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

Henry Chai & Matt Gormley

9/13/23

Front Matter

- Announcements:
  - HW2 released 9/6, due 9/15 (Friday!) at 11:59 PM
  - HW3 will be released on 9/15, due 9/23 at 11:59 PM
    - HW3 is a written-only homework
    - **You may only use at most 2 late days on HW3**
  - **Important scheduling note:** we will have lecture on 9/15 (Friday!) in lieu of recitation
    - This is to ensure that we cover enough material for you all to make a meaningful start on HW3
    - The HW3 recitation has been moved to 9/20 (next Wednesday)

Recall: Nearest Neighbor Pseudocode

def train($\mathcal{D}$):

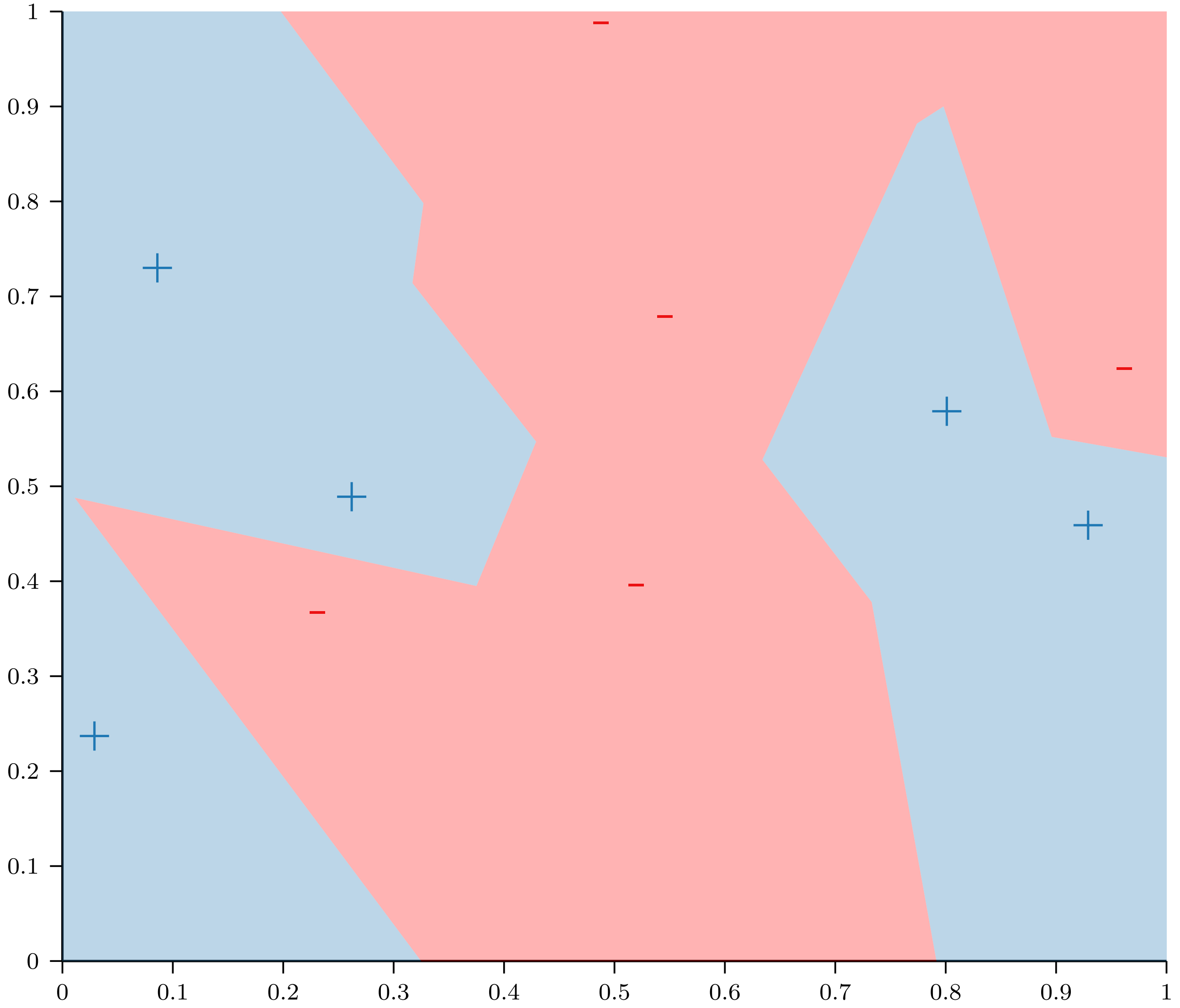
    store $\mathcal{D}$

def predict($\boldsymbol{x}'$):

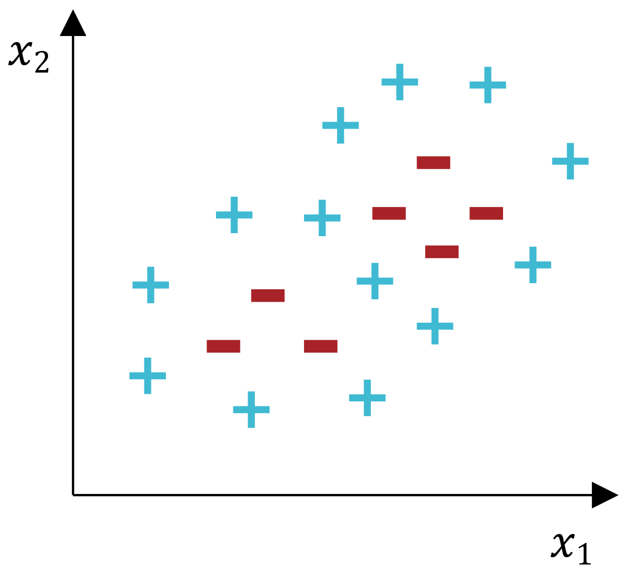
    find the nearest neighbor to $\boldsymbol{x}'$ in $\mathcal{D}$, $\boldsymbol{x}^{(i)}$

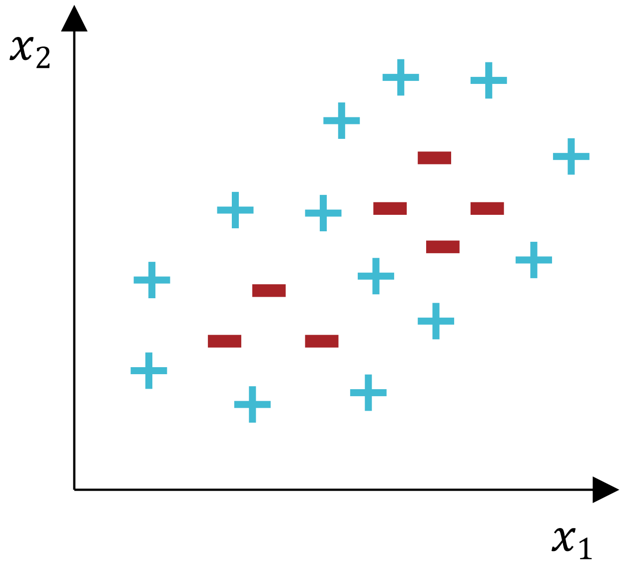
    return $y^{(i)}$

Recall: Nearest Neighbor Decision Boundary



Decision Boundary Exercise





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

Recall: 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)
```

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)

kNN on Fisher Iris Data

