongling Zhu


Tara Lakdawala

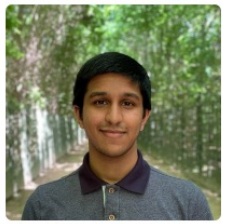

Alex Xie


Lavanya Gupta


Aditi Sharma


Hang Shu


Brynn Edmunds


Abhi Vijayakumar


Yuchen Xu


Qiuyi Yin


Chu Weng

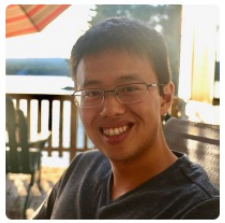

Brandon Wang


Shubham Virmani


Lulu Ricketts


Jack Lyu


Pranay Gundam


Henry Chai


Matt Gormley


Kalvin Chang


Monica Geng


Jeneel Mashru

Recall: The Duck Test for Machine Learning

- Classify a point as the label of the "most similar" training point
- Idea: given real-valued features, we can use a distance metric to determine how similar two data points are
- A common choice is Euclidean distance:

$$d(\boldsymbol{x}, \boldsymbol{x}') = \|\boldsymbol{x} - \boldsymbol{x}'\|_2 = \sqrt{\sum_{d=1}^{D} (x_d - x'_d)^2}$$

- An alternative is the Manhattan distance:

$$d(\boldsymbol{x}, \boldsymbol{x}') = \|\boldsymbol{x} - \boldsymbol{x}'\|_1 = \sum_{d=1}^{D} |x_d - x'_d|$$

Nearest Neighbor: Pseudocode

```
def train(𝒟):
        store 𝒟
def predict(𝒙′):
        find the nearest neighbor to 𝒙′ in 𝒟, 𝒙^(i)
        return 𝑦^(i)
```

Decision Boundary Example

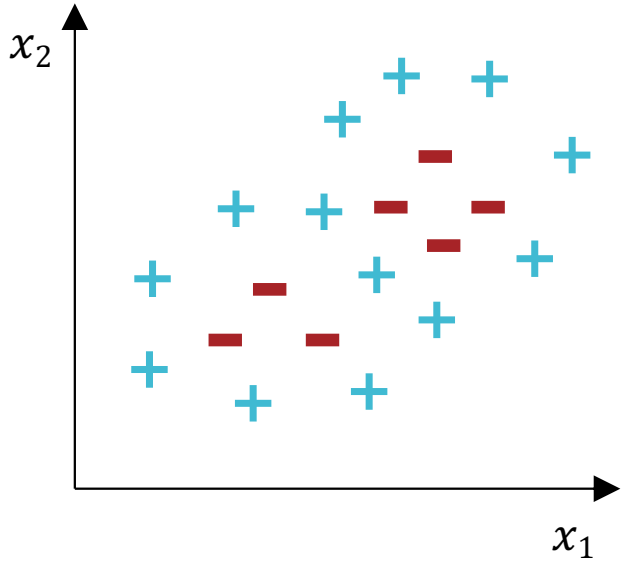
Poll Question 1:

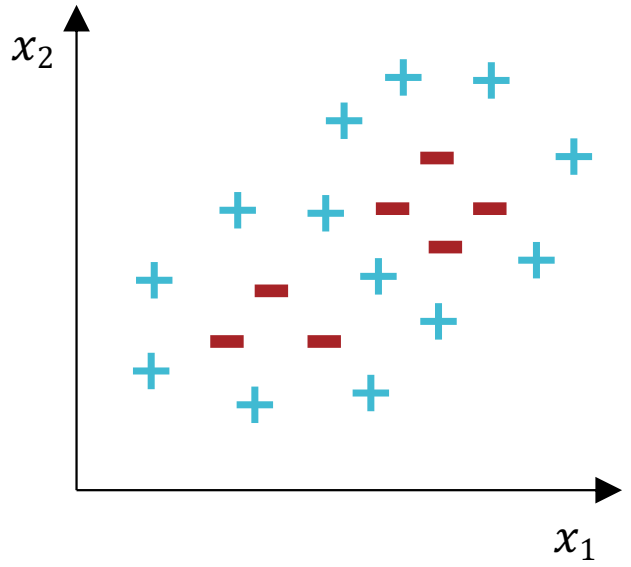
Can a **Nearest Neighbor classifier** achieve **zero training error** on this dataset? If so, draw the decision boundary and if not, briefly explain why.

A. Yes

B. No

C. Yes AND No **(TOXIC)**





Poll Question 2:

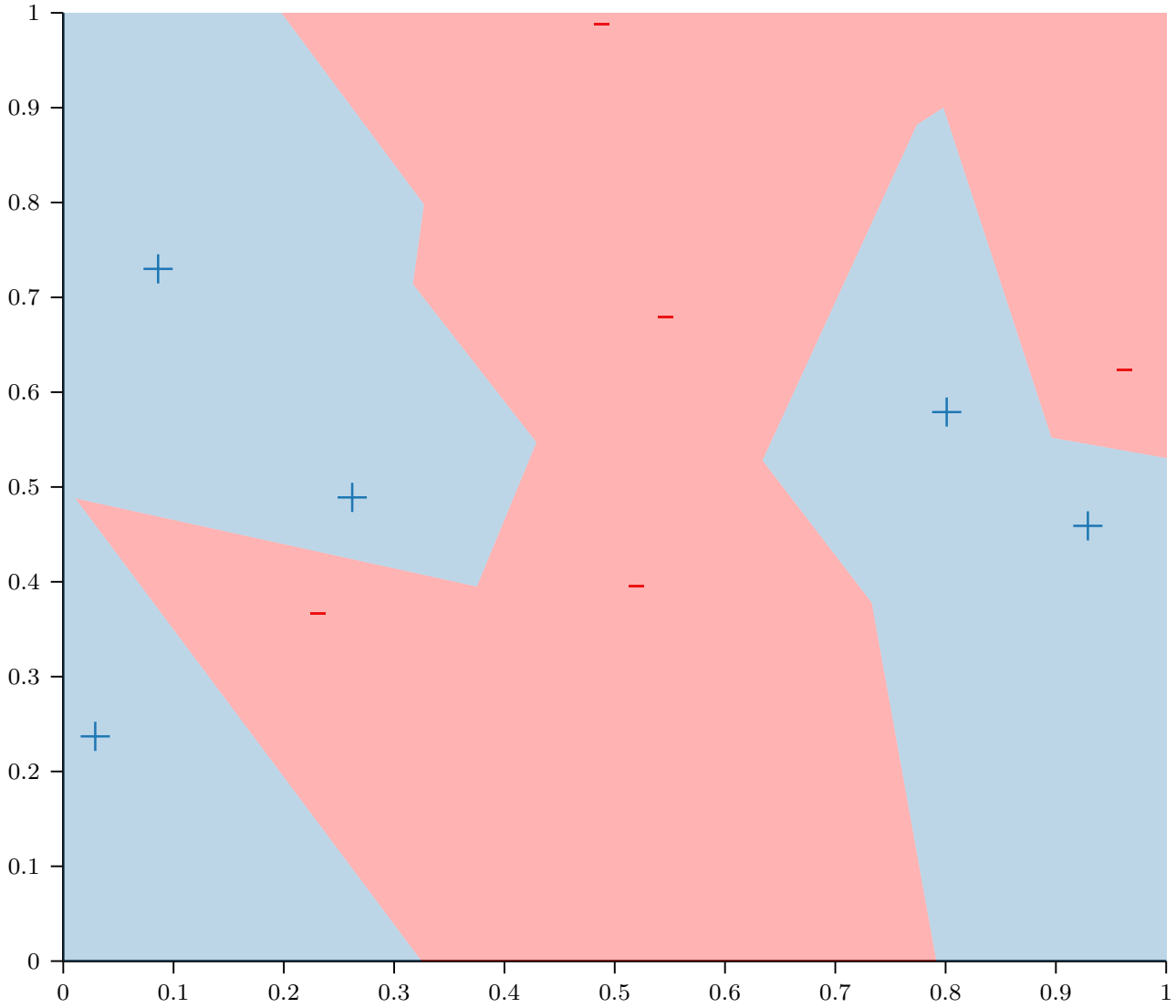
Can a **Decision Tree classifier** achieve **zero training error** on this dataset? If so, draw the decision boundary and if not, briefly explain why.

A. Yes

B. No

C. Yes AND No **(TOXIC)**

Nearest Neighbor: Example



The Nearest Neighbor Model

- Requires no training!
- Always has zero training error!
  - ***A data point is always its own nearest neighbor***

⋮

- Always has zero training error…

Generalization of Nearest Neighbor (Cover and Hart, 1967)

- Claim: under certain conditions, as $N \to \infty$, with high probability, the true error rate of the nearest neighbor model $\leq 2 * $ the Bayes error rate (the optimal classifier)
- Interpretation: "In this sense, it may be said that half the classification information in an infinite sample set is contained in the nearest neighbor."

Source: https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=1053964

Why stop at just one neighbor?

- Claim: under certain conditions, as $N \to \infty$, with high probability, the true error rate of the nearest neighbor model $\leq 2 * $ the Bayes error rate (the optimal classifier)
- Interpretation: “In this sense, it may be said that half the classification information in an infinite sample set is contained in the nearest neighbor.”

Source: https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=1053964

k-Nearest Neighbors (kNN)

- Classify a point as the most common label among the labels of the k nearest training points
- Tie-breaking (in case of even k and/or more than 2 classes)
  - Weight votes by distance
  - Remove furthest neighbor
  - Add next closest neighbor
  - Use a different distance metric

k-Nearest Neighbors (kNN): Pseudocode

```
def train(𝒟):
    store 𝒟
def predict(𝒙′):
    return majority_vote(labels of the k
    nearest neighbors to 𝒙′ in 𝒟)
```

