



# 10-301/10-601 Introduction to Machine Learning

Machine Learning Department  
School of Computer Science  
Carnegie Mellon University

# k-Nearest Neighbors + Model Selection

Matt Gormley  
Lecture 5  
Feb. 1, 2023

# Reminders

- **Homework 2: Decision Trees**
  - **Out: Wed, Jan. 25**
  - **Due: Fri, Feb. 3 at 11:59pm**
- **Schedule Note:**
  - **Fri, Feb. 3: Lecture 6: Perceptron**
  - **Wed, Feb. 8: Recitation: HW3**

# Moss: Code Plagiarism Detection

# What is Moss?

- Moss (Measure Of Software Similarity): is an automatic system for determining the similarity of programs. To date, the main application of Moss has been in 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

# What is Moss?

At first glance, the submissions may look different

```
# 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()
```

```
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()
```

# What is Moss?

Moss can quickly find 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()
```

# Q&A

**Q:** I'm now terrified to collaborate with anyone ever again. Can you remind me of what sort of collaboration is allowed?

**A:** Yes!

You should collaborate as follows: (1) sketch out pseudocode on an impermanent surface, e.g., a whiteboard (2) erase said surface and part ways with your collaborator and (3) implement your own code from scratch.

# **K-NEAREST NEIGHBORS**



# Classification & KNN

## *Whiteboard:*

- Binary classification
- 2D examples
- Decision rules / hypotheses

# Nearest Neighbor: Algorithm

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

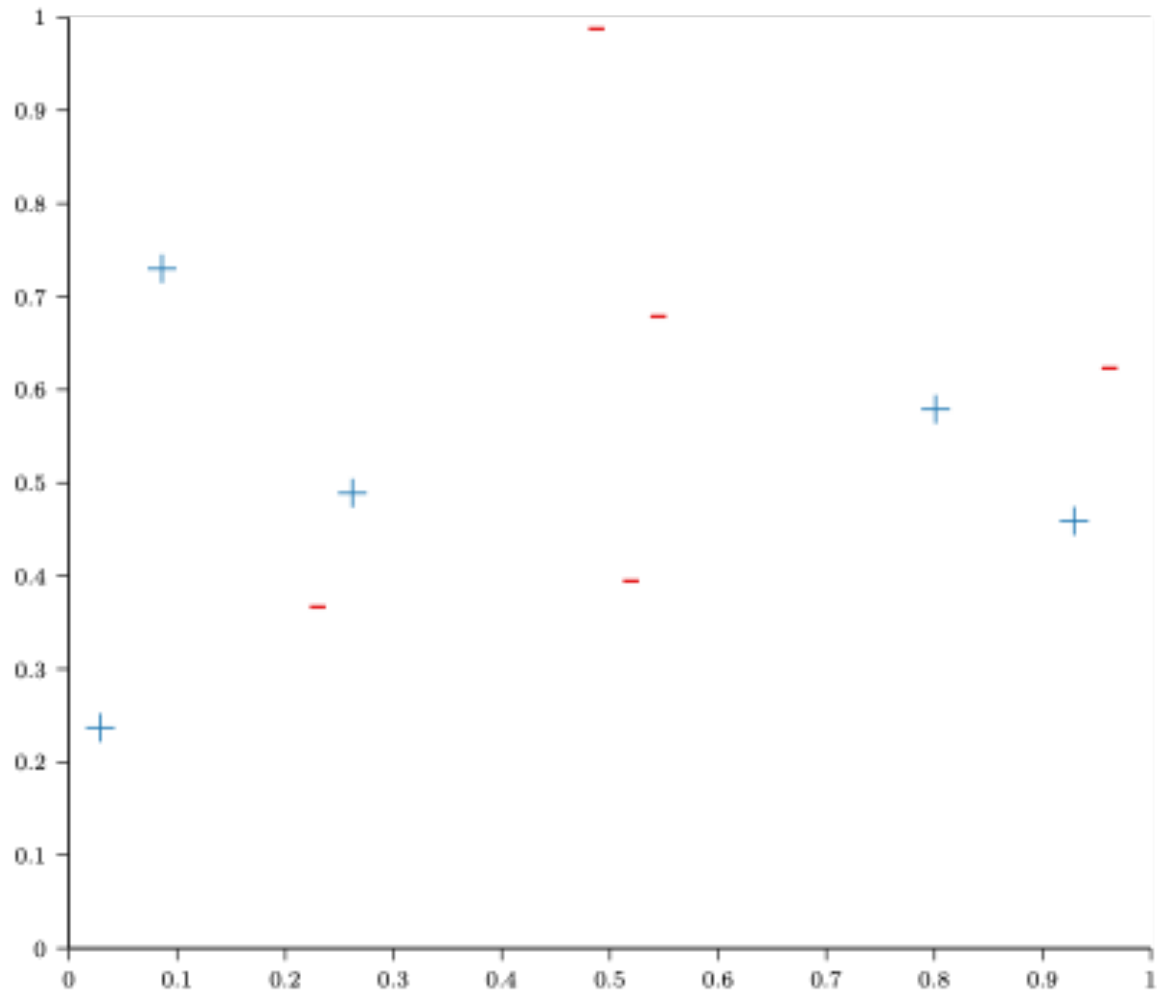
    Store  $\mathcal{D}$

**def h( $x'$ ):**

    Let  $x^{(i)}$  = the point in  $\mathcal{D}$  that is nearest to  $x'$

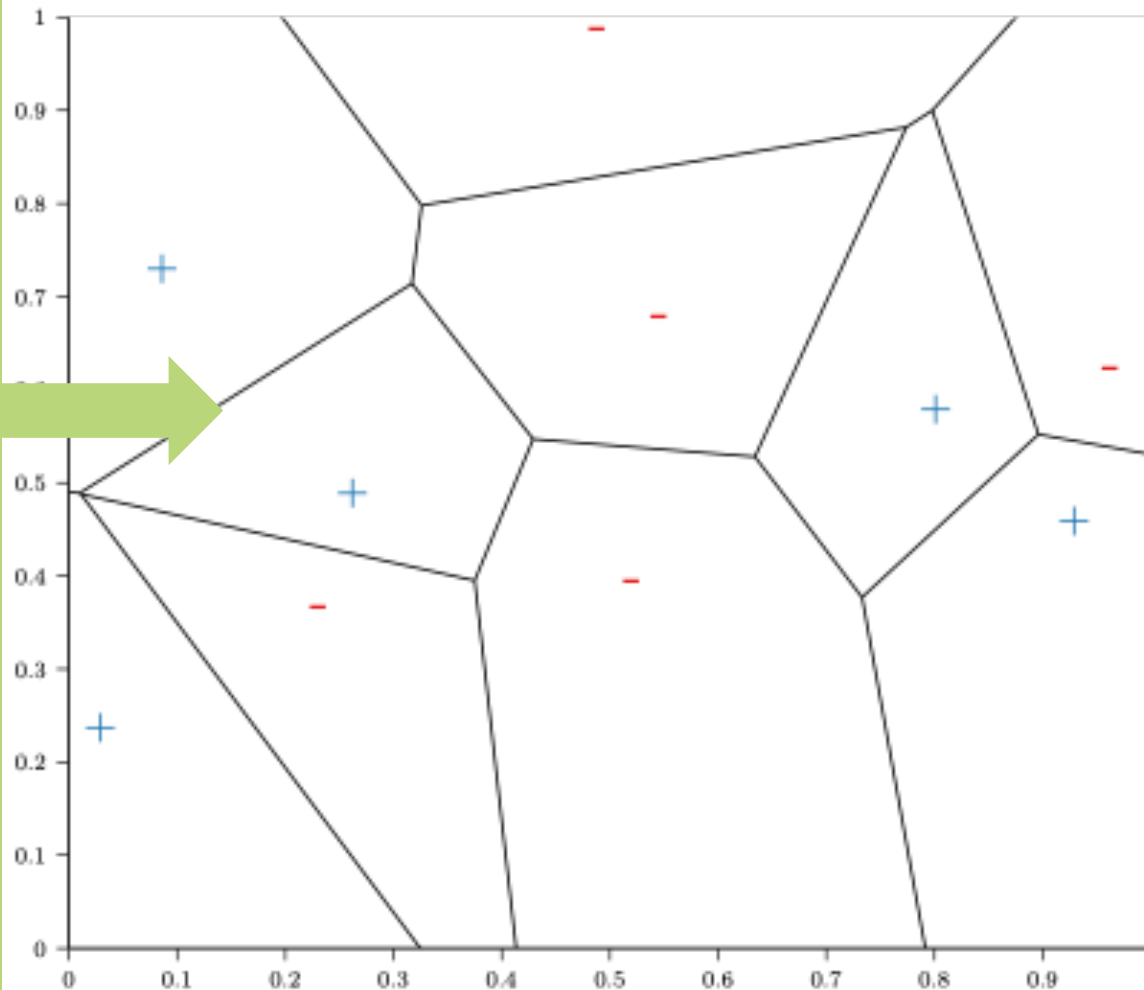
**return**  $y^{(i)}$

# Nearest Neighbor: Example

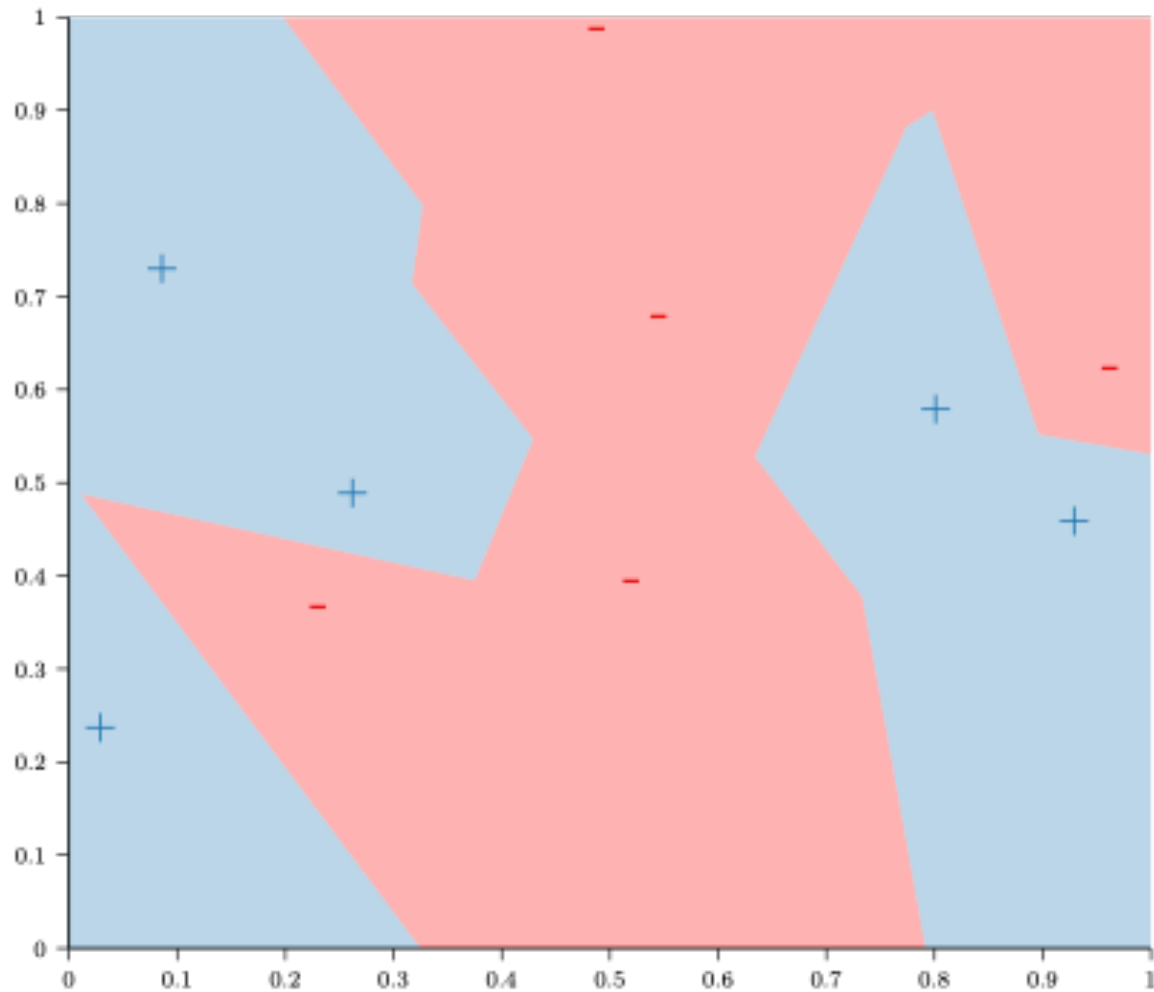


# Nearest Neighbor: Example

- This is a **Voronoi diagram**
- Each **cell** contain one of our training examples
- **All points within a cell** are **closer** to that training example, than to any other training example
- **Points on the Voronoi line segments** are **equidistant** to one or more training examples



# Nearest Neighbor: Example



# The Nearest Neighbor Model

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

# k-Nearest Neighbors: Algorithm

**def set\_hyperparameters(k, d):**

    Store k

    Store  $d(\cdot, \cdot)$

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

    Store  $\mathcal{D}$

**def h( $x'$ ):**

    Let  $S$  = the set of  $k$  points in  $\mathcal{D}$  nearest to  $x'$   
        according to distance function

$d(\mathbf{u}, \mathbf{v})$

    Let  $v$  = majority\_vote( $S$ )

**return**  $v$



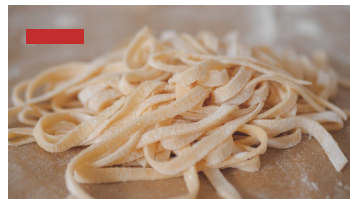
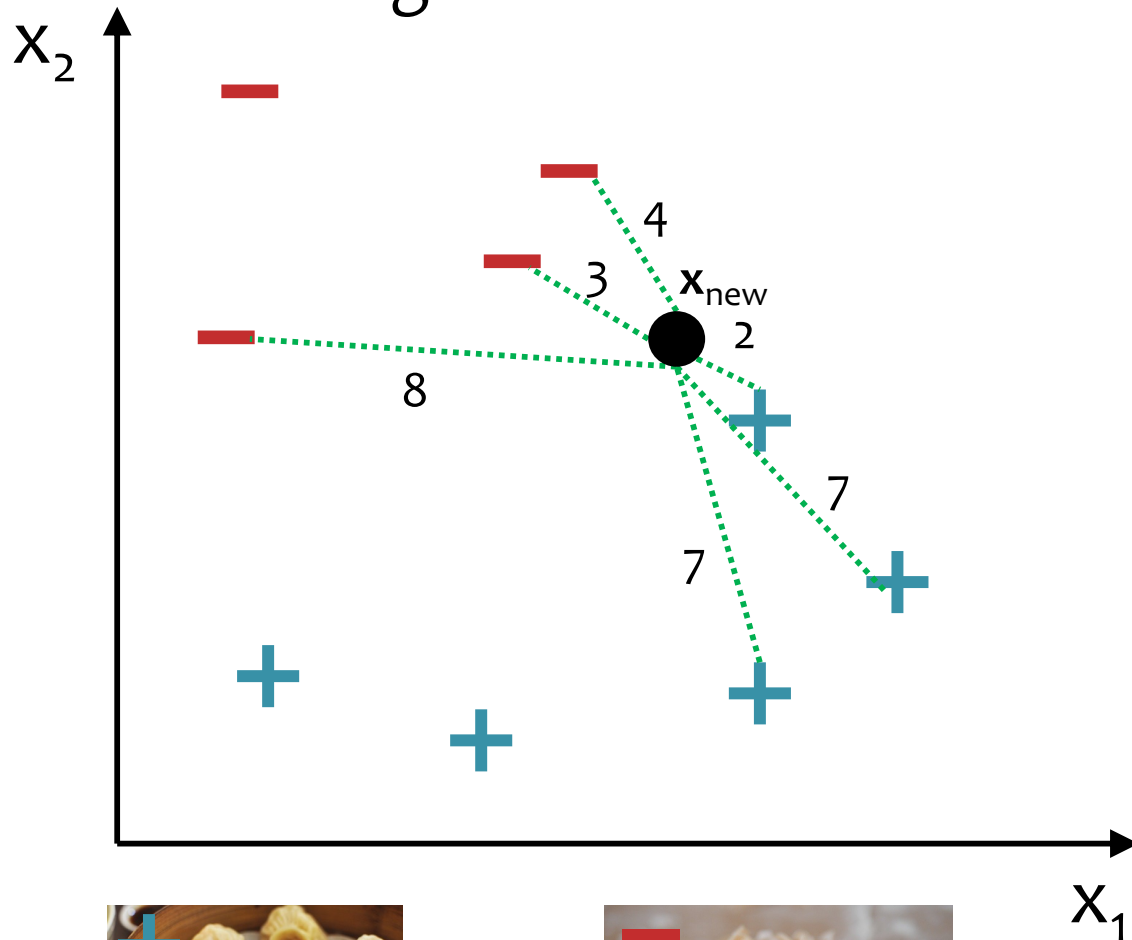
# 新年 快樂

HAPPY NEW YEAR



# k-Nearest Neighbors

Suppose we have the training dataset below.



How should we label the new point?

It depends on k:

if  $k=1$ ,  $h(\mathbf{x}_{\text{new}}) = +1$

if  $k=3$ ,  $h(\mathbf{x}_{\text{new}}) = -1$

if  $k=5$ ,  $h(\mathbf{x}_{\text{new}}) = +1$



# KNN: Remarks

## Distance Functions:

- KNN requires a **distance function**

$$d : \mathbb{R}^M \times \mathbb{R}^M \rightarrow \mathbb{R}$$

- The most common choice is **Euclidean distance**

$$d(\mathbf{u}, \mathbf{v}) = \sqrt{\sum_{m=1}^M (u_m - v_m)^2}$$

- But there are other choices (e.g. **Manhattan distance**)

$$d(\mathbf{u}, \mathbf{v}) = \sum_{m=1}^M |u_m - v_m|$$

# KNN: Remarks

## In-Class Exercises

How can we handle ties for even values of  $k$ ?

Answer(s) Here:

# KNN: Remarks

## In-Class Exercises

How can we handle ties for even values of  $k$ ?

## Answer(s) Here:

- Consider another point
- Remove farthest of  $k$  points
- Weight votes by distance
- Consider another distance metric

# KNN: Inductive Bias

## **In-Class Exercise**

What is the inductive bias of KNN?

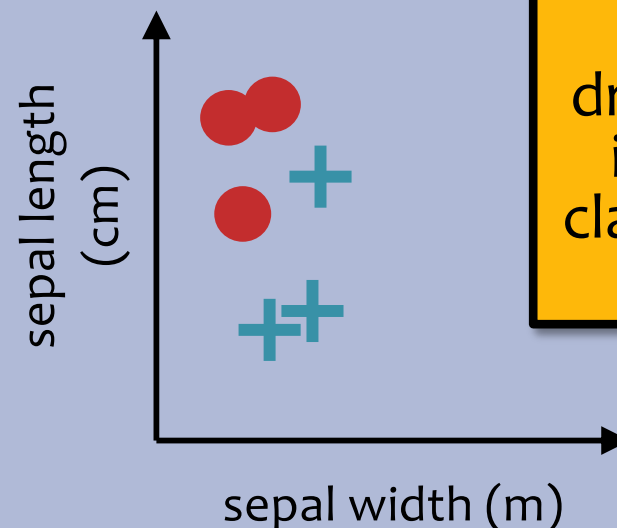
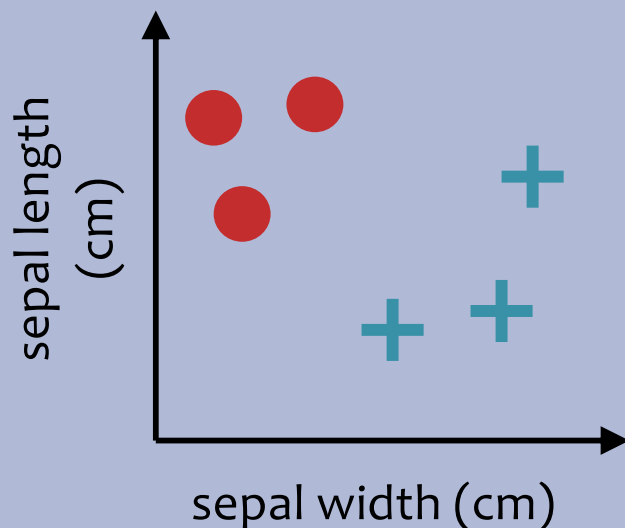
# KNN: Inductive Bias

## In-Class Exercise

What is the inductive bias of KNN?

1. Similar points should have similar labels
2. All dimensions are created equally!

Example: two features for KNN



**big problem:**  
feature scale  
can  
dramatically  
influence  
classification  
results!

# KNN: Computational Efficiency

- Suppose we have  $N$  training examples and each one has  $M$  features
- Computational complexity when  $k=1$ :

Task	Naive	k-d Tree
Train	$O(1)$	$\sim O(M N \log N)$
Predict (one test example)	$O(MN)$	$\sim O(2^M \log N)$ on average



**Problem:** Very fast for small  $M$ , but very slow for large  $M$

**In practice:** use stochastic approximations (very fast, and empirically often as good)

# KNN: Theoretical Guarantees

## Cover & Hart (1967)

Let  $h(x)$  be a Nearest Neighbor ( $k=1$ ) binary classifier. As the number of training examples  $N$  goes to infinity...

$$\text{error}_{\text{true}}(h) < 2 \times \text{Bayes Error Rate}$$

“In this sense, it may be said that half the classification information in an infinite sample set is contained in the nearest neighbor.”



**very informally,**  
Bayes Error Rate can be thought of as:  
*‘the best you could possibly do’*



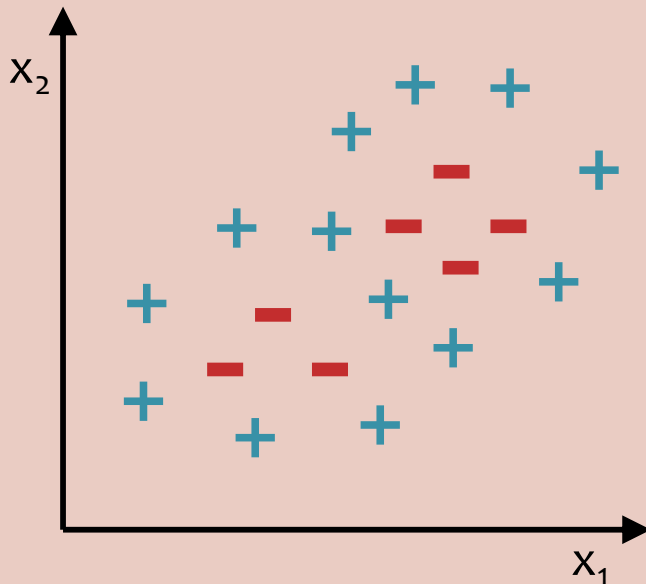
# Decision Boundary Example

**Dataset:** Outputs  $\{+, -\}$ ; Features  $x_1$  and  $x_2$

## In-Class Exercise

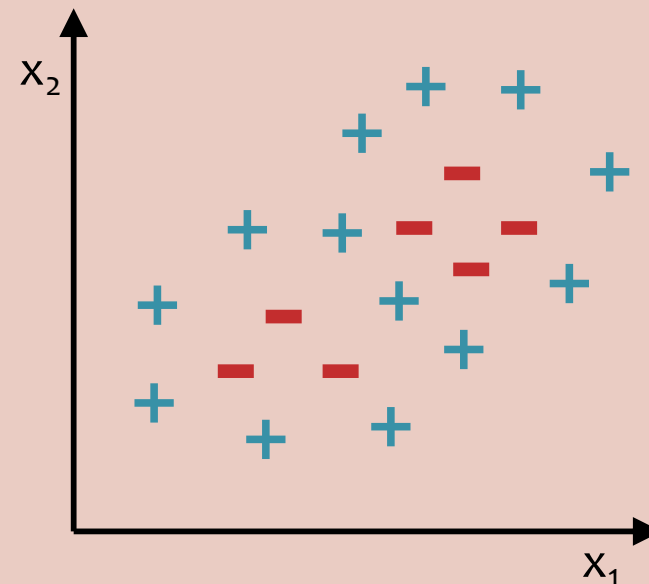
Question:

- A. Can a **k-Nearest Neighbor classifier with  $k=1$**  achieve **zero training error** on this dataset?
- B. If **'Yes'**, draw the learned decision boundary. If **'No'**, why not?



Question:

- A. Can a **Decision Tree classifier** achieve **zero training error** on this dataset?
- B. If **'Yes'**, draw the learned decision boundary. If **'No'**, why not?