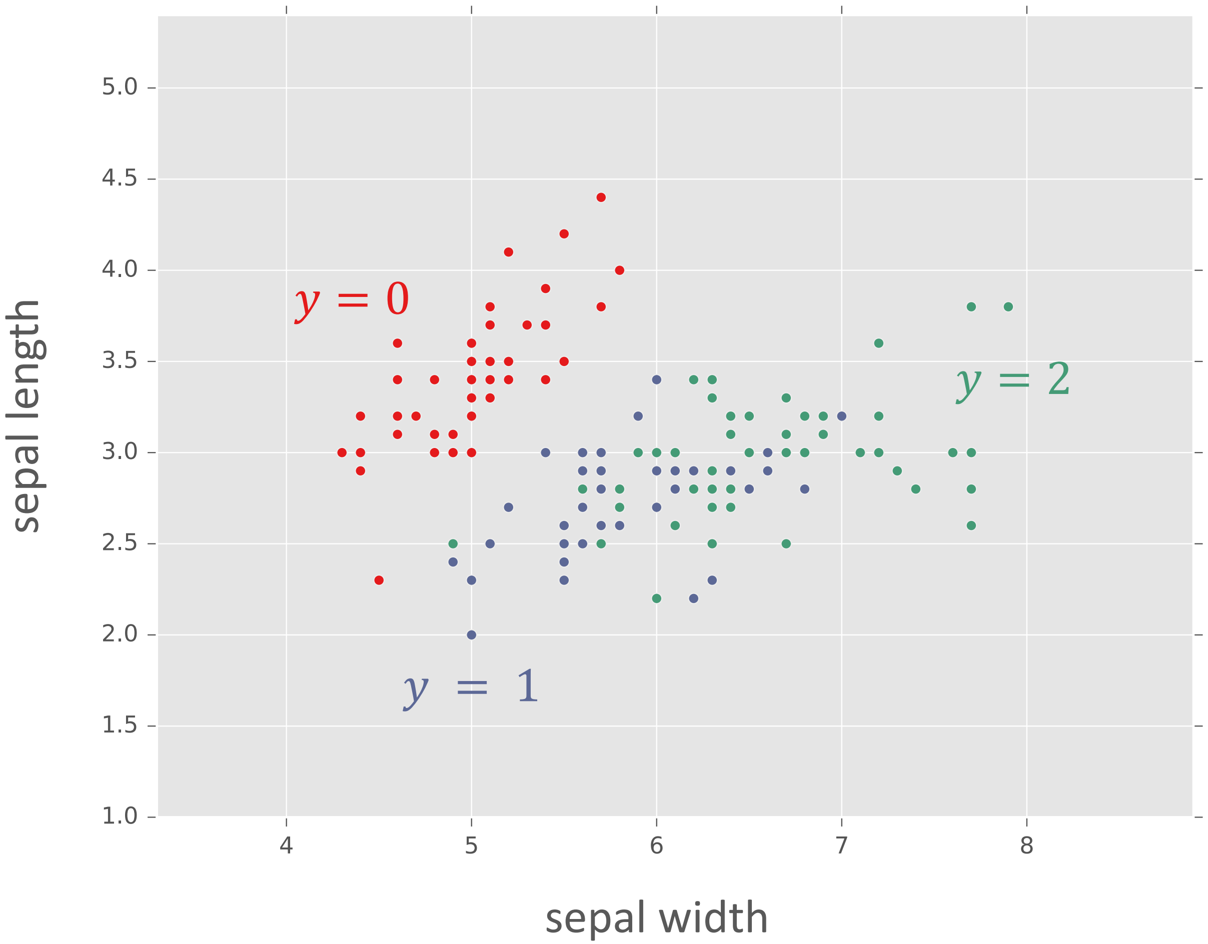


Figure courtesy of Matt Gormley

kNN on Fisher Iris Data

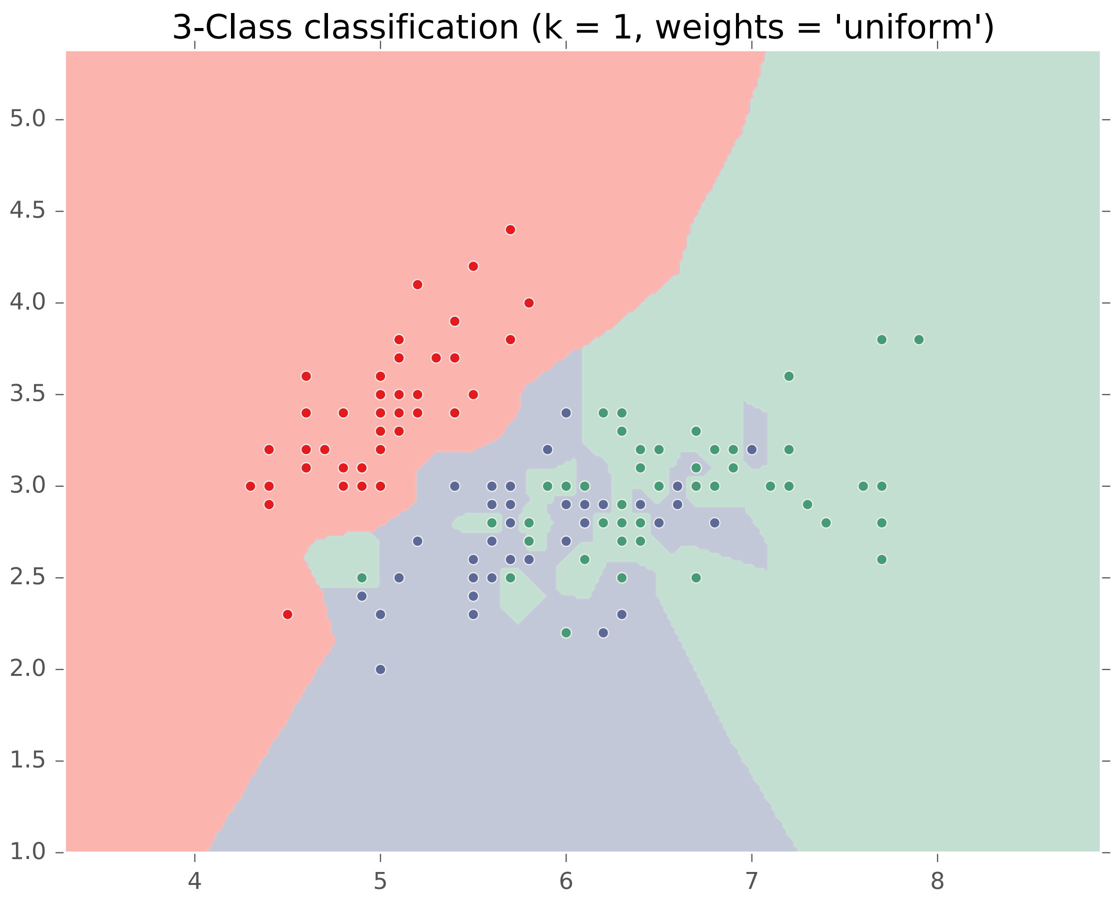


Figure courtesy of Matt Gormley

kNN on Fisher Iris Data

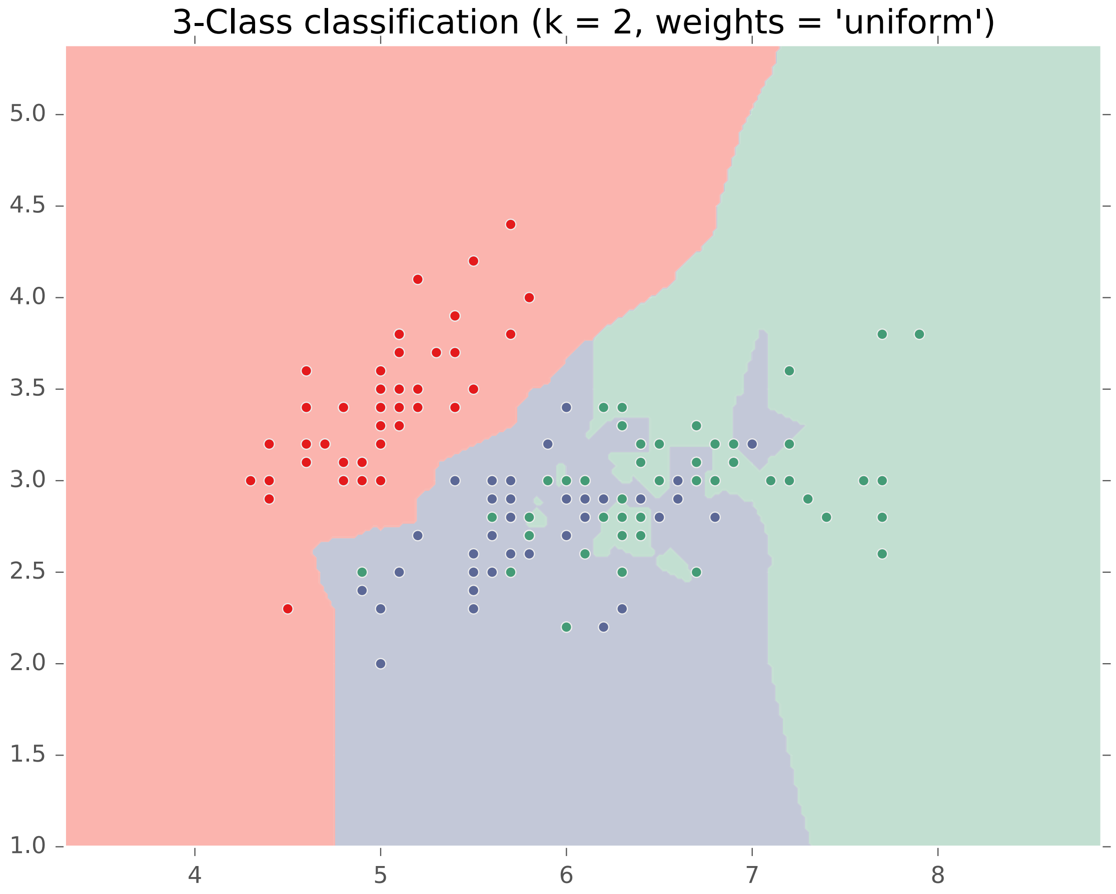


Figure courtesy of Matt Gormley

kNN on Fisher Iris Data

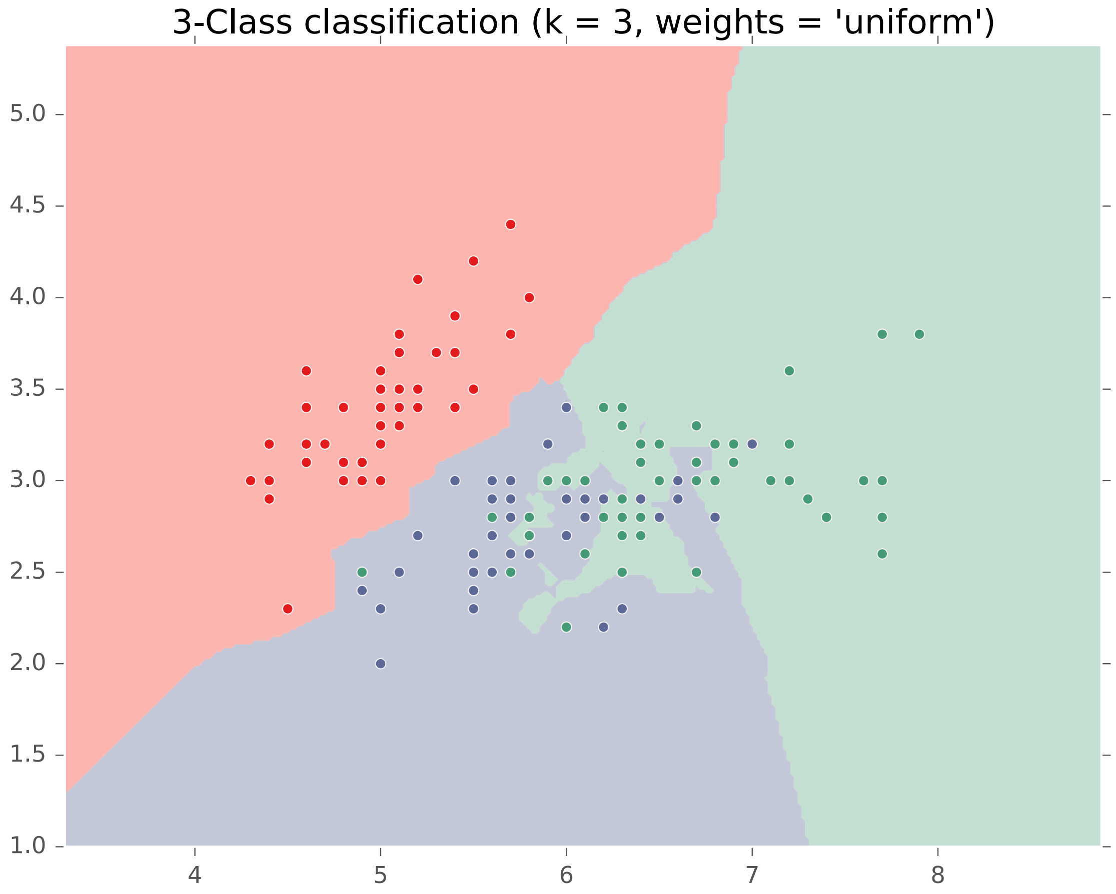


Figure courtesy of Matt Gormley

kNN on Fisher Iris Data

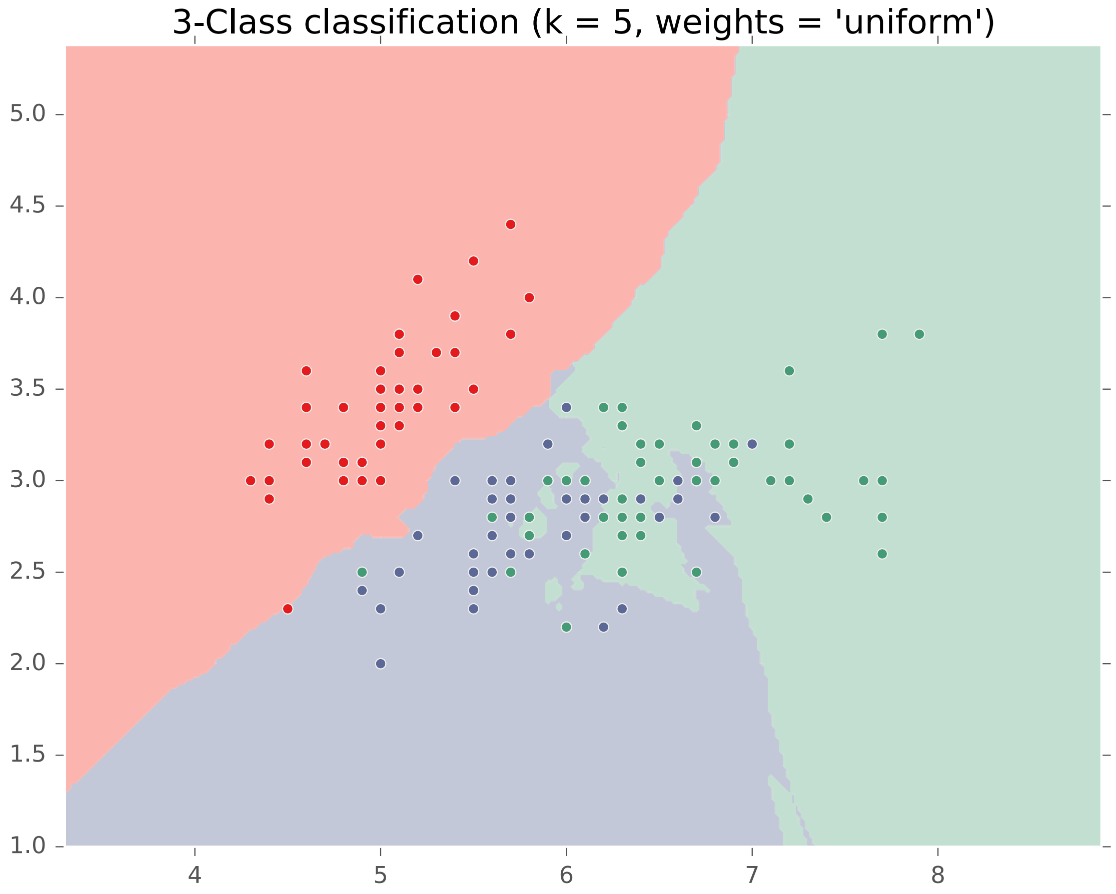


Figure courtesy of Matt Gormley

kNN on Fisher Iris Data

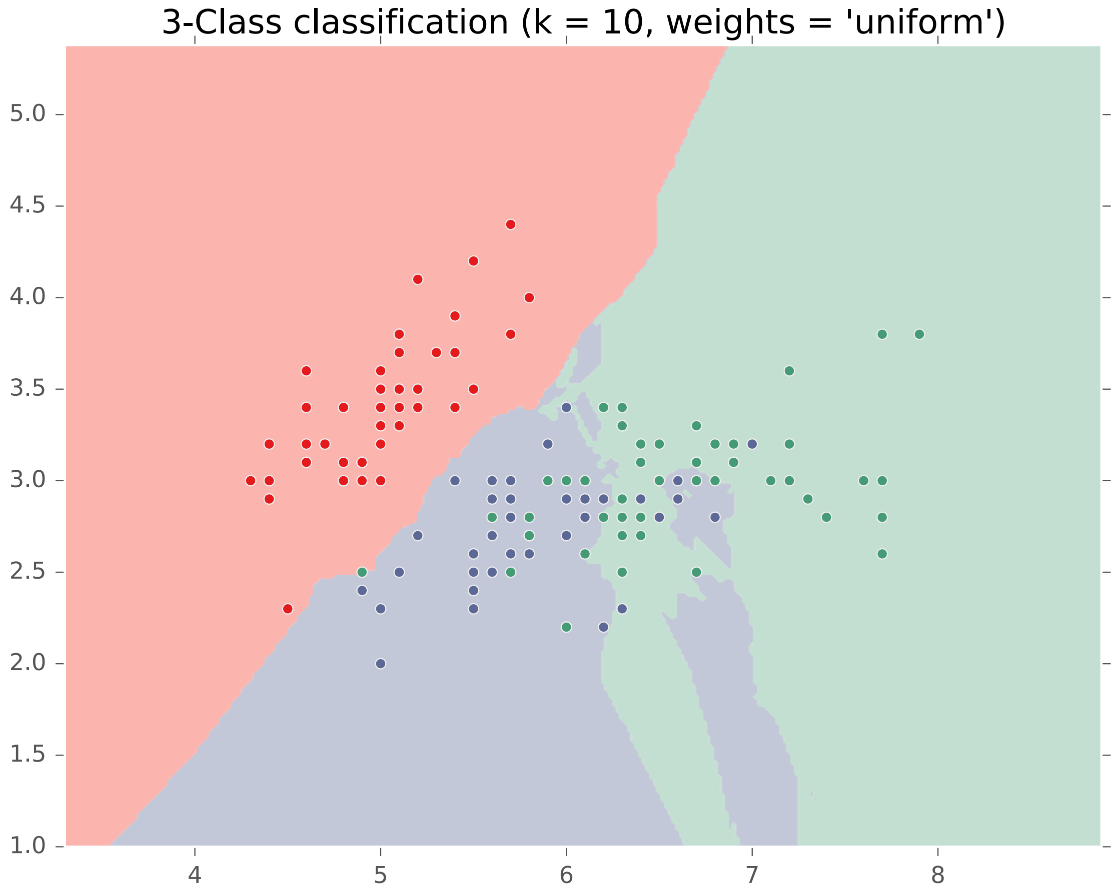


Figure courtesy of Matt Gormley

kNN on Fisher Iris Data

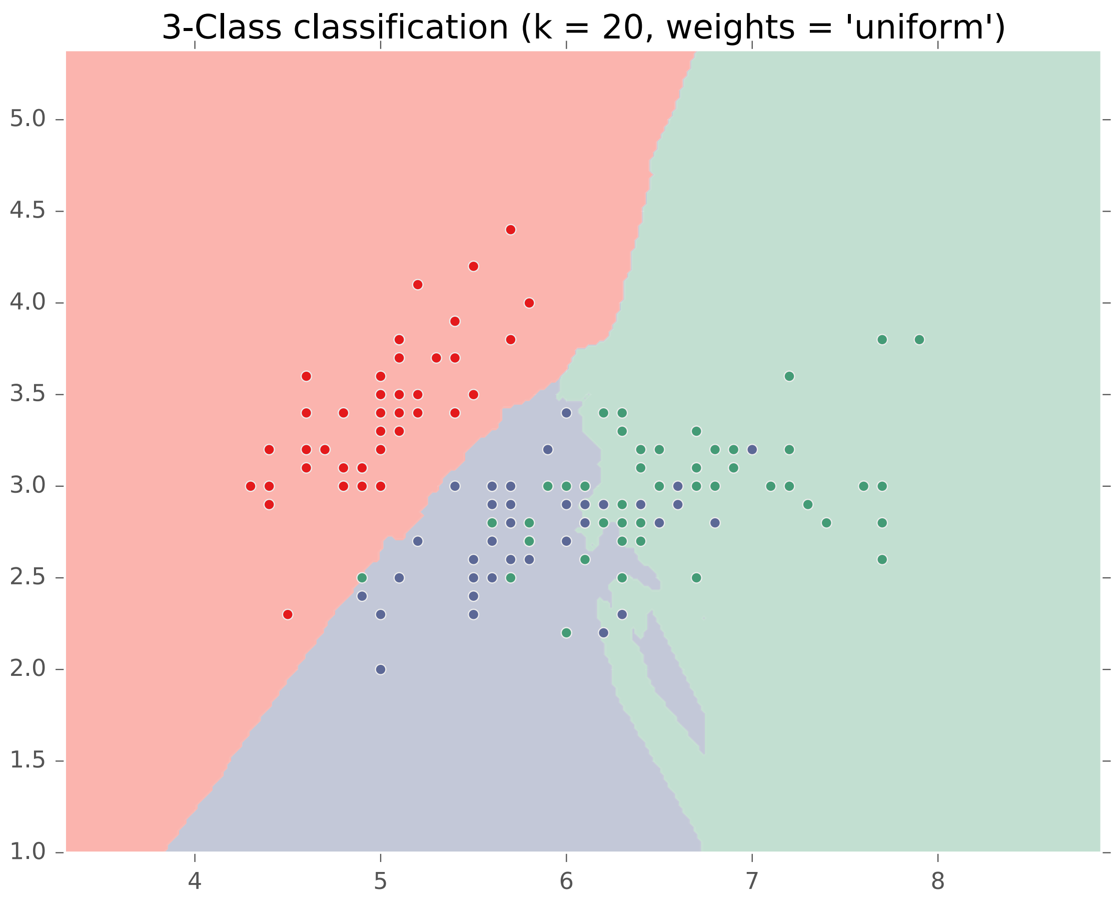