kNN on Fisher Iris Data

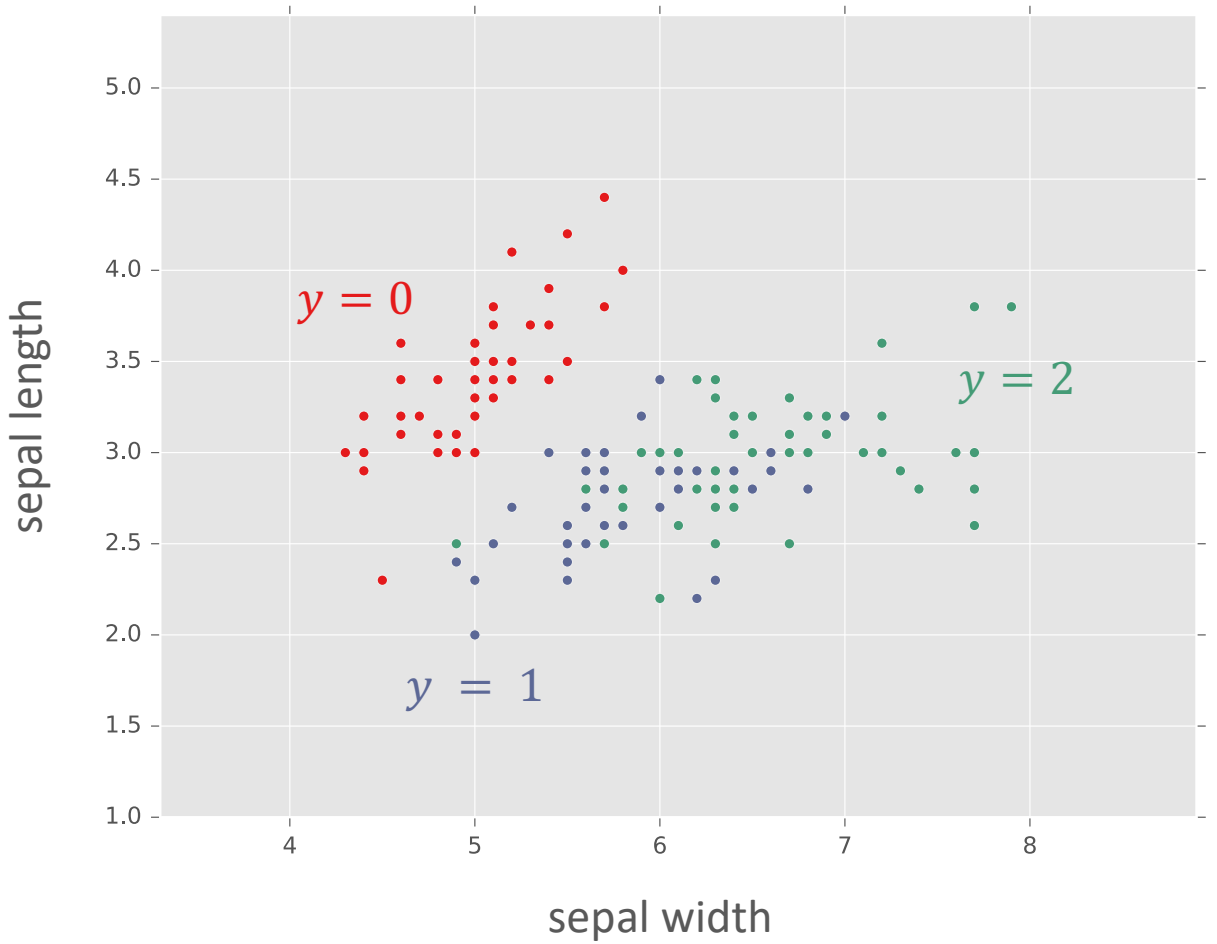


Figure courtesy of Matt Gormley

kNN on Fisher Iris Data

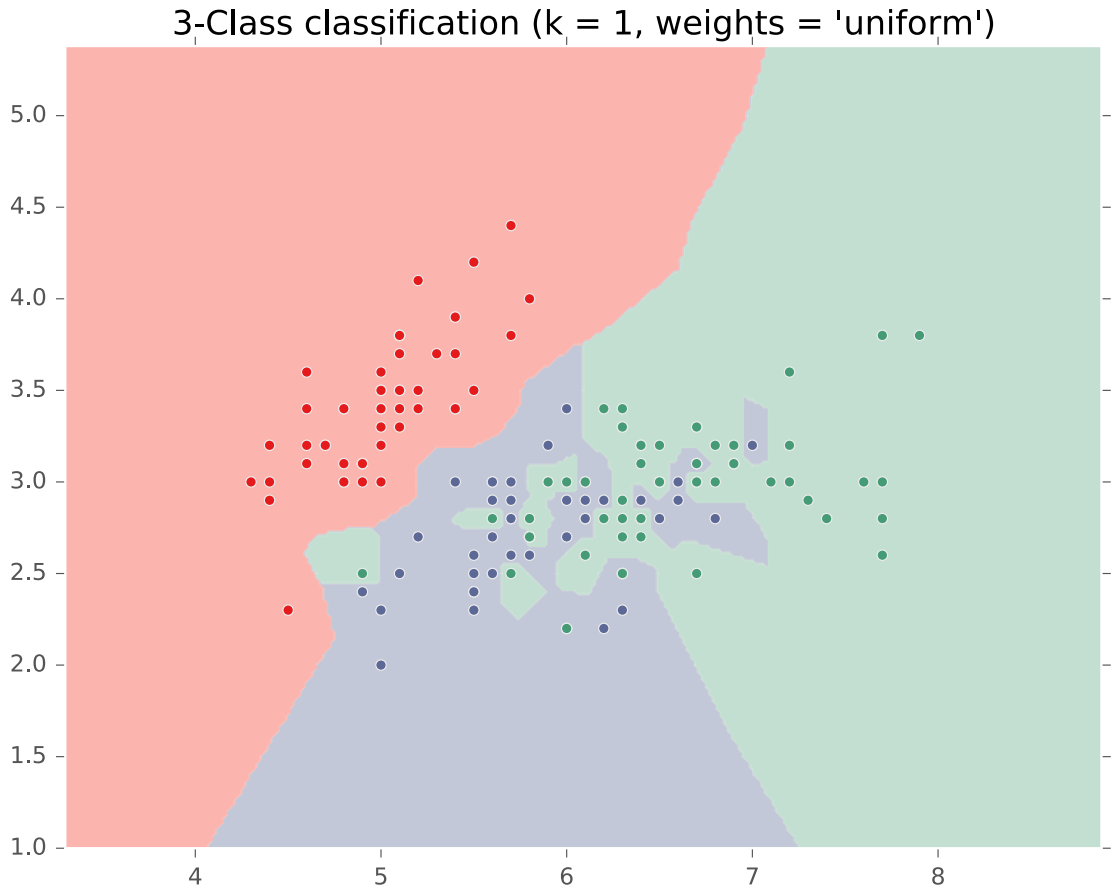


Figure courtesy of Matt Gormley

kNN on Fisher Iris Data

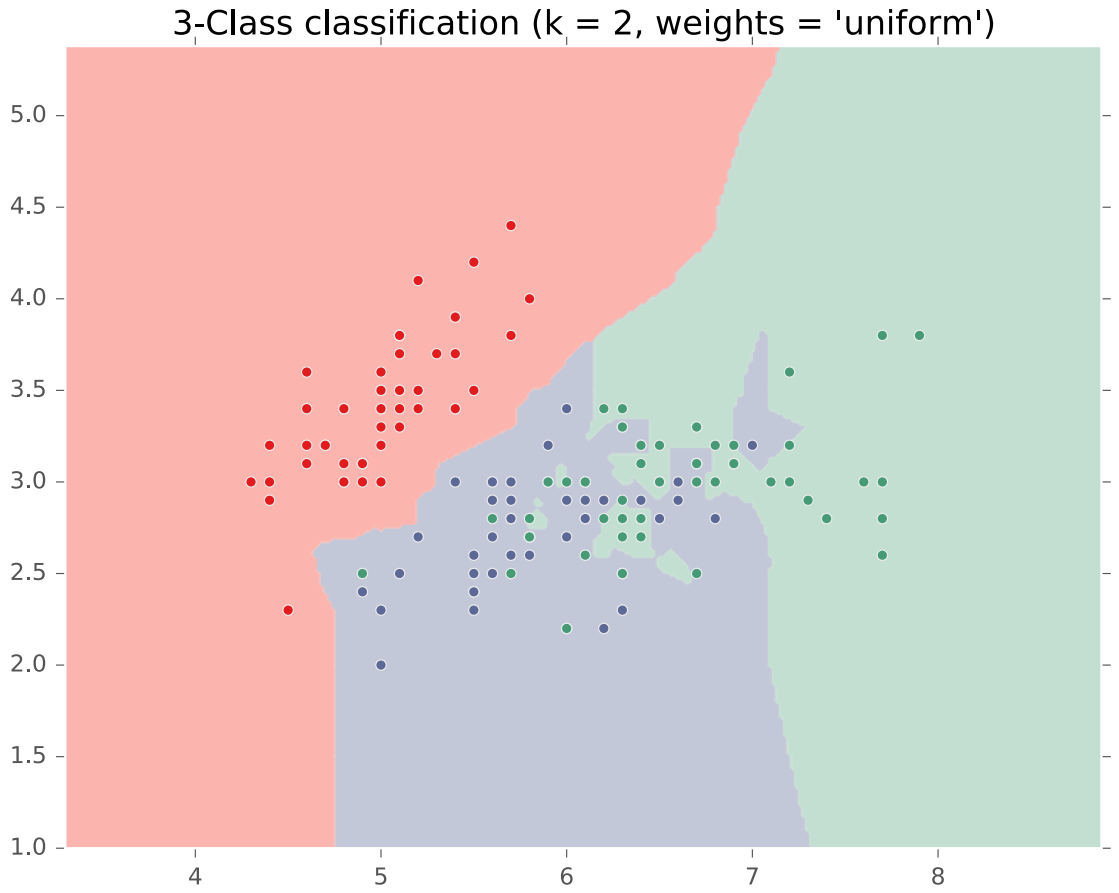


Figure courtesy of Matt Gormley

kNN on Fisher Iris Data

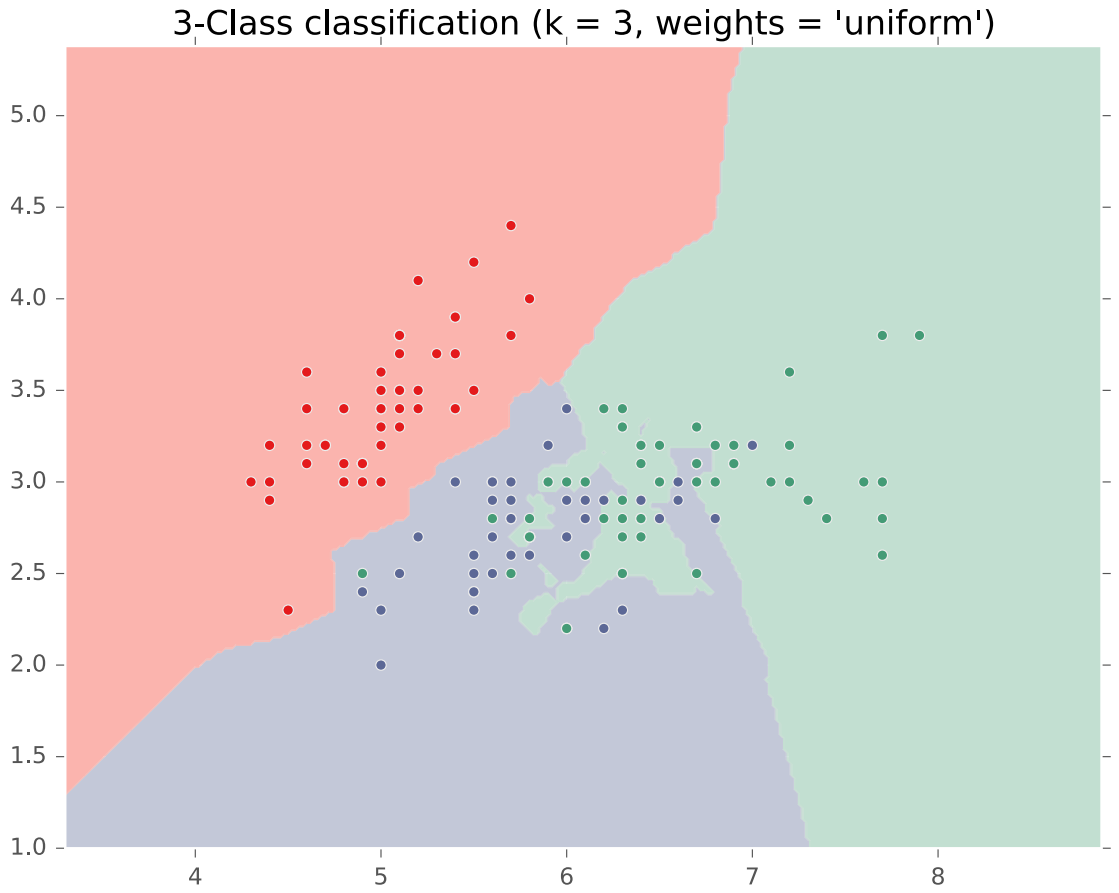


Figure courtesy of Matt Gormley

kNN on Fisher Iris Data

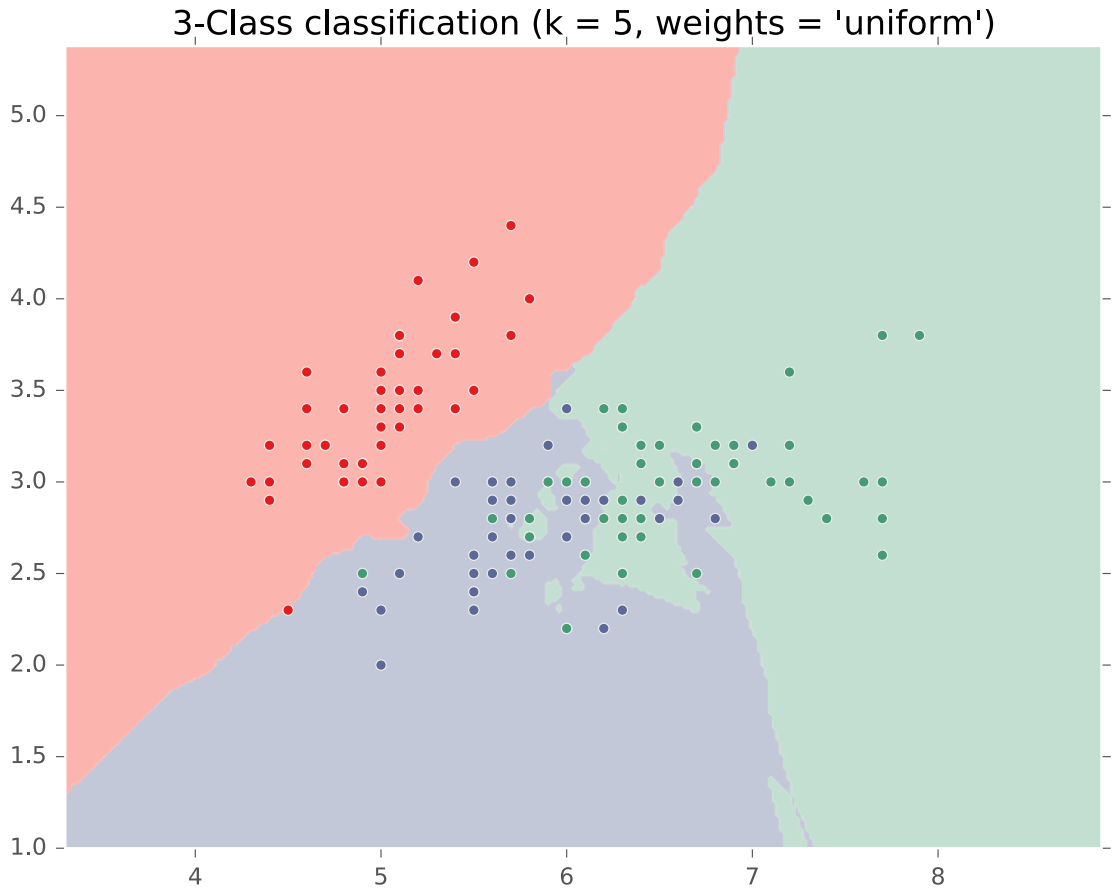