# Decision Boundary Example

*Whiteboard:*

- Decision Tree boundary with continuous features (saving this for a short video)

# **KNN ON FISHER IRIS DATA**



# Fisher Iris Dataset

Fisher (1936) used 150 measurements of flowers from 3 different species: Iris setosa (0), Iris virginica (1), Iris versicolor (2) collected by Anderson (1936)

Species	Sepal Length	Sepal Width	Petal Length	Petal Width
0	4.3	3.0	1.1	0.1
0	4.9	3.6	1.4	0.1
0	5.3	3.7	1.5	0.2
1	4.9	2.4	3.3	1.0
1	5.7	2.8	4.1	1.3
1	6.3	3.3	4.7	1.6
1	6.7	3.0	5.0	1.7

# Fisher Iris Dataset

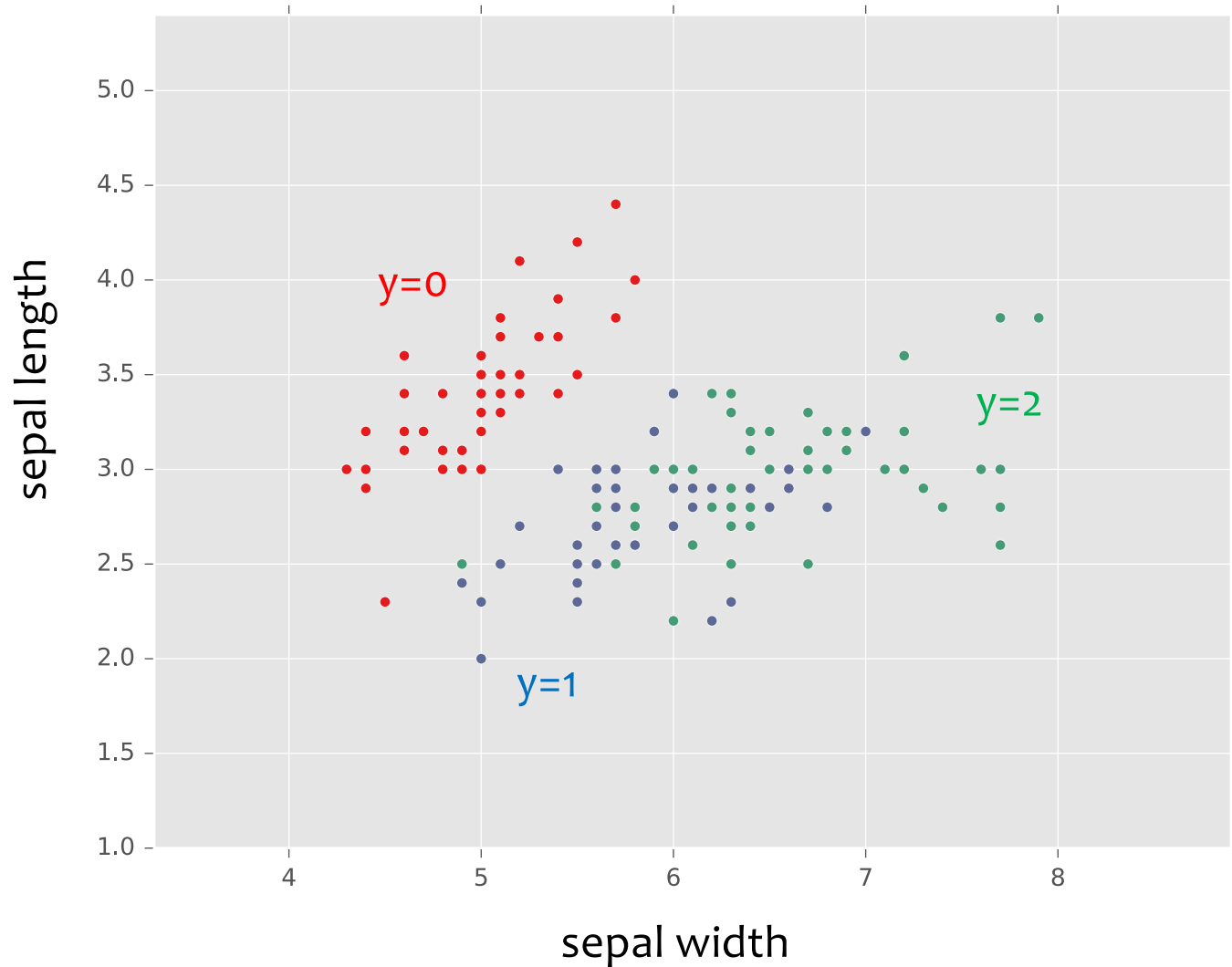
Fisher (1936) used 150 measurements of flowers from 3 different species: Iris setosa (0), Iris virginica (1), Iris versicolor (2) collected by Anderson (1936)