Figure courtesy of Matt Gormley

kNN on Fisher Iris Data

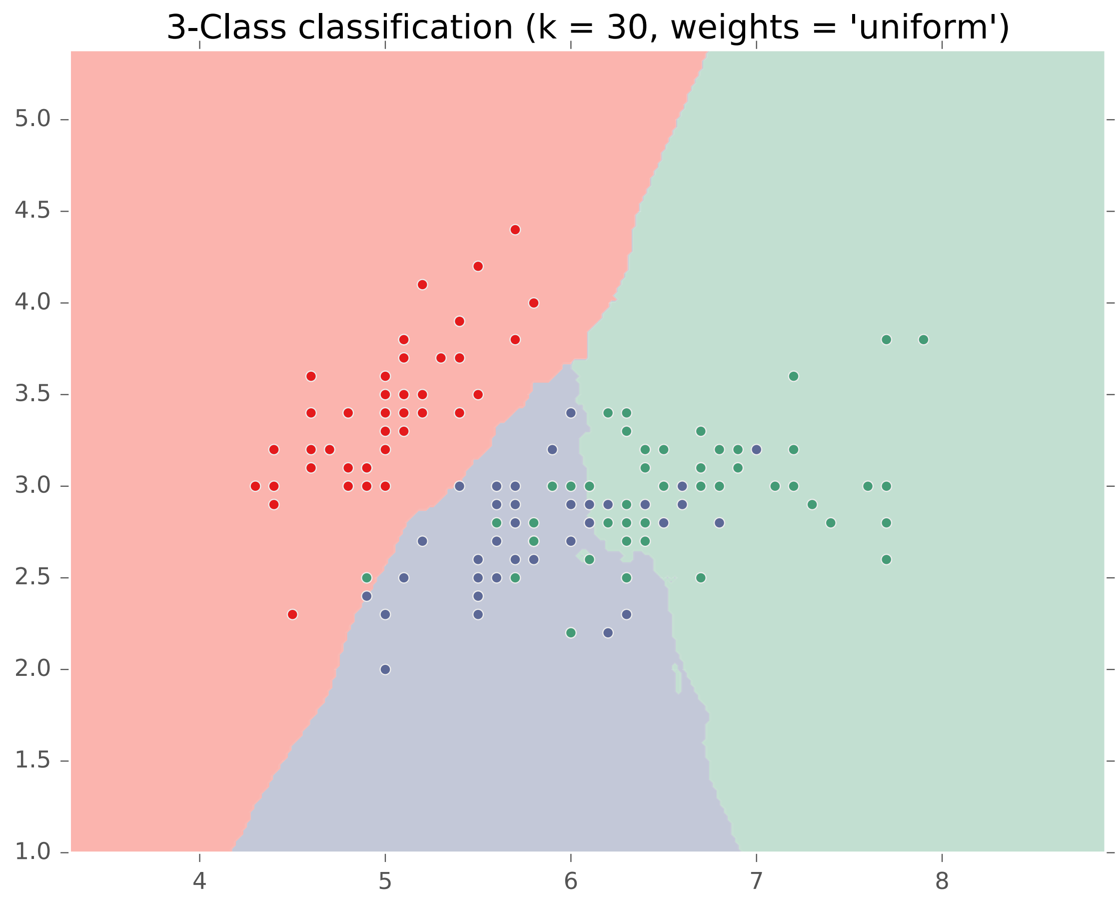


Figure courtesy of Matt Gormley

kNN on Fisher Iris Data

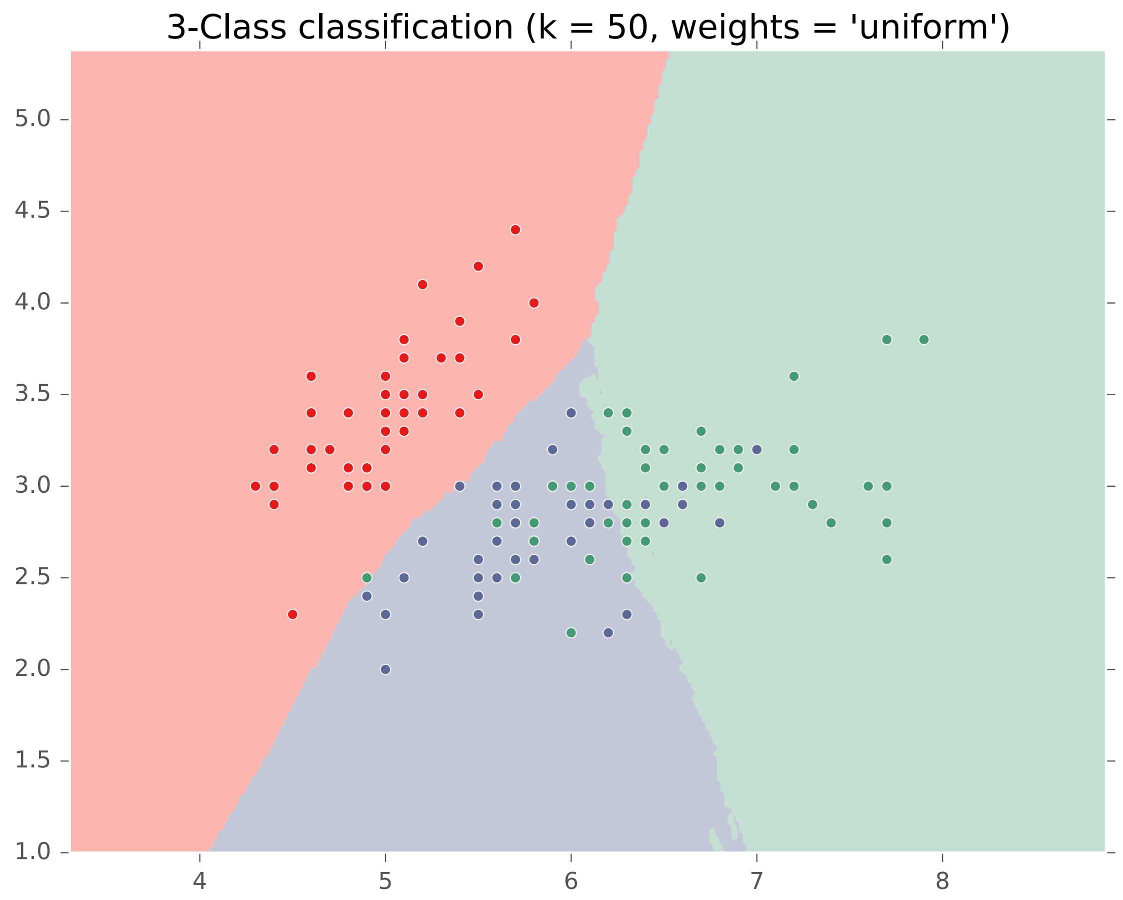


Figure courtesy of Matt Gormley

kNN on Fisher Iris Data

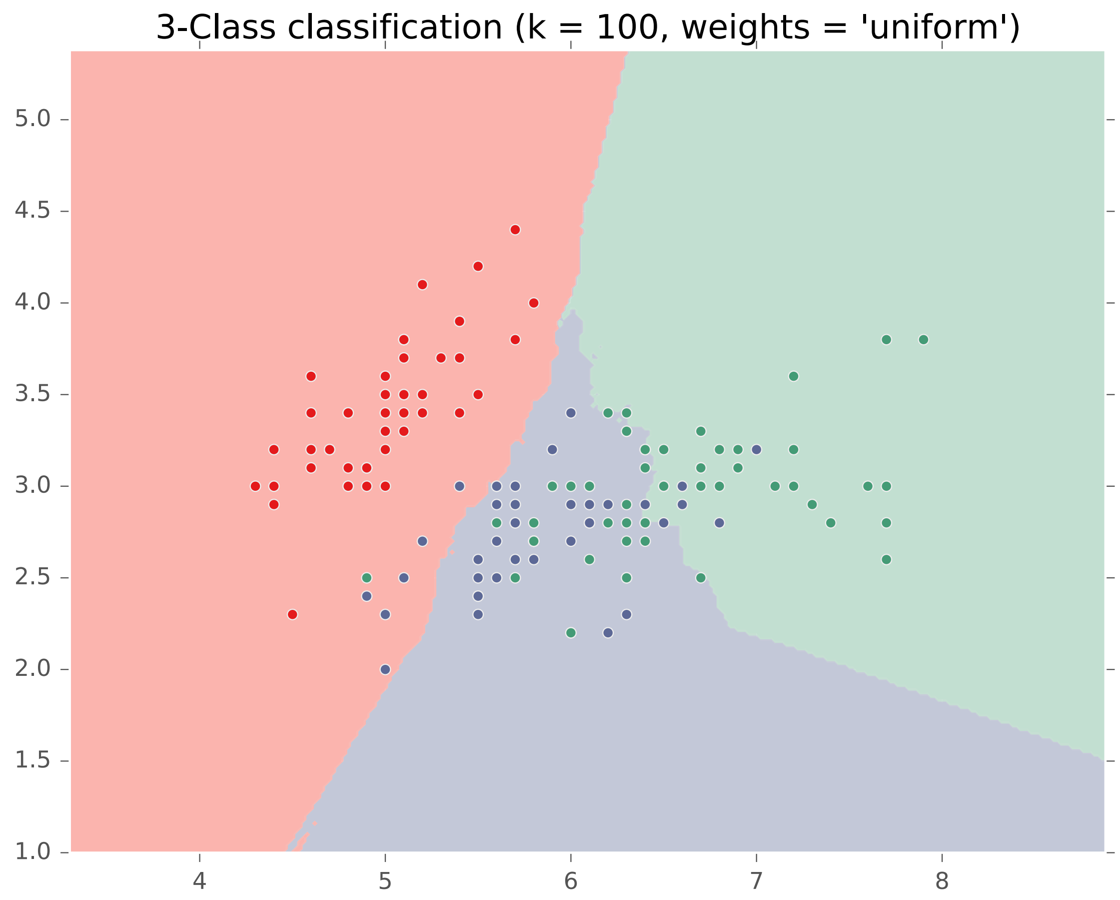


Figure courtesy of Matt Gormley

kNN on Fisher Iris Data

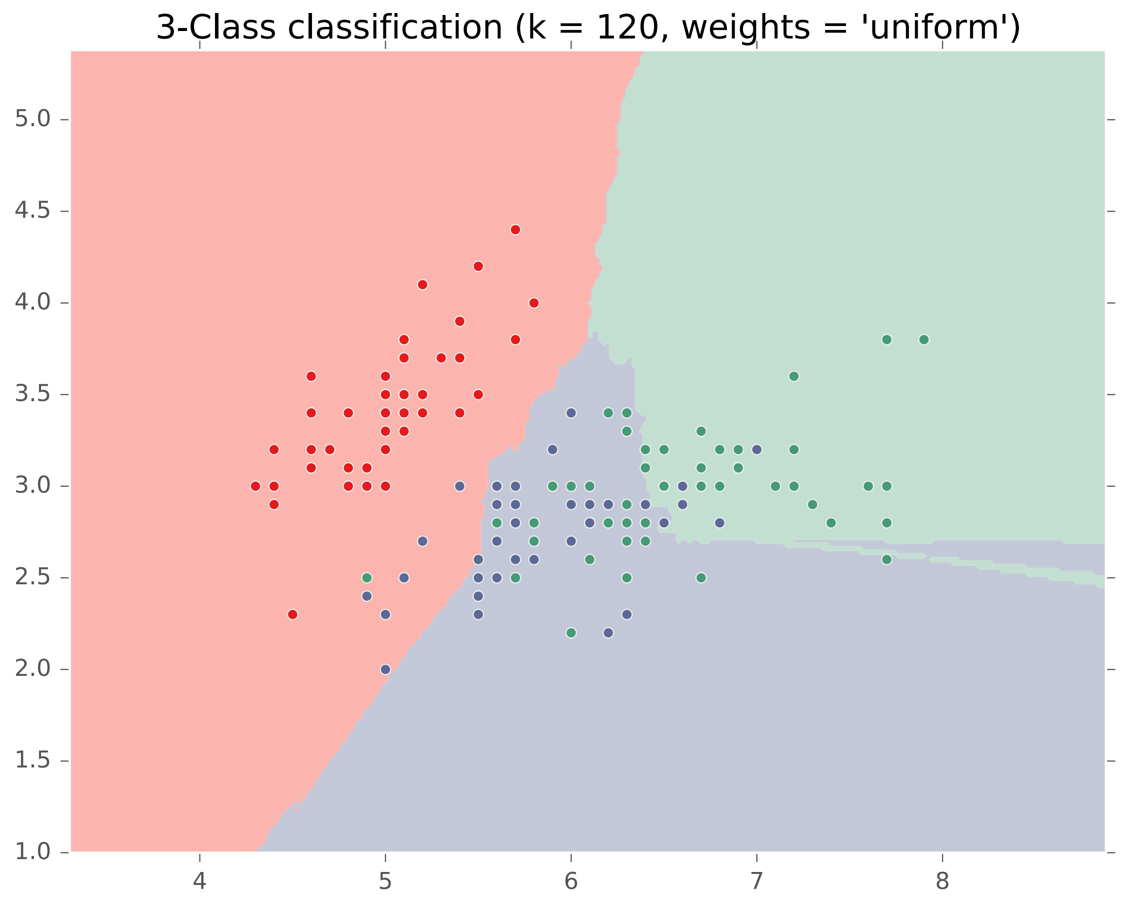


Figure courtesy of Matt Gormley

kNN on Fisher Iris Data

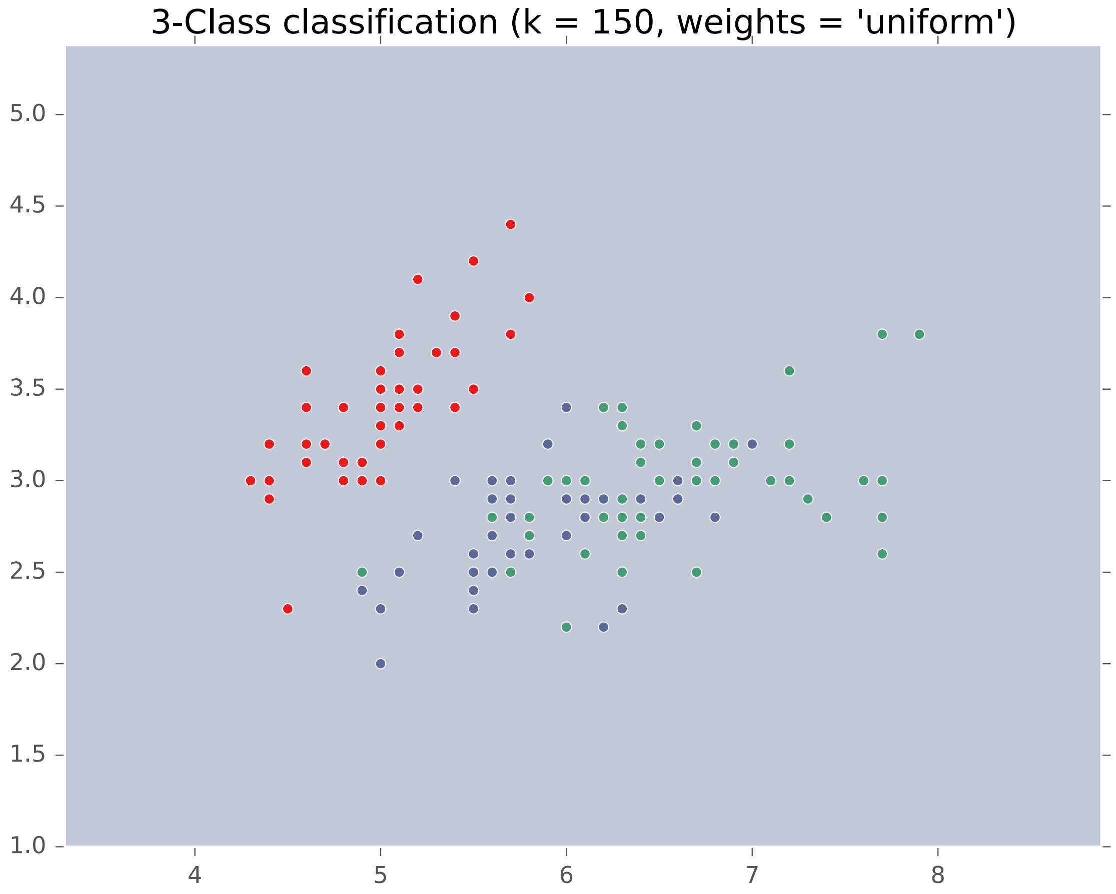


Figure courtesy of Matt Gormley

kNN with Euclidean Distance: Inductive Bias

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?

- 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.
  - In the real world, the models and the many decisions available to you will be much more complex than what we've seen so far.

Model Selection

- Terminology:
  - **Model** $\approx$ 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 (see below)
  - 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** $\approx$ 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

Parametric vs. Nonparametric Models

- Parametric models (e.g., decision trees)
  - Have a parametrized form with parameters learned from training data
  - Can discard training data after parameters have been learned.
  - Cannot exactly model every target function
- Nonparametric models (e.g., kNN)
  - Have no parameters that are learned from training data; can still have *hyperparameters*
  - Training data generally needs to be stored in order to make predictions
  - Can recover any target function given enough data

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

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 $\to$ the Bayes error rate
- Practical heuristics:
  - $k = \lfloor \sqrt{N} \rfloor$
  - $k = 3$
- Perform model selection!

Model Selection with Test Sets?