Figure courtesy of Matt Gormley

kNN on Fisher Iris Data

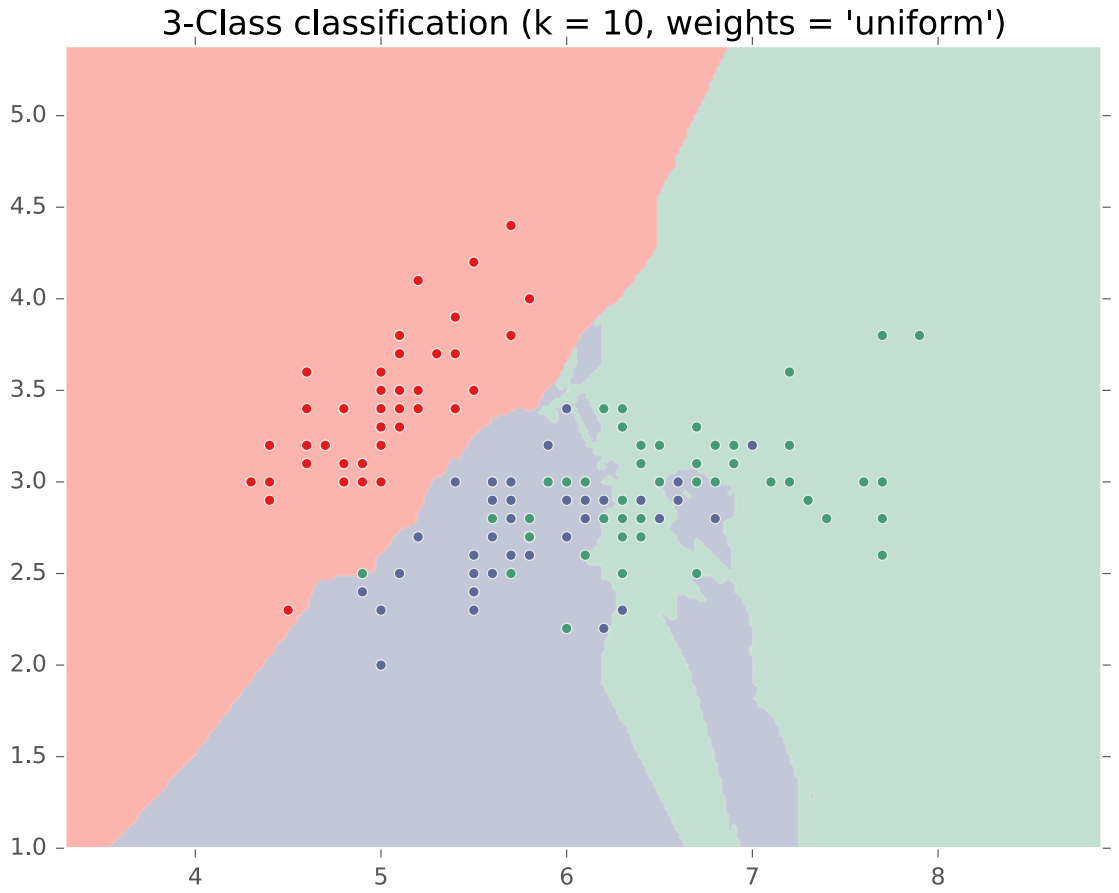


Figure courtesy of Matt Gormley

kNN on Fisher Iris Data

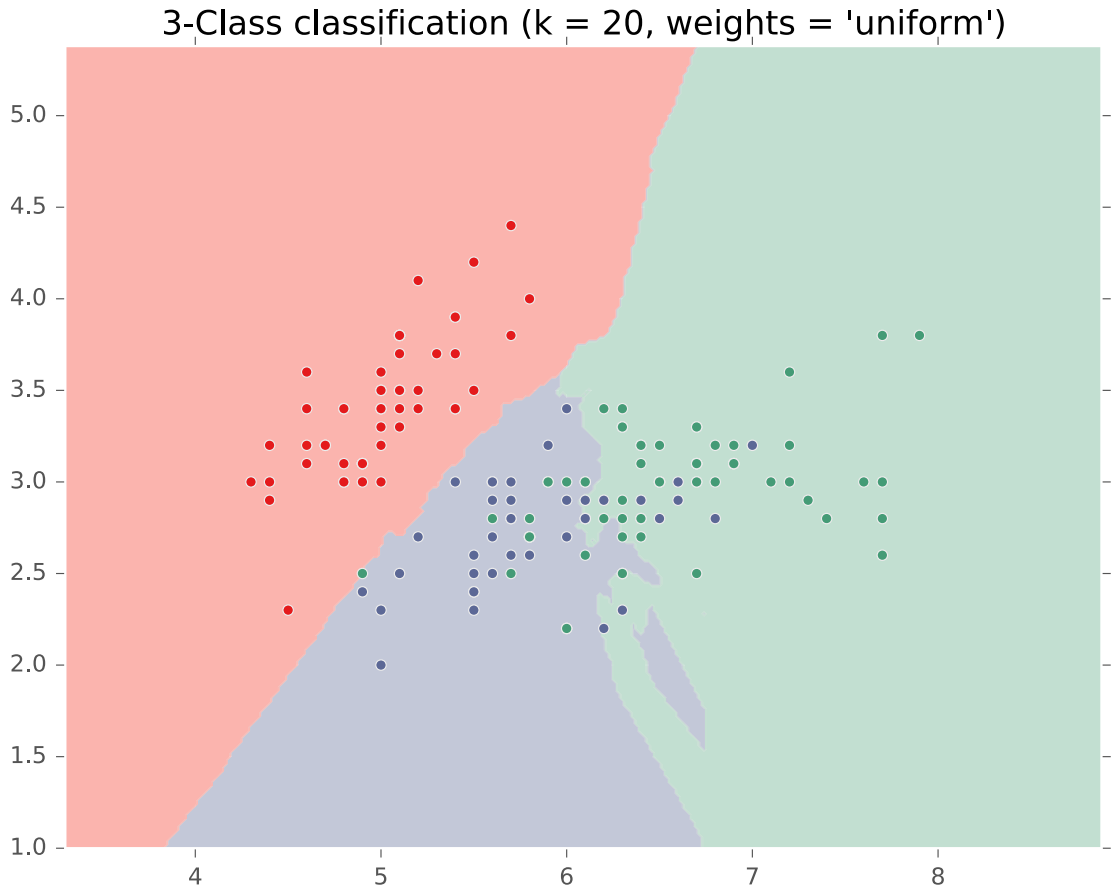


Figure courtesy of Matt Gormley

kNN on Fisher Iris Data

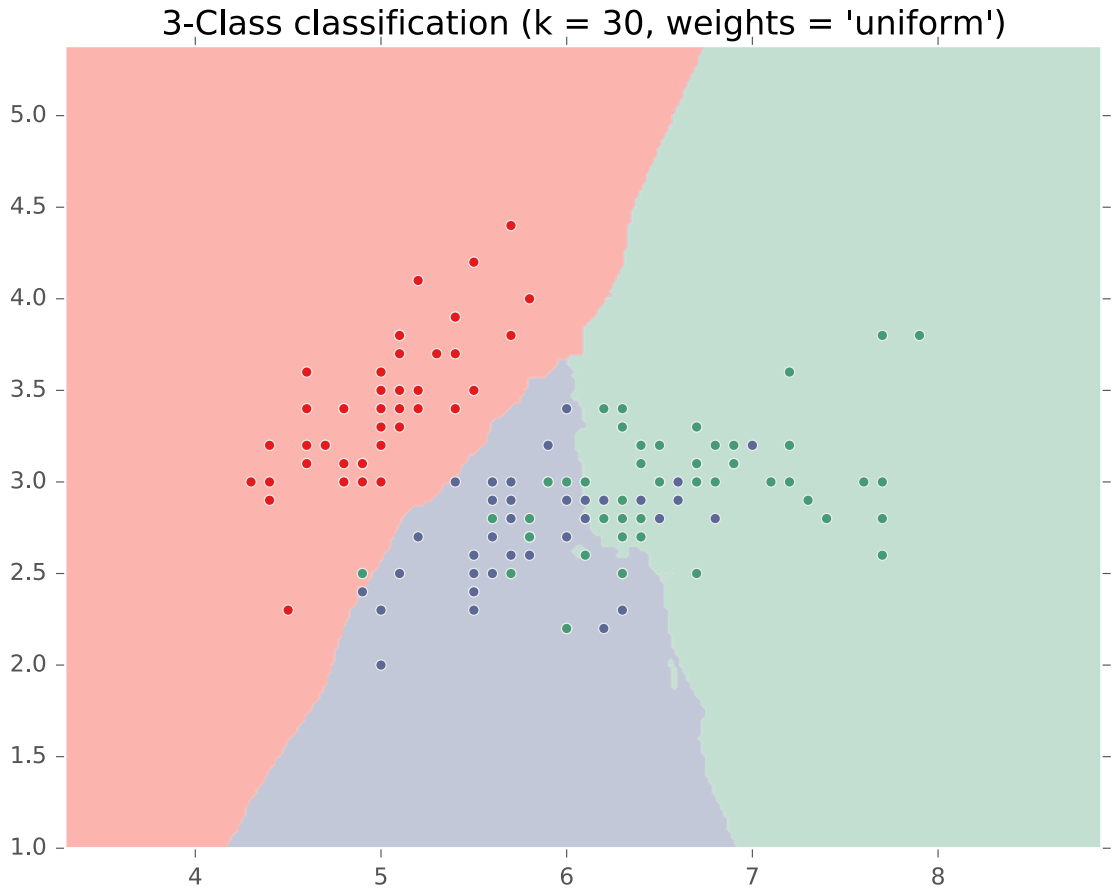


Figure courtesy of Matt Gormley

kNN on Fisher Iris Data

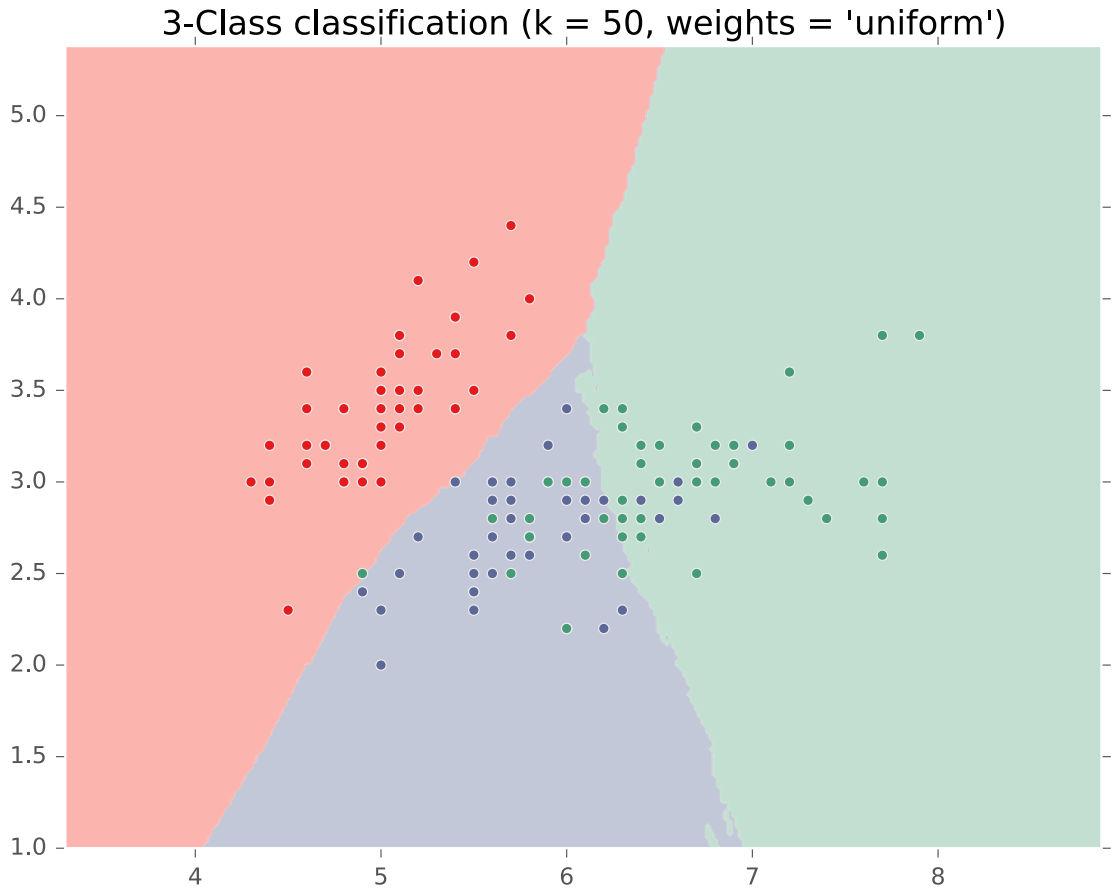


Figure courtesy of Matt Gormley

kNN on Fisher Iris Data

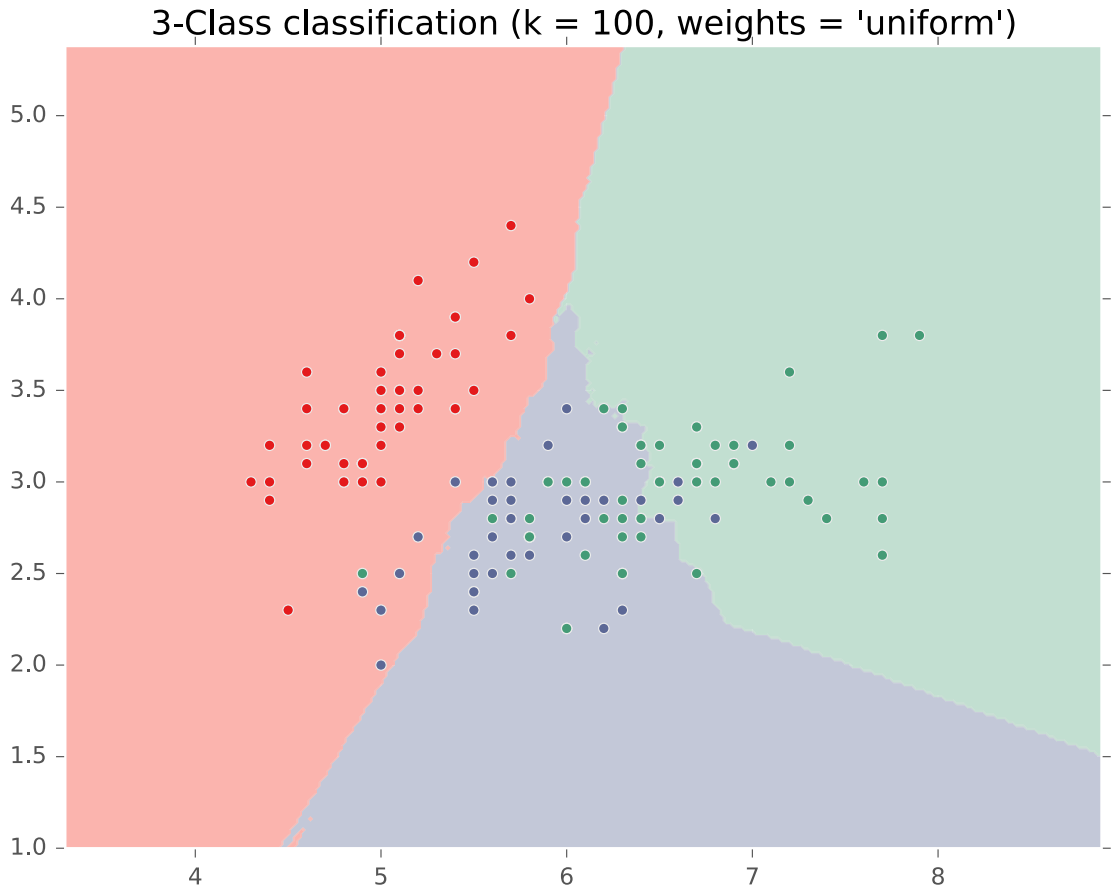


