Decision trees

- Twenty questions/dichotomous key/decision theory
- Useful for discrete features and target function
- ID3 induces decision trees, using information gain to choose splits
- C4.5 improves on this, using gain ratio to choose splits, plus reduced-error pruning
- Relatively cheap and effective, widely available implementation
- Many variations: decision lists, decision stumps

Homework

- Use decision trees to correct easily confused words:

There planning to build they're house their.

⇓

They're planning to build their house there.

- We can treat this as a classification problem by removing members of the set:

____ planning to build ____ house ____.

and assigning the class *there, their,* or *they're* to each blank.

Homework

- The first step is to produce annotated training data, using a tagged version of the Brown corpus (`~malouf/ling696/brown.txt`):

```
We_PRP marvel_VB at_IN their_PRP$ blindness_NN for_IN not_RB...
```

- Convert this to the C4.5 data format, using the two preceding and following tags as features:

```
VB,IN,NN,IN,their.
```

- Create `.names`, `.data`, and `.test` files, following the format in the man pages.

Homework

- Use a filler tag if the word to be class too early in the sentence
- The characters `,?:.` can't occur as part of a feature value or class name unless escaped: `\,\?\:\.`
- Ignore case (i.e., mark *Their* as *their*)
- You'll need to retokenize *they're*:

```
‘‘_‘‘ They_PRP ’re_VBP going_VBG to_TO louse_VB me_PRP up_RB...
```

as something like:

```
‘‘_‘‘ They’re_VBP going_VBG to_TO louse_VB me_PRP up_RB...
```

Homework

- To keep held out data for evaluation, you need to randomly send some instances to the `.data` file and some to the `.test` file
- In perl, to do one thing to 90% of examples and another thing to 10%:

```
if (rand(10) < 9) {
  ... training file
} else {
  ... test file
}
```

- For reproducable results, call `srand(0);` early in the program

Homework

- The `.names` file:

```
their,there,they’re.
tag-2 : discrete 200.
tag-1 : discrete 200.
tag+1 : discrete 200.
tag+2 : discrete 200.
```

- Or, if you are feeling ambitious, you can list possible values for each feature.

Homework

- Once you've got the data in the right format (which is 99% of the work), build and evaluate some decision trees
- Try different settings of `c4.5`'s options (especially the `-s`, `-g`, and `-c` options. What effect do they have on the classification accuracy?
- Feel free to try different feature sets as well.
- It's surprisingly difficult to get classification results, so don't bother with error analysis
- Turn a transcript, plus any programs you write, along with a paragraph of discussion.
- **Due February 5**

Instance-based learning

- Aka: Memory-based learning, case-based reasoning, lazy learning, . . .
- Relate new instances to the previously seen instances that they most resemble (Cover and Hart 1967, Duda and Hart 1973, Aha and Kibler 1991, Daelemans 1992–now)
- Applicable to problems with continuous or discrete feature vectors $\mathbf{x}$ and continuous or discrete target function $f(\mathbf{x})$
- Intuitively attractive, conceptually simple, can be computationally expensive
- Success depends entirely on the choice of representation (as always)

k Nearest Neighbors

- Place training instances and query $\mathbf{x}_q$ in a *vector space*
- The simplest version (IB1) classifies new instances by letting the k nearest training instances vote
- Or, for *regression*, it can average the neighbors:

$$\hat{f}(\mathbf{x}_q) = \frac{1}{k}\sum_i f(\mathbf{x}_i)$$

- We need an appropriate way measuring *distance*

k Nearest Neighbors

- For continuous features, the *Euclidean distance* works well:

$$d(\mathbf{x},\mathbf{y}) = \sqrt{\sum (x_i - y_i)^2}$$

- The *cosine* is similar, but not dependent on the lengths of the vectors:

$$\cos(\mathbf{x},\mathbf{y}) = \frac{\sum x_i y_i}{\sqrt{\sum x_i^2}\sqrt{\sum y_i^2}}$$

- Other measures: Manhattan distance, Chebyshev distance

k Nearest Neighbors

- For discrete features, we need a measure of overlap
- The *Hamming distance* is the number of features which match
- The *modified value difference metric (MVDM)* measures the difference between two feature values:

$$\delta(v_1, v_2) = \sum_j |P(C_j|v_1) - P(C_j|v_2)|$$

  which can be used to measure the distance between instances:

$$d(\mathbf{x}, \mathbf{y}) = \sum_i \delta(x_i, y_i)$$

- Or, we can map discrete values into a continuous space (e.g., vowel space) and use a continuous measure

k Nearest Neighbors

- IB1 is easily distracted by irrelevant features (imagine one feature is the class, 20 features are random)
- A very effective variation (IB1-IG) weights features by their *information gain* or *gain ratio* (like C4.5) or *chi square*
- Another variant weights the contribution of training instances inversely with their distance

k Nearest Neighbors

- IB2 and IB3 reduce computational cost by deleteing redundant or noisy training instances
- Start with a small subset of training examples
- For IB2, add each additional training example to the subset if it is misclassified by the current model
- For IB3, add each additional training example to the subset if it decreases the expected error
- Very similar to decision tree pruning

Lazy learning

- IBL is *lazy* in that it doesn't try to model the target function until there is a query $\mathbf{x}_q$
- *Eager* learners (e.g., C4.5) construct an approximation $\hat{f}$ immediately
- A lazy learner's $\hat{f}$ only needs to be accurate in the immediate neighborhood of $\mathbf{x}_q$, while an eager learner's $\hat{f}$ must be accurate everywhere