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

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

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

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

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

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

- 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:

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

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:

$$\theta_1, \theta_2, \dots, \theta_M$$

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

$$h_1, h_2, \dots, h_M$$

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

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

Setting k for kNN with Validation Sets

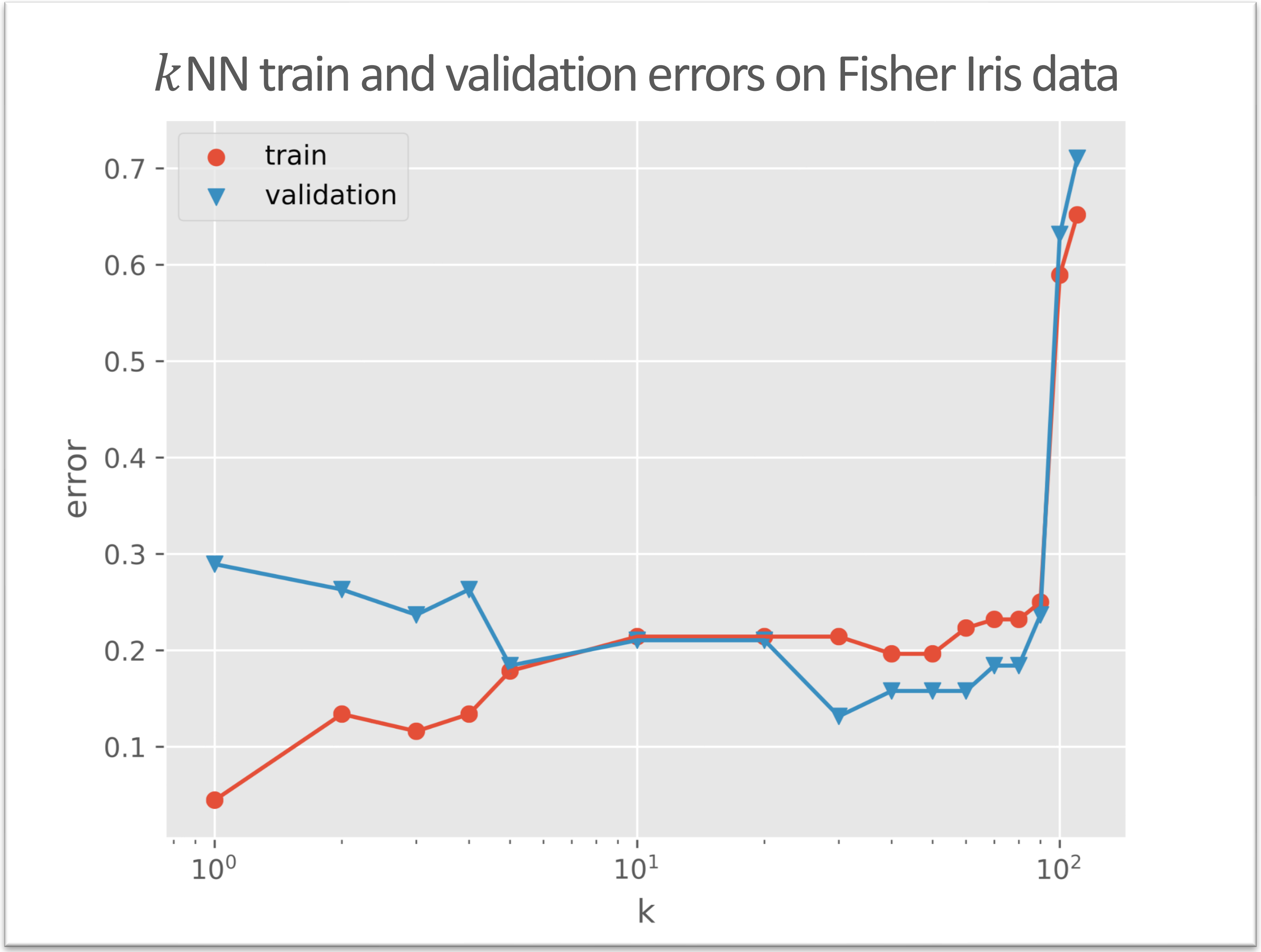


Figure courtesy of Matt Gormley

How should we partition our dataset?

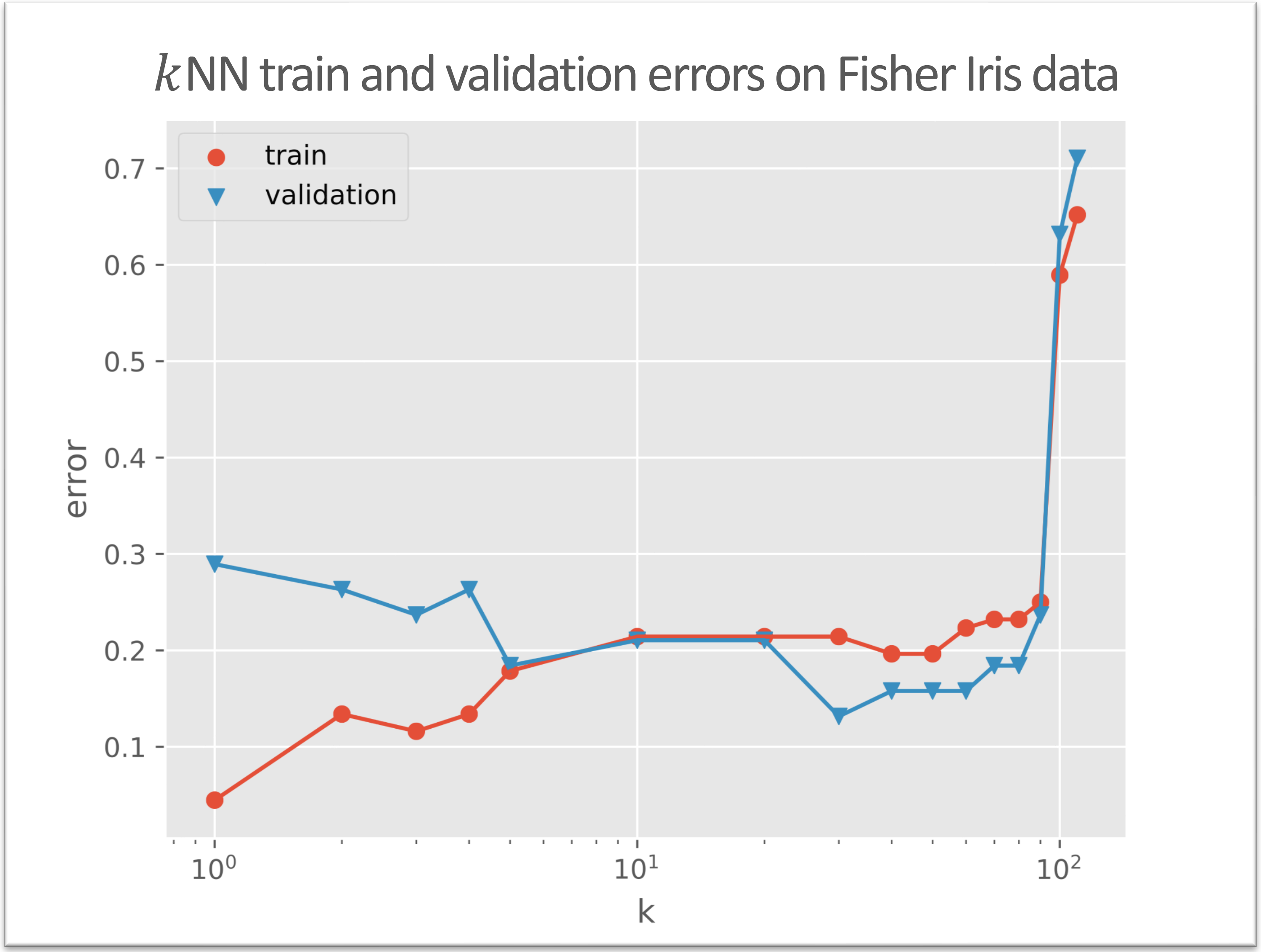
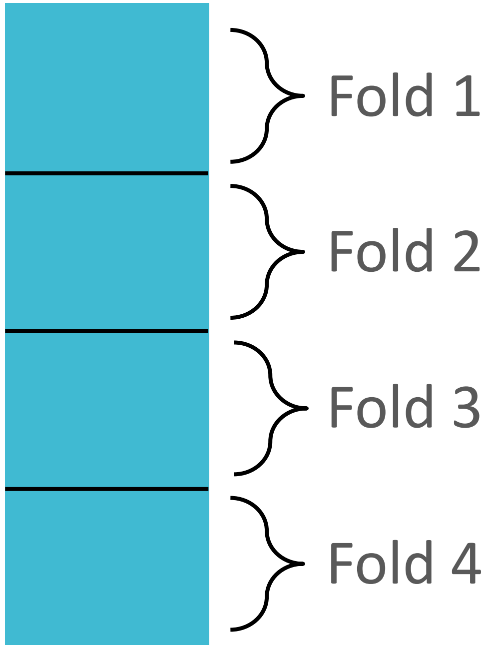


Figure courtesy of Matt Gormley

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:

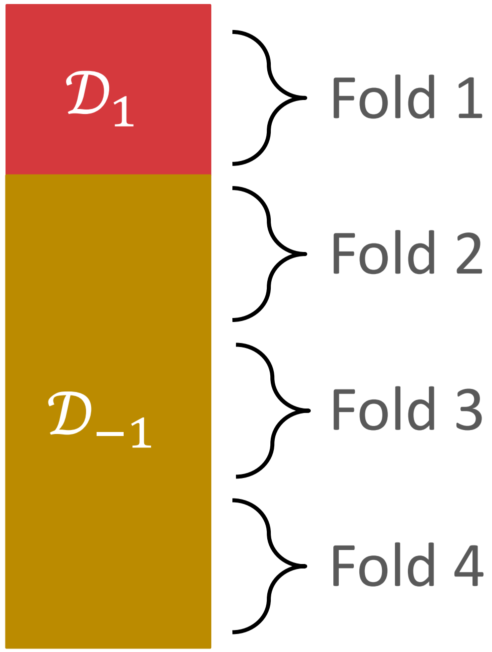


- Let h_{-i} be the classifier learned using $\mathcal{D}_{-i} = \mathcal{D} \backslash \mathcal{D}_i$ (all folds other than $\mathcal{D}_i$) and let $e_i = err(h_{-i}, \mathcal{D}_i)$
- The K-fold cross validation error is

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

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:

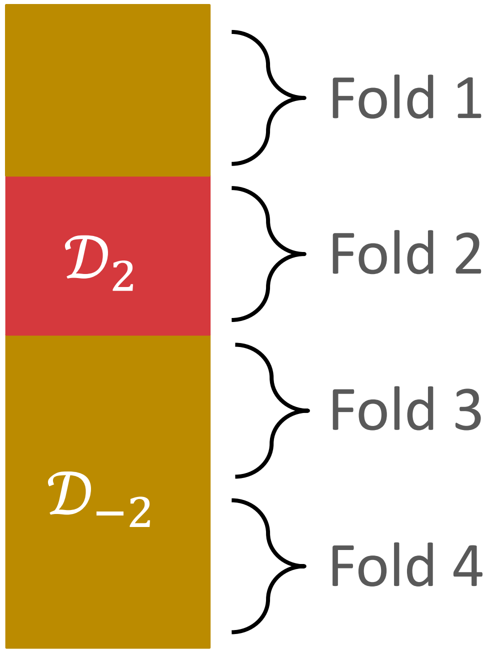


- Let h_{-i} be the classifier learned using $\mathcal{D}_{-i} = \mathcal{D} \backslash \mathcal{D}_i$ (all folds other than $\mathcal{D}_i$) and let $e_i = err(h_{-i}, \mathcal{D}_i)$
- The K-fold cross validation error is

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

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:

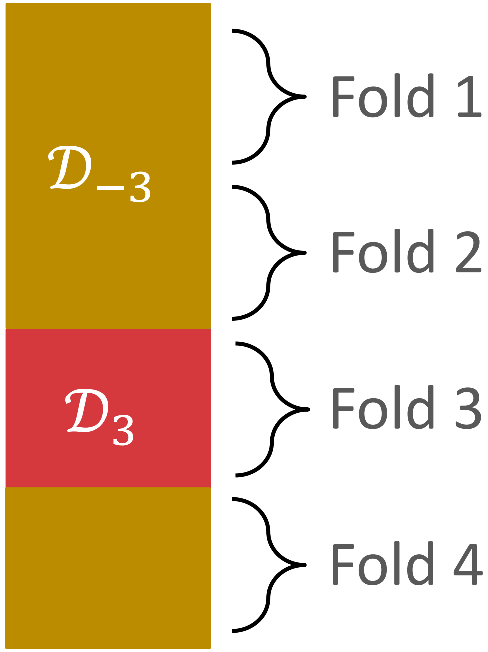