Figure courtesy of Matt Gormley

kNN on Fisher Iris Data

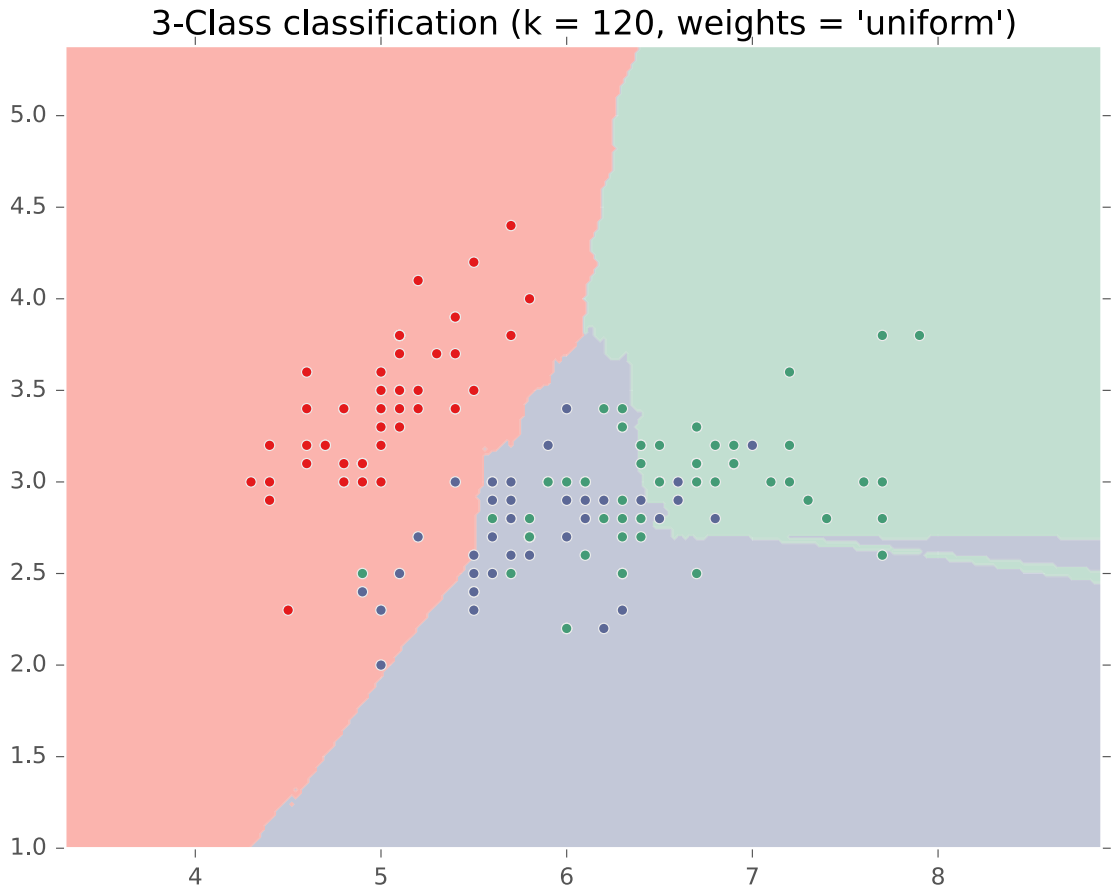


Figure courtesy of Matt Gormley

kNN on Fisher Iris Data

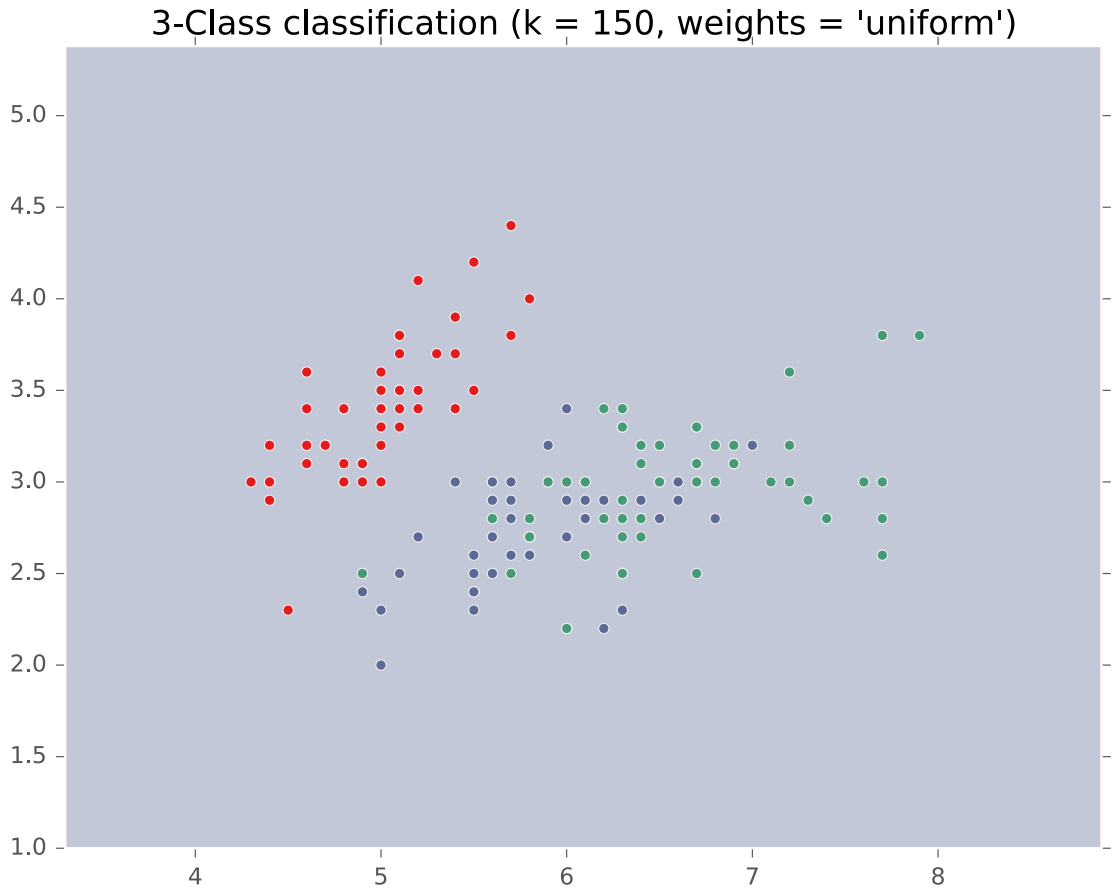
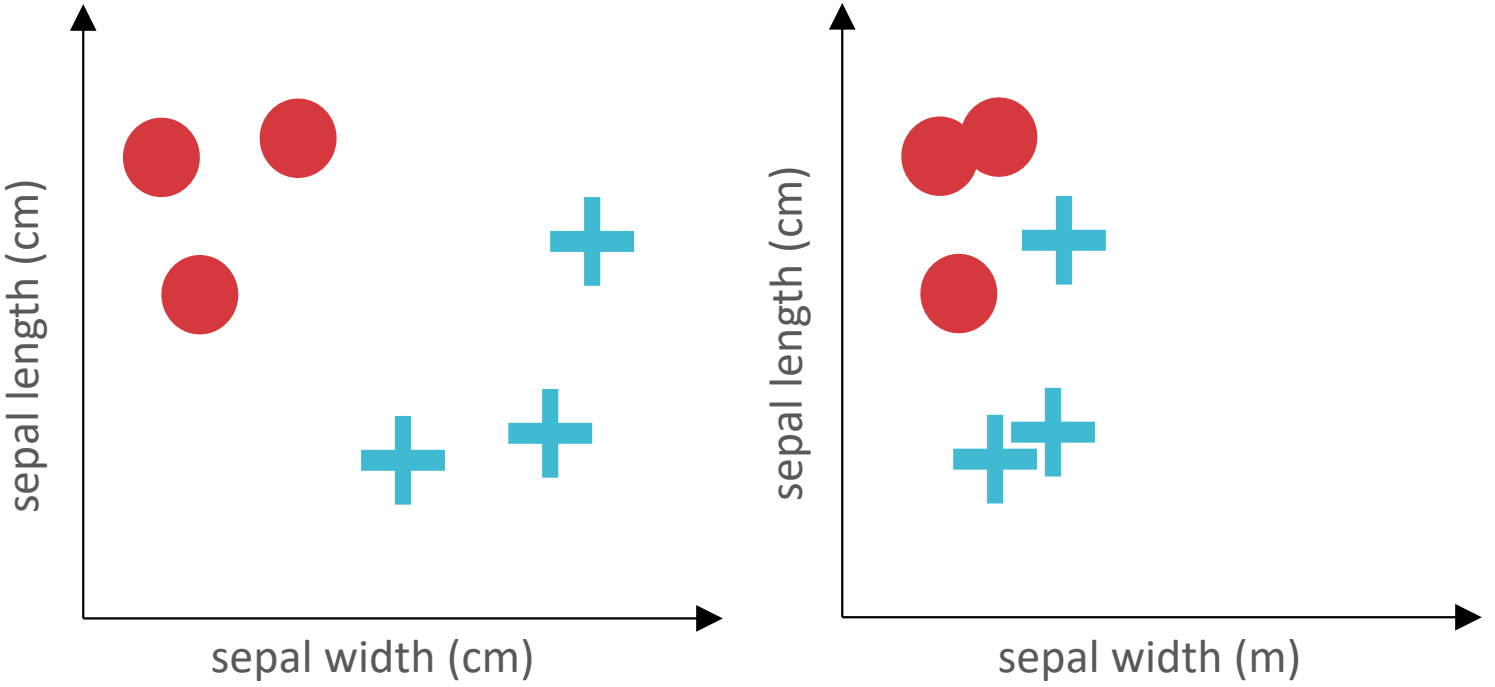


Figure courtesy of Matt Gormley

kNN: Inductive Bias

- Similar points should have similar labels and *all features are equivalently important for determining similarity*



- Feature scale can dramatically influence results!