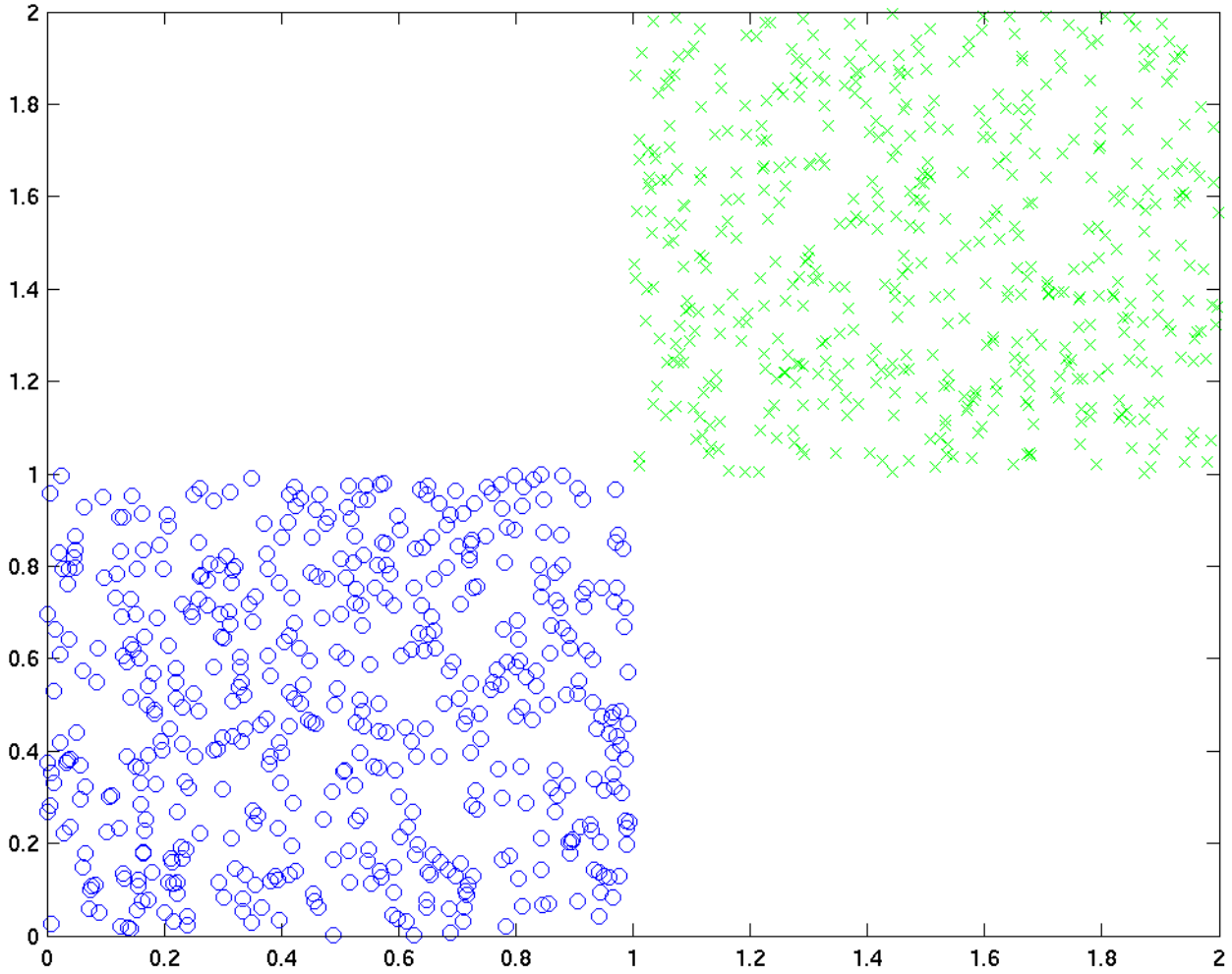
Lazy learning

- Given the same hypothesis space, a lazy learner can represent more complex concepts than an eager learner
- Or, a lazy learner can use a simpler class of hypotheses to learn similar concepts
- IB1 is essentially a baseline classifier (always assign the majority class), applied to a local neighborhood of the query instance
- It pays to be lazy!