Species	Sepal Length	Sepal Width
0	4.3	3.0
0	4.9	3.6
0	5.3	3.7
1	4.9	2.4
1	5.7	2.8
1	6.3	3.3
1	6.7	3.0

Deleted two of the four features, so that input space is 2D

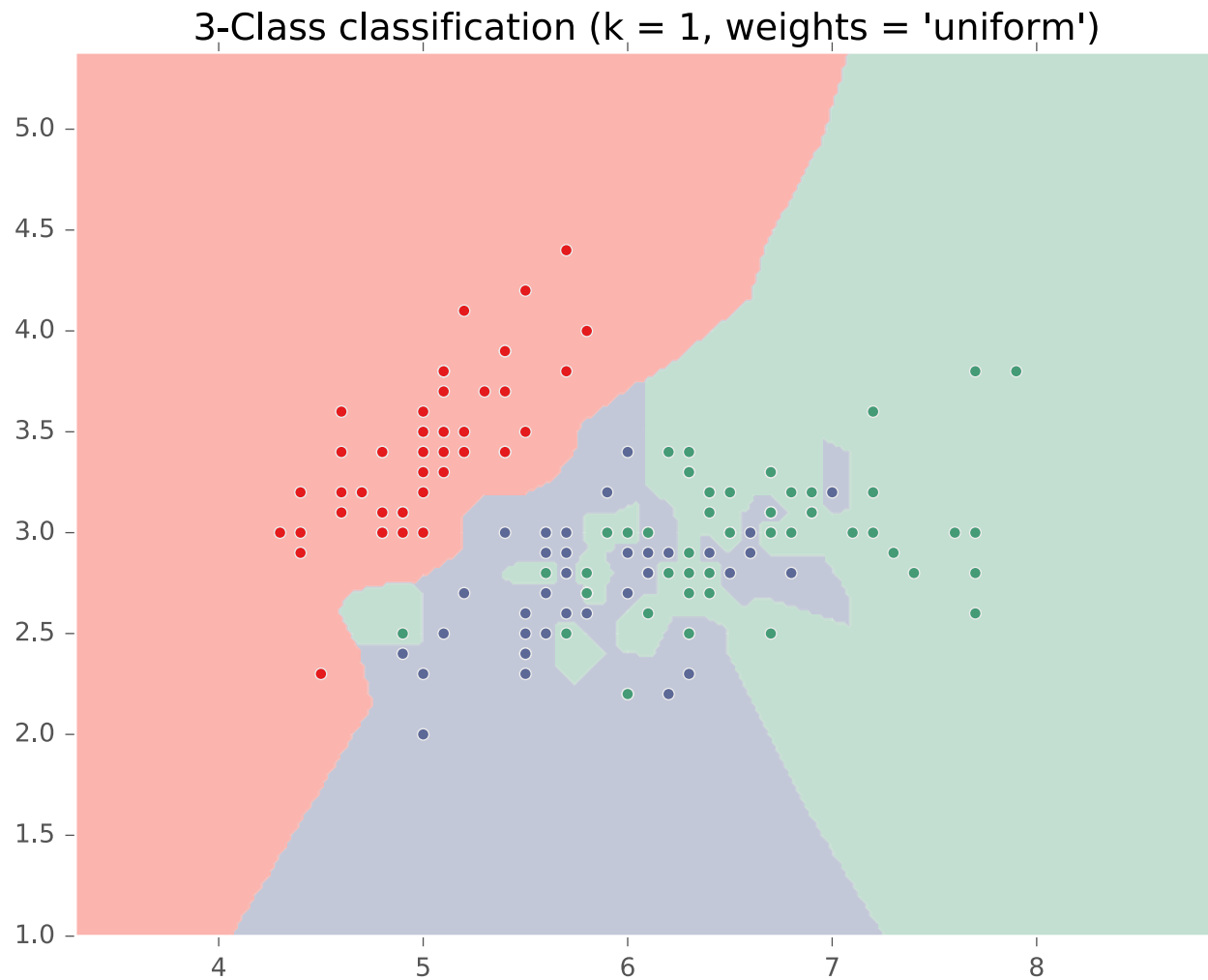


# KNN on Fisher Iris Data



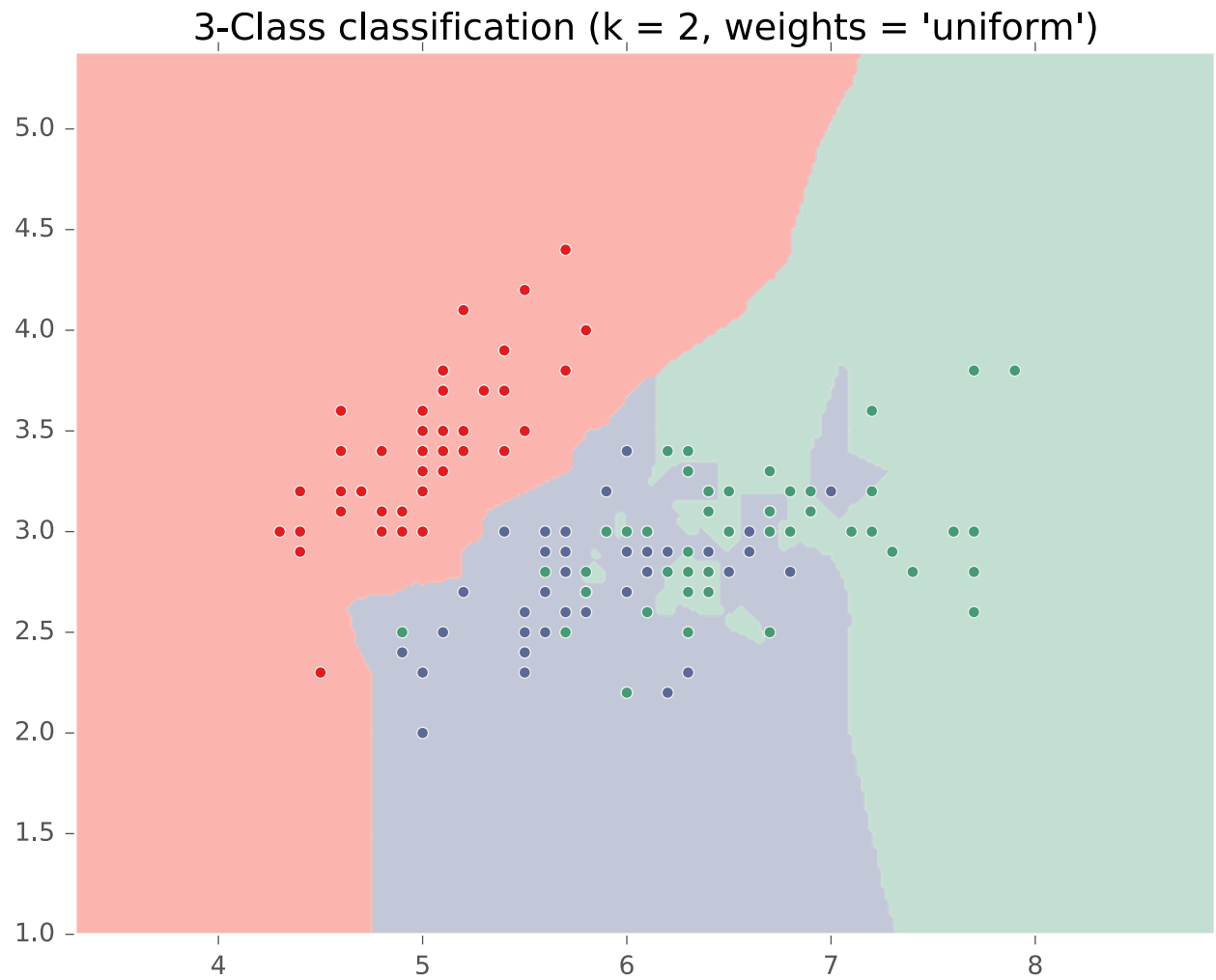
# KNN on Fisher Iris Data

## Special Case: Nearest Neighbor

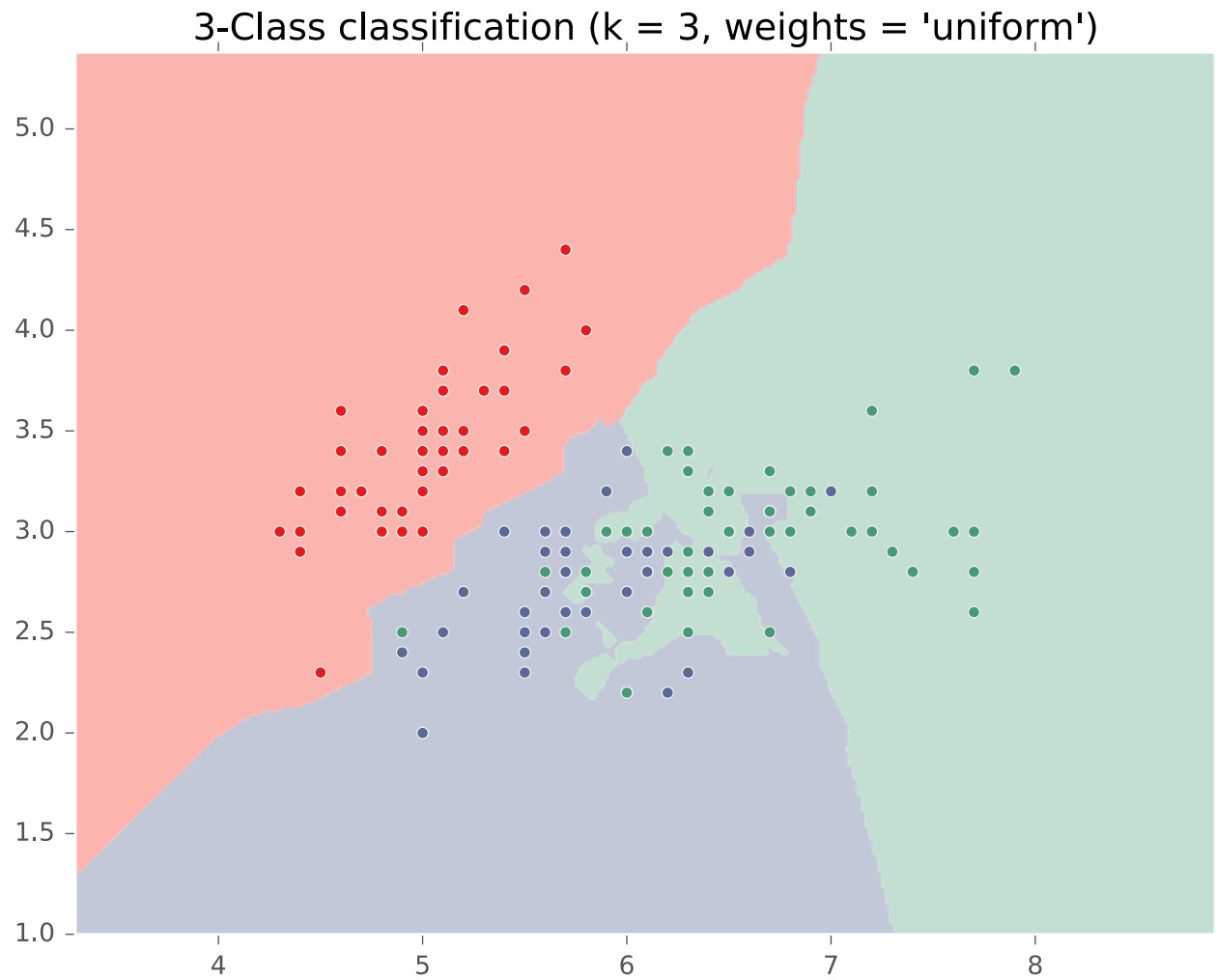




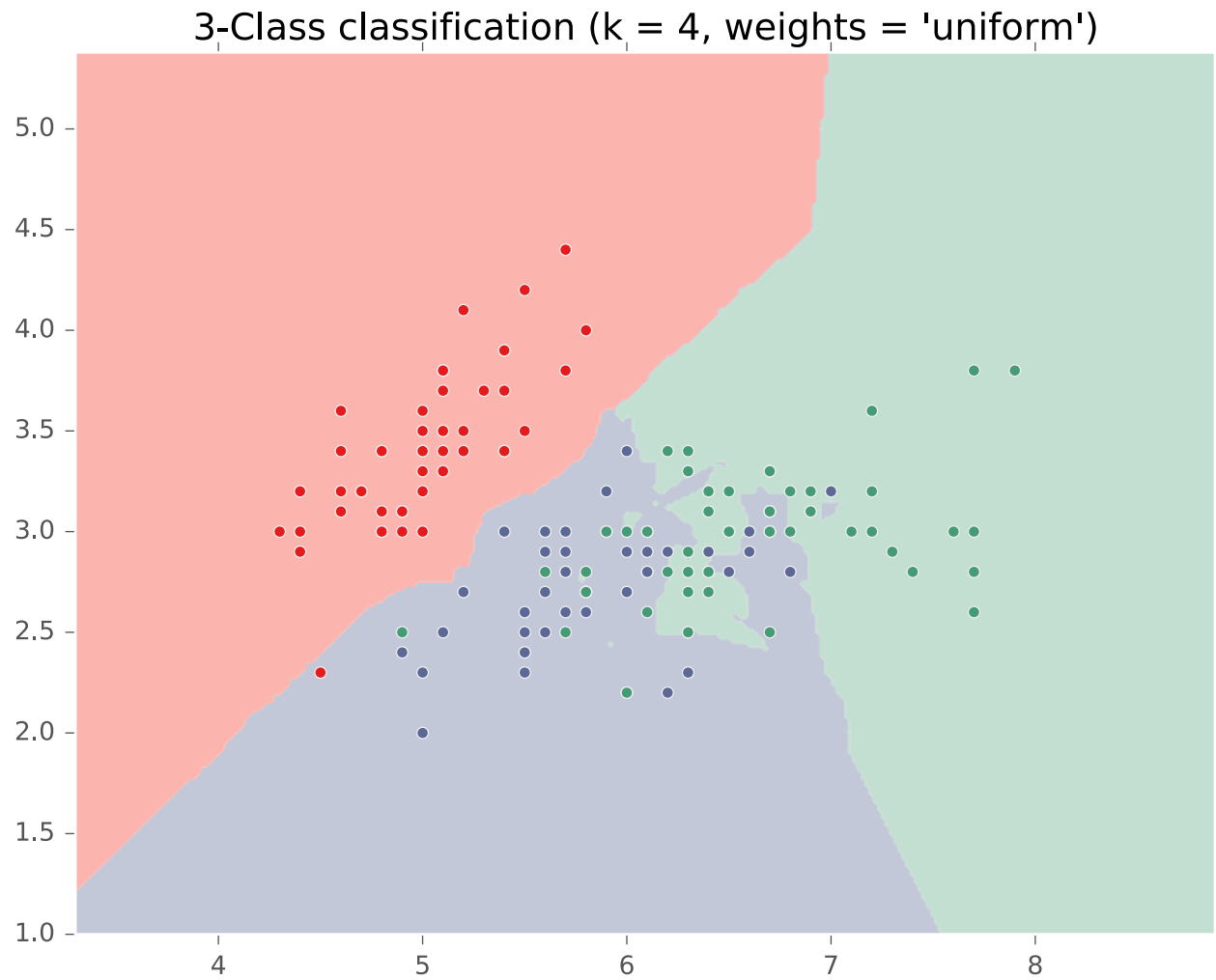
# KNN on Fisher Iris Data



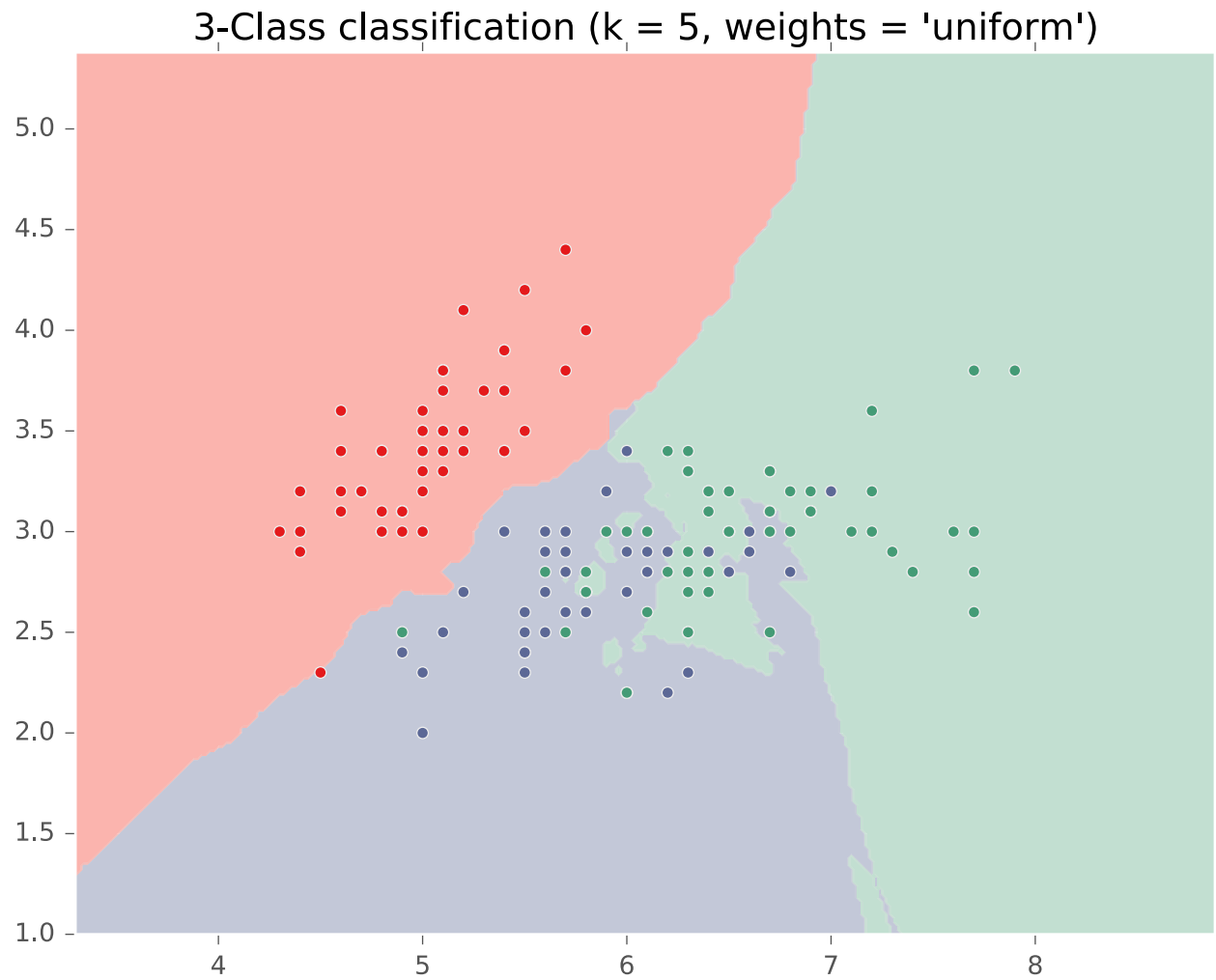
# KNN on Fisher Iris Data



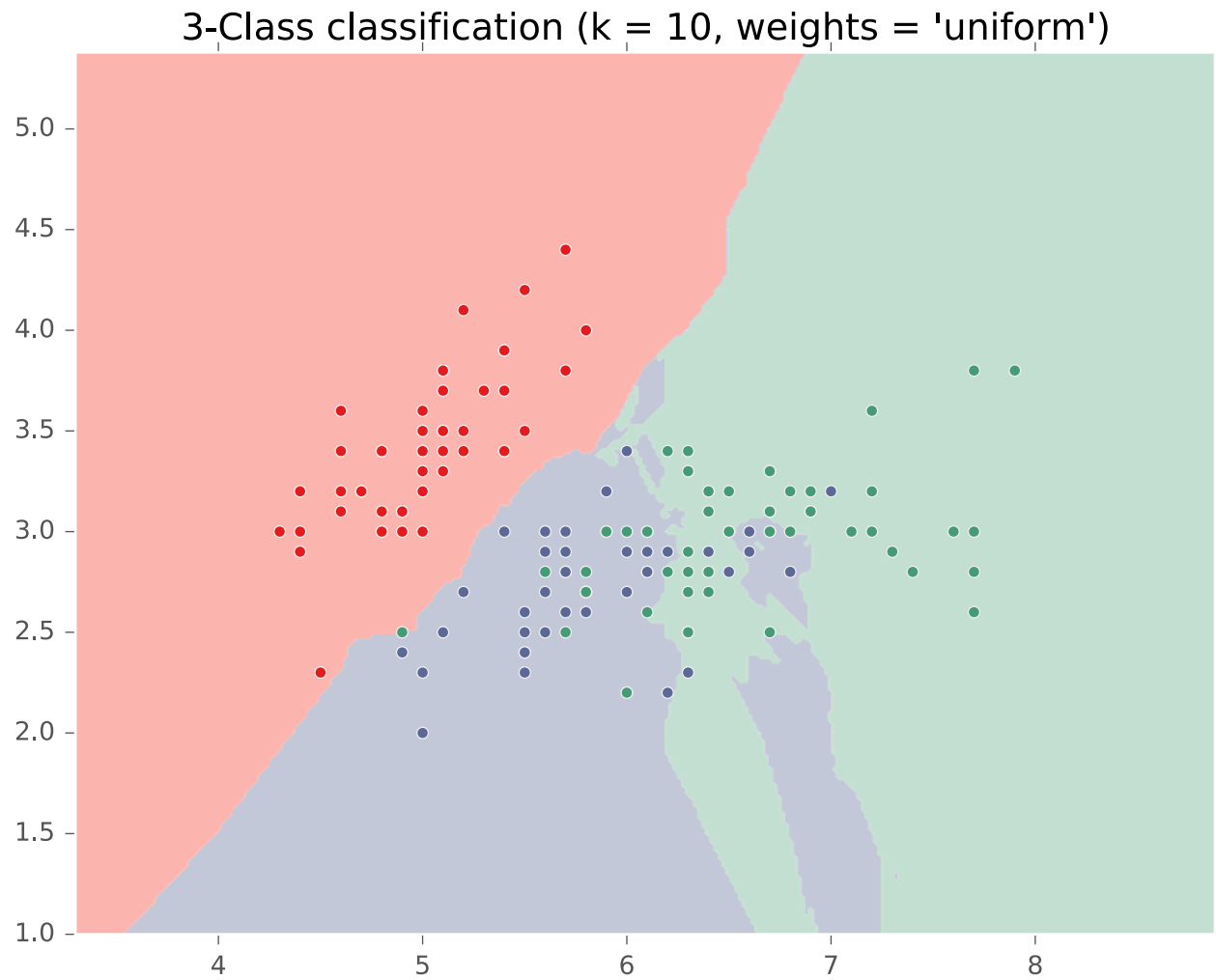
# KNN on Fisher Iris Data



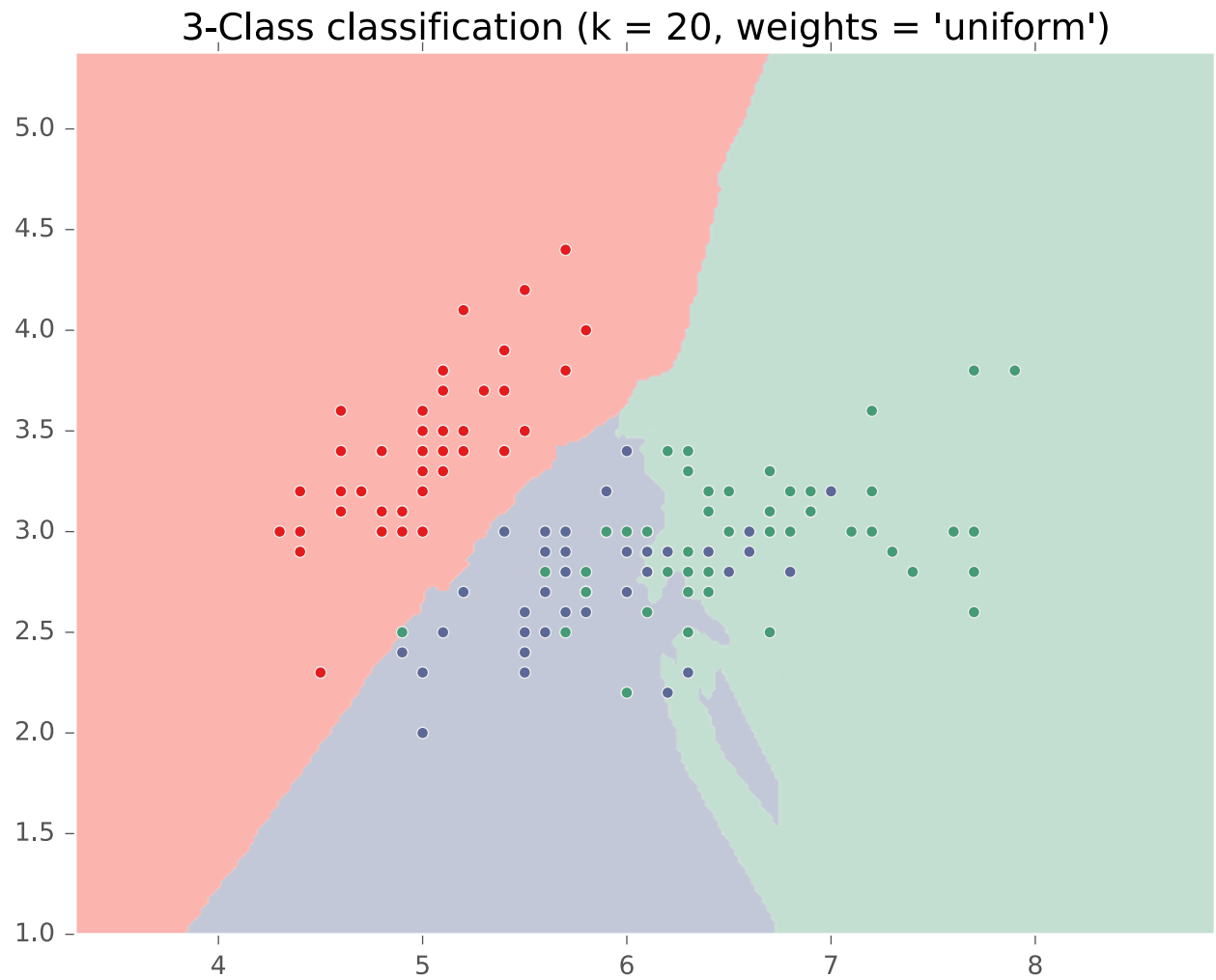
# KNN on Fisher Iris Data



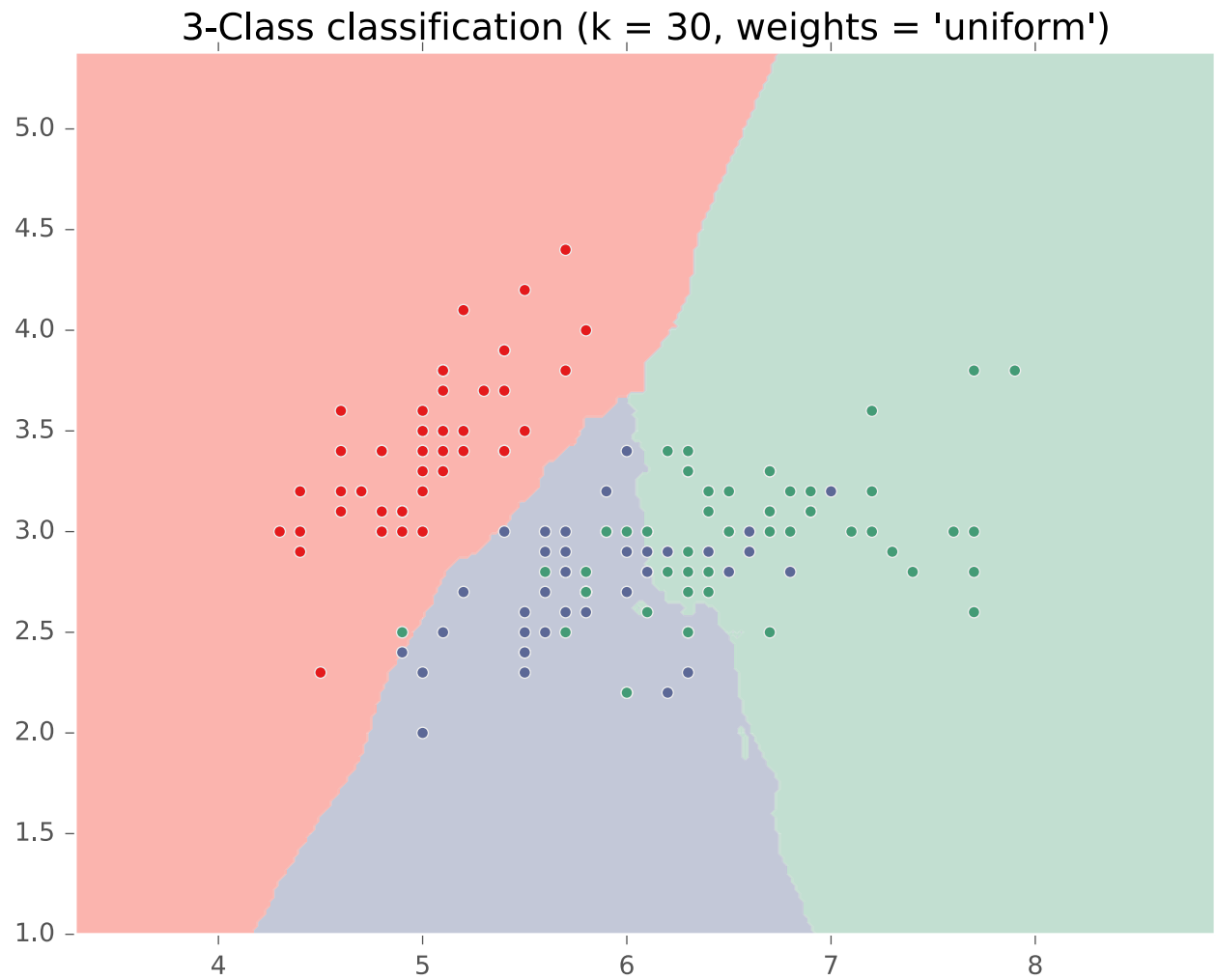