Figure courtesy of Matt Gormley

kNN: Pros and Cons

- Pros:
  - Intuitive / explainable
  - No training / retraining
  - Provably near-optimal in terms of true error rate
- Cons:
  - Computationally expensive
    - Always needs to store all data: $O(ND)$
    - Finding the k closest points in D dimensions: $O(ND + N\log(k))$
      - Can be sped up through clever use of data structures (trades off training and test costs)
      - Can be approximated using stochastic methods
  - Affected by feature scale

KNN Learning Objectives

You should be able to…

- Describe a dataset as points in a high dimensional space [CIML]
- Implement k-Nearest Neighbors with O(N) prediction
- Describe the inductive bias of a k-NN classifier and relate it to feature scale [a la. CIML]
- Sketch the decision boundary for a learning algorithm (compare k-NN and DT)
- State Cover & Hart (1967)'s large sample analysis of a nearest neighbor classifier
- Invent "new" k-NN learning algorithms capable of dealing with even k

How on earth do we go about setting k?

You should be able to…

- Describe a dataset as points in a high dimensional space [CIML]
- Implement k-Nearest Neighbors with O(N) prediction
- Describe the inductive bias of a k-NN classifier and relate it to feature scale [a la. CIML]
- Sketch the decision boundary for a learning algorithm (compare k-NN and DT)
- State Cover & Hart (1967)'s large sample analysis of a nearest neighbor classifier
- Invent "new" k-NN learning algorithms capable of dealing with even k

How on earth do we go about setting k?

- This is effectively a question of model selection: every value of k corresponds to a different model.

- **WARNING**:
  - In some sense, our discussion of model selection is premature.
  - The models we have considered thus far are fairly simple.
  - The models and the many decisions available to the data scientist wielding them will grow to be much more complex than what we've seen so far.

Model Selection

- Terminology:
  - Model ≈ the hypothesis space in which the learning algorithm searches for a classifier to return
  - Parameters = numeric values or structure selected by the learning algorithm
  - Hyperparameters = tunable aspects of the model that need to be specified before learning can happen, set outside of the training procedure
- Example – Decision Trees:
  - Model = the set of all possible trees, potentially limited by some hyperparameter, e.g., max depth
  - Parameters = structure of a specific tree, i.e., the order in which features are split on
  - Hyperparameters = max depth, splitting criterion, etc…

Model Selection

- Terminology:
  - Model ≈ the hypothesis space in which the learning algorithm searches for a classifier to return
  - Parameters = numeric values or structure selected by the learning algorithm
  - Hyperparameters = tunable aspects of the model that need to be specified before learning can happen, set outside of the training procedure

- Example – kNN:
  - Model = the set of all possible nearest neighbor classifiers
  - Parameters = none! kNN is a non-parametric model
  - Hyperparameters = k

Model Selection vs Hyperparameter Optimization

- Hyperparameter optimization can be considered a special case of model selection
  - Changing the hyperparameters changes the hypothesis space or the set of potential classifiers returned by the learning algorithm
- Deciding between a decision tree and kNN (model selection) vs. selecting a value of k for kNN (hyperparameter optimization)
- Both model selection and hyperparameter optimization happen outside the regular training procedure

Setting k

- When $k = 1$:
  - many, complicated decision boundaries
  - liable to overfit
- When $k = N$:
  - no decision boundaries; always predicts the most common label in the training data (majority vote)
  - liable to underfit
- k controls the complexity of the hypothesis set $\Longrightarrow$ k affects how well the learned hypothesis will generalize