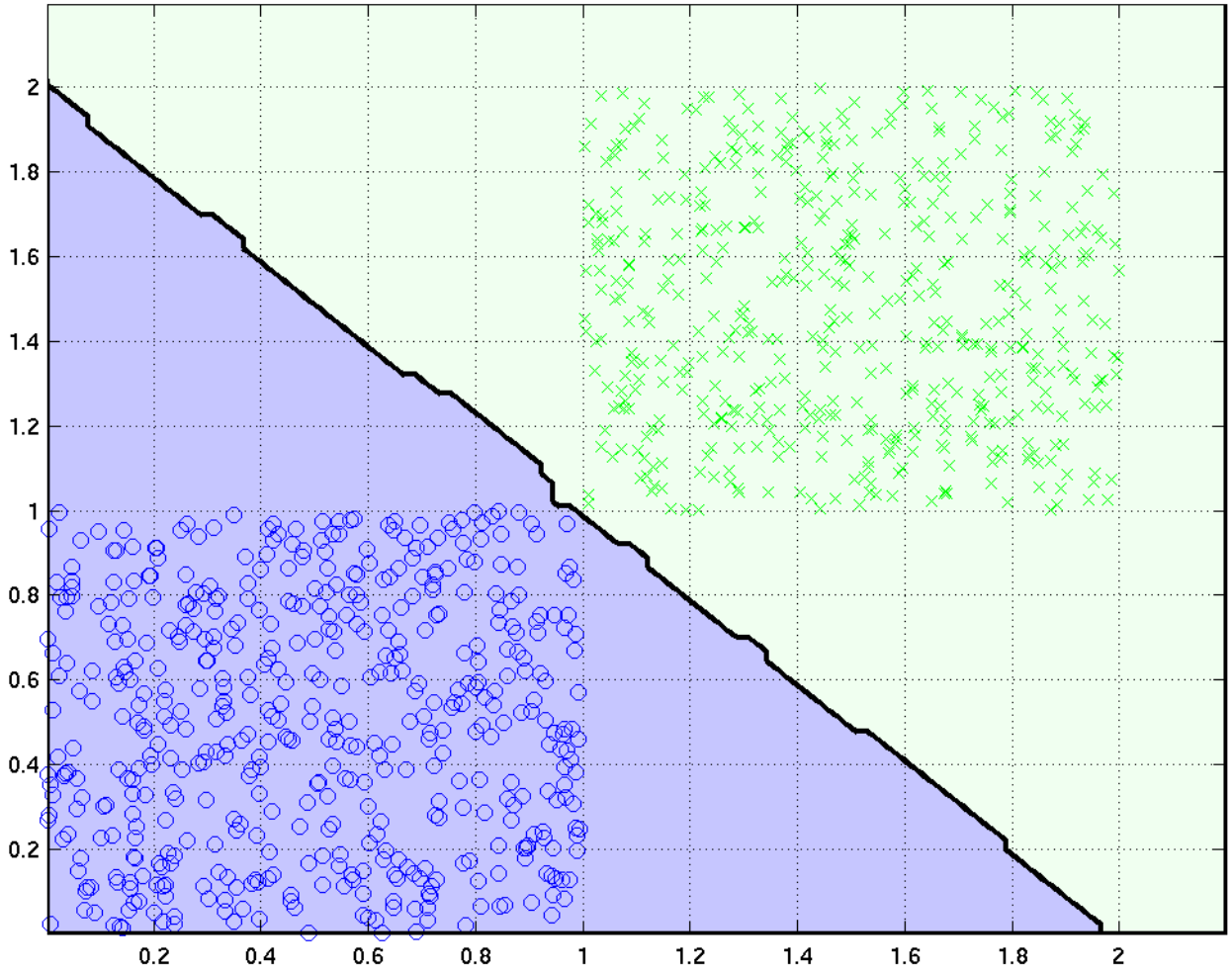
Lazy learning

- Daelemans, et al. (1999) claims lazy learners are particularly well suited to NLP problems
- Language is ‘lumpy’, with lots of different fairly regular patterns: *-ed, -d*
- Even linguistic exceptions include lots of subregularities: *sell*/*sold, tell*/*told*
- Even highly exceptional forms need to be remembered: *be*/*was*
- Irregularities can be hard to distinguish from noise