# KNN on Fisher Iris Data



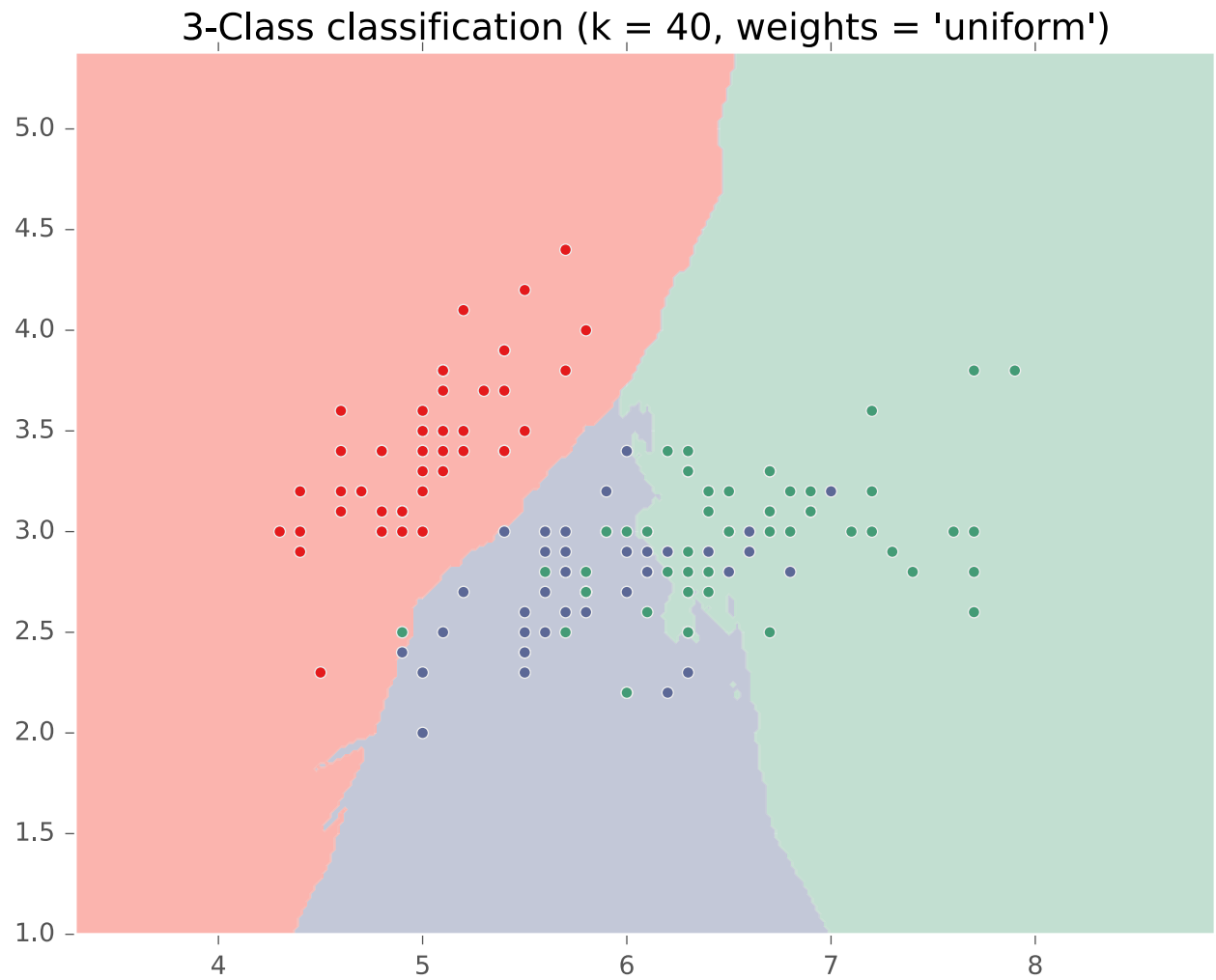
# KNN on Fisher Iris Data



# KNN on Fisher Iris Data

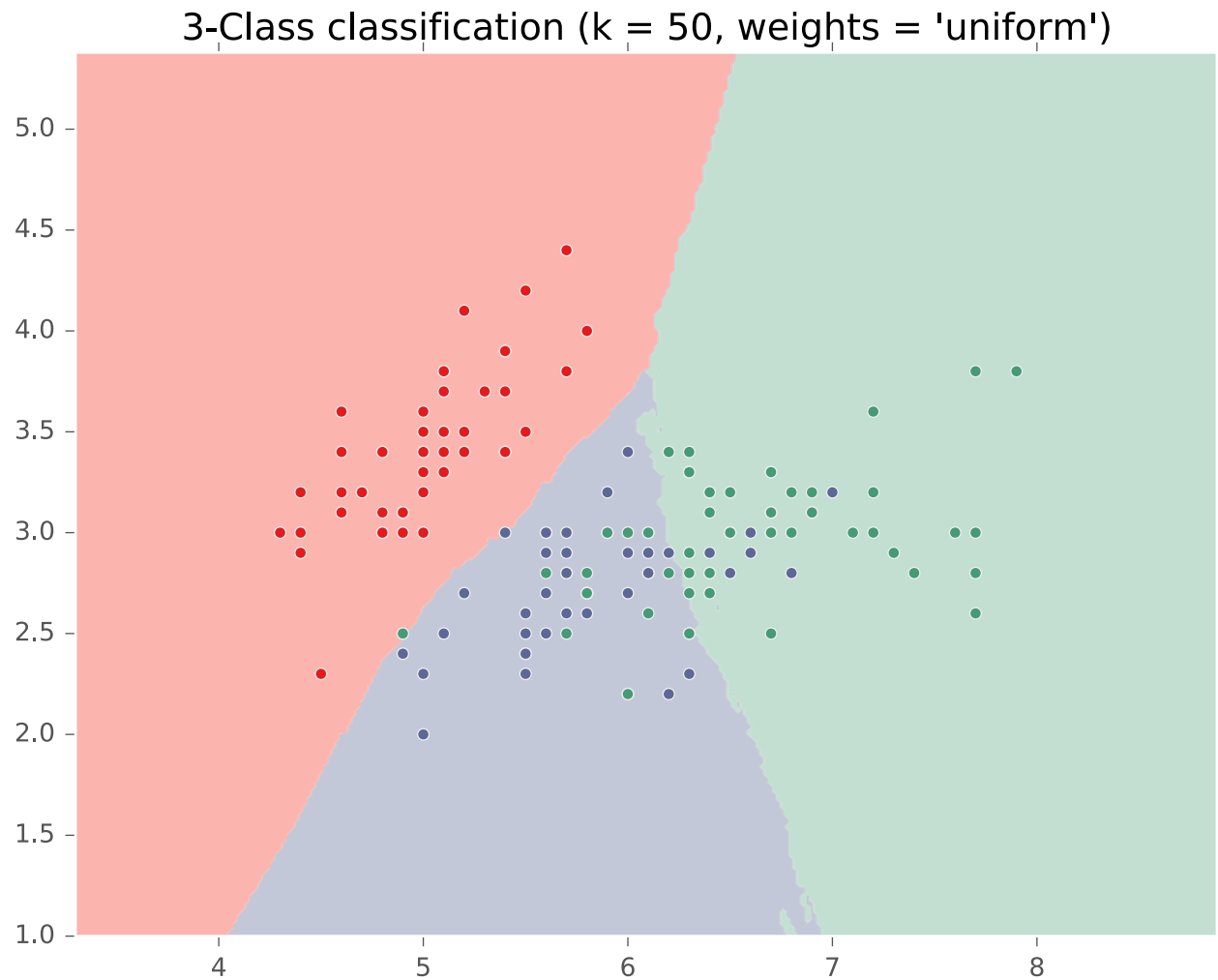


# KNN on Fisher Iris Data

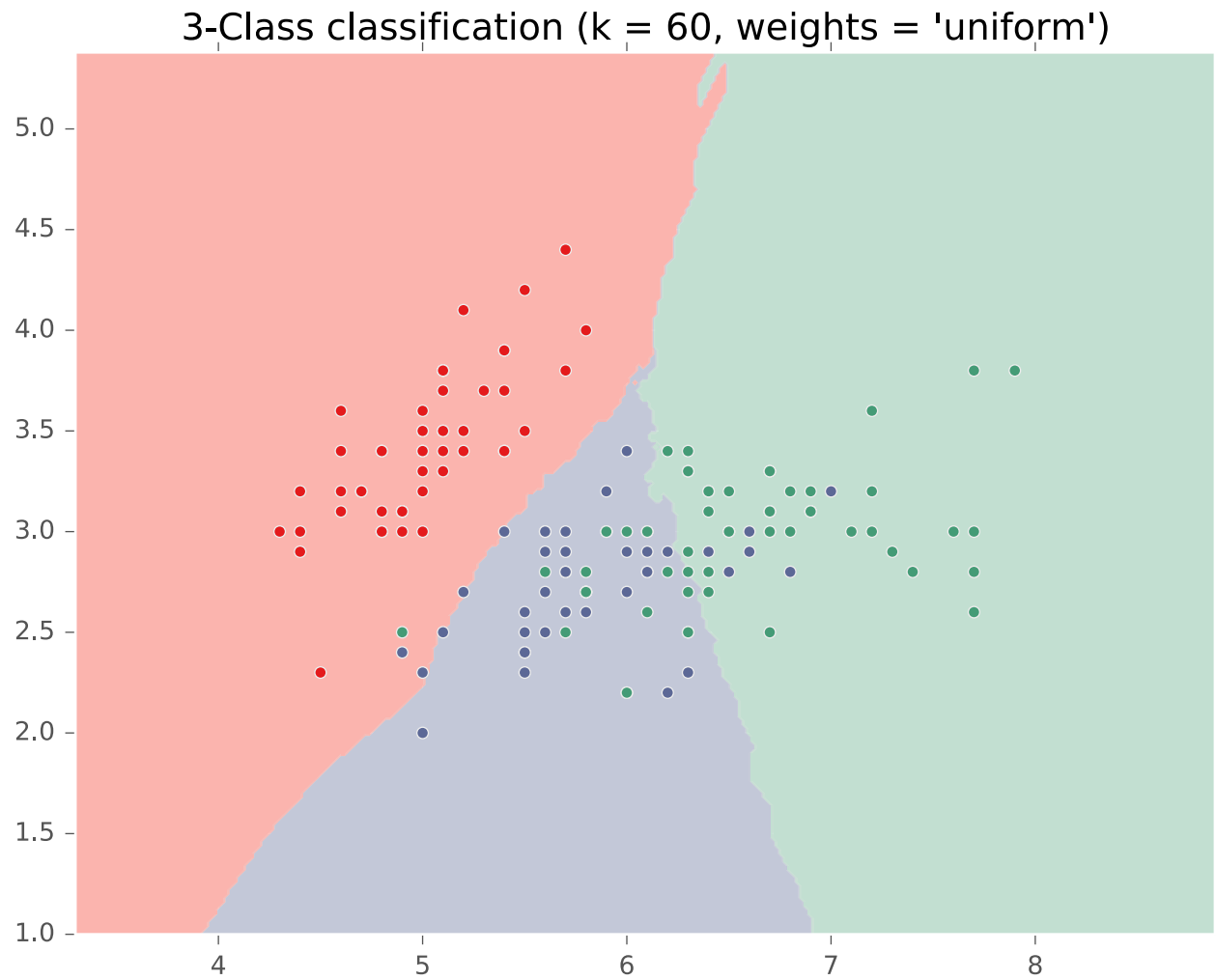




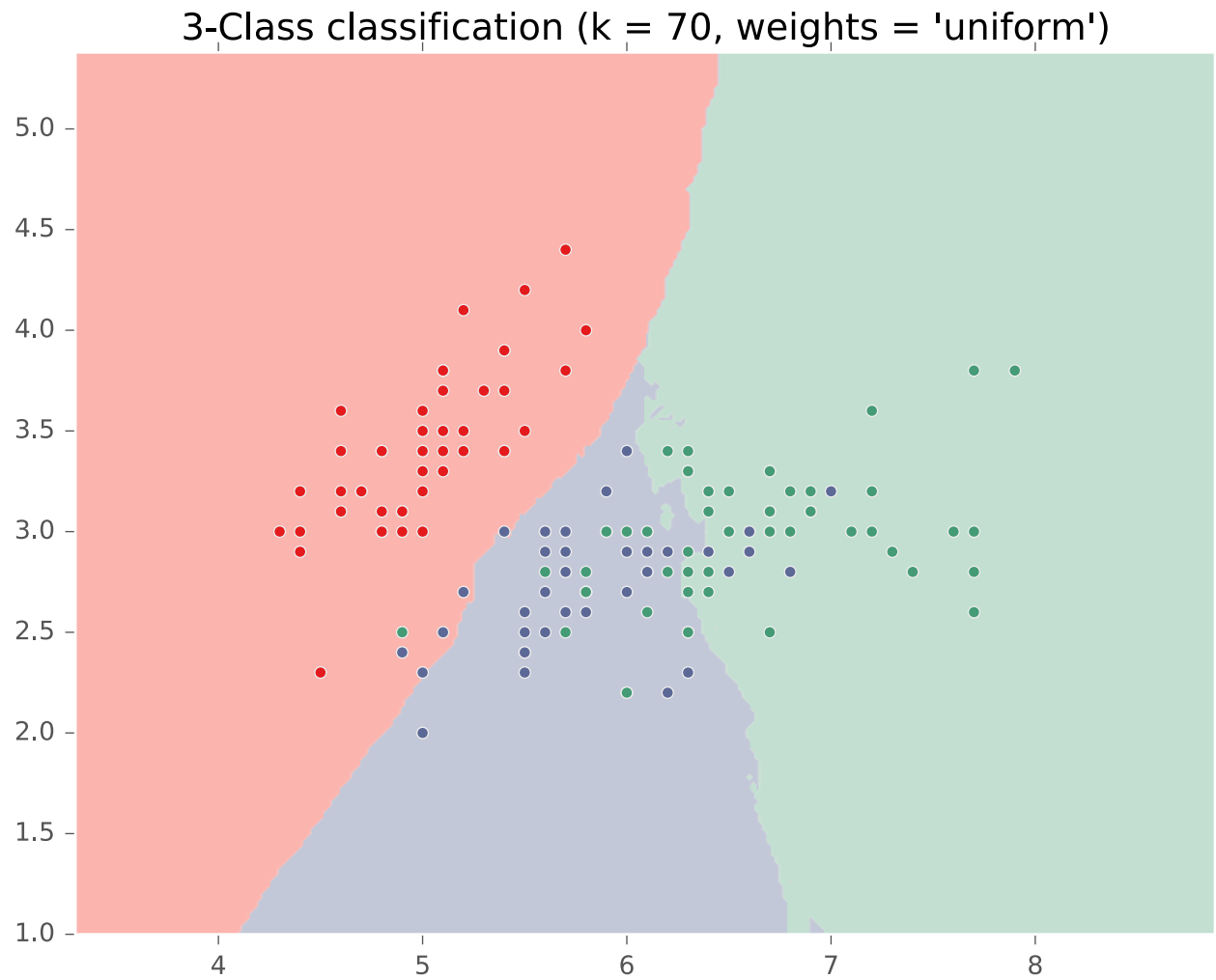
# KNN on Fisher Iris Data



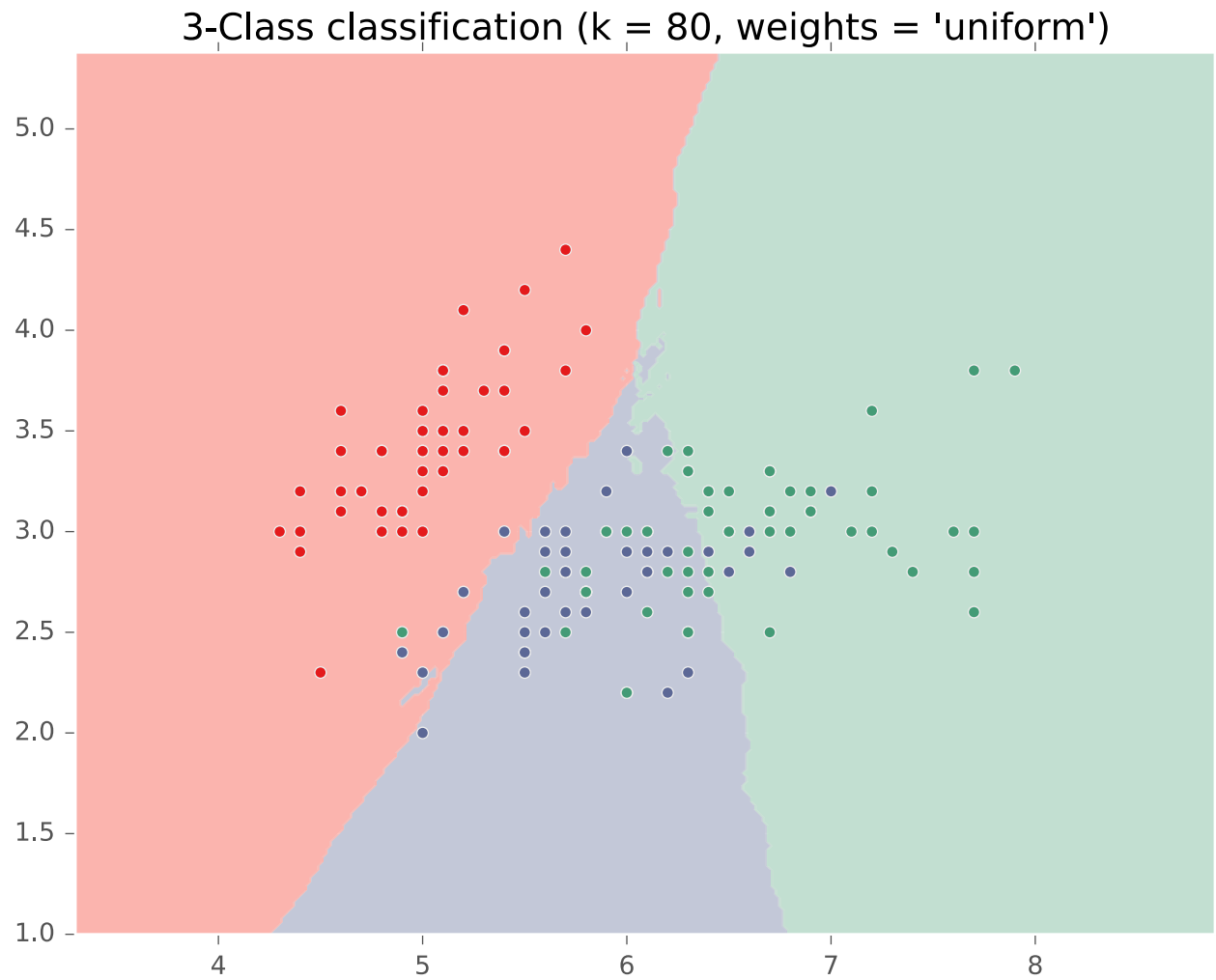
# KNN on Fisher Iris Data



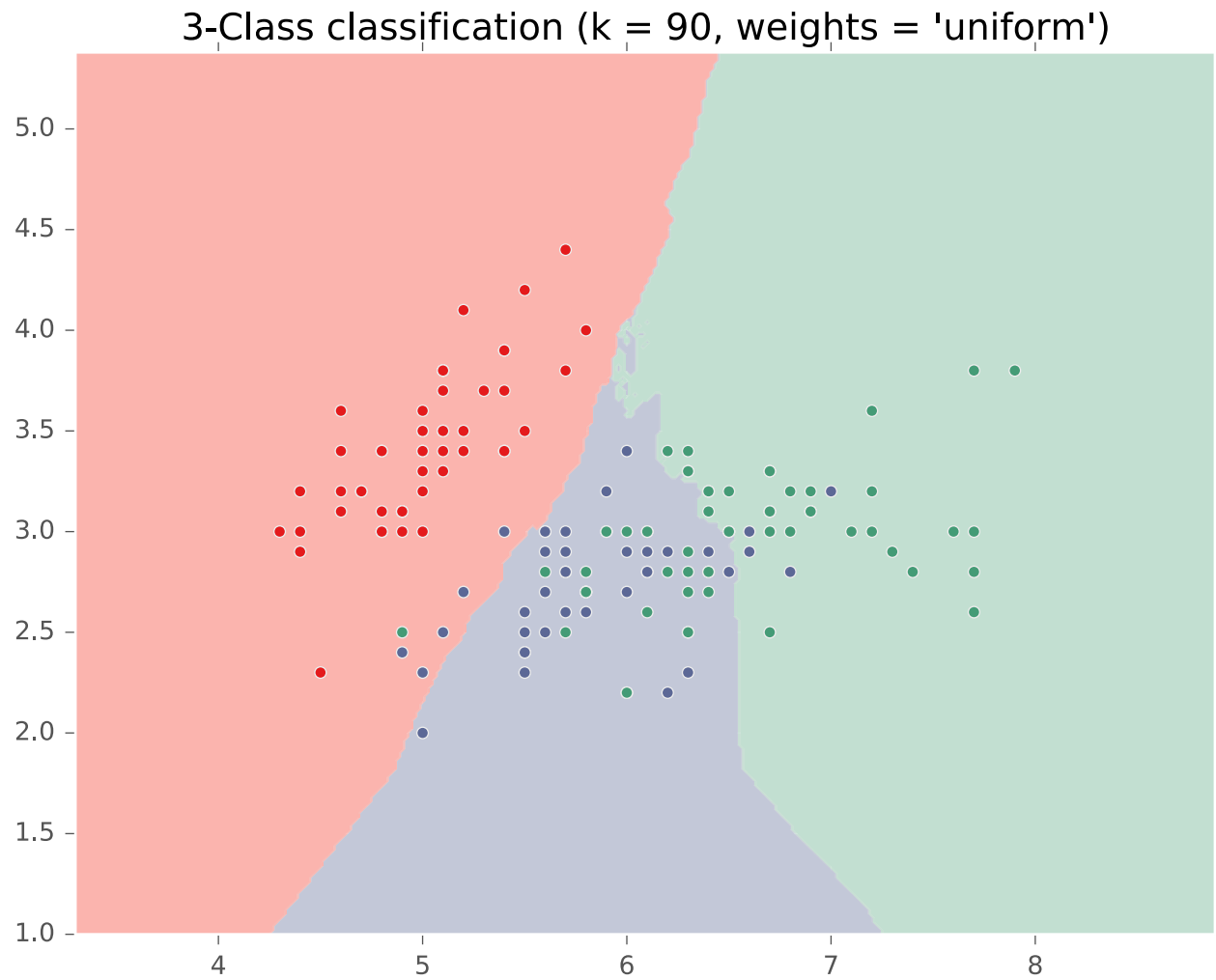
# KNN on Fisher Iris Data



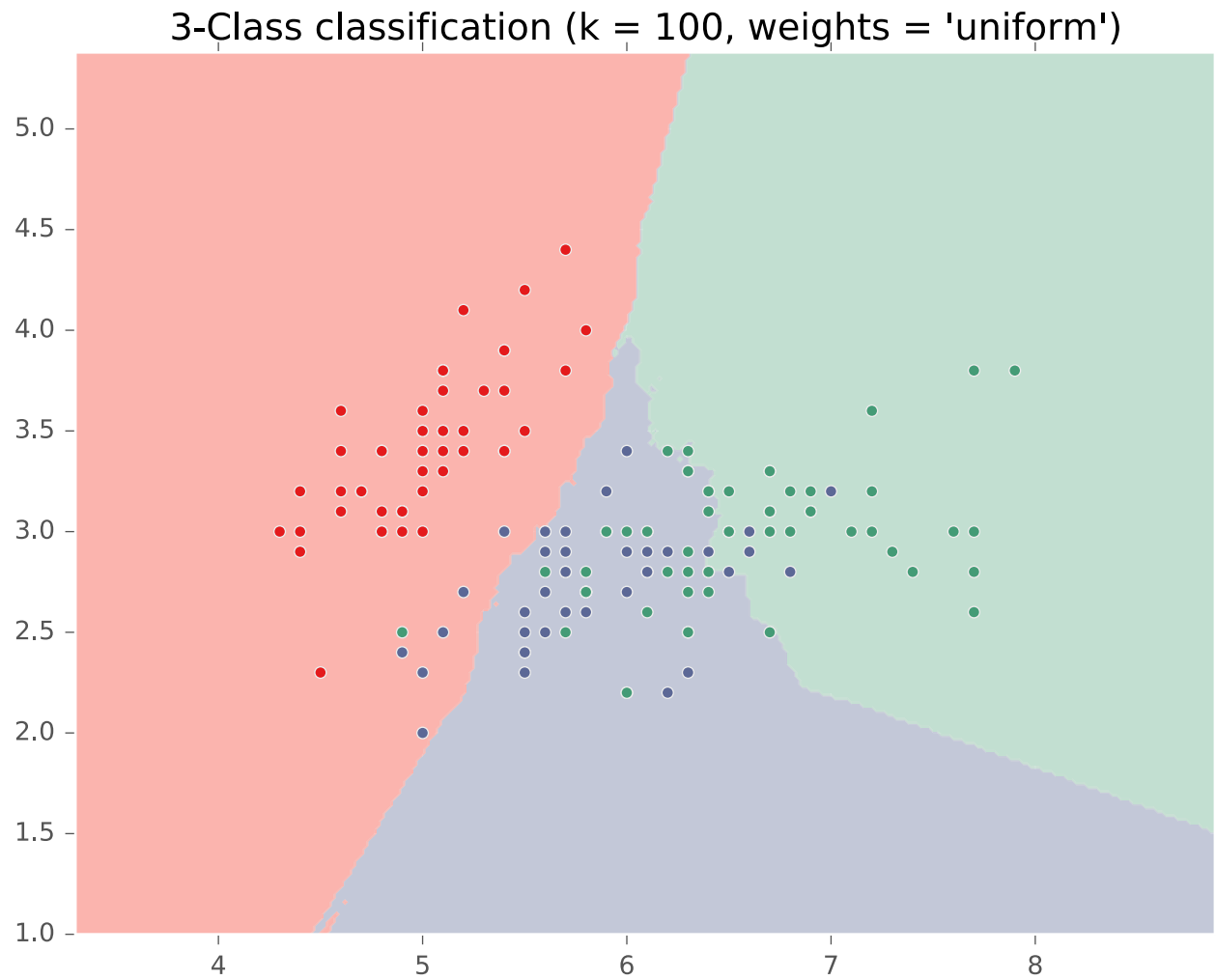
# KNN on Fisher Iris Data



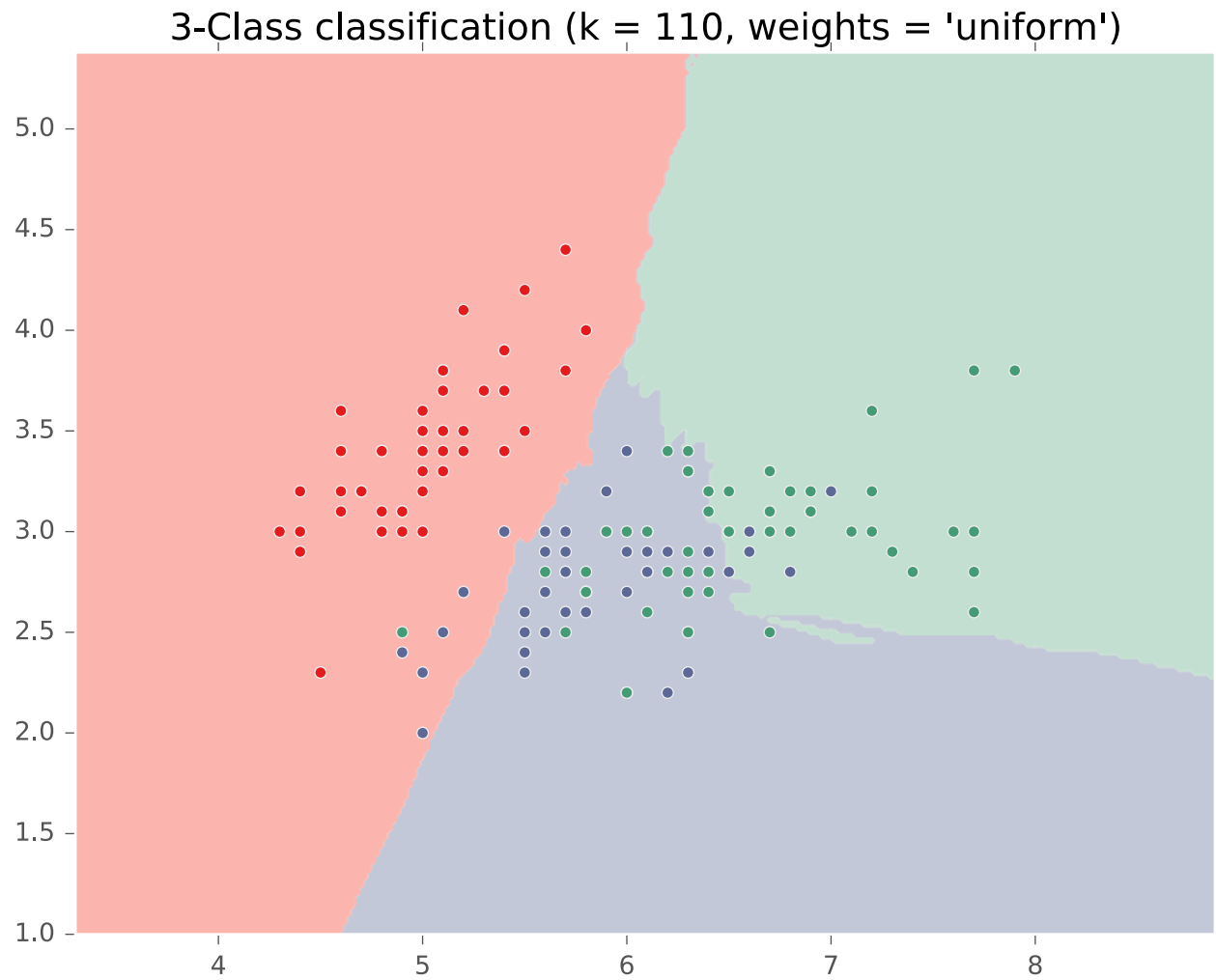
# KNN on Fisher Iris Data



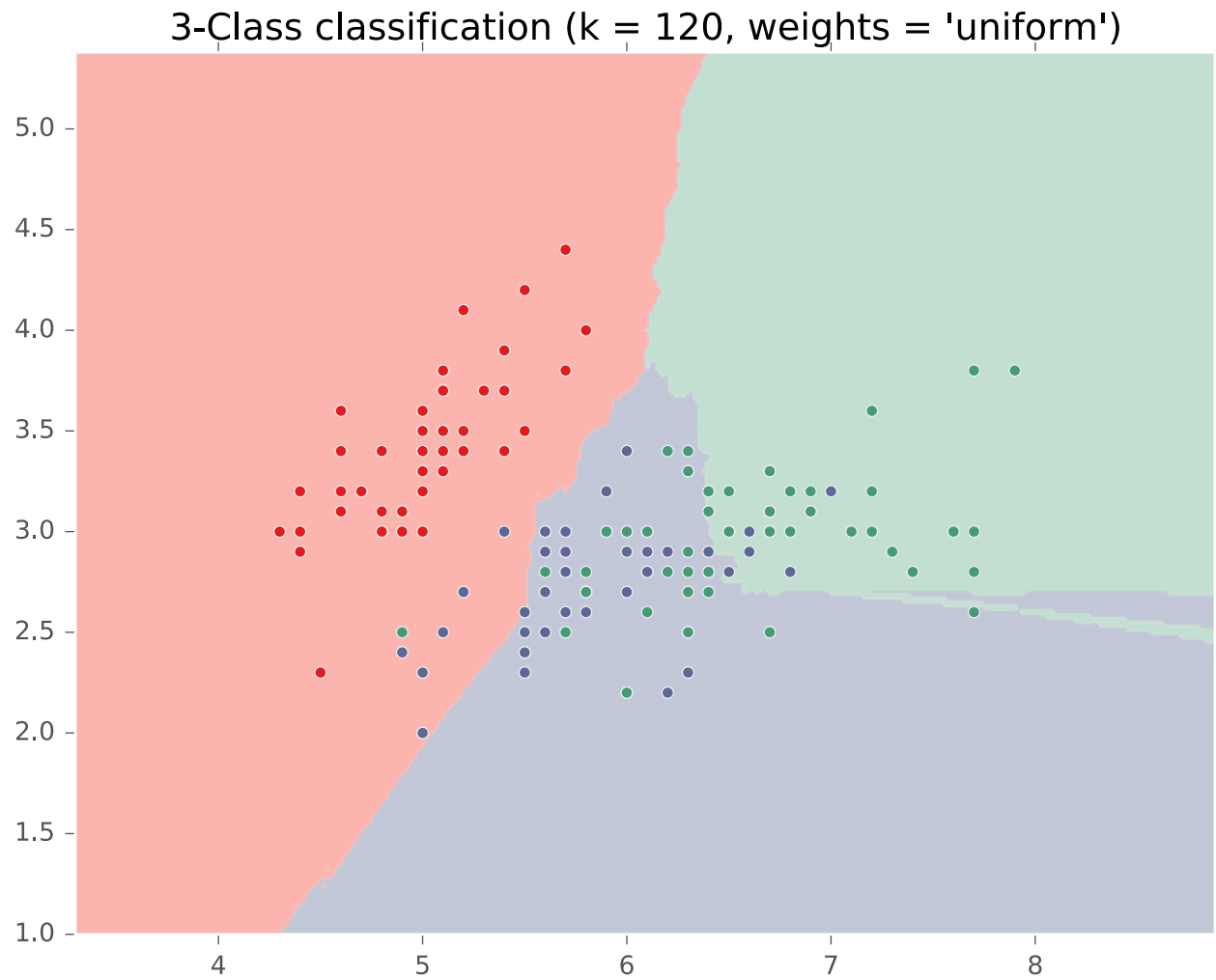
# KNN on Fisher Iris Data



# KNN on Fisher Iris Data

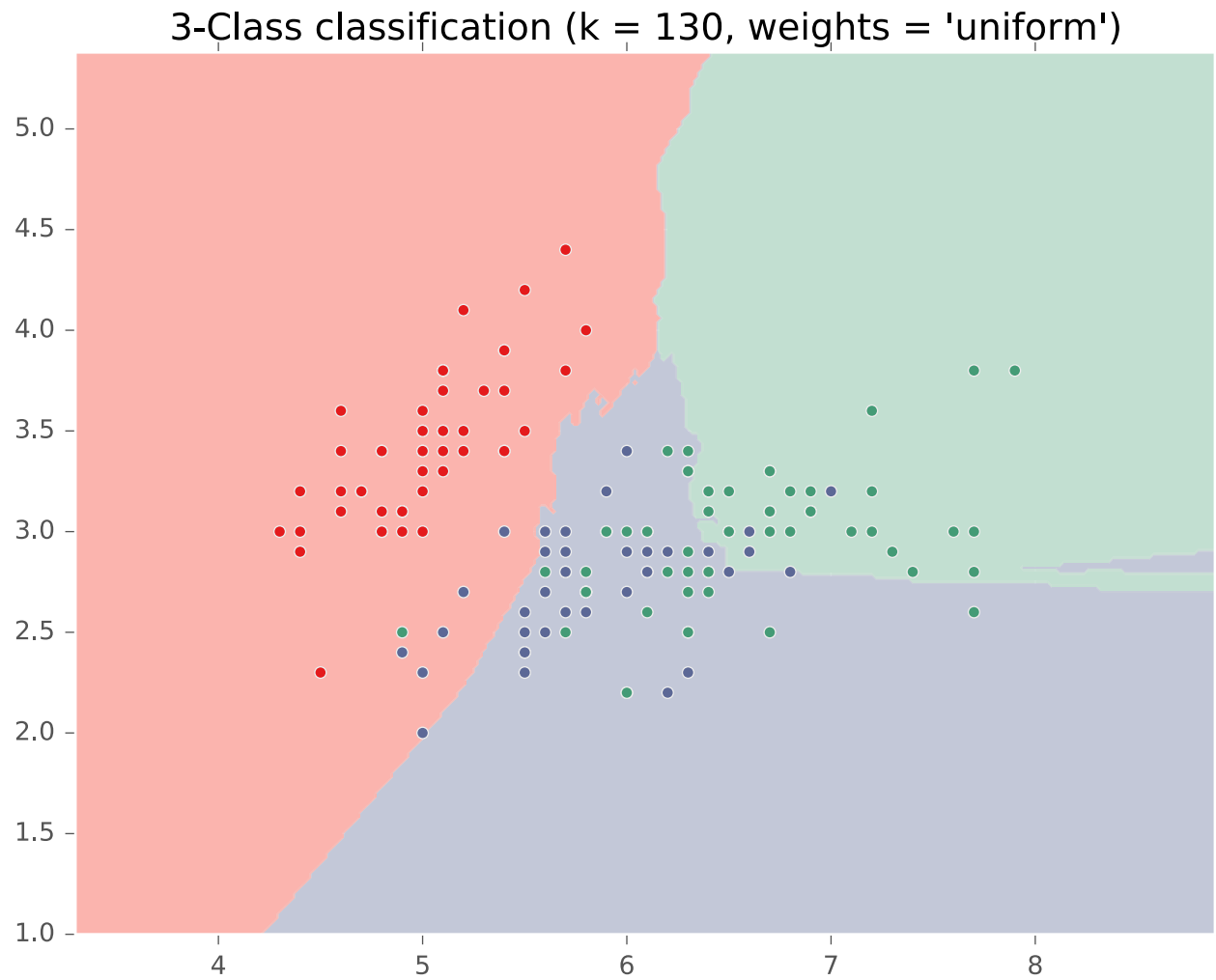