kNN on Gaussian Data

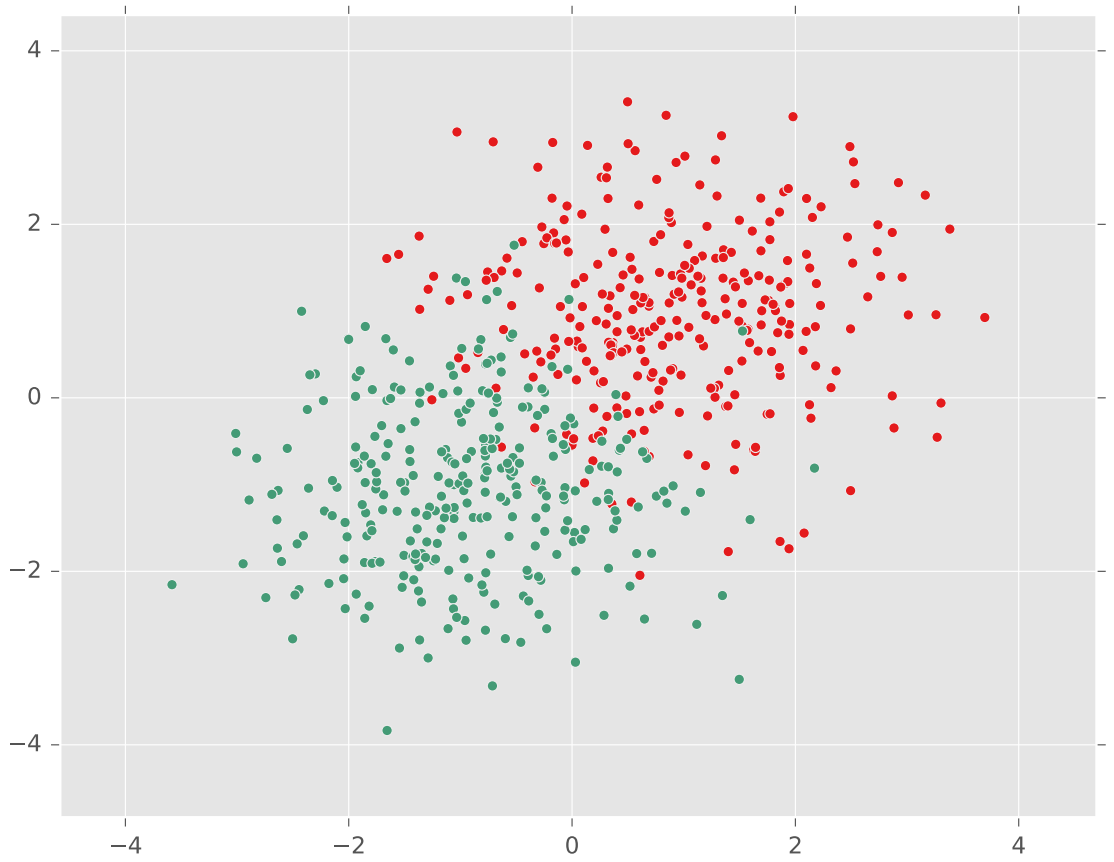


Figure courtesy of Matt Gormley

kNN on Gaussian Data

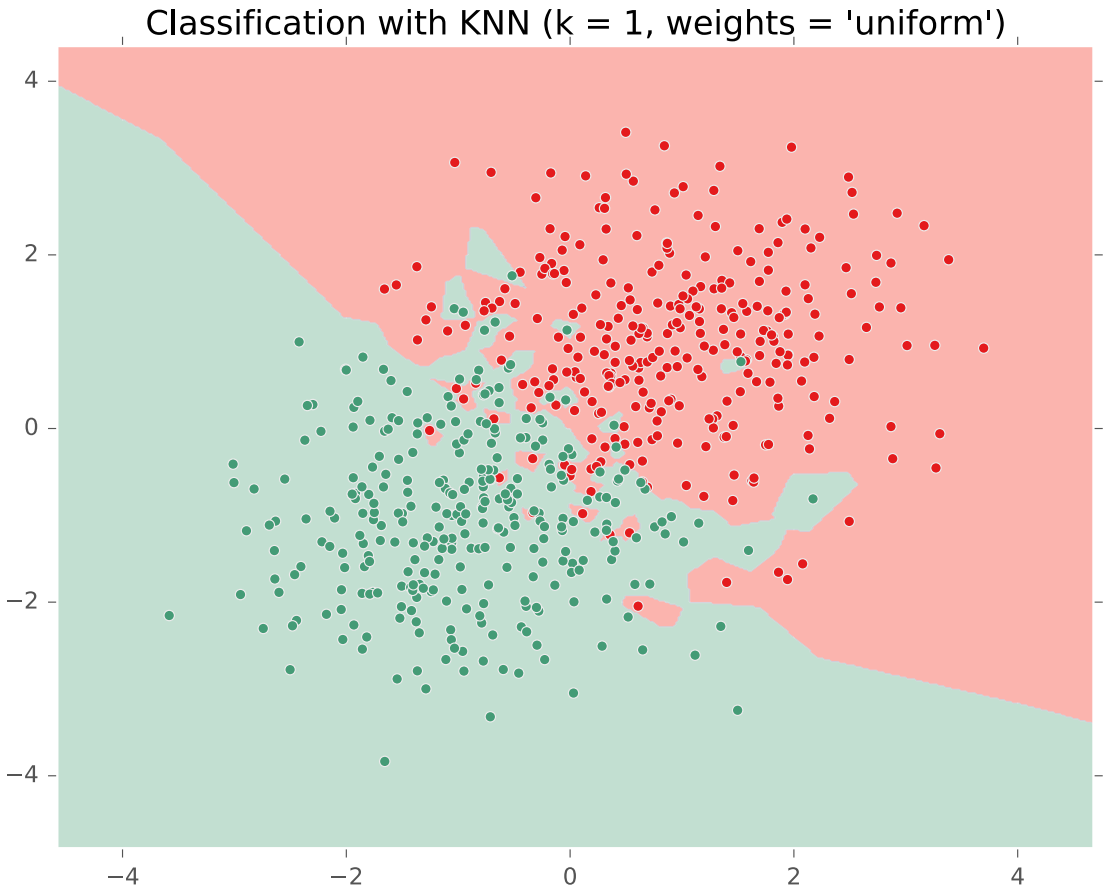


Figure courtesy of Matt Gormley

kNN on Gaussian Data

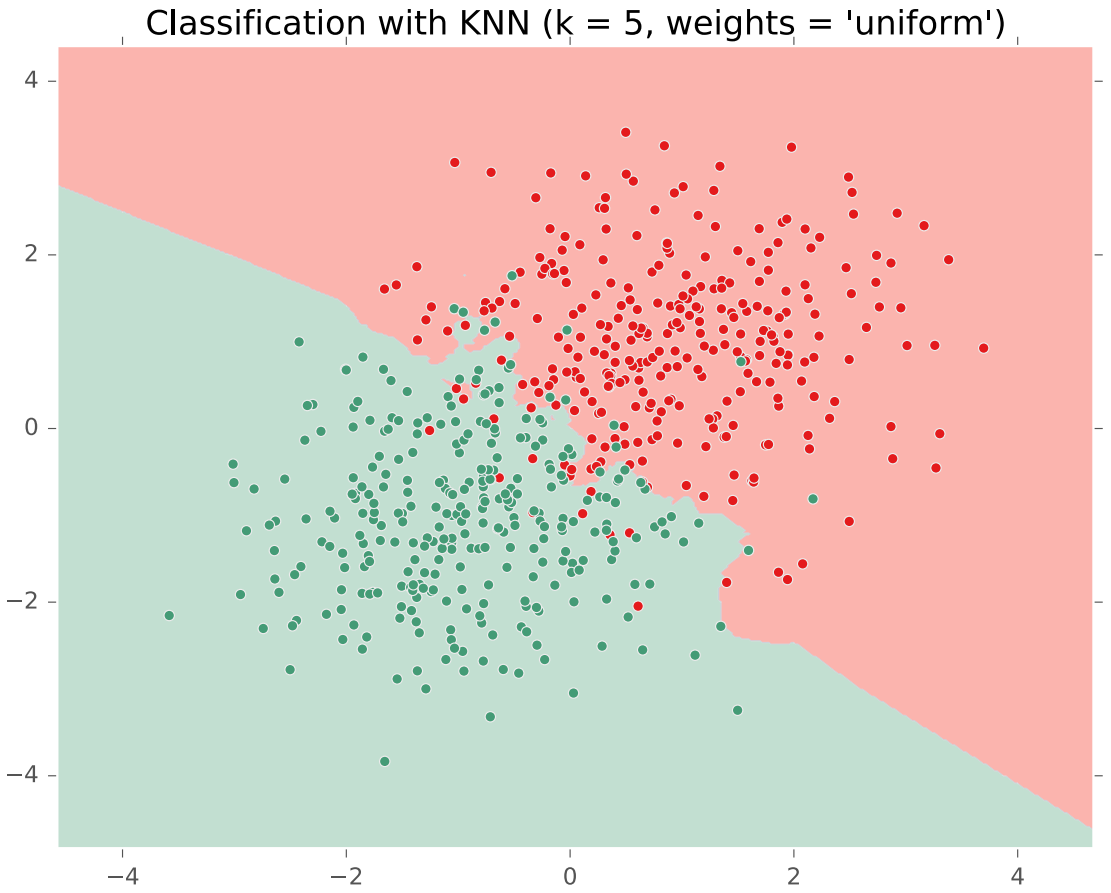


Figure courtesy of Matt Gormley

kNN on Gaussian Data

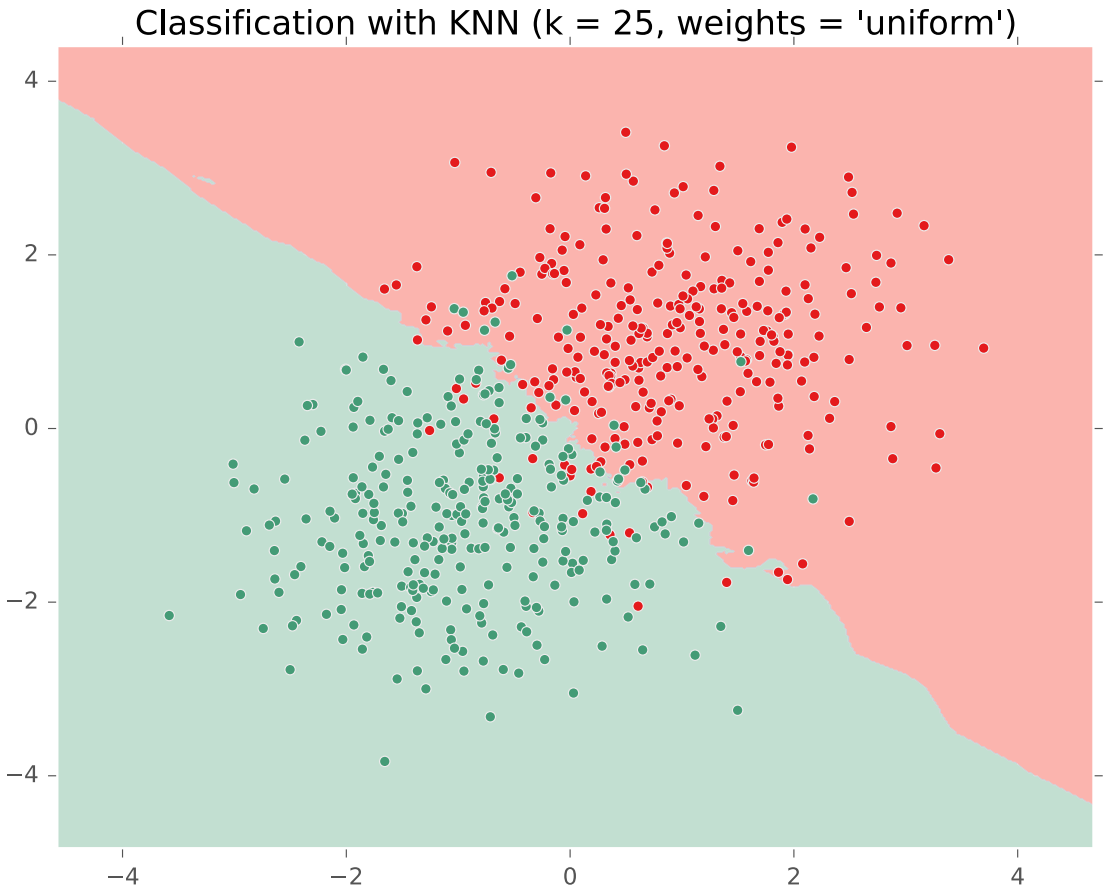


Figure courtesy of Matt Gormley

kNN on Gaussian Data

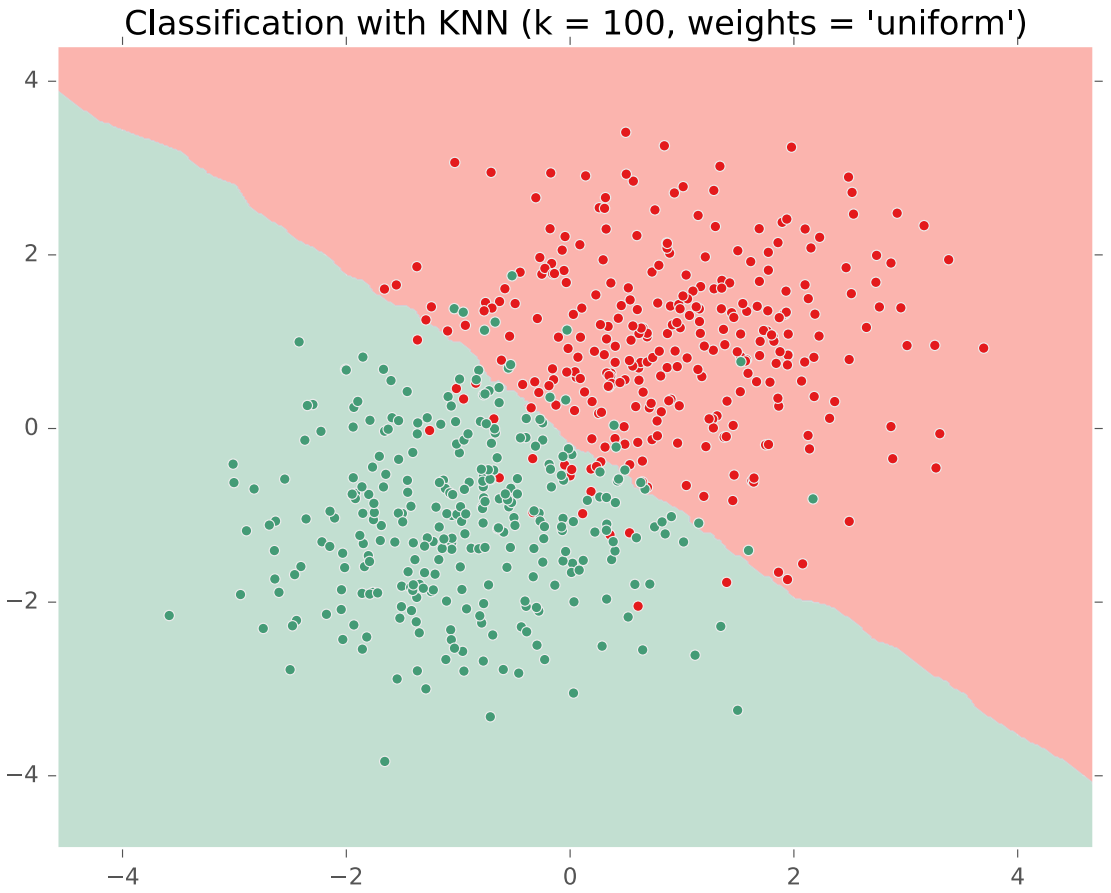


Figure courtesy of Matt Gormley

kNN on Gaussian Data

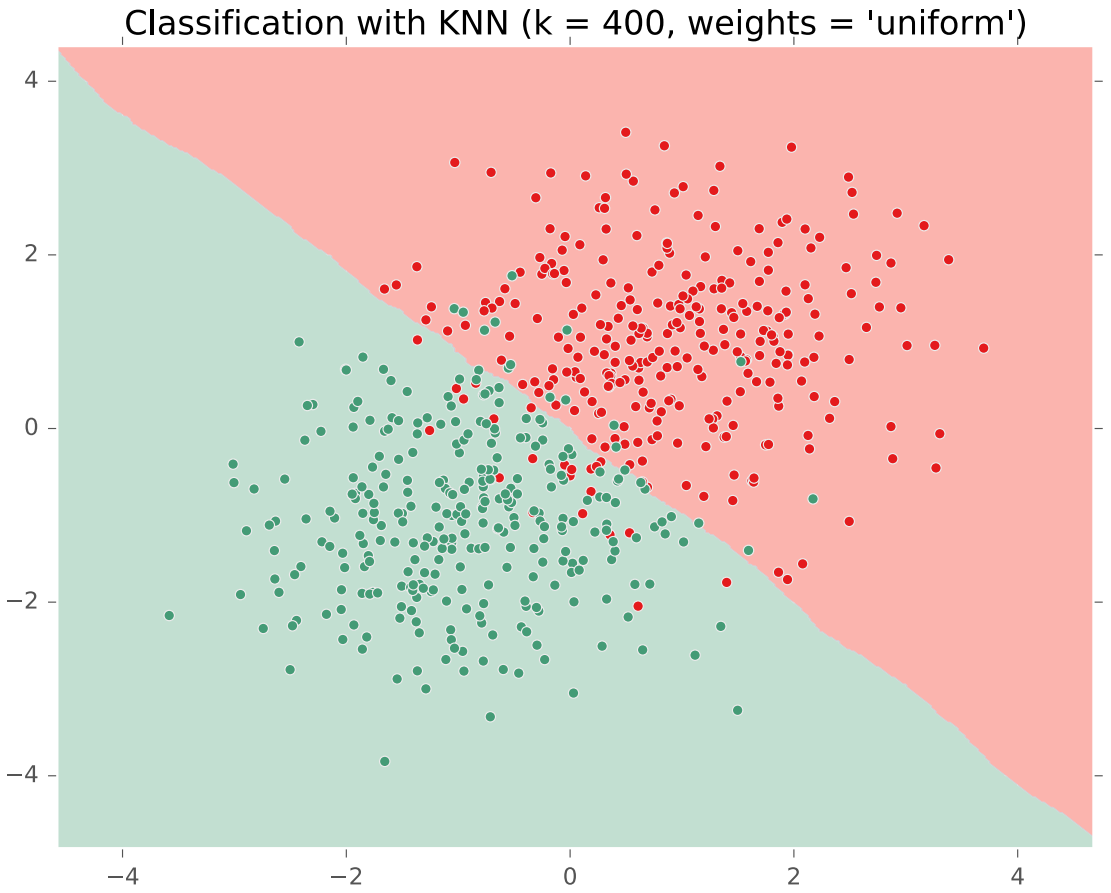


Figure courtesy of Matt Gormley

Setting k

- Theorem:
  - If k is some function of N s.t. $k(N) \to \infty$ and $\frac{k(N)}{N} \to 0$ as $N \to \infty$ …
  - … then (under certain assumptions) the true error of a kNN model → the Bayes error rate
- Practical heuristics:
  - $k = \lfloor \sqrt{N} \rfloor$
  - $k = 3$
- Use a held out "validation" dataset

Model Selection with Test Sets?

- Given $\mathcal{D} = \mathcal{D}_{train} \cup \mathcal{D}_{test}$, suppose we have multiple candidate models:

$$\mathcal{H}_1, \mathcal{H}_2, \ldots, \mathcal{H}_M$$

- Learn a classifier from each model using only $\mathcal{D}_{train}$:

$$h_1 \in \mathcal{H}_1, h_2 \in \mathcal{H}_2, \ldots, h_M \in \mathcal{H}_M$$

- Evaluate each one using $\mathcal{D}_{test}$ and choose the one with lowest test error:

$$\hat{m} = \underset{m \in \{1,\ldots,M\}}{\operatorname{argmin}} \; err(h_m, \mathcal{D}_{test})$$

- Is $err(h_{\hat{m}}, \mathcal{D}_{test})$ a good estimate of $err(h_{\hat{m}})$?

Model Selection with Validation Sets

- Given $\mathcal{D} = \mathcal{D}_{train} \cup \mathcal{D}_{val} \cup \mathcal{D}_{test}$, suppose we have multiple candidate models:

$$\mathcal{H}_1, \mathcal{H}_2, \ldots, \mathcal{H}_M$$

- Learn a classifier from each model using only $\mathcal{D}_{train}$:

$$h_1 \in \mathcal{H}_1, h_2 \in \mathcal{H}_2, \ldots, h_M \in \mathcal{H}_M$$

- Evaluate each one using $\mathcal{D}_{val}$ and choose the one with lowest *validation* error:

$$\hat{m} = \underset{m \in \{1,\ldots,M\}}{\operatorname{argmin}} \; err(h_m, \mathcal{D}_{val})$$

- Now $err(h_{\hat{m}}, \mathcal{D}_{test})$ is a good estimate of $err(h_{\hat{m}})$!

Hyperparameter Optimization with Validation Sets

- Given $\mathcal{D} = \mathcal{D}_{train} \cup \mathcal{D}_{val} \cup \mathcal{D}_{test}$, suppose we have multiple candidate hyperparameter settings:

$$\theta_1, \theta_2, \ldots, \theta_M$$

- Learn a classifier for each setting using only $\mathcal{D}_{train}$:

$$h_1, h_2, \ldots, h_M$$

- Evaluate each one using $\mathcal{D}_{val}$ and choose the one with lowest *validation* error:

$$\hat{m} = \underset{m \in \{1,\ldots,M\}}{\operatorname{argmin}} \; err(h_m, \mathcal{D}_{val})$$

- Now $err(h_{\hat{m}}, \mathcal{D}_{test})$ is a good estimate of $err(h_{\hat{m}})$!