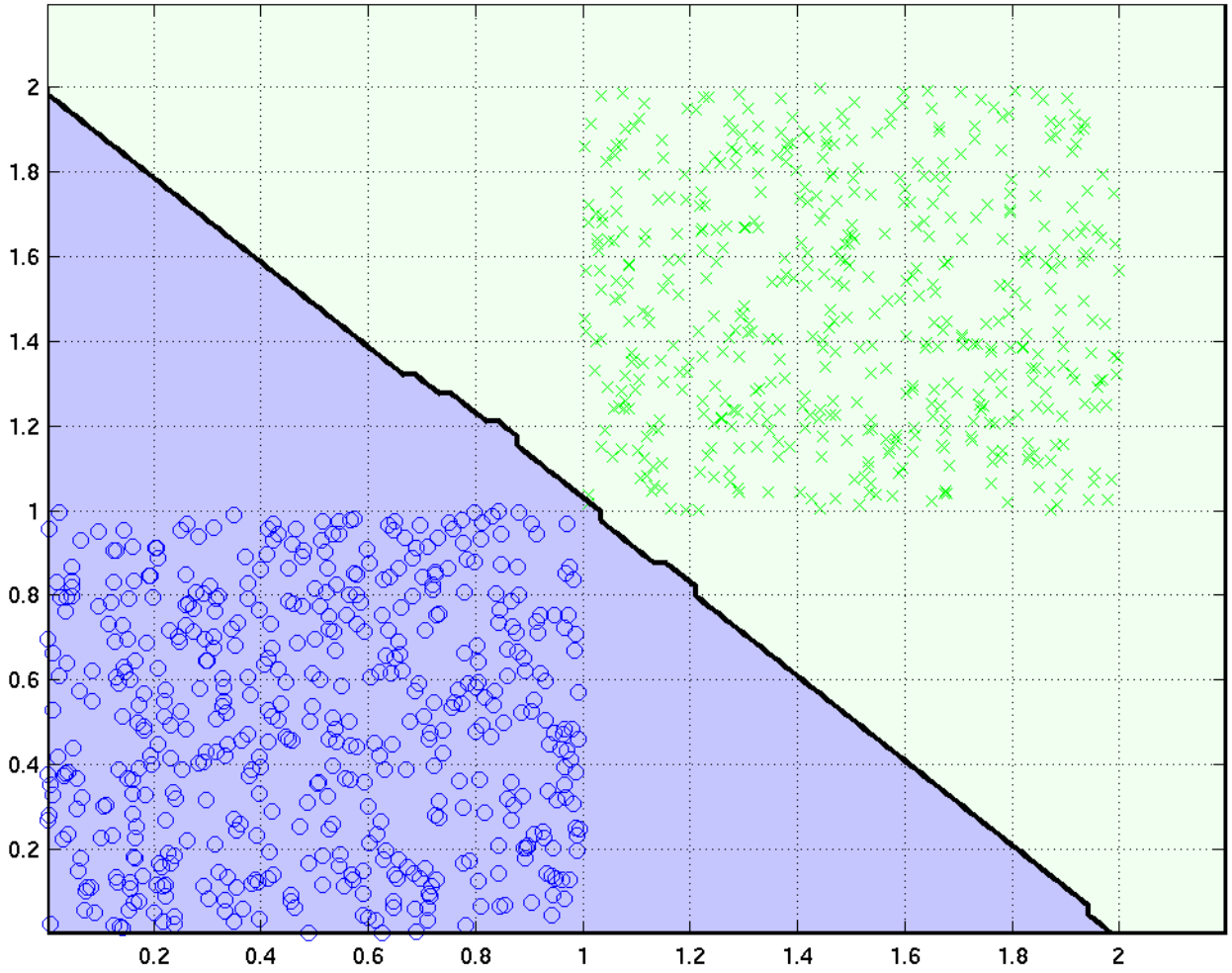
Decision boundaries



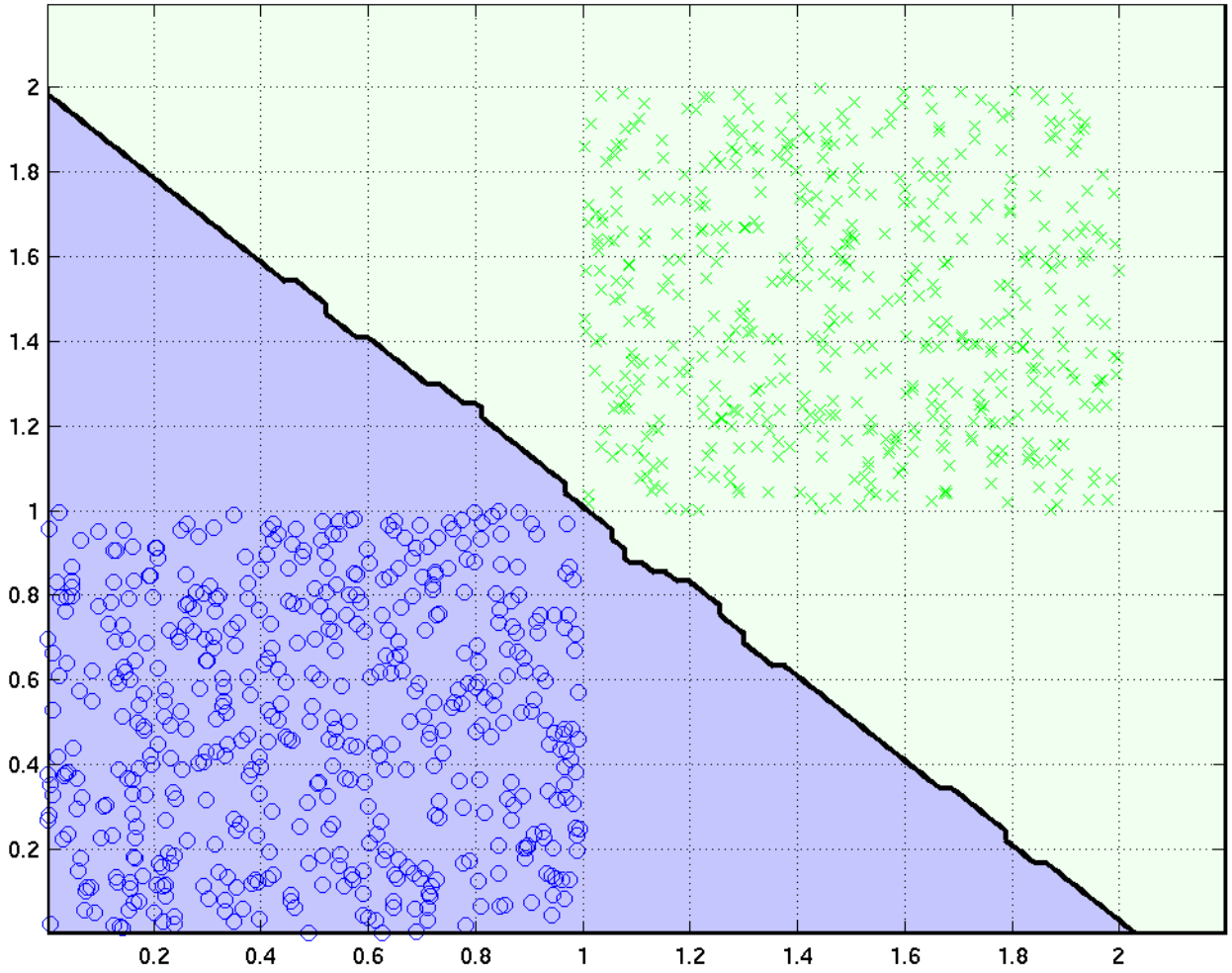
Decision boundaries $k = 1$