# KNN on Fisher Iris Data

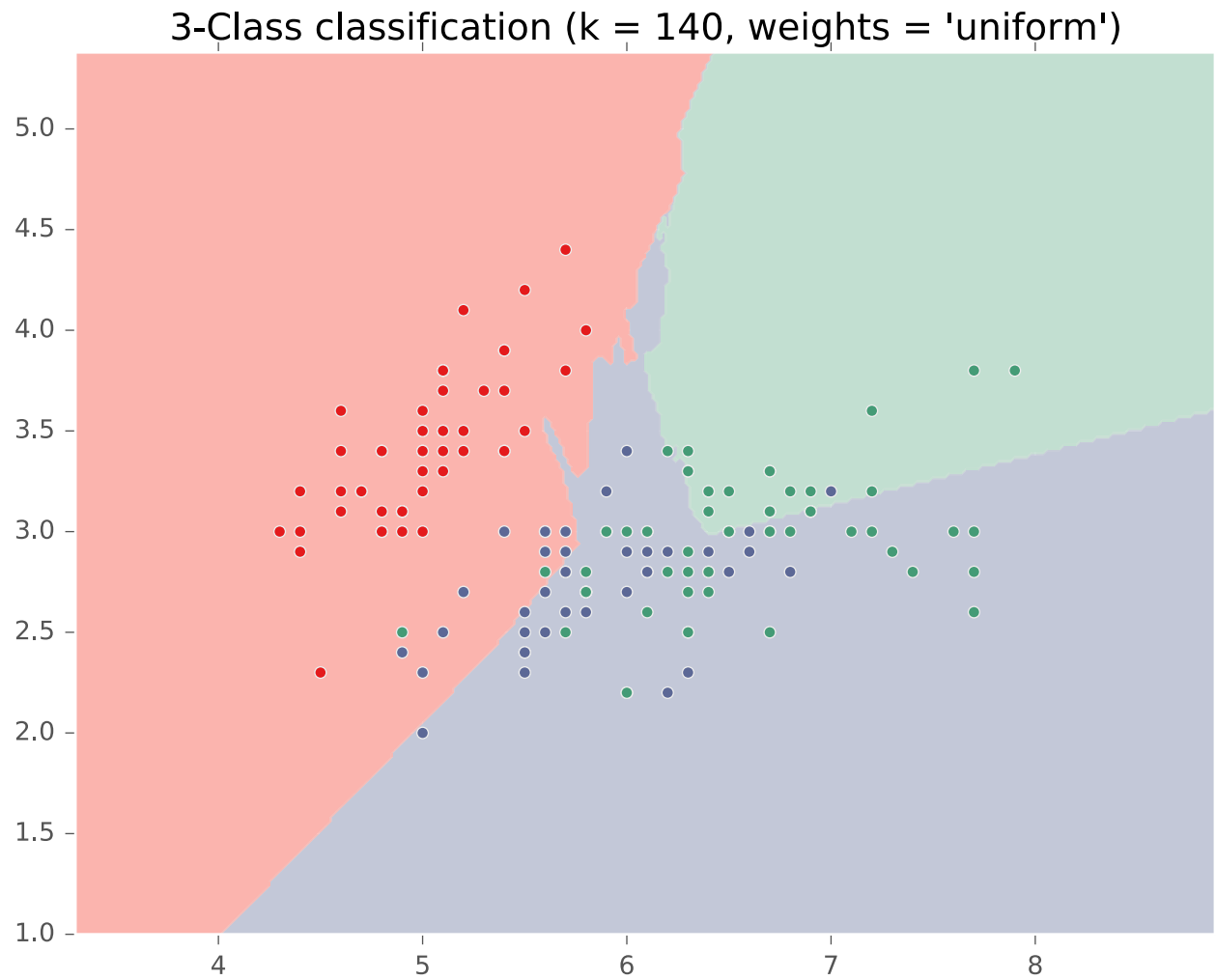




# KNN on Fisher Iris Data

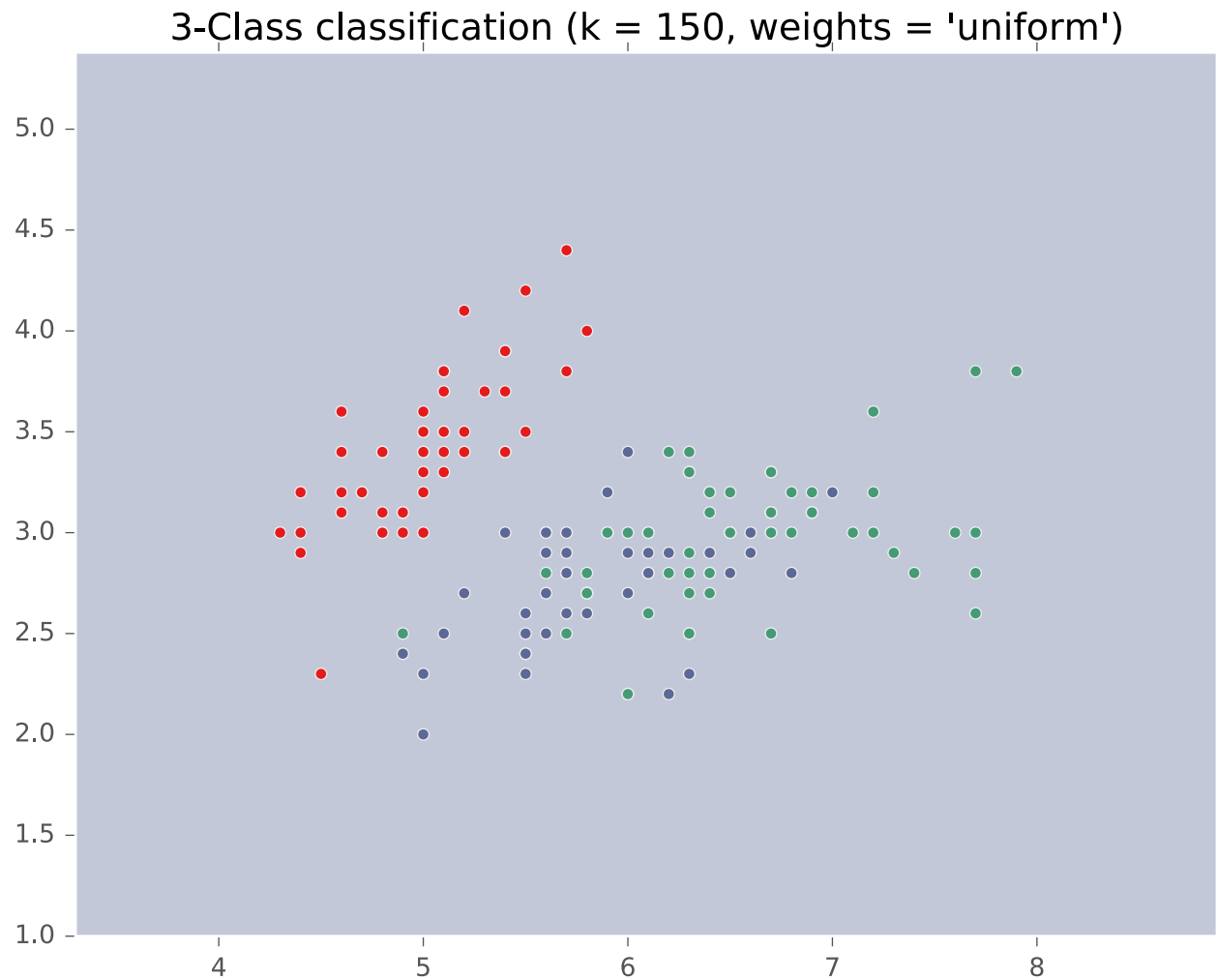


# KNN on Fisher Iris Data



# KNN on Fisher Iris Data

## Special Case: Majority Vote

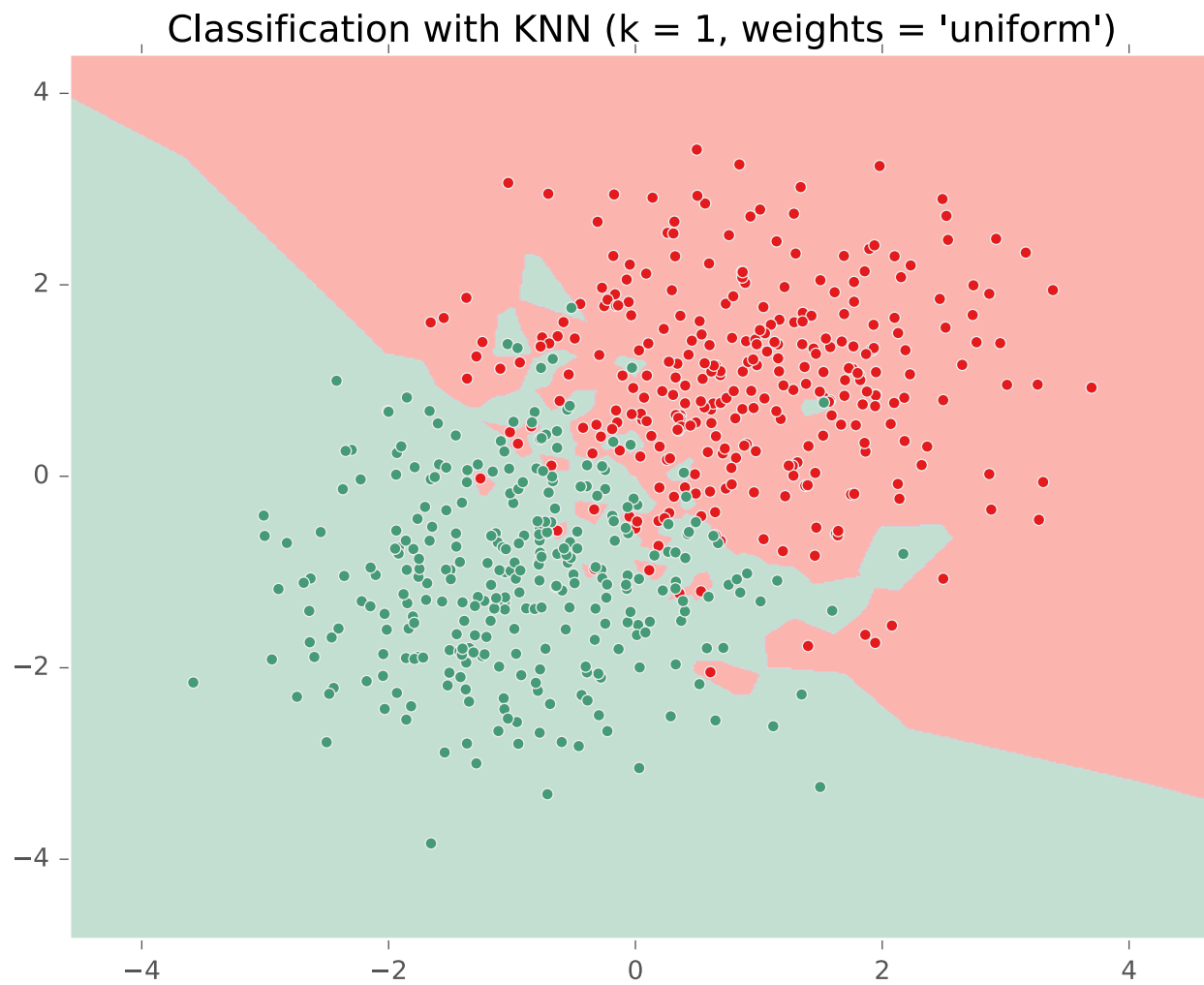


# **KNN ON GAUSSIAN DATA**

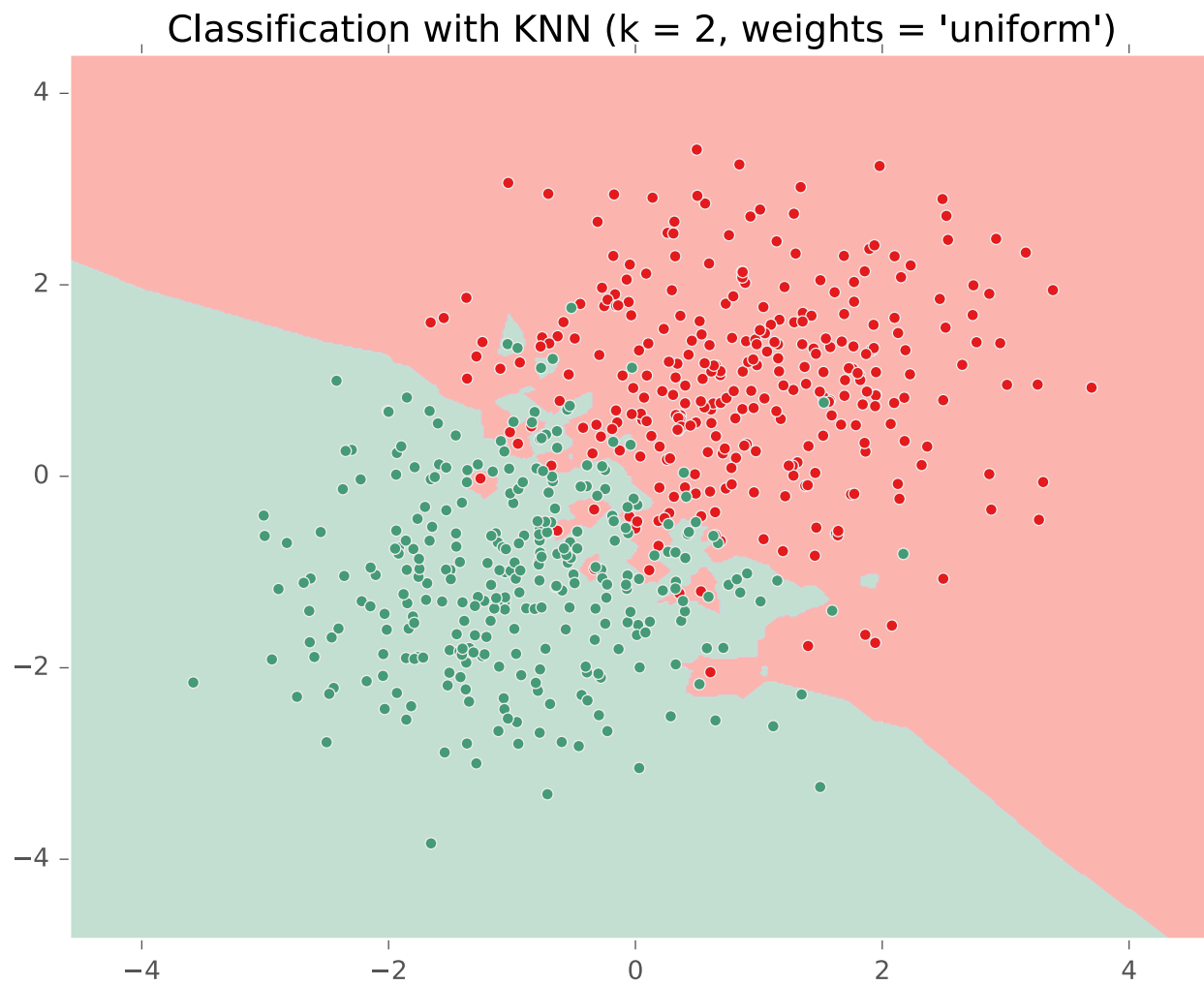
# KNN on Gaussian Data



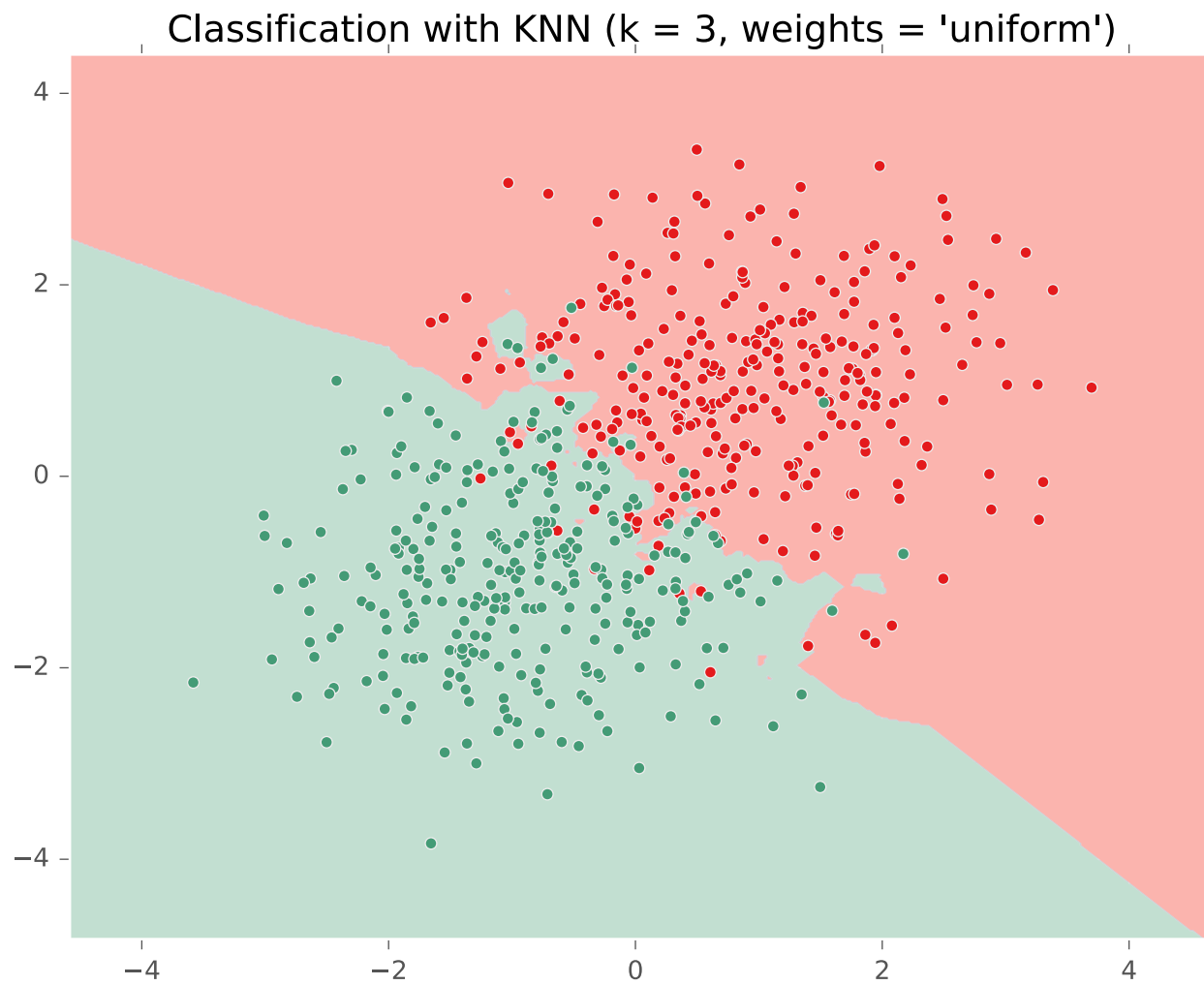
# KNN on Gaussian Data



# KNN on Gaussian Data

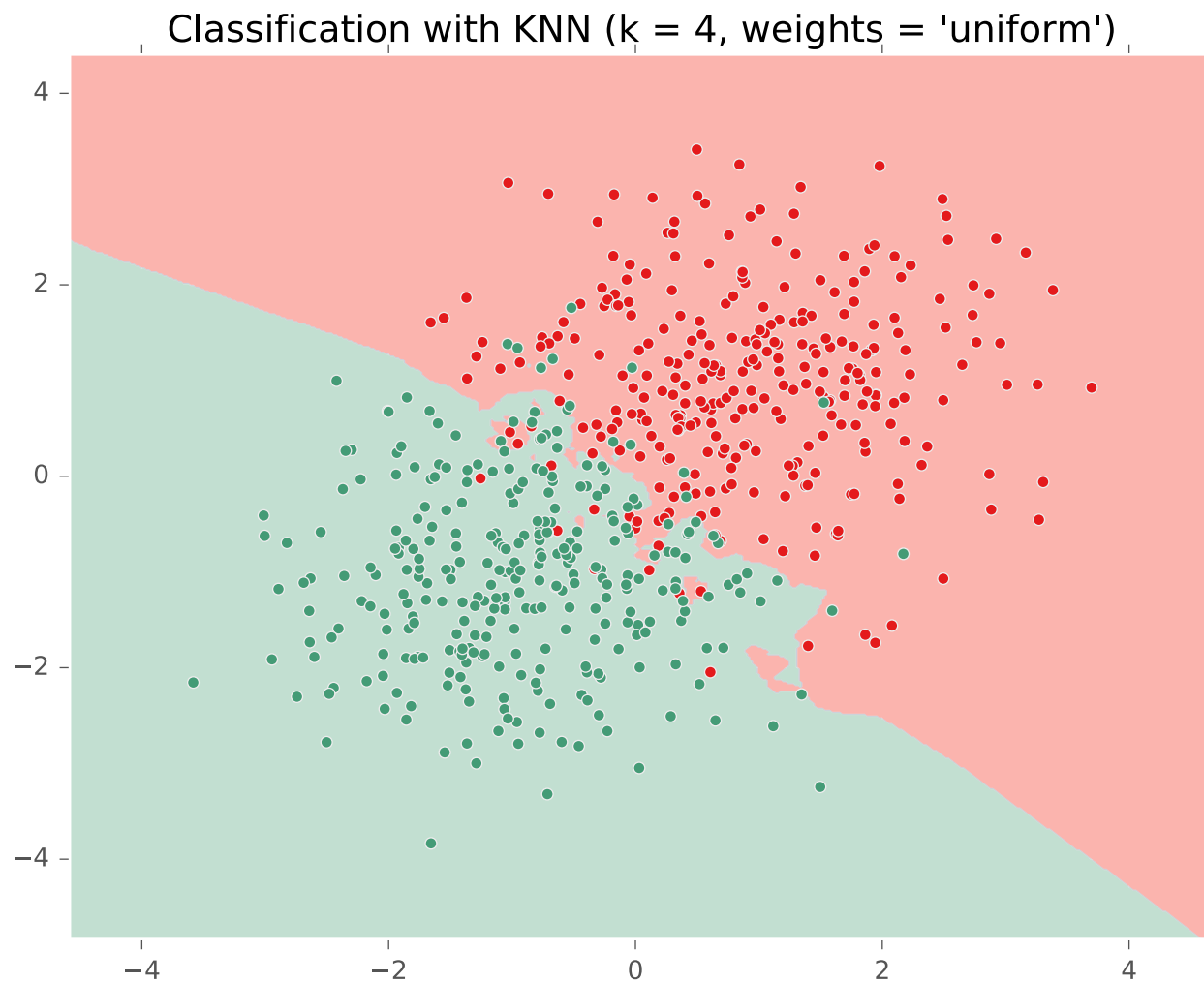


# KNN on Gaussian Data

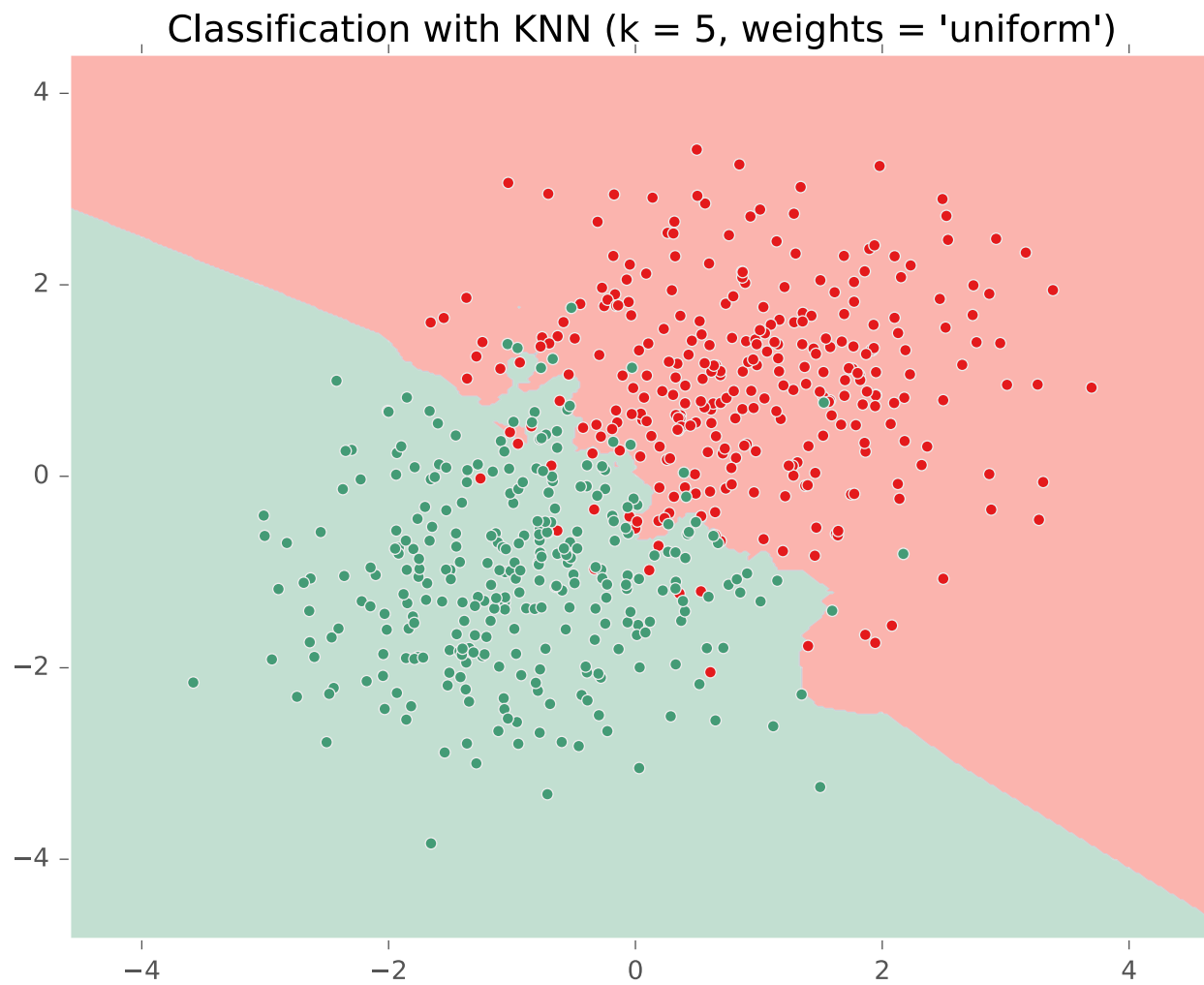




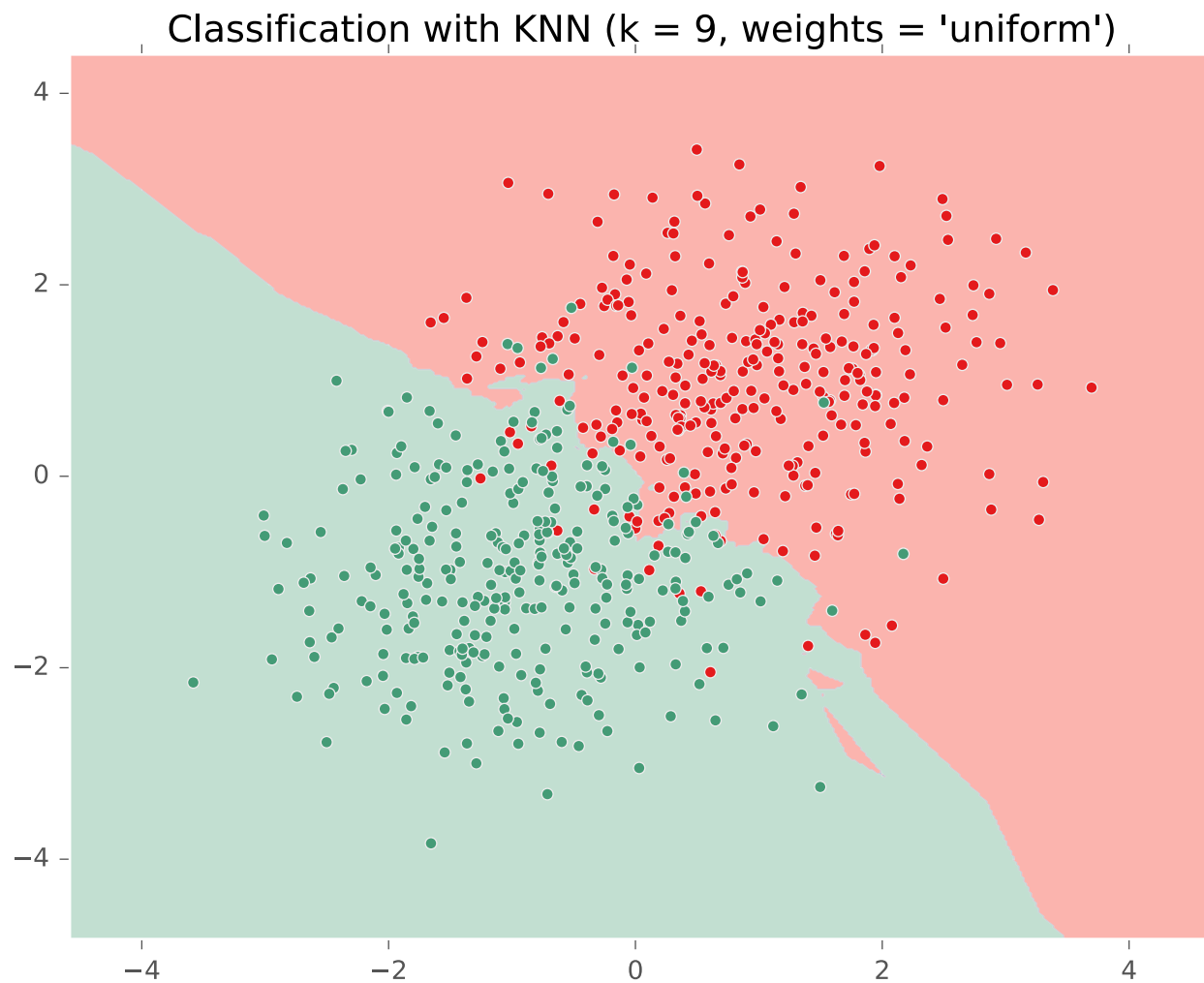
# KNN on Gaussian Data



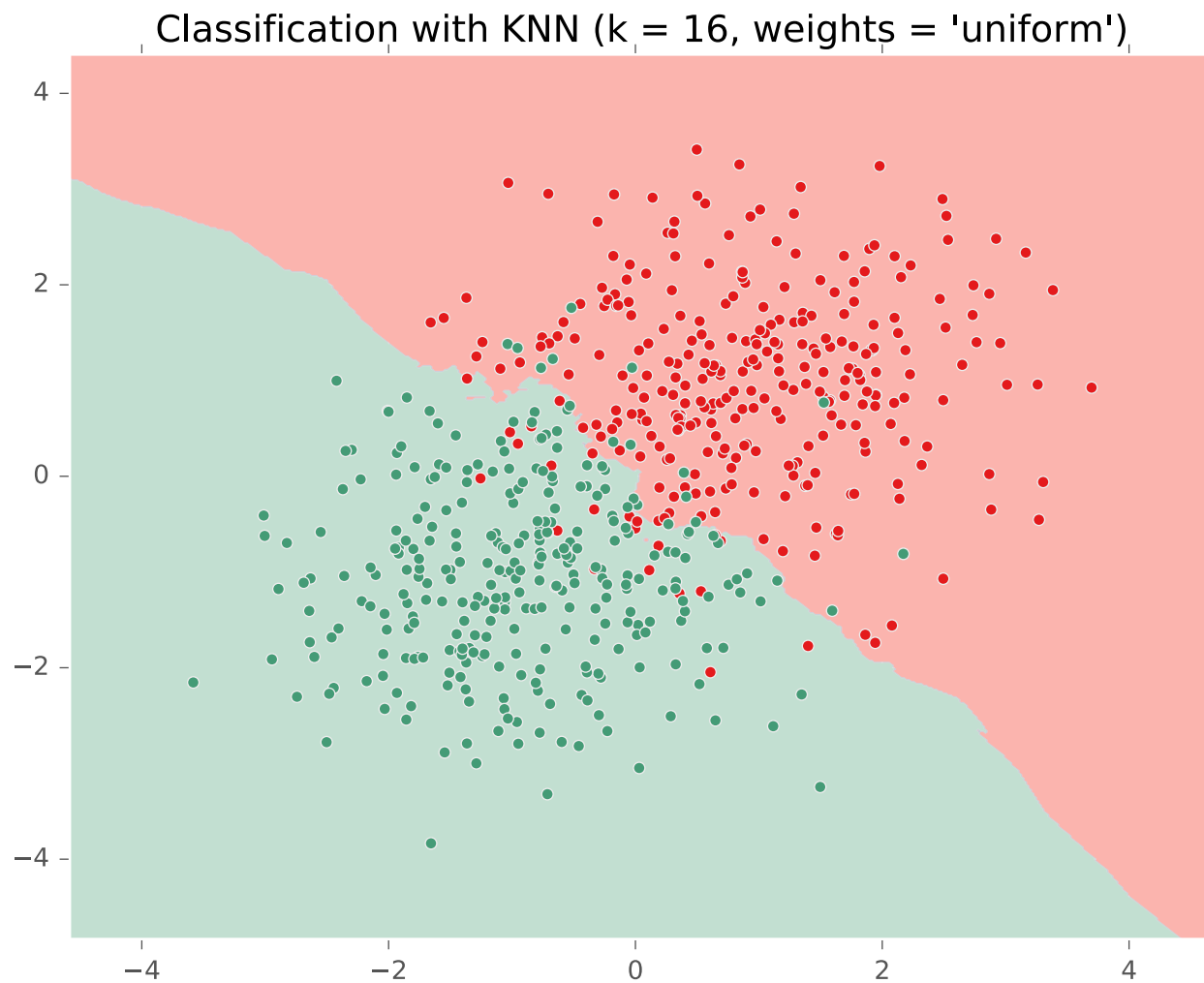
# KNN on Gaussian Data



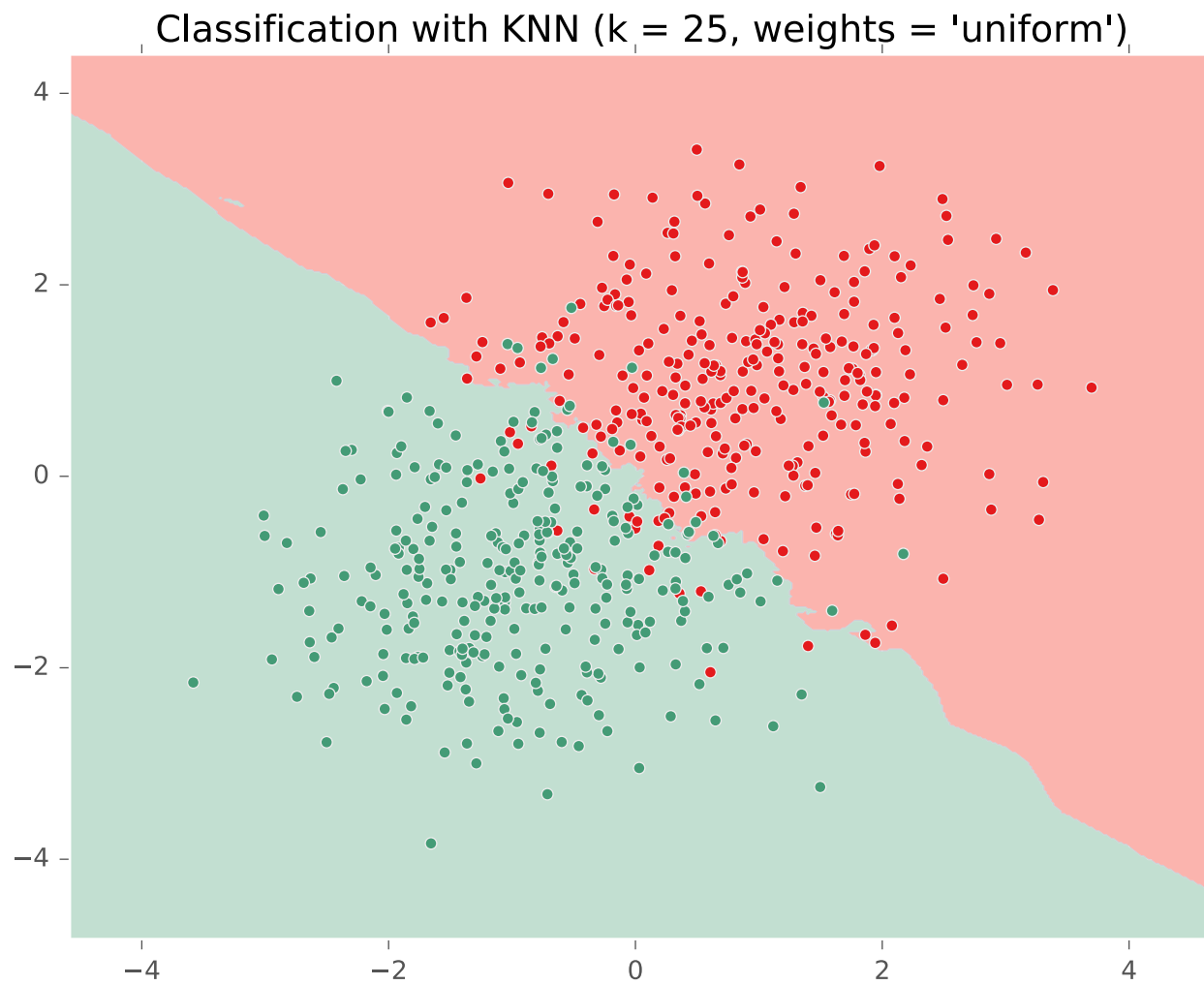