- Let h_{-i} be the classifier learned using $\mathcal{D}_{-i} = \mathcal{D} \backslash \mathcal{D}_i$ (all folds other than $\mathcal{D}_i$) and let $e_i = err(h_{-i}, \mathcal{D}_i)$
- The K-fold cross validation error is

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

K-fold cross-validation

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

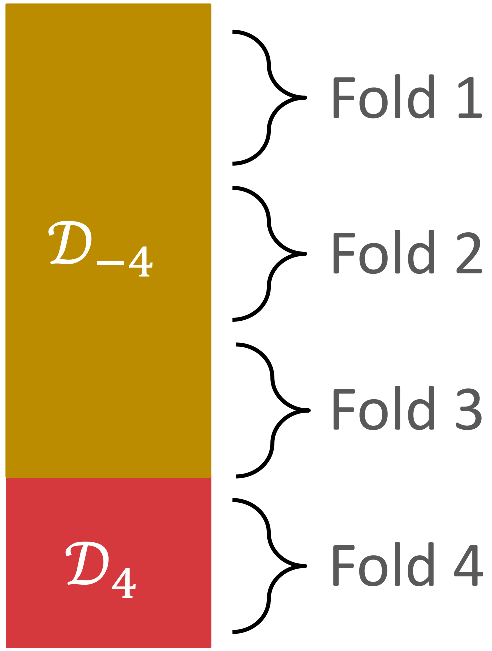


- Let h_{-i} be the classifier learned using $\mathcal{D}_{-i} = \mathcal{D} \backslash \mathcal{D}_i$ (all folds other than $\mathcal{D}_i$) and let $e_i = err(h_{-i}, \mathcal{D}_i)$
- The K-fold cross validation error is

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

K-fold cross-validation

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

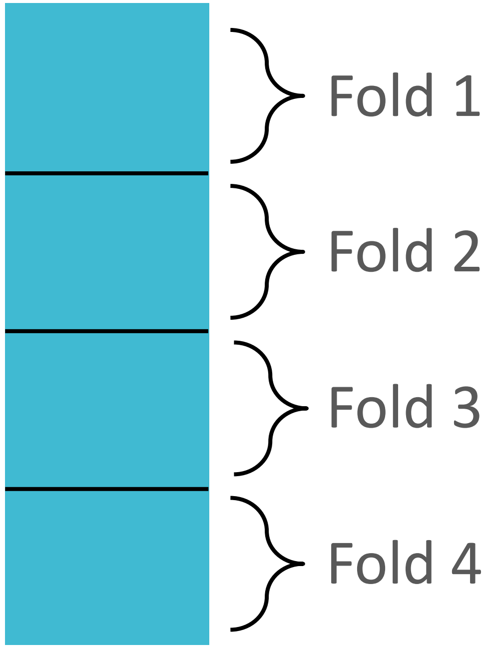


- Let h_{-i} be the classifier learned using $\mathcal{D}_{-i} = \mathcal{D} \backslash \mathcal{D}_i$ (all folds other than $\mathcal{D}_i$) and let $e_i = err(h_{-i}, \mathcal{D}_i)$
- The K-fold cross validation error is

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

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:

$$\theta_1, \theta_2, \dots, \theta_M$$

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

$$h_1, h_2, \dots, h_M$$

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

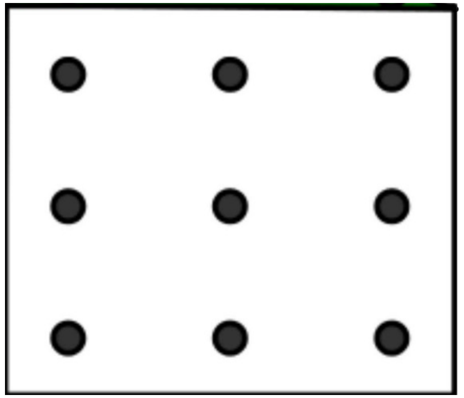
$$\hat{m} = \underset{m \in \{1,\dots,M\}}{\operatorname{argmin}} \; err(h_m, \mathcal{D}_{val})$$

General Methods for Hyperparameter Optimization

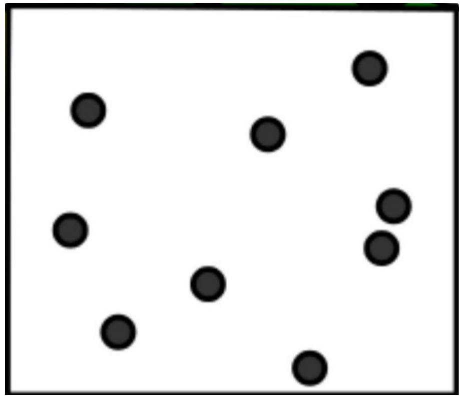
- Idea: set the hyperparameters to optimize some performance metric of the model
- Issue: if we have many hyperparameters that can all take on lots of different values, we might not be able to test all possible combinations
- Commonly used methods:
  - Grid search
  - Random search
  - Bayesian optimization (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

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