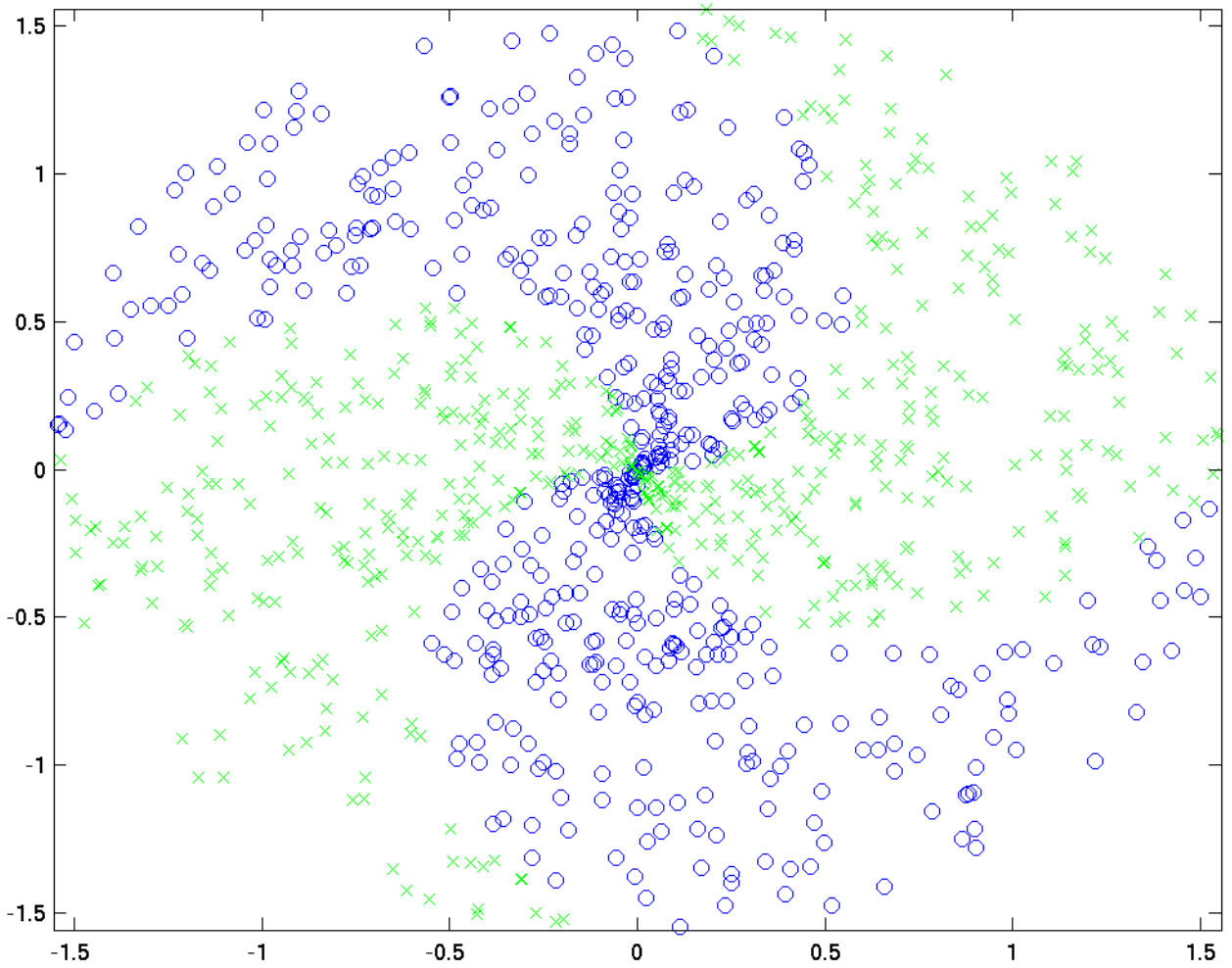
Decision boundaries $k = 5$



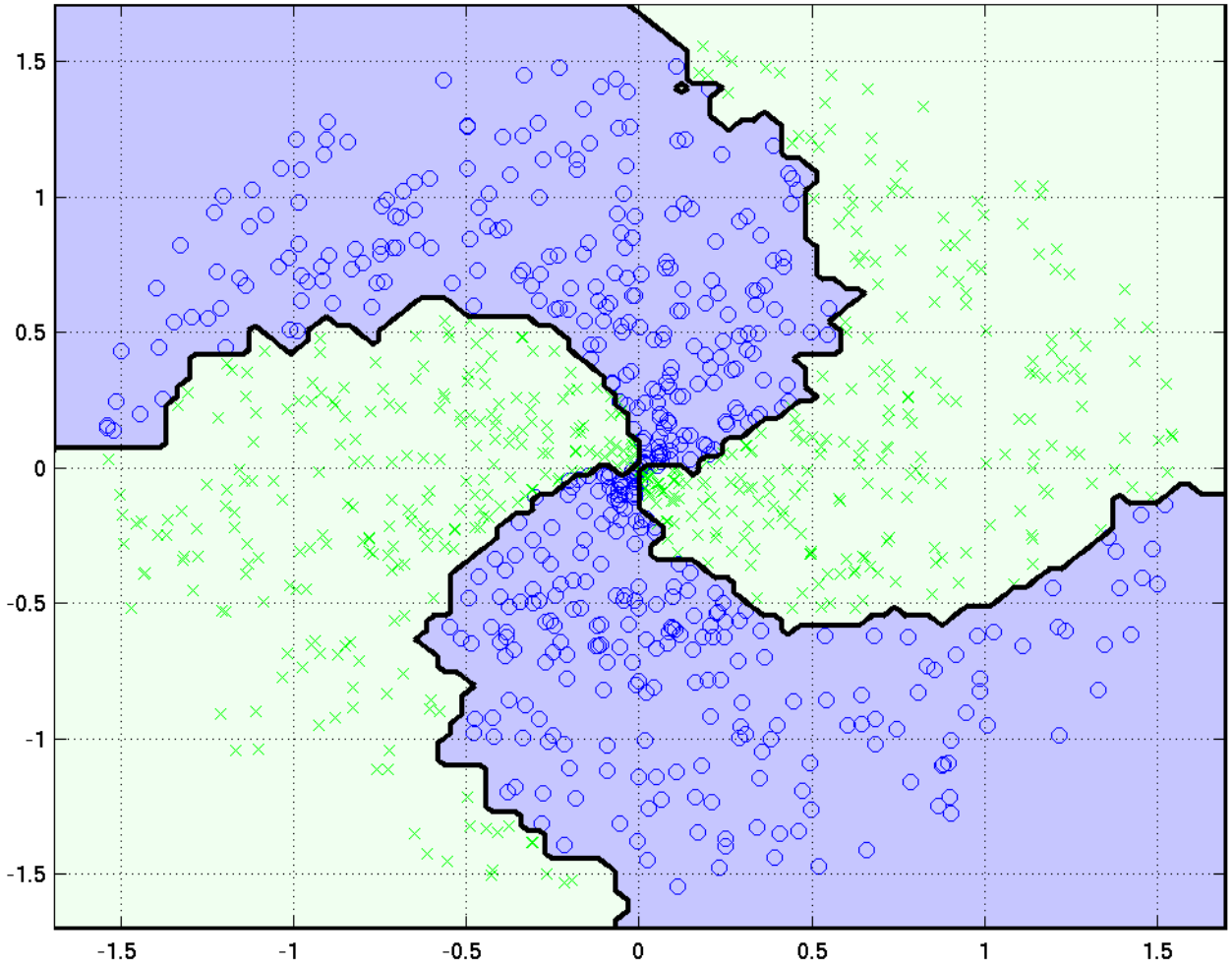