# KNN on Gaussian Data



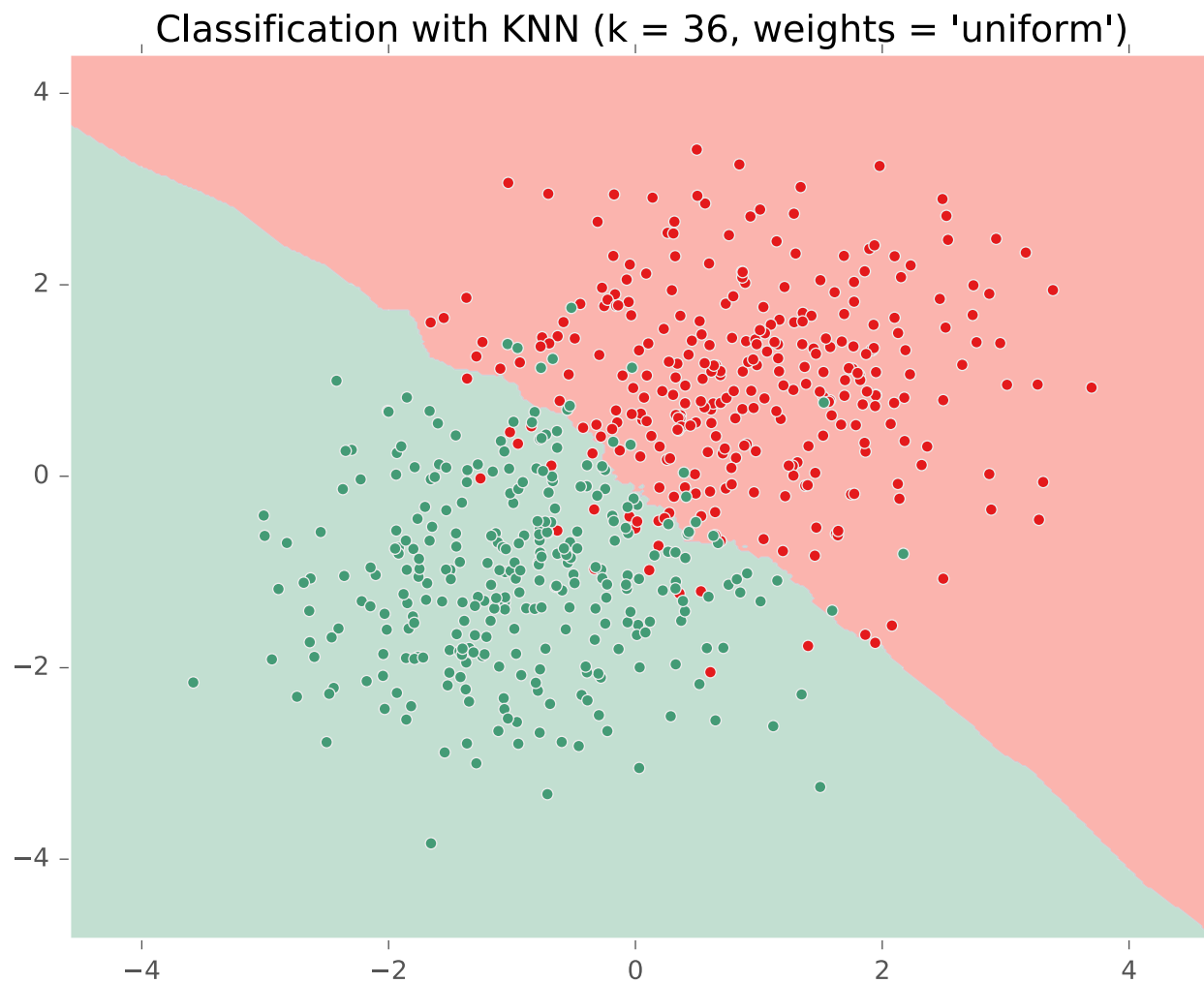
# KNN on Gaussian Data



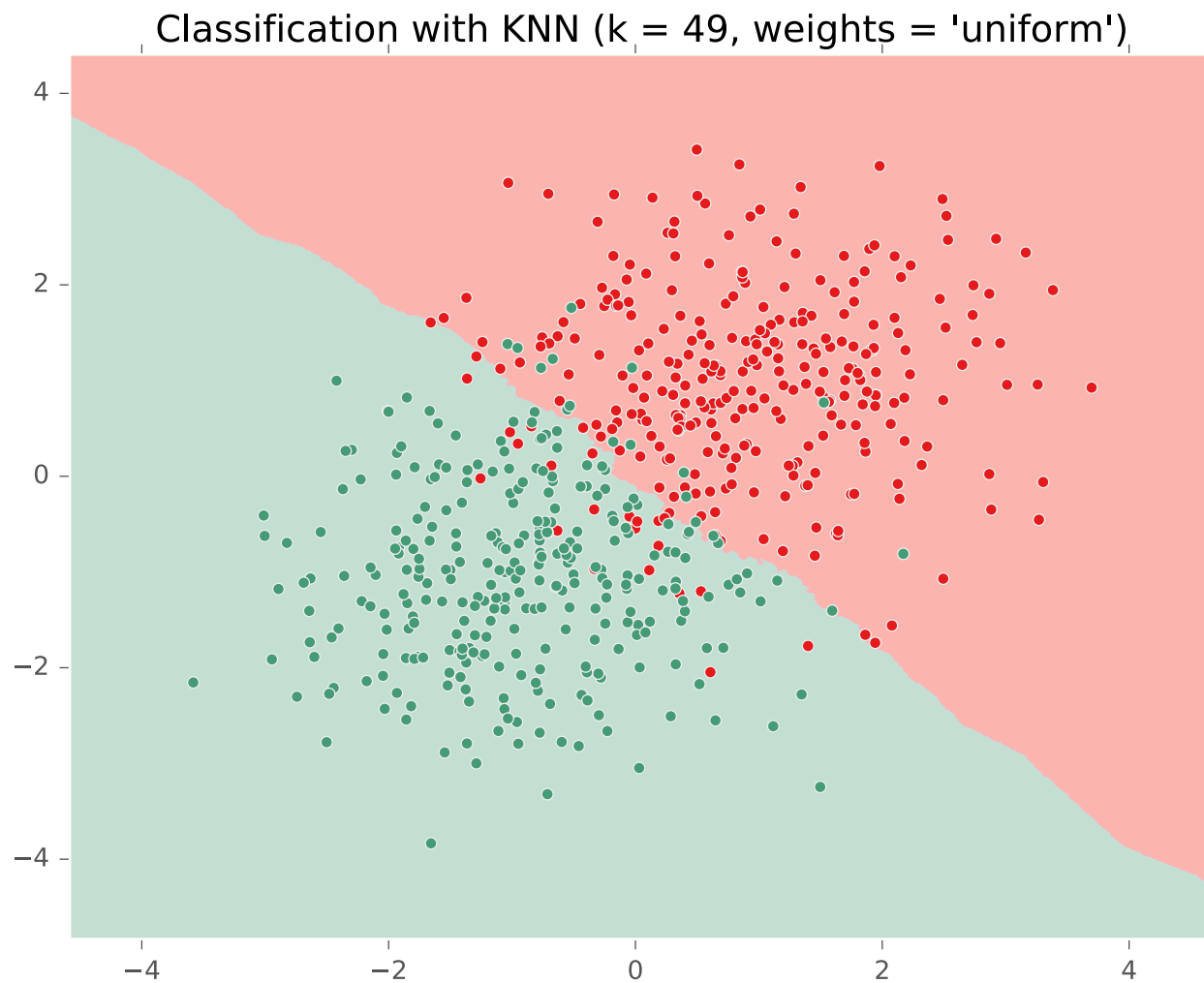
# KNN on Gaussian Data



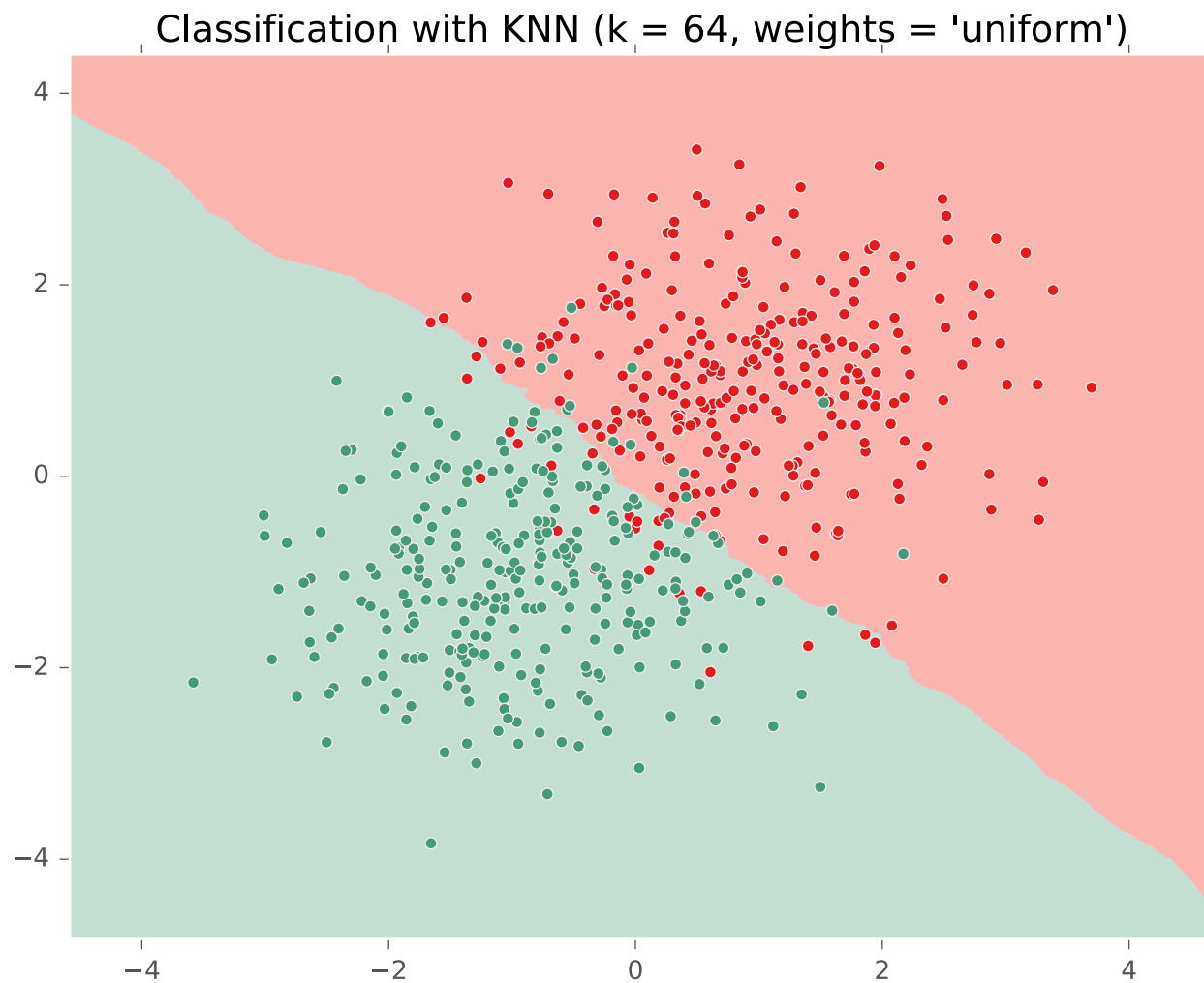
# KNN on Gaussian Data



# KNN on Gaussian Data

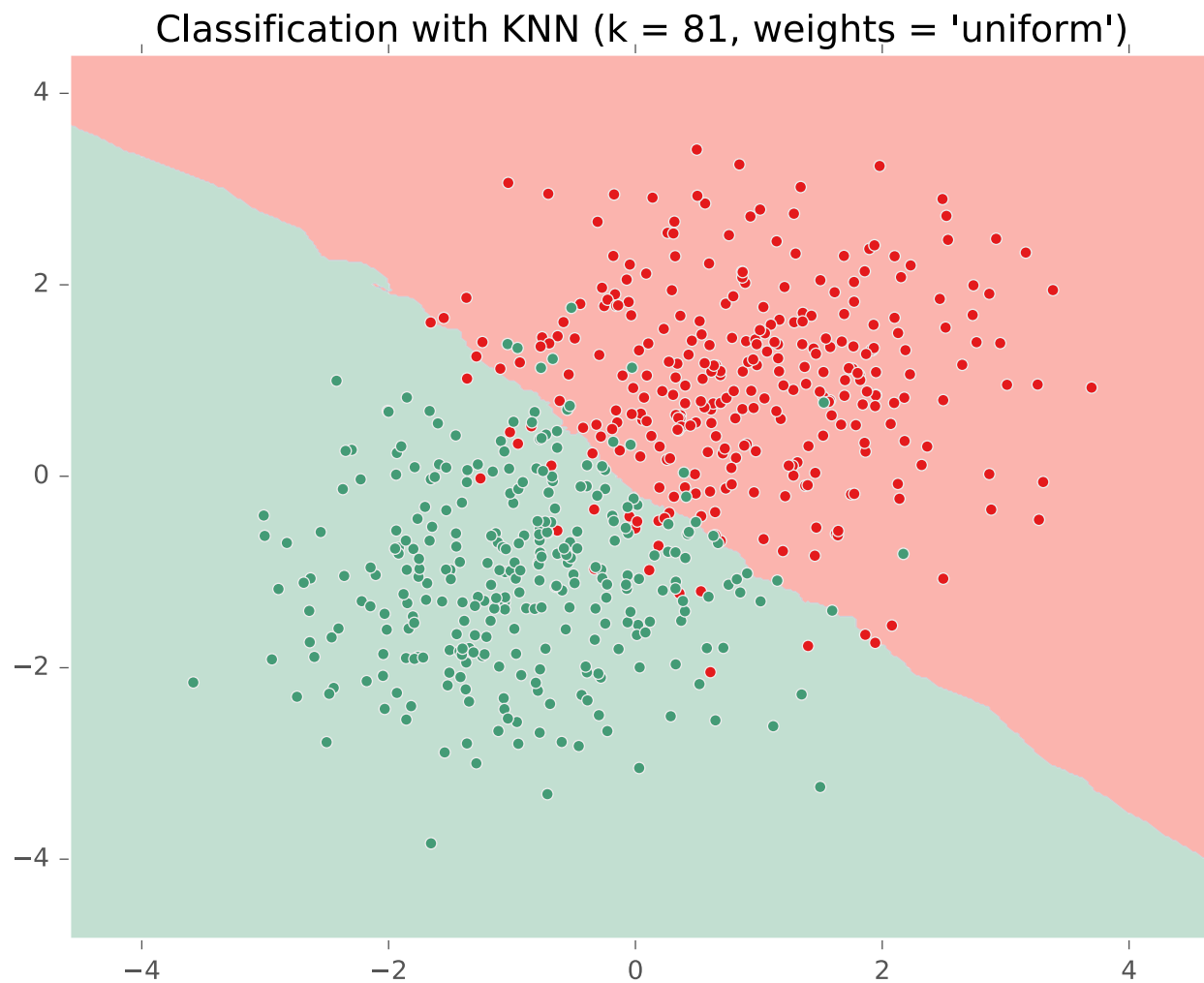


# KNN on Gaussian Data

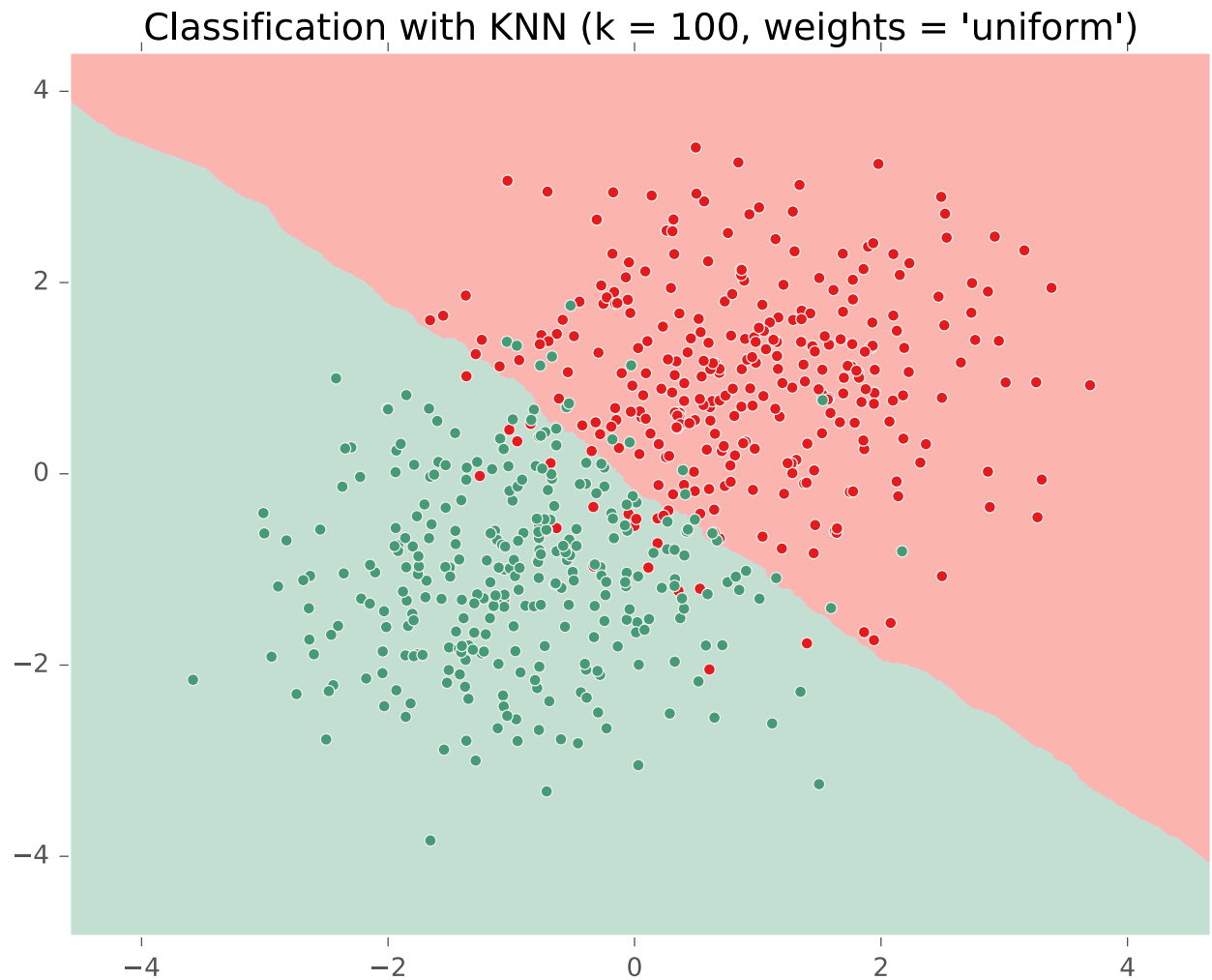




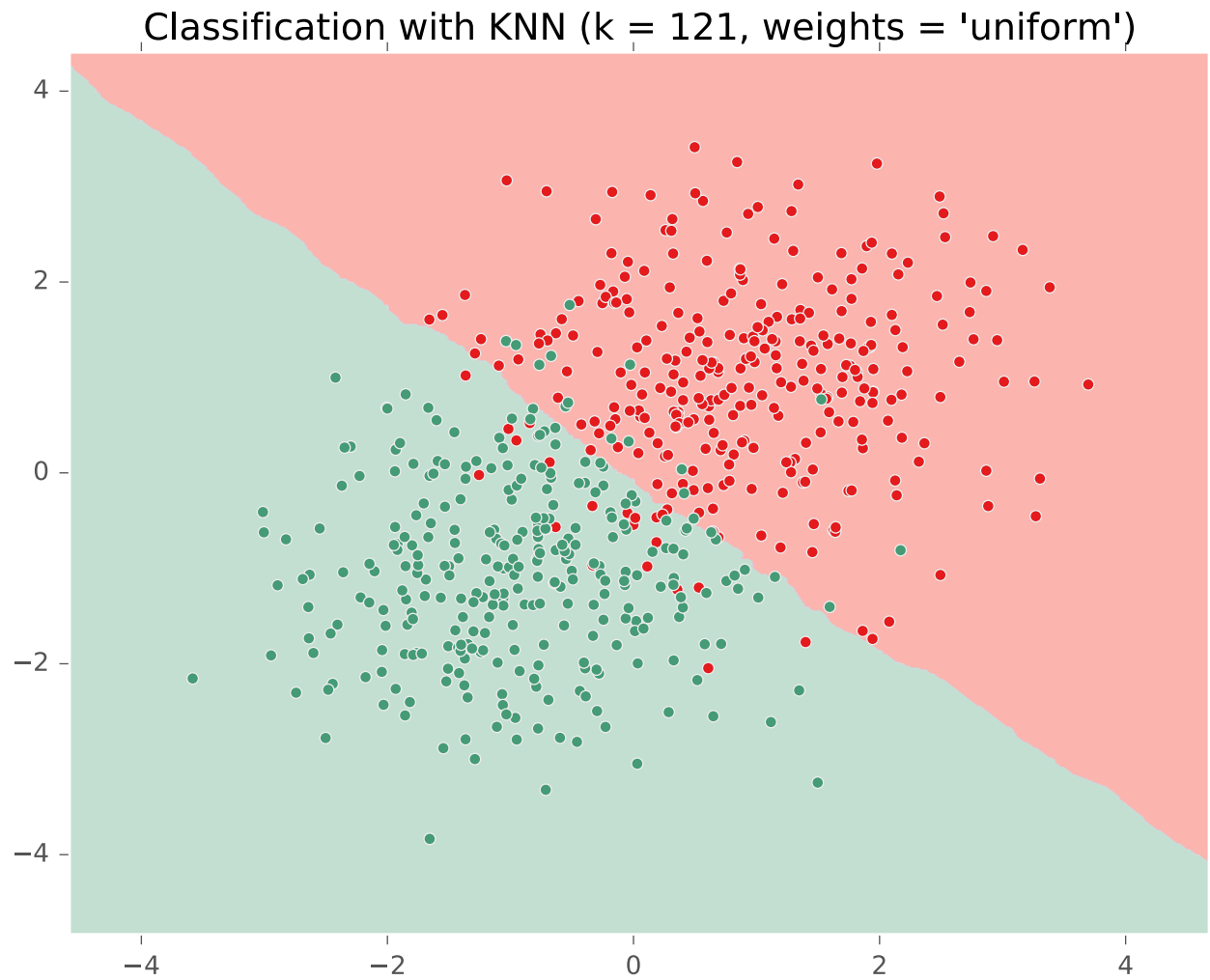
# KNN on Gaussian Data



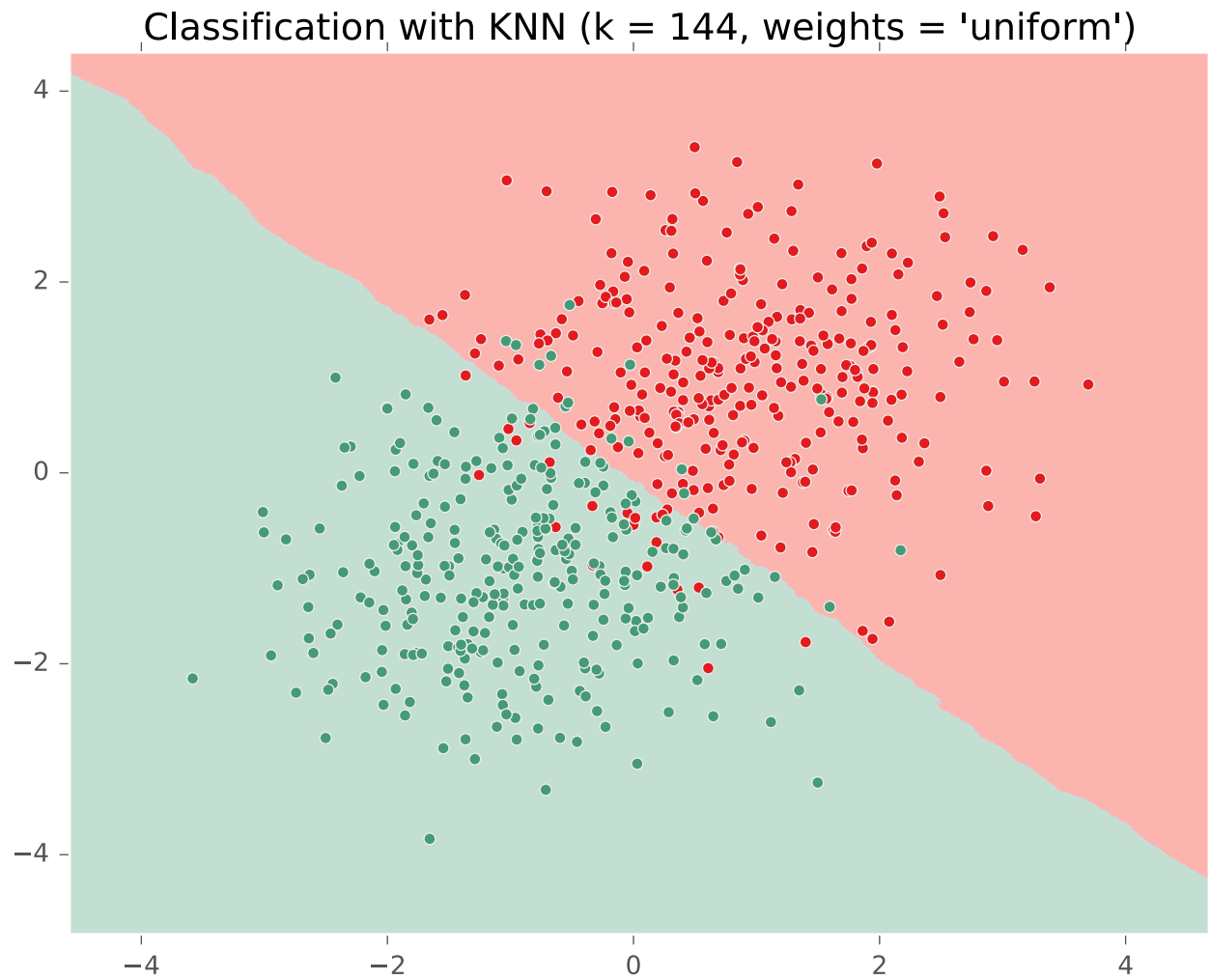
# KNN on Gaussian Data



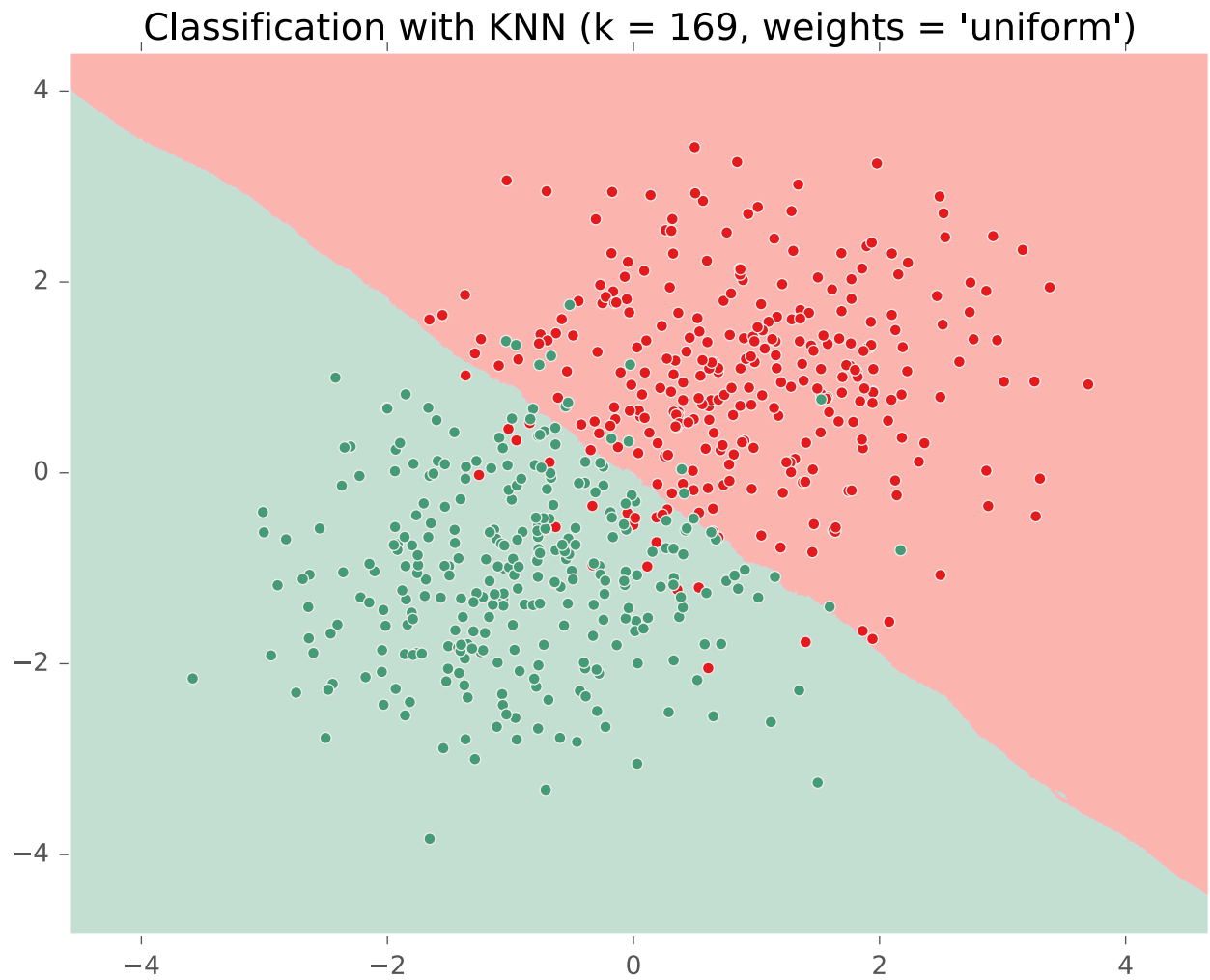
# KNN on Gaussian Data



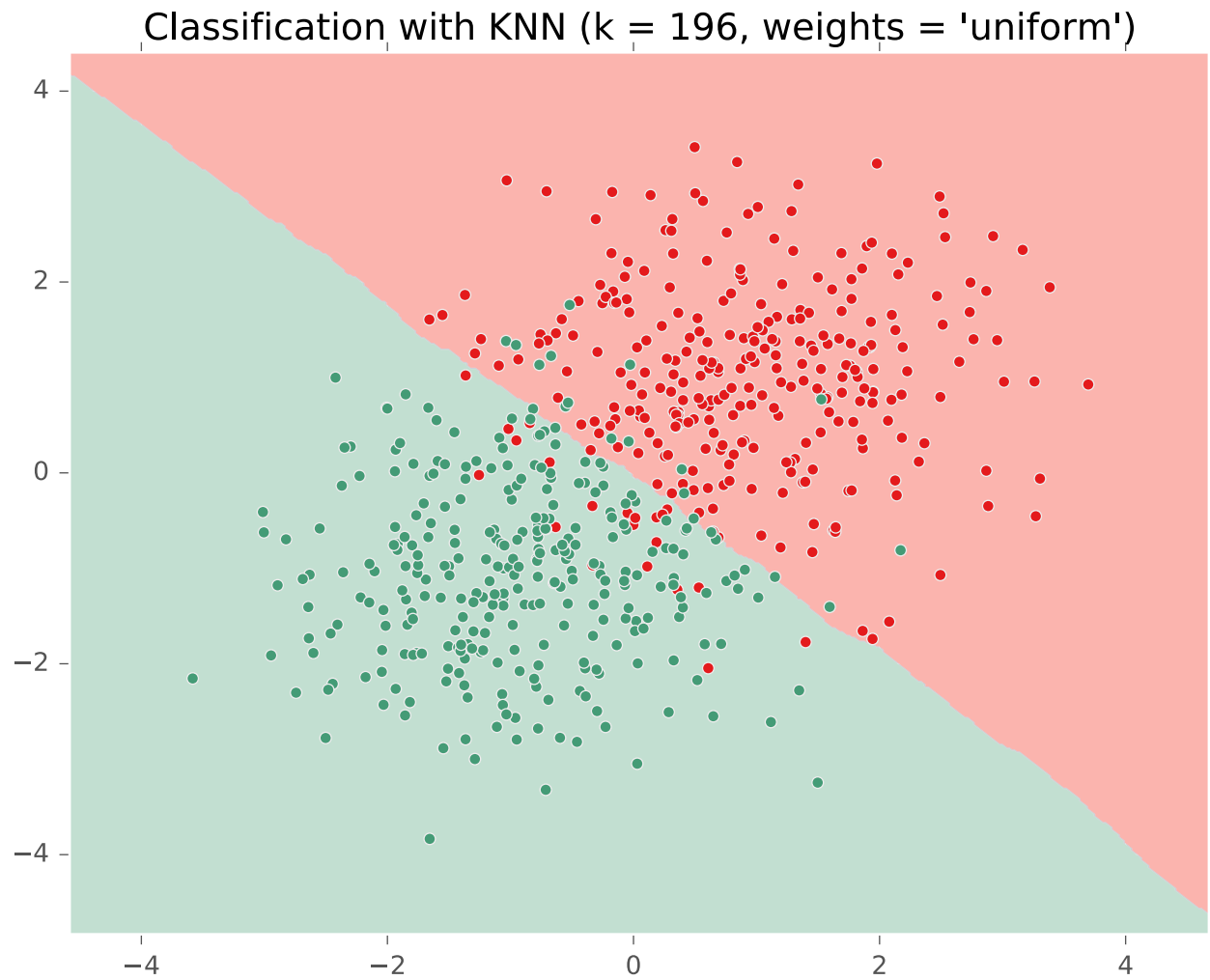
# KNN on Gaussian Data



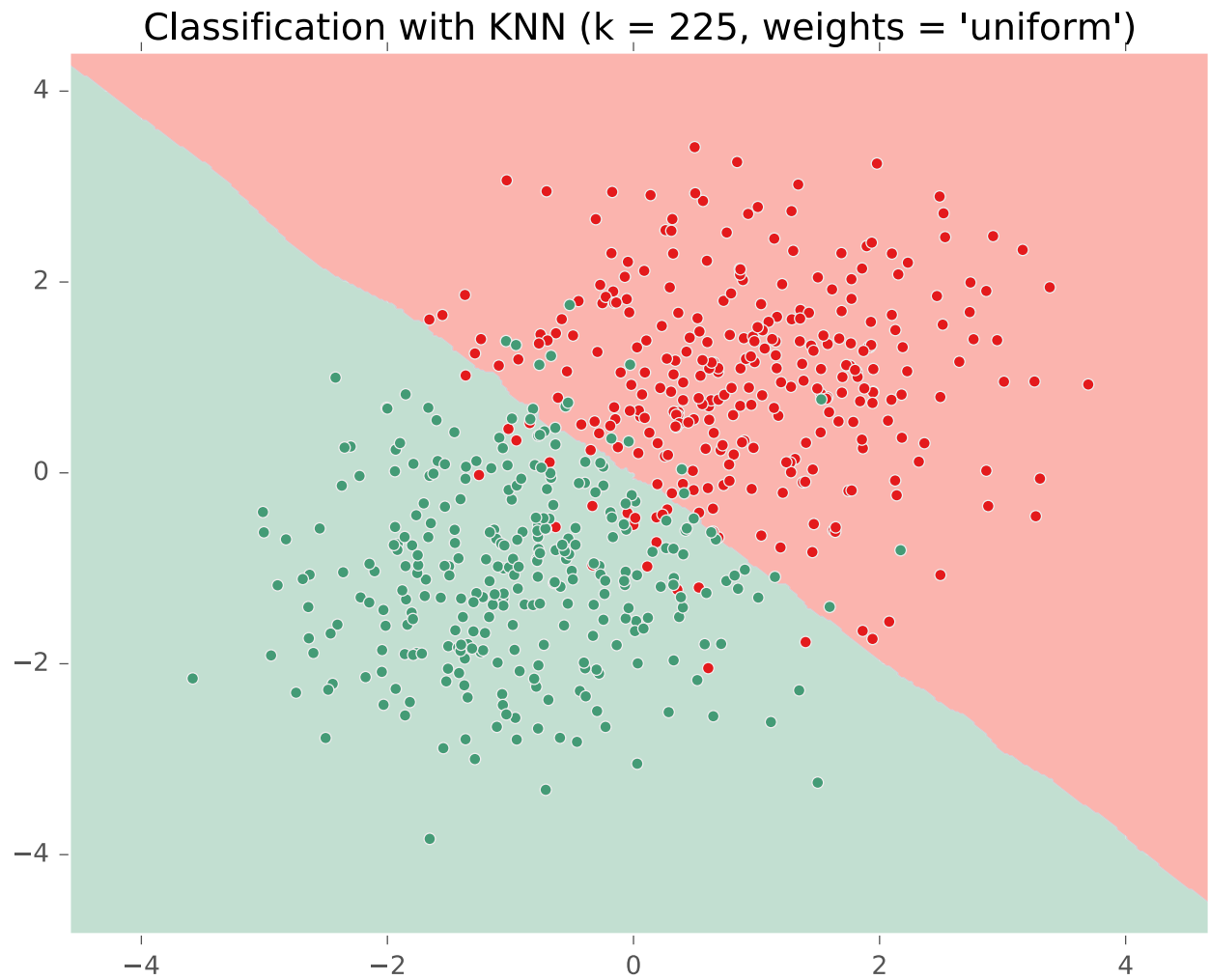
# KNN on Gaussian Data



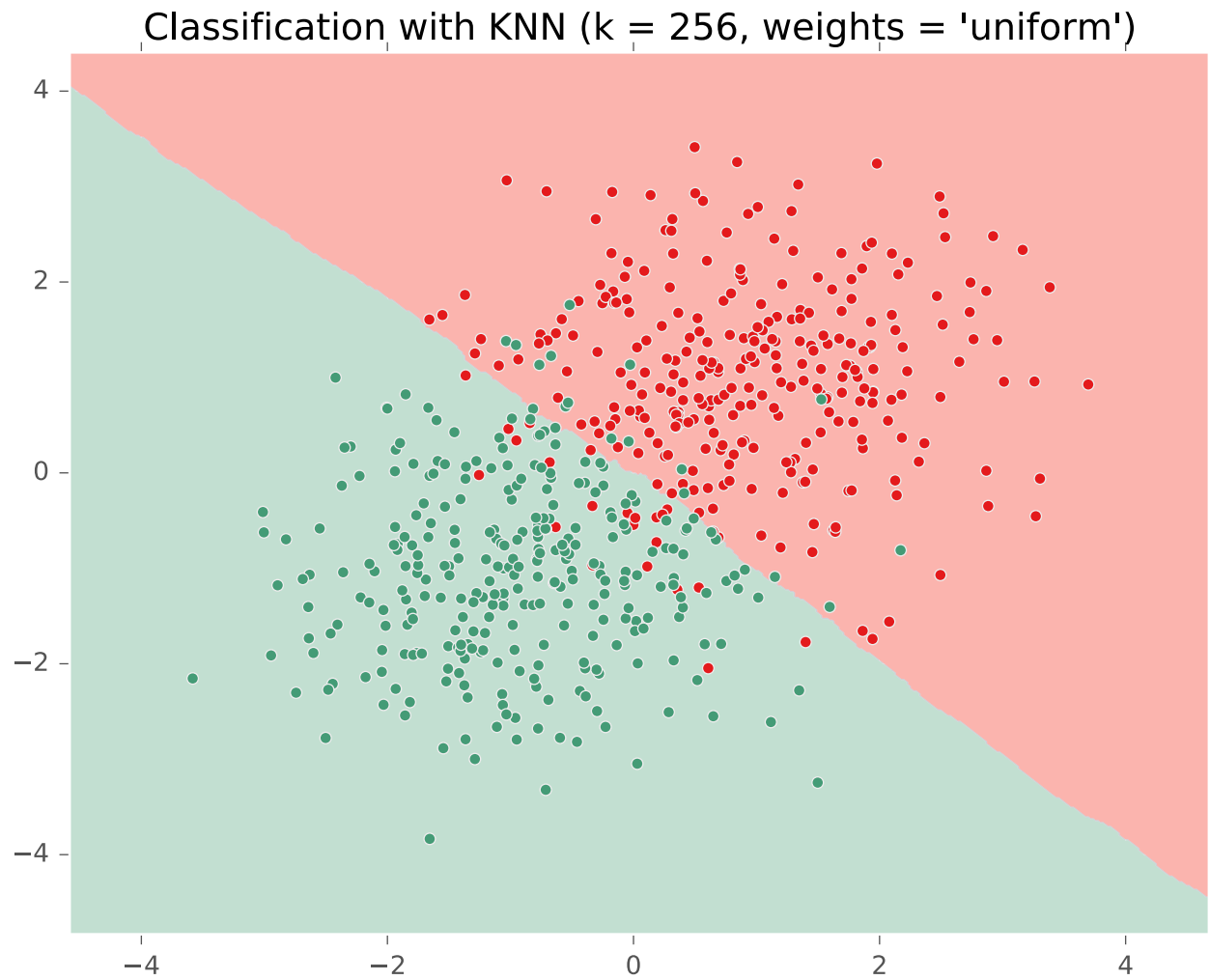