Setting k for kNN with Validation Sets

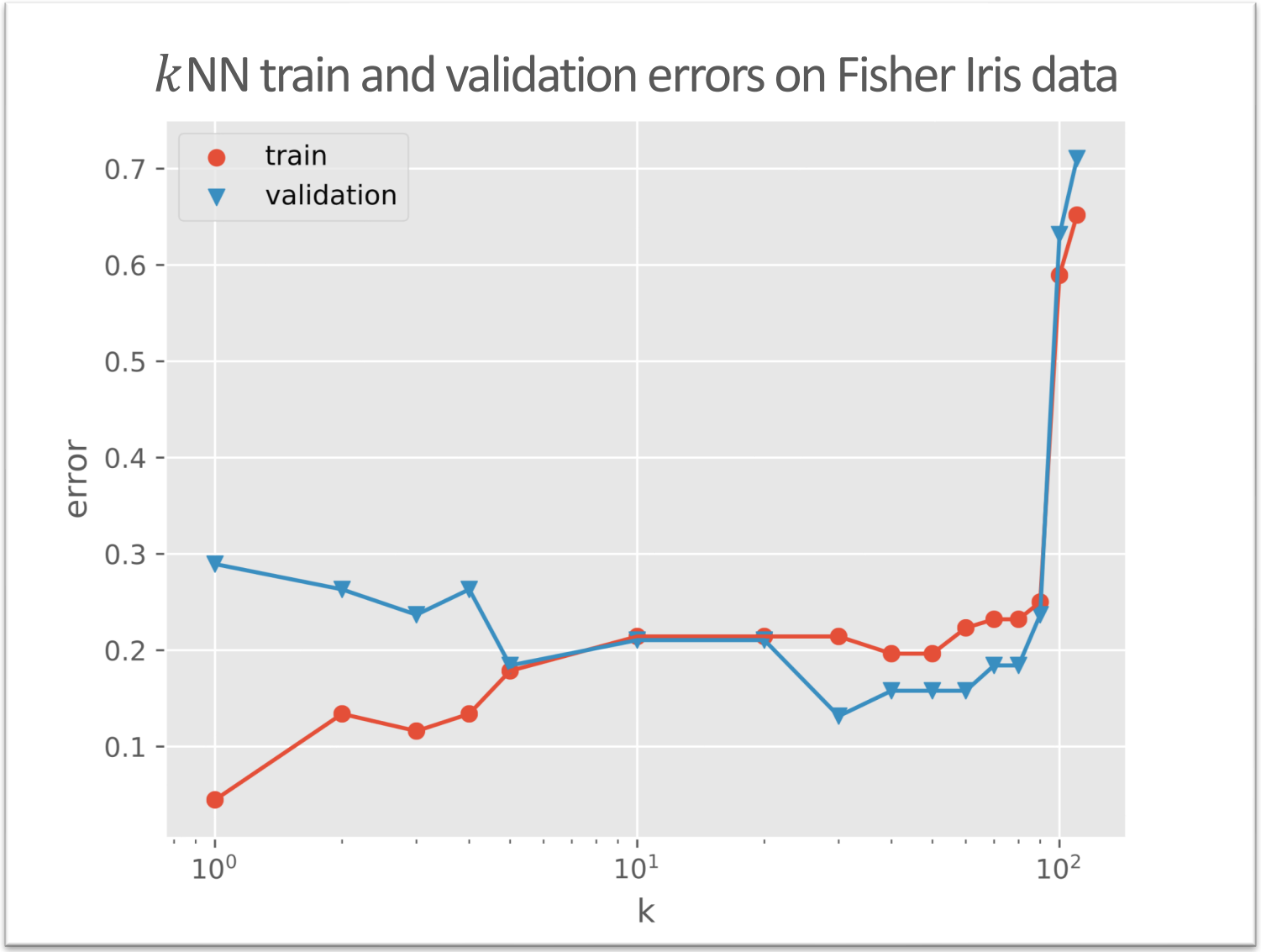


Figure courtesy of Matt Gormley

How should we partition our dataset?

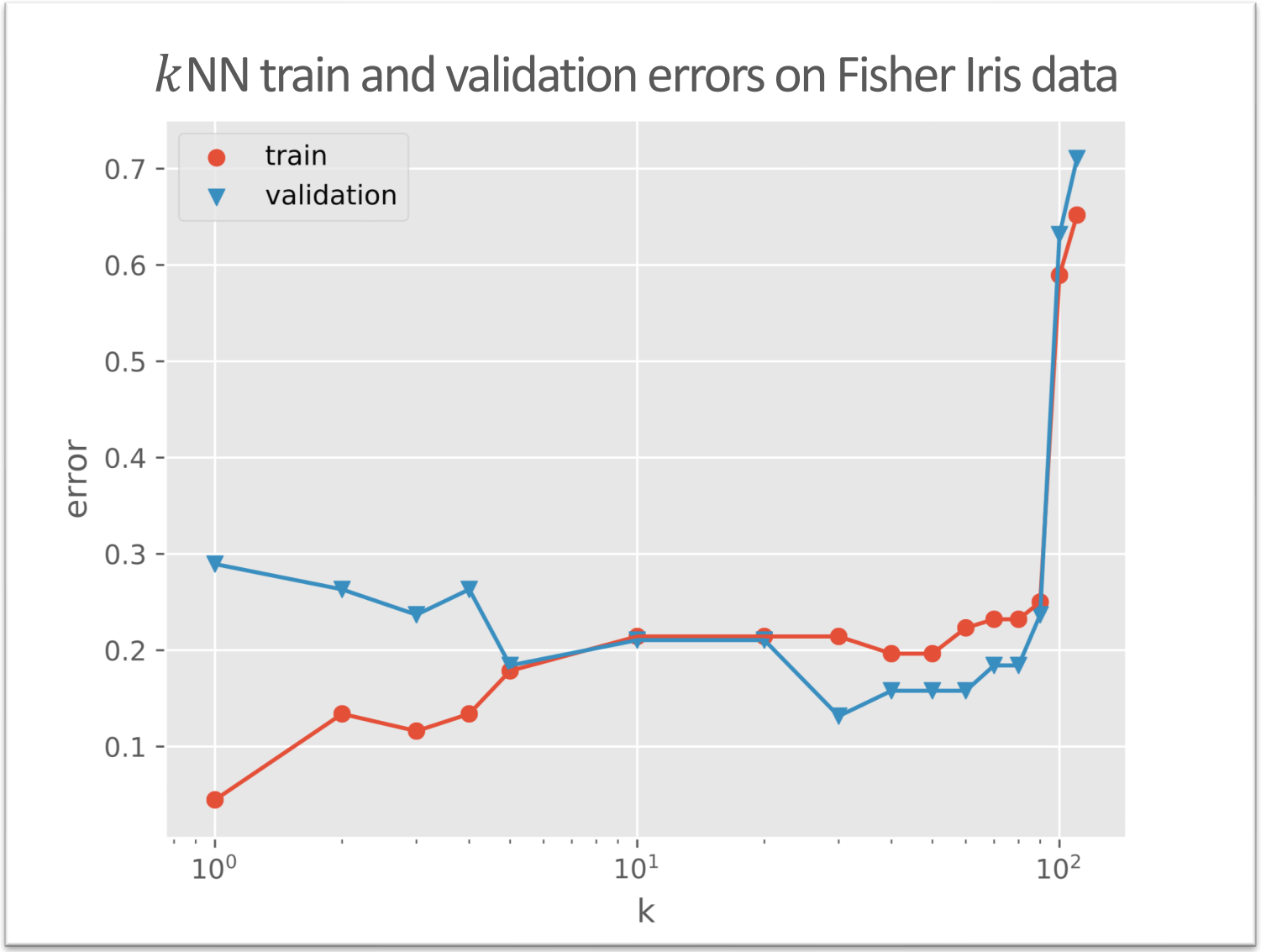
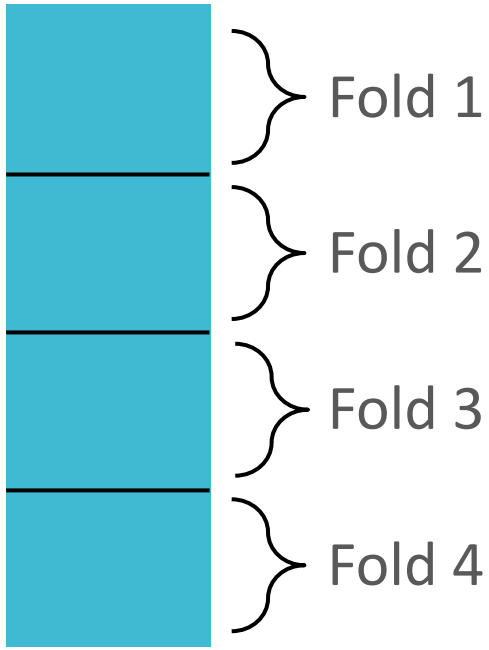


Figure courtesy of Matt Gormley

K-fold cross-validation

- Given $\mathcal{D}$, split $\mathcal{D}$ into K equally sized datasets or folds: $\mathcal{D}_1, \mathcal{D}_2, \ldots, \mathcal{D}_K$
- Use each one as a validation set once:

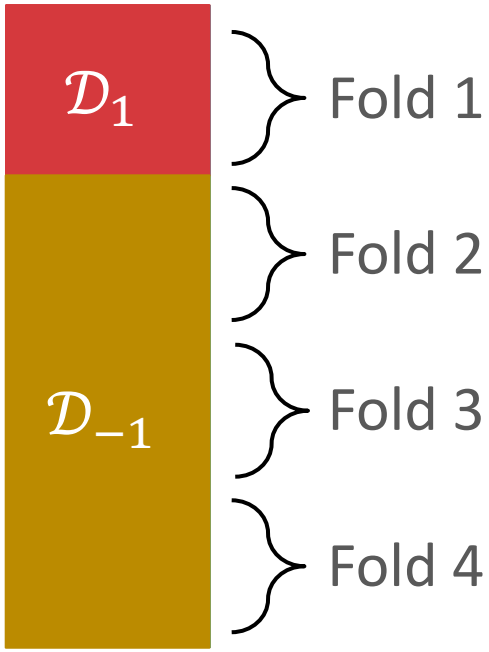


- Let h_{-i} be the classifier learned using $\mathcal{D}_{-i} = \mathcal{D} \backslash \mathcal{D}_i$ (all folds other than $\mathcal{D}_i$) and let $e_i = err(h_{-i}, \mathcal{D}_i)$
- The K-fold cross validation error is

$$err_{cv_K} = \frac{1}{K} \sum_{i=1}^{K} e_i$$

K-fold cross-validation

- Given $\mathcal{D}$, split $\mathcal{D}$ into K equally sized datasets or folds: $\mathcal{D}_1, \mathcal{D}_2, \ldots, \mathcal{D}_K$
- Use each one as a validation set once:

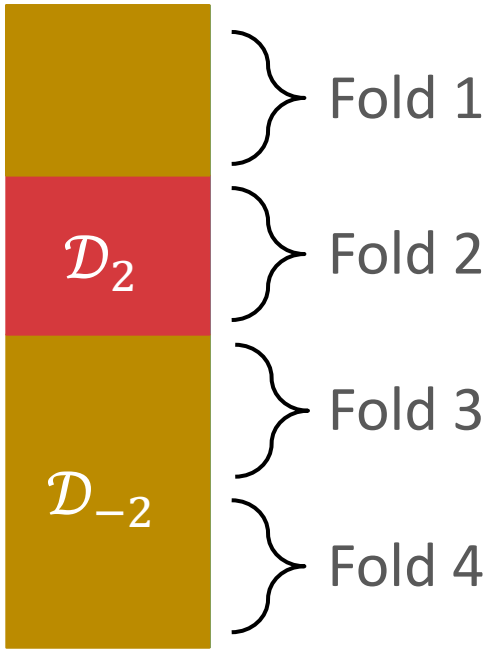


- Let h_{-i} be the classifier learned using $\mathcal{D}_{-i} = \mathcal{D} \backslash \mathcal{D}_i$ (all folds other than $\mathcal{D}_i$) and let $e_i = err(h_{-i}, \mathcal{D}_i)$
- The K-fold cross validation error is

$$err_{cv_K} = \frac{1}{K} \sum_{i=1}^{K} e_i$$

K-fold cross-validation

- Given $\mathcal{D}$, split $\mathcal{D}$ into K equally sized datasets or folds: $\mathcal{D}_1, \mathcal{D}_2, \ldots, \mathcal{D}_K$
- Use each one as a validation set once:

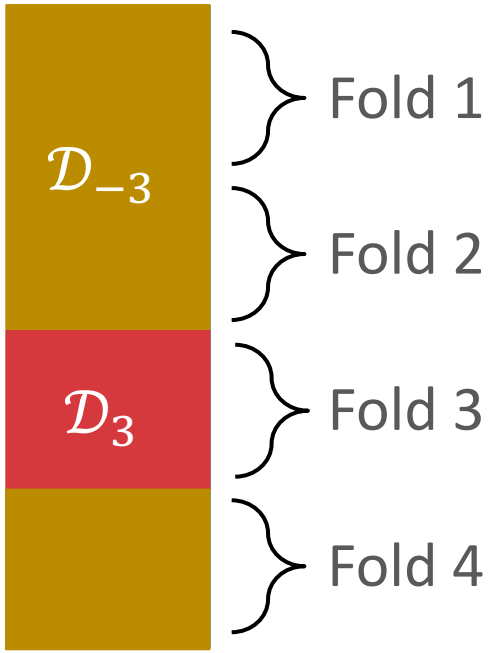


- Let h_{-i} be the classifier learned using $\mathcal{D}_{-i} = \mathcal{D} \backslash \mathcal{D}_i$ (all folds other than $\mathcal{D}_i$) and let $e_i = err(h_{-i}, \mathcal{D}_i)$
- The K-fold cross validation error is

$$err_{cv_K} = \frac{1}{K} \sum_{i=1}^{K} e_i$$

K-fold cross-validation

- Given $\mathcal{D}$, split $\mathcal{D}$ into K equally sized datasets or folds: $\mathcal{D}_1, \mathcal{D}_2, \ldots, \mathcal{D}_K$
- Use each one as a validation set once:

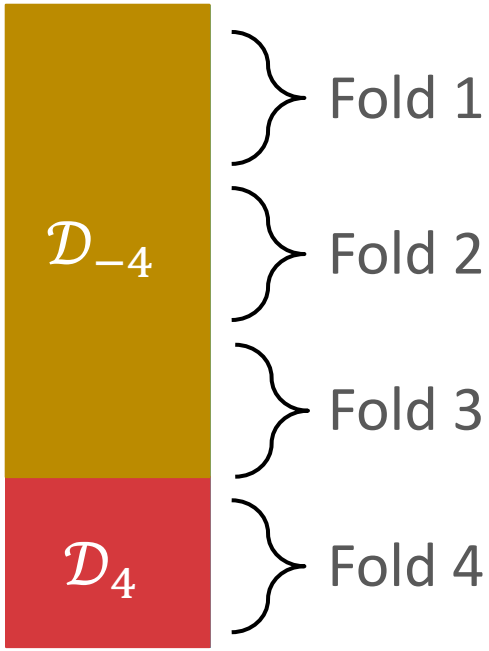


- Let h_{-i} be the classifier learned using $\mathcal{D}_{-i} = \mathcal{D} \backslash \mathcal{D}_i$ (all folds other than $\mathcal{D}_i$) and let $e_i = err(h_{-i}, \mathcal{D}_i)$
- The K-fold cross validation error is

$$err_{cv_K} = \frac{1}{K} \sum_{i=1}^{K} e_i$$

K-fold cross-validation

- Given $\mathcal{D}$, split $\mathcal{D}$ into K equally sized datasets or folds: $\mathcal{D}_1, \mathcal{D}_2, \ldots, \mathcal{D}_K$
- Use each one as a validation set once:

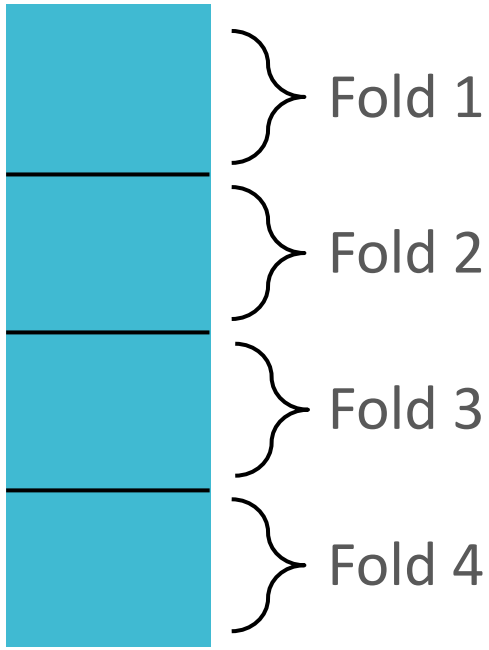


- Let h_{-i} be the classifier learned using $\mathcal{D}_{-i} = \mathcal{D} \backslash \mathcal{D}_i$ (all folds other than $\mathcal{D}_i$) and let $e_i = err(h_{-i}, \mathcal{D}_i)$
- The K-fold cross validation error is

$$err_{cv_K} = \frac{1}{K} \sum_{i=1}^{K} e_i$$

K-fold cross-validation

- Given $\mathcal{D}$, split $\mathcal{D}$ into K equally sized datasets or folds: $\mathcal{D}_1, \mathcal{D}_2, \ldots, \mathcal{D}_K$
- Use each one as a validation set once:



- Let h_{-i} be the classifier learned using $\mathcal{D}_{-i} = \mathcal{D} \backslash \mathcal{D}_i$ (all folds other than $\mathcal{D}_i$) and let $e_i = err(h_{-i}, \mathcal{D}_i)$
- The K-fold cross validation error is

$$err_{cv_K} = \frac{1}{K} \sum_{i=1}^{K} e_i$$

- Special case when $K = N$: Leave-one-out cross-validation
- Choosing between m candidates requires training mK times

Summary

	Input	Output
Training	• training dataset • hyperparameters	• best model parameters
Hyperparameter Optimization	• training dataset • validation dataset	• best hyperparameters
Cross-Validation	• training dataset • validation dataset	• cross-validation error
Testing	• test dataset • classifier	• test error

Hyperparameter Optimization

- Given $\mathcal{D} = \mathcal{D}_{train} \cup \mathcal{D}_{val} \cup \mathcal{D}_{test}$, suppose we have multiple candidate hyperparameter settings:

$$\theta_1, \theta_2, \ldots, \theta_M$$

- Learn a classifier for each setting using only $\mathcal{D}_{train}$:

$$h_1, h_2, \ldots, h_M$$

- Evaluate each one using $\mathcal{D}_{val}$ and choose the one with lowest *validation* error:

$$\hat{m} = \underset{m \in \{1,\ldots,M\}}{\operatorname{argmin}} \; err(h_m, \mathcal{D}_{val})$$

- Now $err(h_{\hat{m}}, \mathcal{D}_{test})$ is a good estimate of $err(h_{\hat{m}})$!

How do we pick hyperparameter settings to try?

- Given $\mathcal{D} = \mathcal{D}_{train} \cup \mathcal{D}_{val} \cup \mathcal{D}_{test}$, suppose we have multiple candidate hyperparameter settings:

$$\theta_1, \theta_2, \ldots, \theta_M$$

- Learn a classifier for each setting using only $\mathcal{D}_{train}$:

$$h_1, h_2, \ldots, h_M$$

- Evaluate each one using $\mathcal{D}_{val}$ and choose the one with lowest *validation* error:

$$\hat{m} = \underset{m \in \{1,\ldots,M\}}{\operatorname{argmin}} \; err(h_m, \mathcal{D}_{val})$$

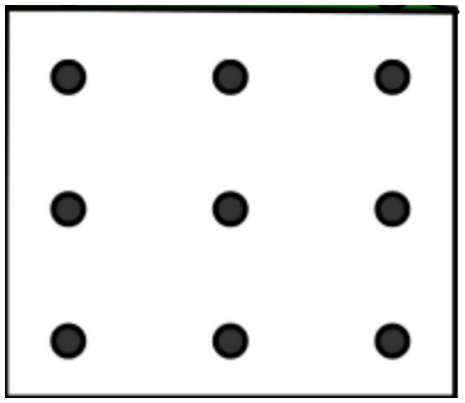
- Now $err(h_{\hat{m}}, \mathcal{D}_{test})$ is a good estimate of $err(h_{\hat{m}})$!

General Methods for Hyperparameter Optimization

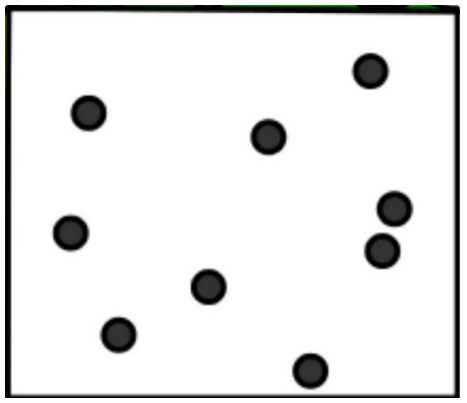
- Idea: set the hyperparameters to optimize some performance metric of the model
- Issue: if we have many hyperparameters that can all take on lots of different values, we might not be able to test all possible combinations
- Commonly used methods:
  - Grid search
  - Random search
  - Bayesian optimization (used by Google DeepMind to optimize the hyperparameters of AlphaGo: https://arxiv.org/pdf/1812.06855v1.pdf)
  - Evolutionary algorithms
  - Graduate-student descent

Grid Search vs. Random Search (Bergstra and Bengio, 2012)

Grid Layout



Random Layout

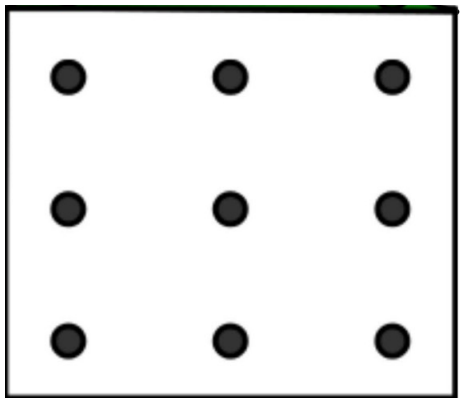


Source: https://jmlr.csail.mit.edu/papers/volume13/bergstra12a/bergstra12a.pdf

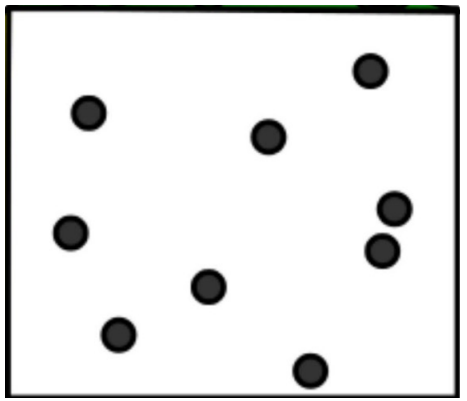
Poll Question 3:

Which hyperparameter optimization method do you think will perform better?

Grid Layout



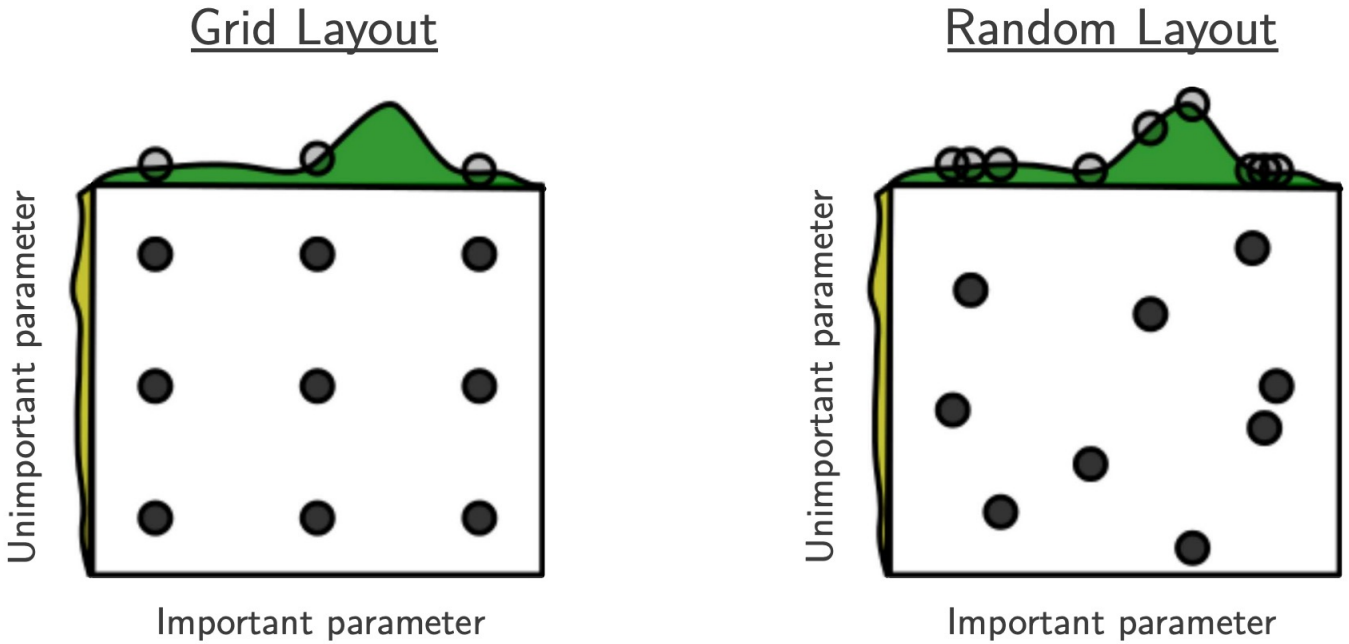
Random Layout



A. Graduate student descent
B. Grid search
C. Random search

Source: https://jmlr.csail.mit.edu/papers/volume13/bergstra12a/bergstra12a.pdf

Grid Search vs. Random Search (Bergstra and Bengio, 2012)



Grid and random search of nine trials for optimizing a function $f(x, y) = g(x) + h(y) \approx g(x)$ with *low effective dimensionality*. Above each square $g(x)$ is shown in green, and left of each square $h(y)$ is shown in yellow. With grid search, nine trials only test $g(x)$ in three distinct places. With random search, all nine trials explore distinct values of g. This failure of grid search is the rule rather than the exception in high dimensional hyper-parameter optimization.

Source: https://jmlr.csail.mit.edu/papers/volume13/bergstra12a/bergstra12a.pdf

Model Selection Learning Objectives

You should be able to…

- Plan an experiment that uses training, validation, and test datasets to predict the performance of a classifier on unseen data (without cheating)
- Explain the difference between (1) training error, (2) validation error, (3) cross-validation error, (4) test error, and (5) true error
- For a given learning technique, identify the model, learning algorithm, parameters, and hyperparamters
- Select an appropriate algorithm for optimizing (aka. learning) hyperparameters