Decision boundaries $k = 10$



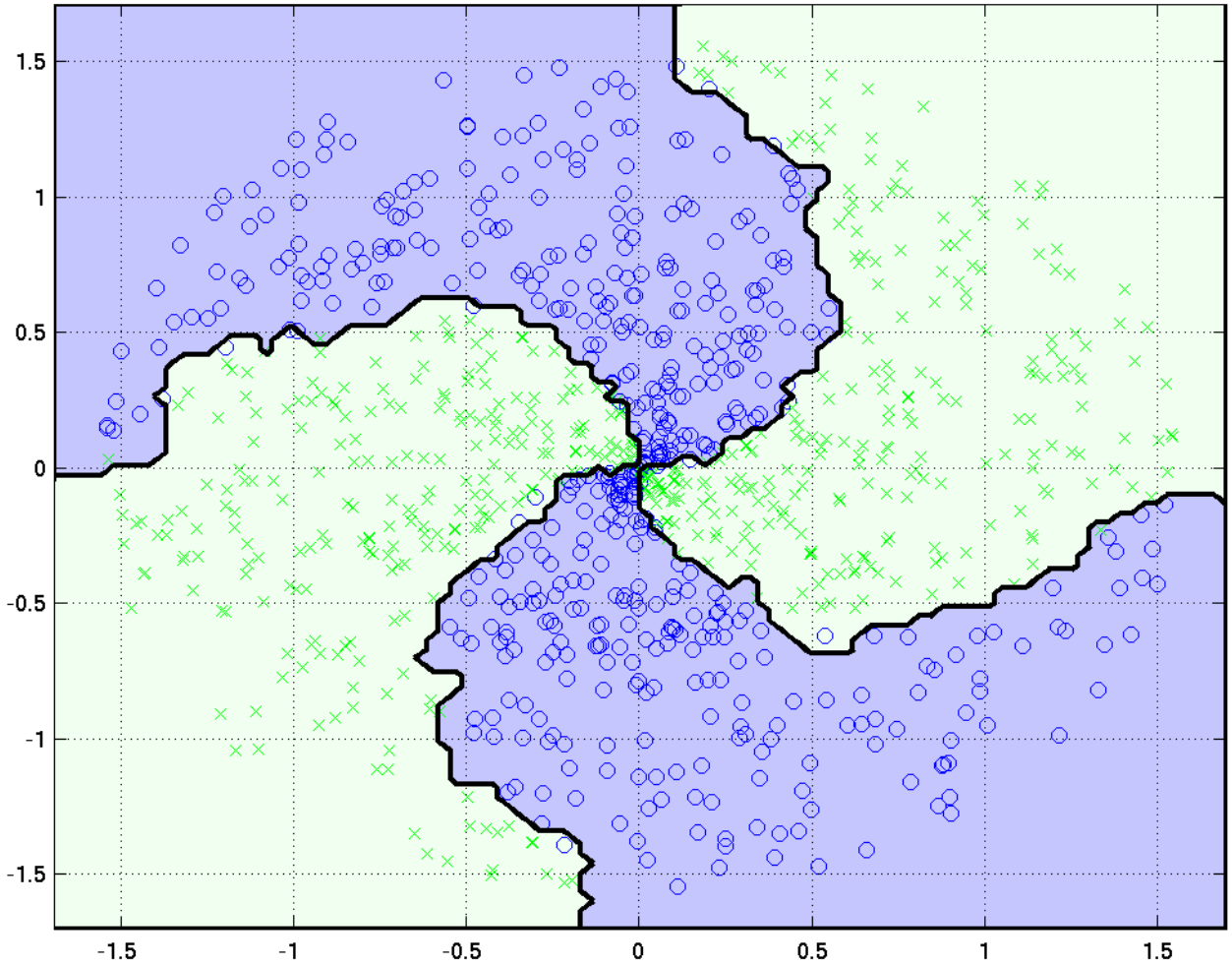
Decision boundaries



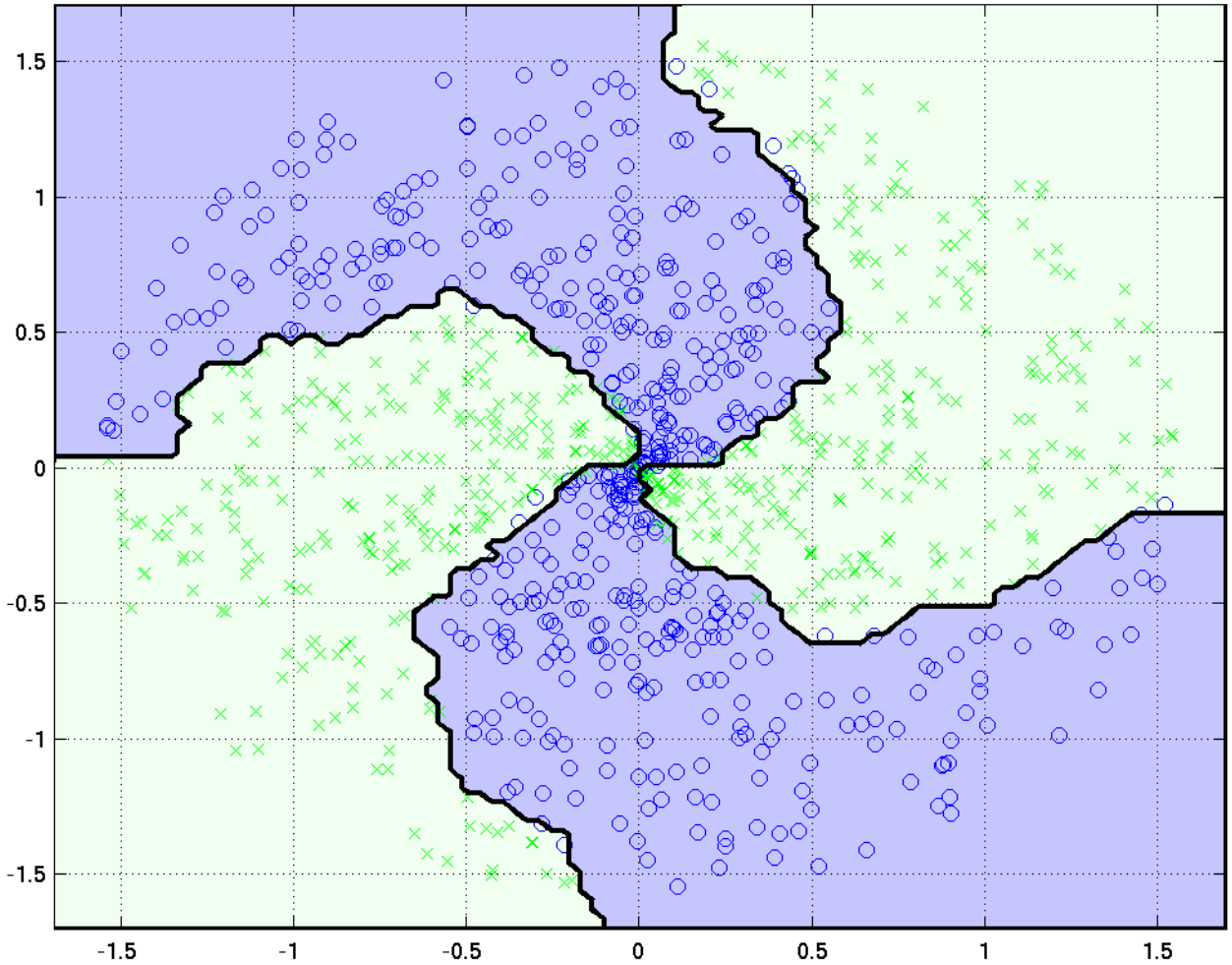