# KNN on Gaussian Data



# KNN on Gaussian Data

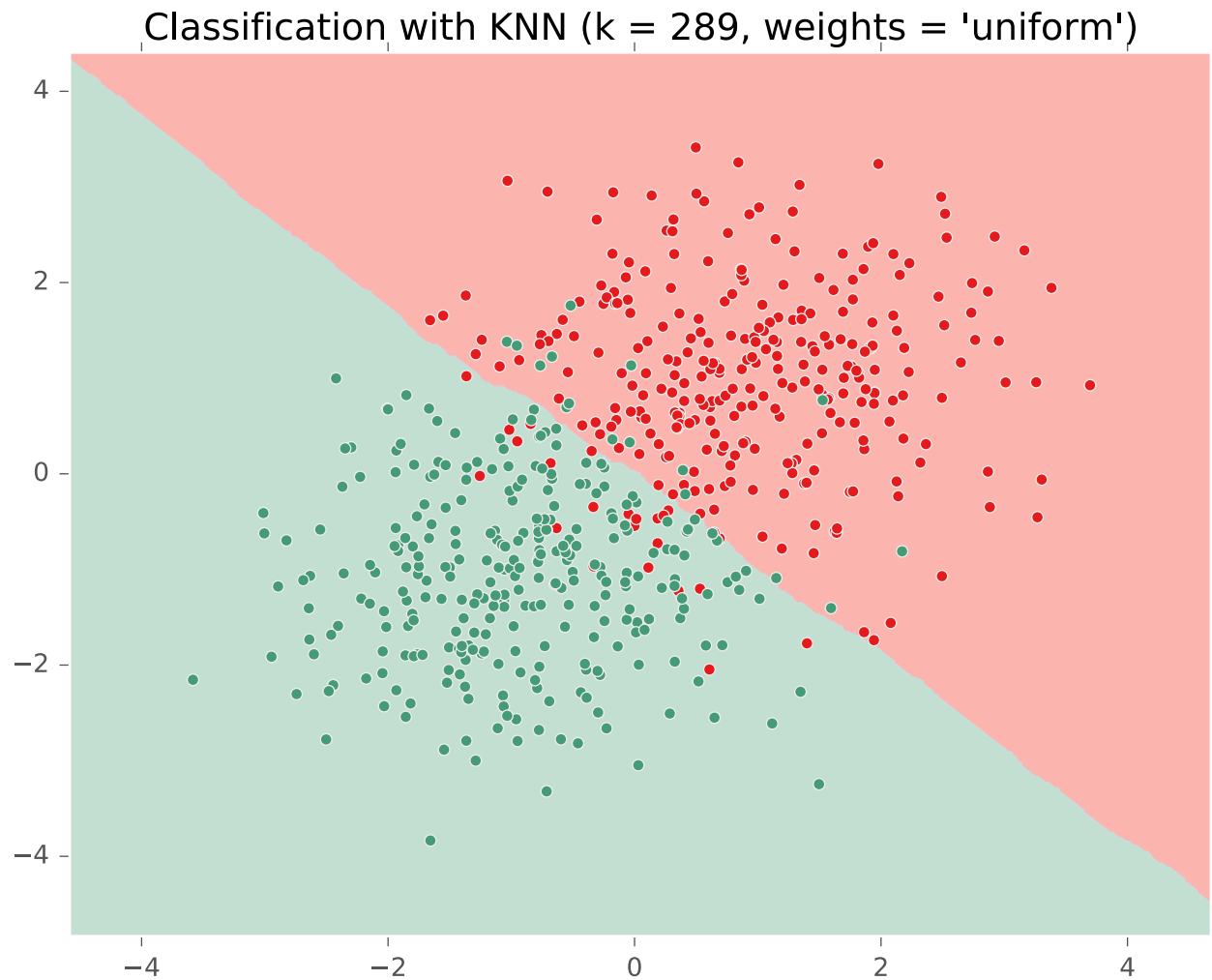


# KNN on Gaussian Data

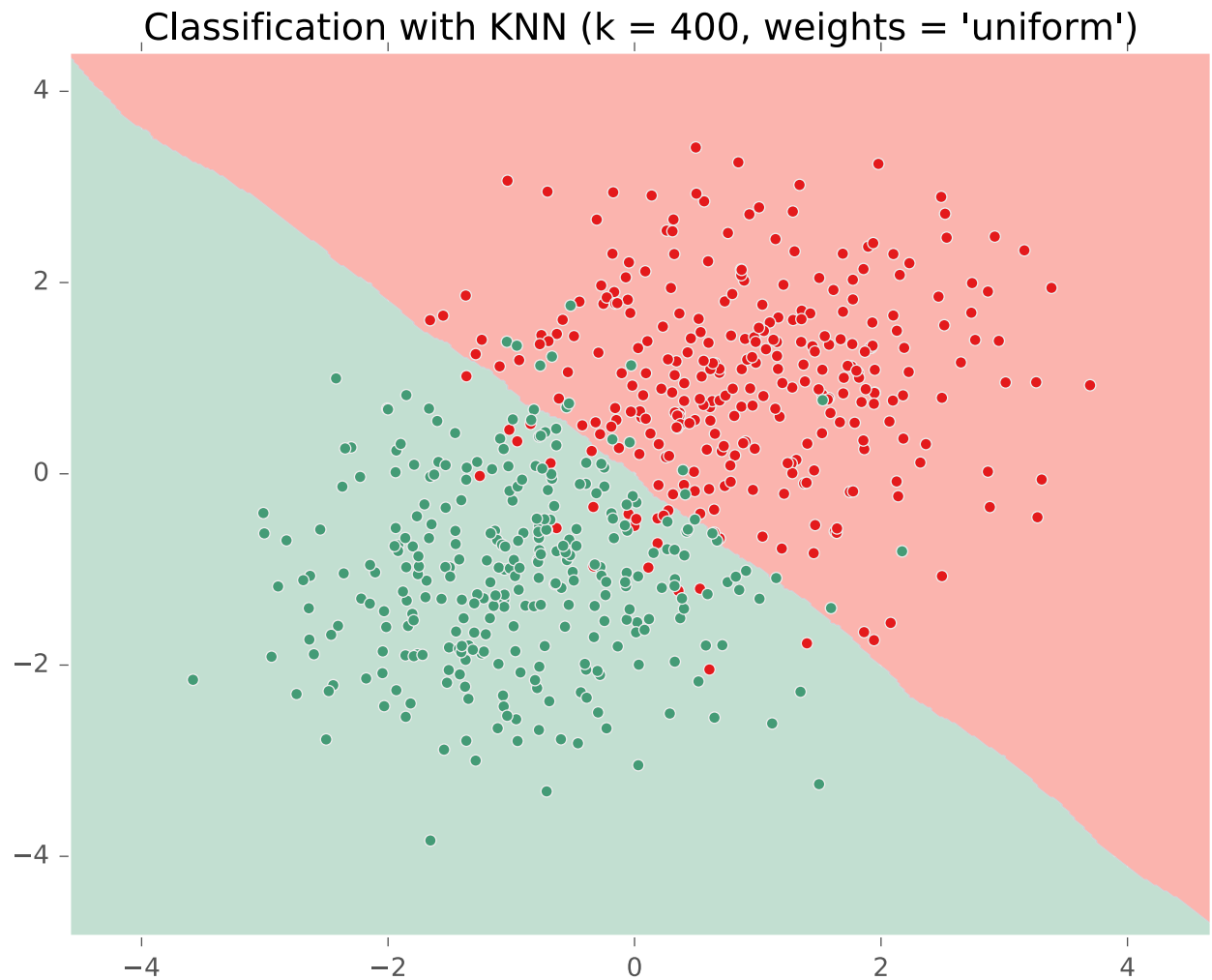




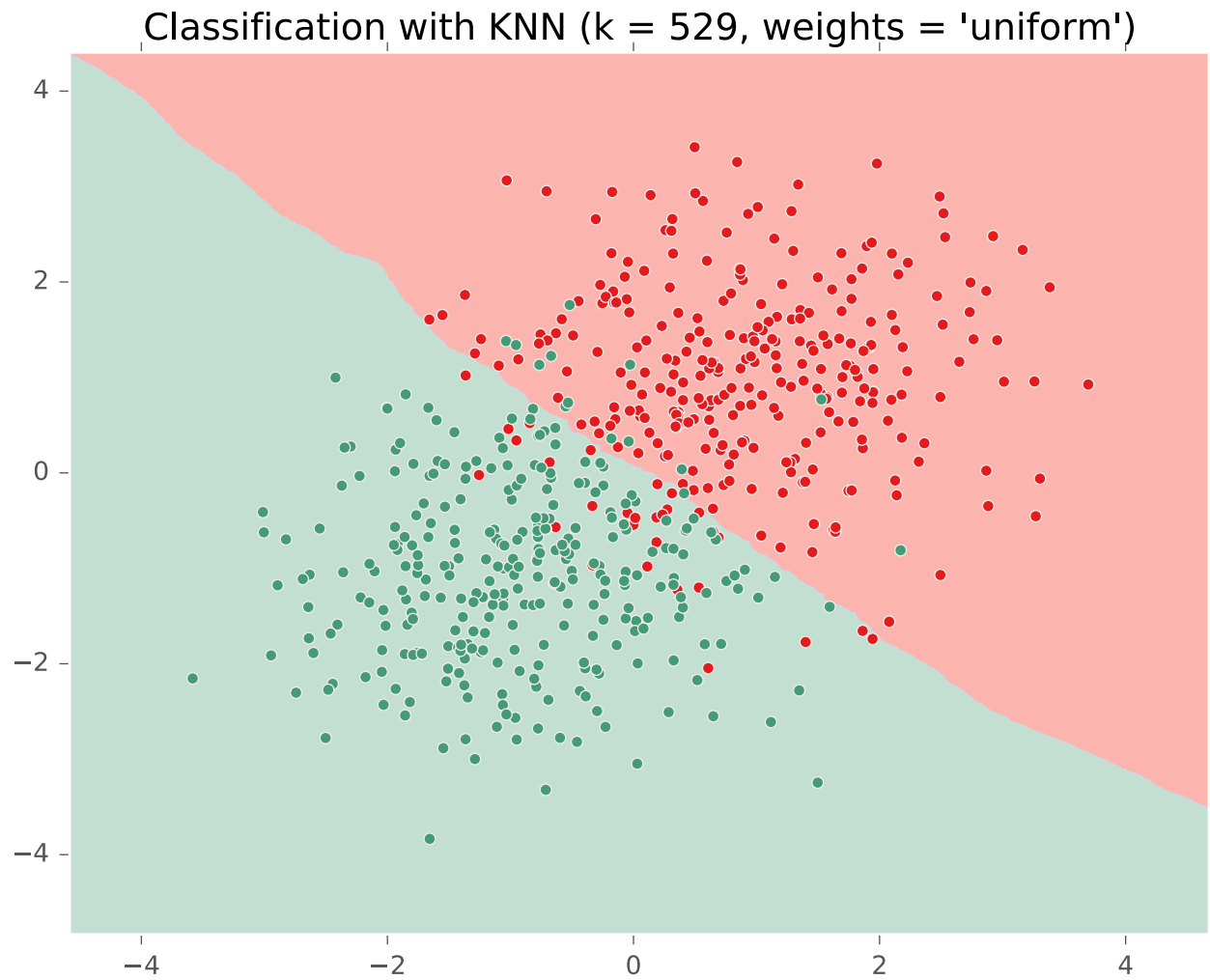
# KNN on Gaussian Data



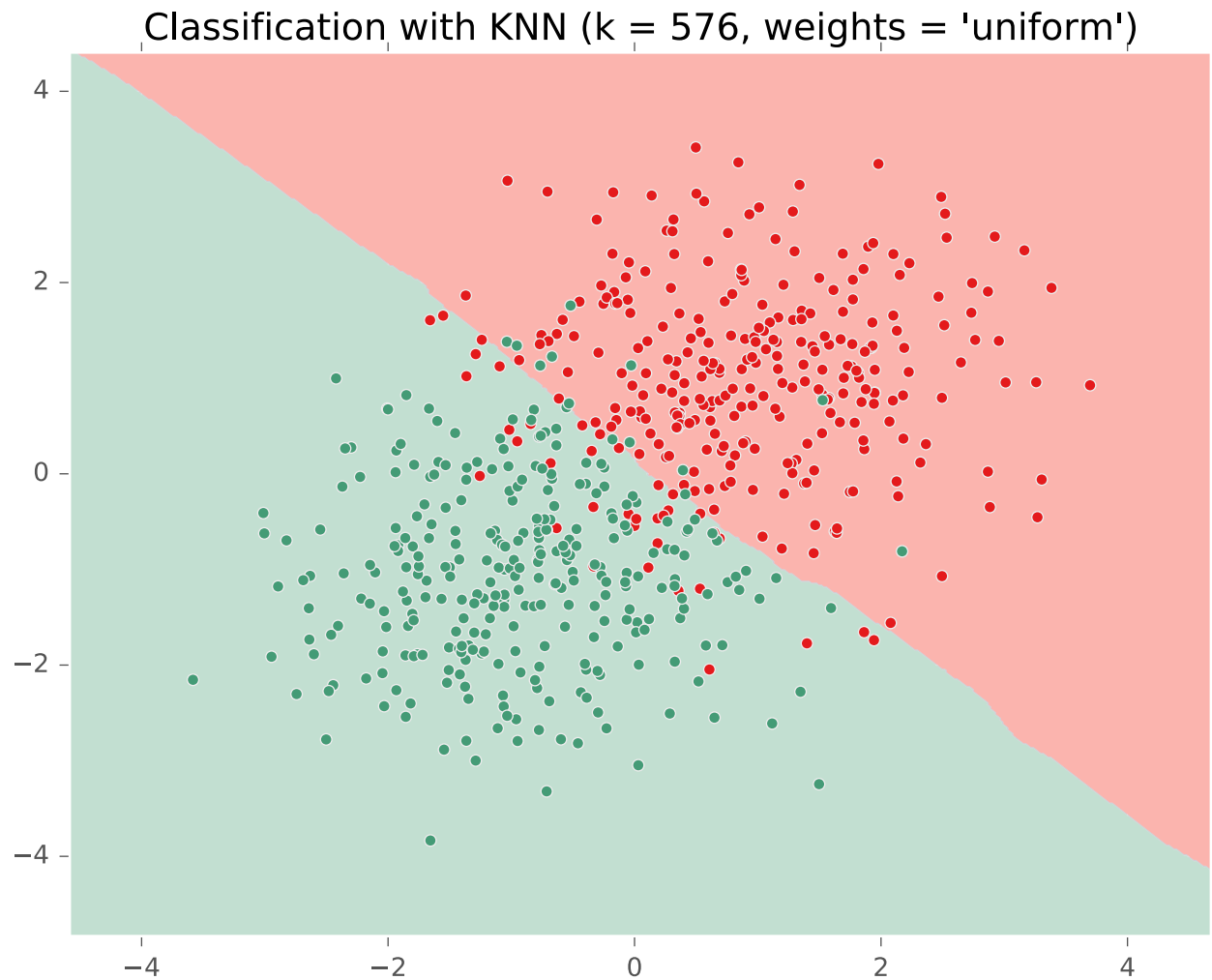
# KNN on Gaussian Data



# KNN on Gaussian Data



# KNN on Gaussian Data



# 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

# **MODEL SELECTION**

# Model Selection

## **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

## Example: Decision Tree

- model = set of all possible trees, possibly restricted by some hyperparameters (e.g. max depth)
- parameters = structure of a specific decision tree
- learning algorithm = ID3, CART, etc.