Decision boundaries $k = 1$



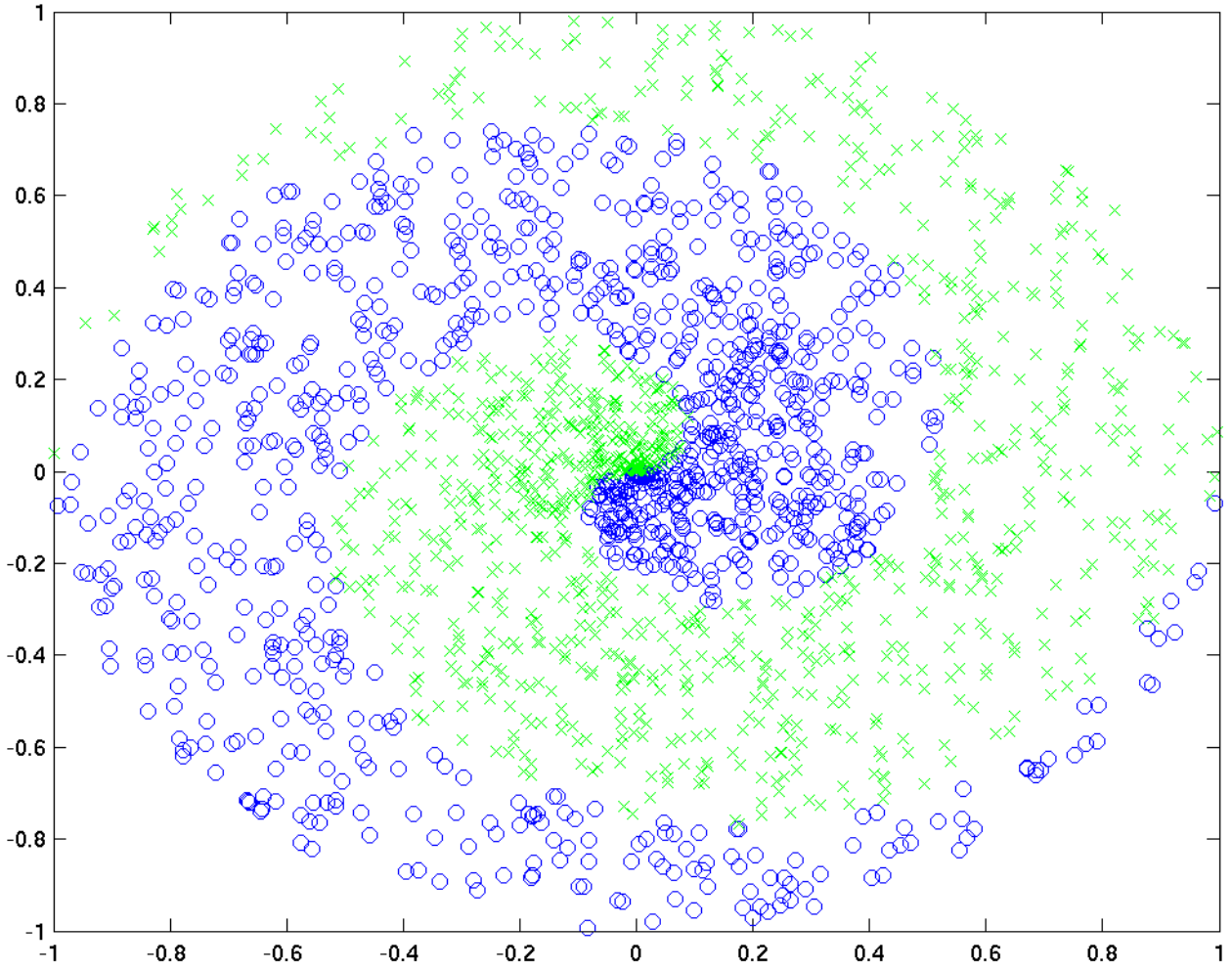
Decision boundaries $k = 5$