- hyperparameters = max-depth, threshold for splitting criterion, etc.

## Machine Learning

- *Def:* (loosely) a **model** defines the hypothesis space over which learning performs its search
- *Def:* **model parameters** are the numeric values or structure selected by the learning algorithm that give rise to a hypothesis
- *Def:* the **learning algorithm** defines the data-driven search over the hypothesis space (i.e. search for good parameters)
- *Def:* **hyperparameters** are the tunable aspects of the model, that the learning algorithm does not select



# Model Selection

## Example: k-Nearest Neighbors

- model = set of all possible nearest neighbors classifiers
- parameters = none (KNN is an instance-based or non-parametric method)
- learning algorithm = for naïve setting, just storing the data
- hyperparameters =  $k$ , the number of neighbors to consider

## Machine Learning

- *Def:* (loosely) a **model** defines the hypothesis space over which learning performs its search
- *Def:* **model parameters** are the numeric values or structure selected by the learning algorithm that give rise to a hypothesis
- *Def:* the **learning algorithm** defines the data-driven search over the hypothesis space (i.e. search for good parameters)
- *Def:* **hyperparameters** are the tunable aspects of the model, that the learning algorithm does not select

# Model Selection

## Example: Perceptron

- model = set of all linear separators
- parameters = vector of weights (one for each feature)
- learning algorithm = mistake based updates to the parameters
- hyperparameters = none (unless using some variant such as averaged perceptron)

## Machine Learning

- *Def:* (loosely) a **model** defines the hypothesis space over which learning performs its search
- *Def:* **model parameters** are the numeric values or structure selected by the learning algorithm that give rise to a hypothesis
- *Def:* the **learning algorithm** defines the data-driven search over the hypothesis space (i.e. search for good parameters)
- *Def:* **hyperparameters** are the tunable aspects of the model, that the learning algorithm does not select

# Model Selection

## Statistics

- *Def:* a **model** defines the data generation process (i.e. a set or family of parametric probability distributions)
- *Def:* **model parameters** are the values that give rise to a particular probability distribution in the model family
- *Def:* **learning** (aka. estimation) is the process of finding the parameters that best fit the data
- *Def:* **hyperparameters** are the parameters of a prior distribution over parameters

## Machine Learning

- *Def:* (loosely) a **model** defines the hypothesis space over which learning performs its search
- *Def:* **model parameters** are the numeric values or structure selected by the learning algorithm that give rise to a hypothesis
- *Def:* the **learning algorithm** defines the data-driven search over the hypothesis space (i.e. search for good parameters)
- *Def:* **hyperparameters** are the tunable aspects of the model, that the learning algorithm does not select

# Model Selection

## Statistics

- *Def:* a **model** defines the data generation process (i.e. a family of parameter distributions)
- *Def:* **model parameters** are the values that give rise to a particular probability distribution in the model
- *Def:* **learning** (aka. estimation) is the process of finding the parameters that best fit the data
- *Def:* **hyperparameters** are the parameters of a prior distribution over parameters

## Machine Learning

- *Def:* (loosely) a **model** defines the hypothesis space over which the learning algorithm performs its search
- **hyperparameters** are the tunable aspects of the model, that the learning algorithm does not select
- **learning algorithm** defines the data-driven search over the hypothesis space (i.e. search for good parameters)

If “learning” is all about picking the best parameters how do we pick the best hyperparameters?

# Model Selection

- Two very similar definitions:
  - *Def: model selection* is the process by which we choose the “best” model from among a set of candidates
  - *Def: hyperparameter optimization* is the process by which we choose the “best” hyperparameters from among a set of candidates (**could be called a special case of model selection**)
- **Both** assume access to a function capable of measuring the quality of a model
- **Both** are typically done “outside” the main training algorithm --- typically training is treated as a black box

# **EXPERIMENTAL DESIGN**

# Experimental Design

	Input	Output	Notes
Training	<ul style="list-style-type: none"><li>• training dataset</li><li>• hyperparameters</li></ul>	<ul style="list-style-type: none"><li>• best model parameters</li></ul>	We pick the best model parameters by learning on the training dataset for a fixed set of hyperparameters
Hyperparameter Optimization	<ul style="list-style-type: none"><li>• training dataset</li><li>• validation dataset</li></ul>	<ul style="list-style-type: none"><li>• best hyperparameters</li></ul>	We pick the best hyperparameters by learning on the training data and evaluating error on the validation error
Testing	<ul style="list-style-type: none"><li>• test dataset</li><li>• hypothesis (i.e. fixed model parameters)</li></ul>	<ul style="list-style-type: none"><li>• test error</li></ul>	We evaluate a hypothesis corresponding to a decision rule with fixed model parameters on a test dataset to obtain test error

# Example of Hyperparameter Opt.

## *Whiteboard:*

- Special cases of k-Nearest Neighbors
- Choosing k with validation data
- Choosing k with cross-validation



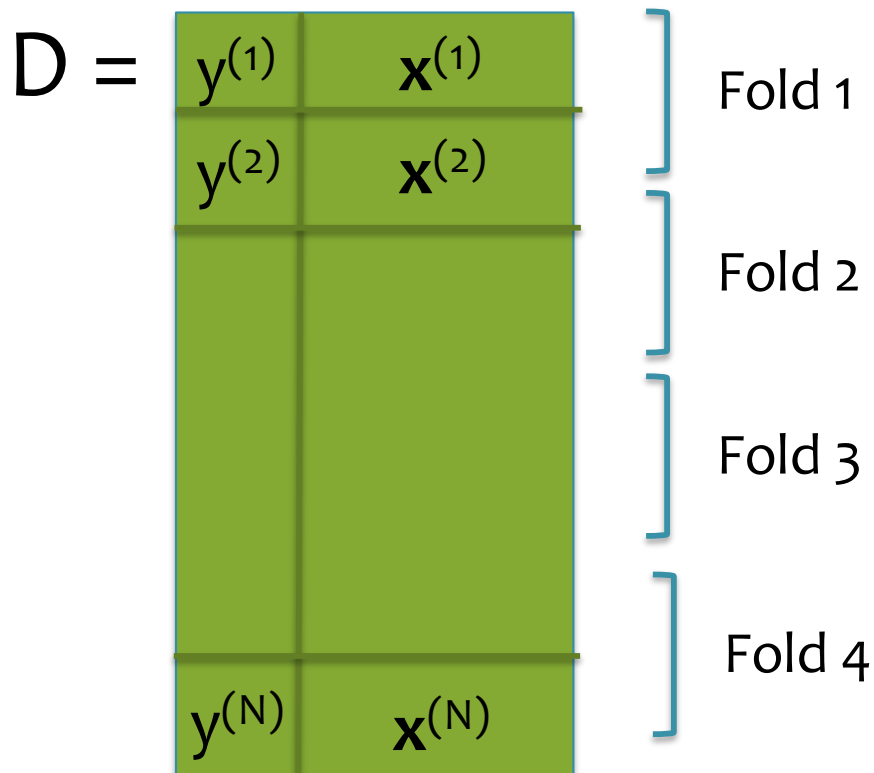
# Cross-Validation

**Cross validation** is a method of estimating loss on held out data

**Input:** training data, learning algorithm, loss function (e.g. 0/1 error)

**Output:** an estimate of loss function on held-out data

**Key idea:** rather than just a single “validation” set, use many!  
(Error is more stable. Slower computation.)



## Algorithm:

Divide data into folds (e.g. 4)

1. Train on folds  $\{1,2,3\}$  and predict on  $\{4\}$
2. Train on folds  $\{1,2,4\}$  and predict on  $\{3\}$
3. Train on folds  $\{1,3,4\}$  and predict on  $\{2\}$
4. Train on folds  $\{2,3,4\}$  and predict on  $\{1\}$

Concatenate all the predictions and evaluate loss (*almost* equivalent to averaging loss over the folds)

Definition:  
**N-fold cross validation** = cross validation with N folds

# Experimental Design

	Input	Output	Notes
Training	<ul style="list-style-type: none"><li>• training dataset</li><li>• hyperparameters</li></ul>	<ul style="list-style-type: none"><li>• best model parameters</li></ul>	We pick the best model parameters by learning on the training dataset for a fixed set of hyperparameters
Hyperparameter Optimization	<ul style="list-style-type: none"><li>• training dataset</li><li>• validation dataset</li></ul>	<ul style="list-style-type: none"><li>• best hyperparameters</li></ul>	We pick the best hyperparameters by learning on the training data and evaluating error on the validation error
Cross-Validation	<ul style="list-style-type: none"><li>• training dataset</li><li>• validation dataset</li></ul>	<ul style="list-style-type: none"><li>• cross-validation error</li></ul>	We estimate the error on held out data by repeatedly training on N-1 folds and predicting on the held-out fold
Testing	<ul style="list-style-type: none"><li>• test dataset</li><li>• hypothesis (i.e. fixed model parameters)</li></ul>	<ul style="list-style-type: none"><li>• test error</li></ul>	We evaluate a hypothesis corresponding to a decision rule with fixed model parameters on a test dataset to obtain test error

# Experimental Design

**Q:** We pick the best hyperparameters by learning on the training data and evaluating error on the validation data. For our final model, should we also learn from just the training data?

**A:** No!

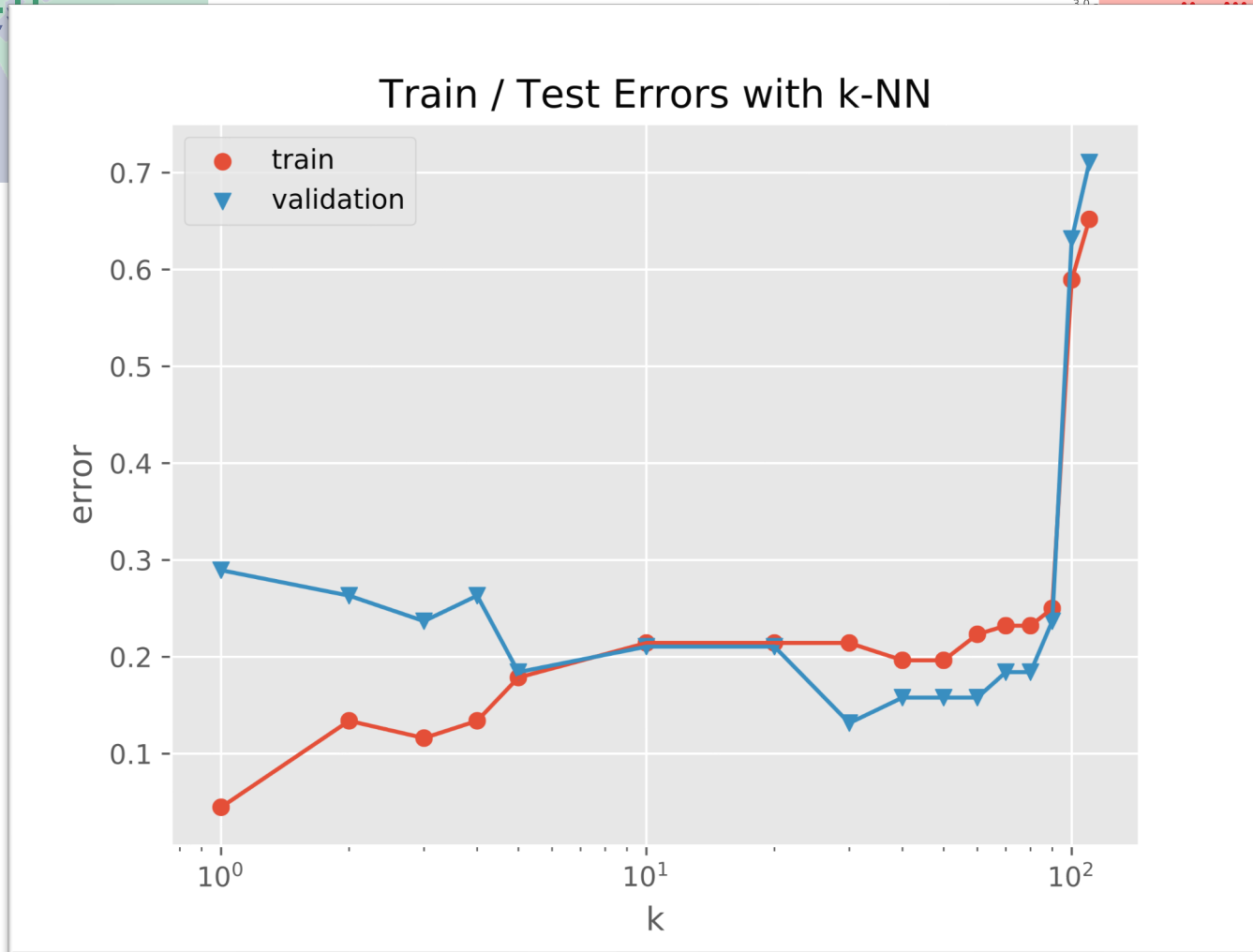
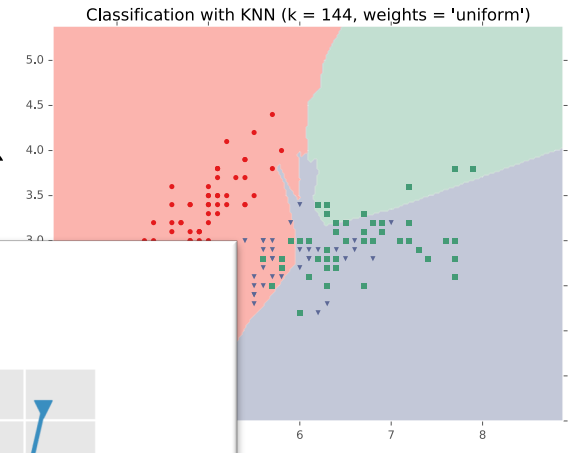
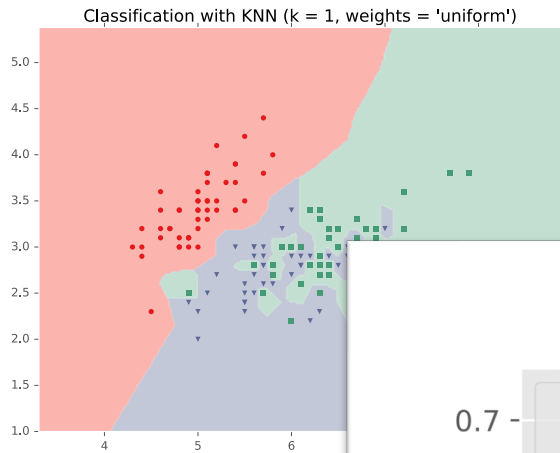
Let's assume that {train-original} is the original training data and {test} is the provided test dataset.

1. Split {train-original} into {train-subset} and {validation}.
2. Pick the hyperparameters that when training on {train-subset} give the lowest error on {validation}. Call these hyperparameters {best-hyper}.
3. Retrain a new model using {best-hyper} on {train-original} = {train-subset}  $\cup$  {validation}.
4. Report test error by evaluating on {test}.

Alternatively, you could replace Steps 1-2 with the following:

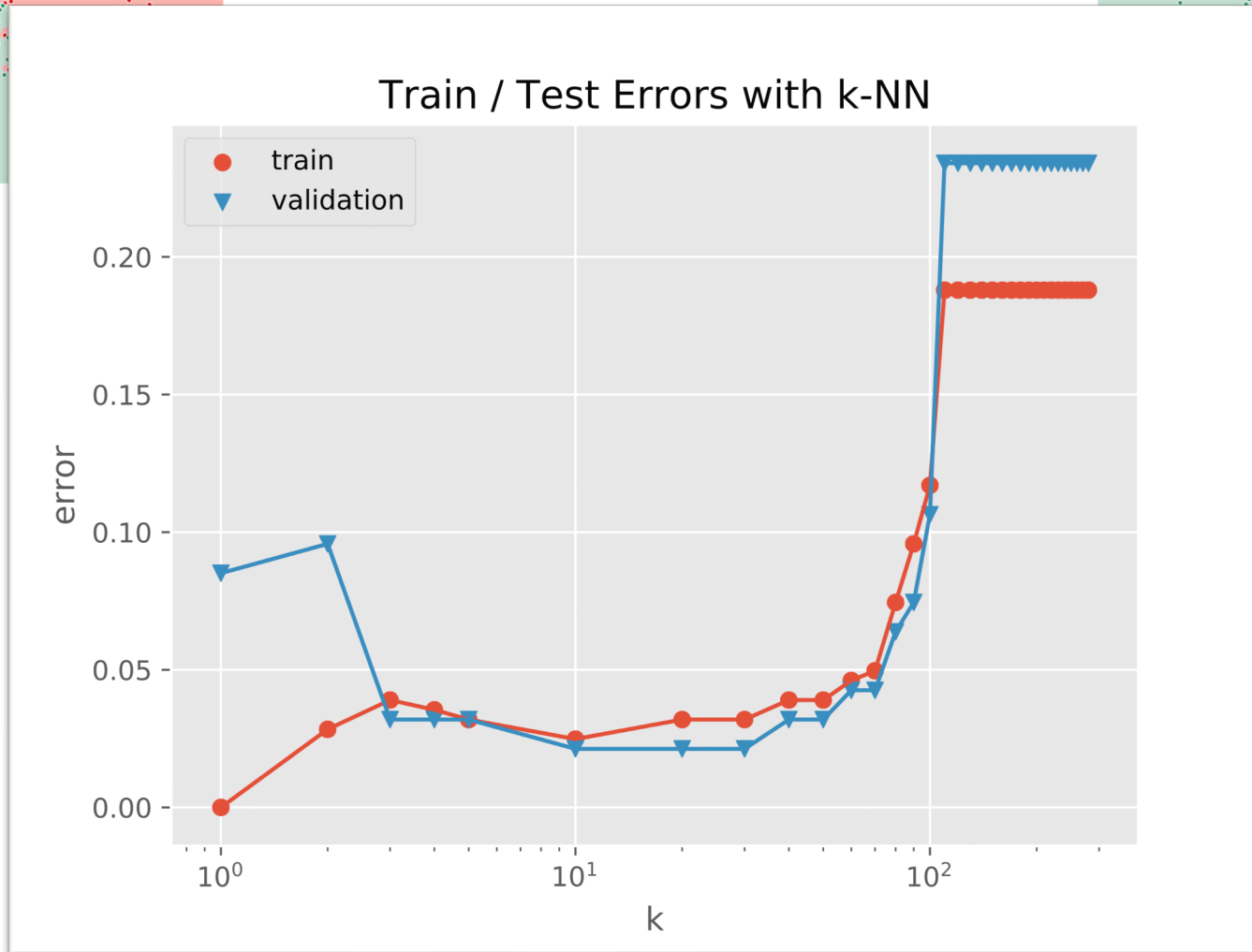
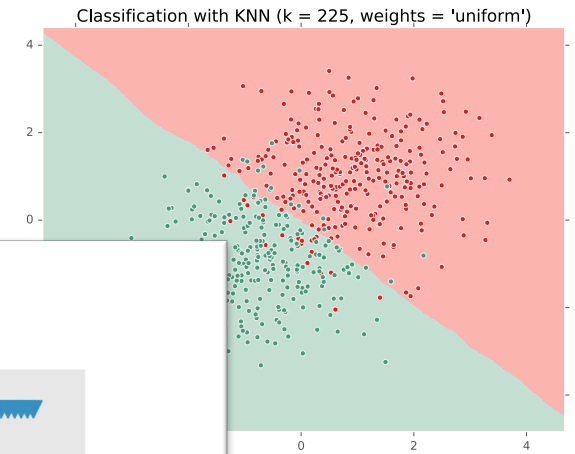
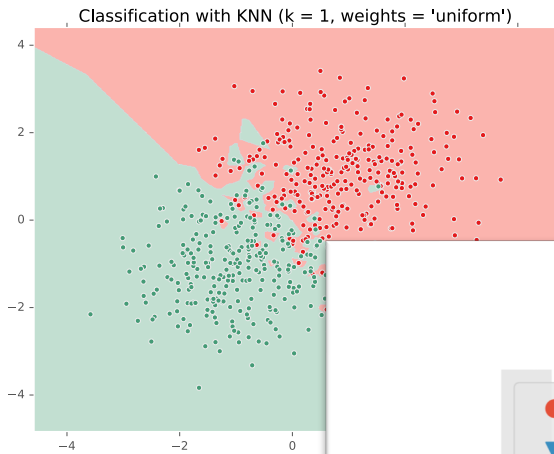
1. Pick the hyperparameters that give the lowest cross-validation error on {train-original}. Call these hyperparameters {best-hyper}.

# k-NN: Choosing k



Fisher Iris Data: varying the value of k

# k-NN: Choosing k



Gaussian Data: varying the value of k

# **HYPERPARAMETER OPTIMIZATION**

# Model Selection

## **WARNING (again):**

- This section is only scratching the surface!
- Lots of methods for hyperparameter optimization: (to talk about later)
  - Grid search
  - Random search
  - Bayesian optimization
  - Graduate-student descent
  - ...

## **Main Takeaway:**

- Model selection / hyperparameter optimization is just another form of learning

# Hyperparameter Optimization

**Setting:** suppose we have hyperparameters  $\alpha$ ,  $\beta$ , and  $\chi$  and we wish to pick the “best” values for each one

## Algorithm 1: Grid Search

- Pick a set of values for each hyperparameter  
 $\alpha \in \{a_1, a_2, \dots, a_n\}$ ,  $\beta \in \{b_1, b_2, \dots, b_n\}$ , and  $\chi \in \{c_1, c_2, \dots, c_n\}$
- Run a grid search

```
for  $\alpha \in \{a_1, a_2, \dots, a_n\}$ :  
  for  $\beta \in \{b_1, b_2, \dots, b_n\}$ :  
    for  $\chi \in \{c_1, c_2, \dots, c_n\}$ :  
       $\theta = \text{train}(D_{\text{train}}; \alpha, \beta, \chi)$   
       $\text{error} = \text{predict}(D_{\text{validation}}; \theta)$ 
```

- return  $\alpha$ ,  $\beta$ , and  $\chi$  with lowest validation error



# Hyperparameter Optimization

**Setting:** suppose we have hyperparameters  $\alpha$ ,  $\beta$ , and  $\chi$  and we wish to pick the “best” values for each one

## Algorithm 2: Random Search

- Pick a range of values for each parameter  
 $\alpha \in \{a_1, a_2, \dots, a_n\}$ ,  $\beta \in \{b_1, b_2, \dots, b_n\}$ , and  $\chi \in \{c_1, c_2, \dots, c_n\}$
- Run a random search

for  $t = 1, 2, \dots, T$ :

sample  $\alpha$  uniformly from  $\{a_1, a_2, \dots, a_n\}$

sample  $\beta$  uniformly from  $\{b_1, b_2, \dots, b_n\}$

sample  $\chi$  uniformly from  $\{c_1, c_2, \dots, c_n\}$

$\theta = \text{train}(D_{\text{train}}; \alpha, \beta, \chi)$

error =  $\text{predict}(D_{\text{validation}}; \theta)$

- return  $\alpha$ ,  $\beta$ , and  $\chi$  with lowest validation error

# Hyperparameter Optimization

## Question:

*True or False:* given a finite amount of computation time, **grid search** is more likely to find good values for hyperparameters than **random search**.

## Answer:

# Hyperparameter Optimization

## Question:

*True or False:* given a finite amount of computation time, grid search is more likely to find good values for hyperparameters than random search.

## Answer:

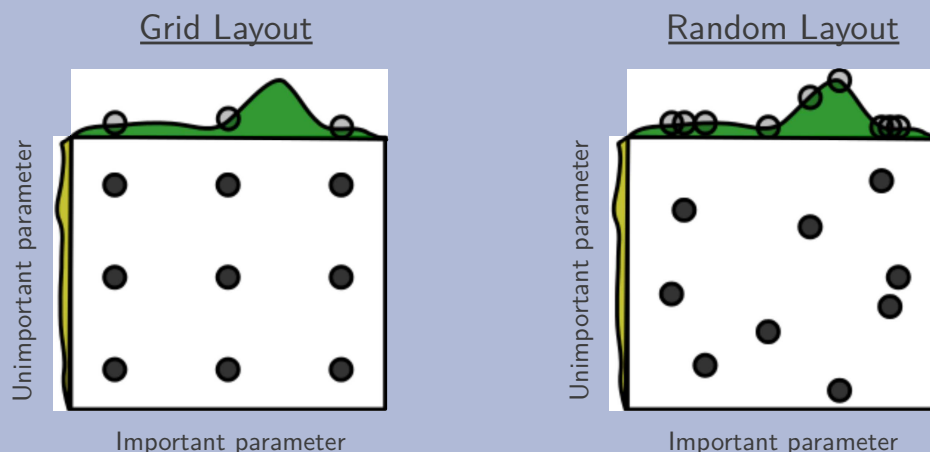


Figure 1: Grid and random search of nine trials for optimizing a function  $f(x,y) = g(x) + h(y) \approx g(x)$  with low effective dimensionality. Above each square  $g(x)$  is shown in green, and left of each square  $h(y)$  is shown in yellow. With grid search, nine trials only test  $g(x)$  in three 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.

# 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 hyperparameters
- Define "instance-based learning" or "nonparametric methods"
- Select an appropriate algorithm for optimizing (aka. learning) hyperparameters