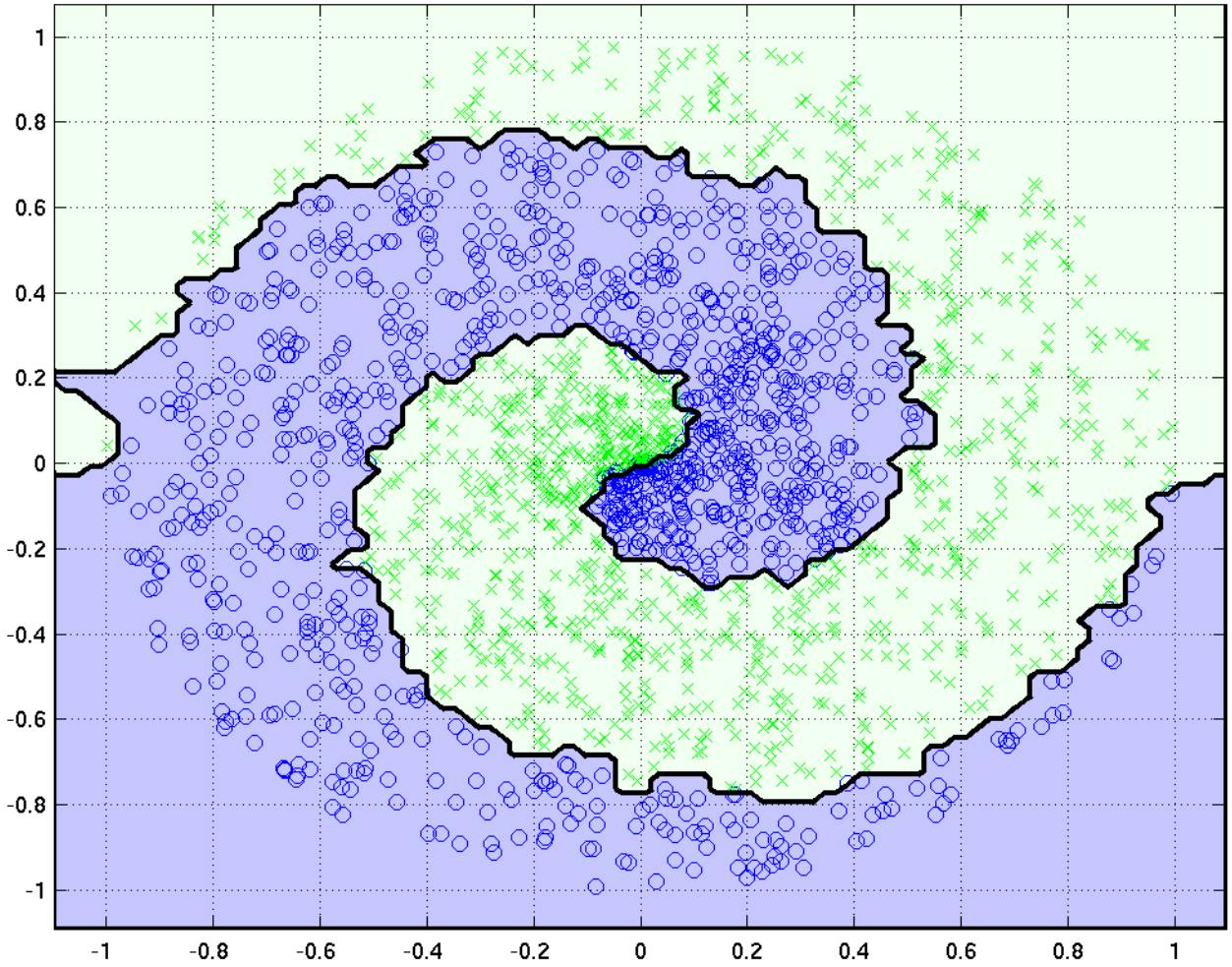
Decision boundaries $k = 10$



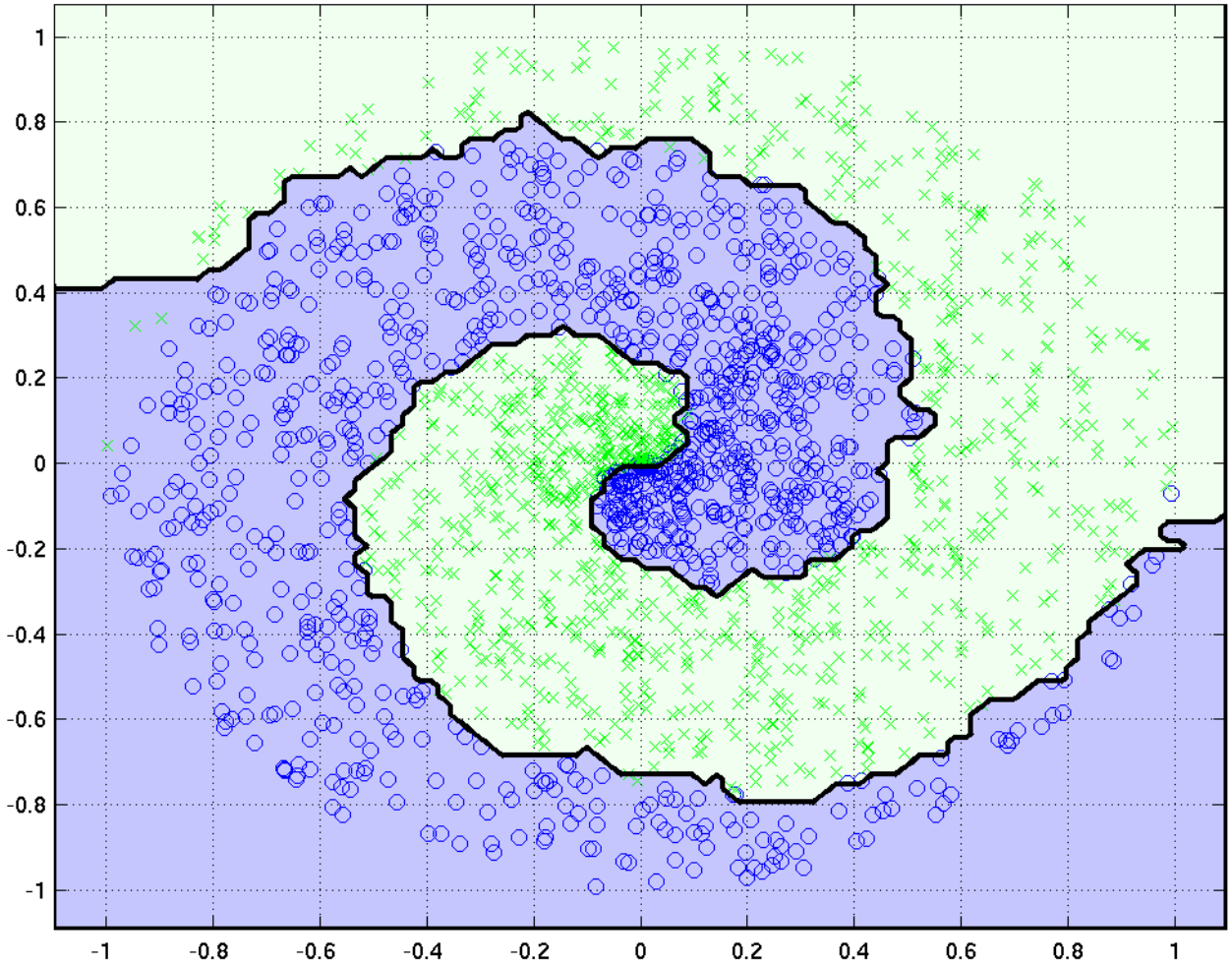
Decision boundaries



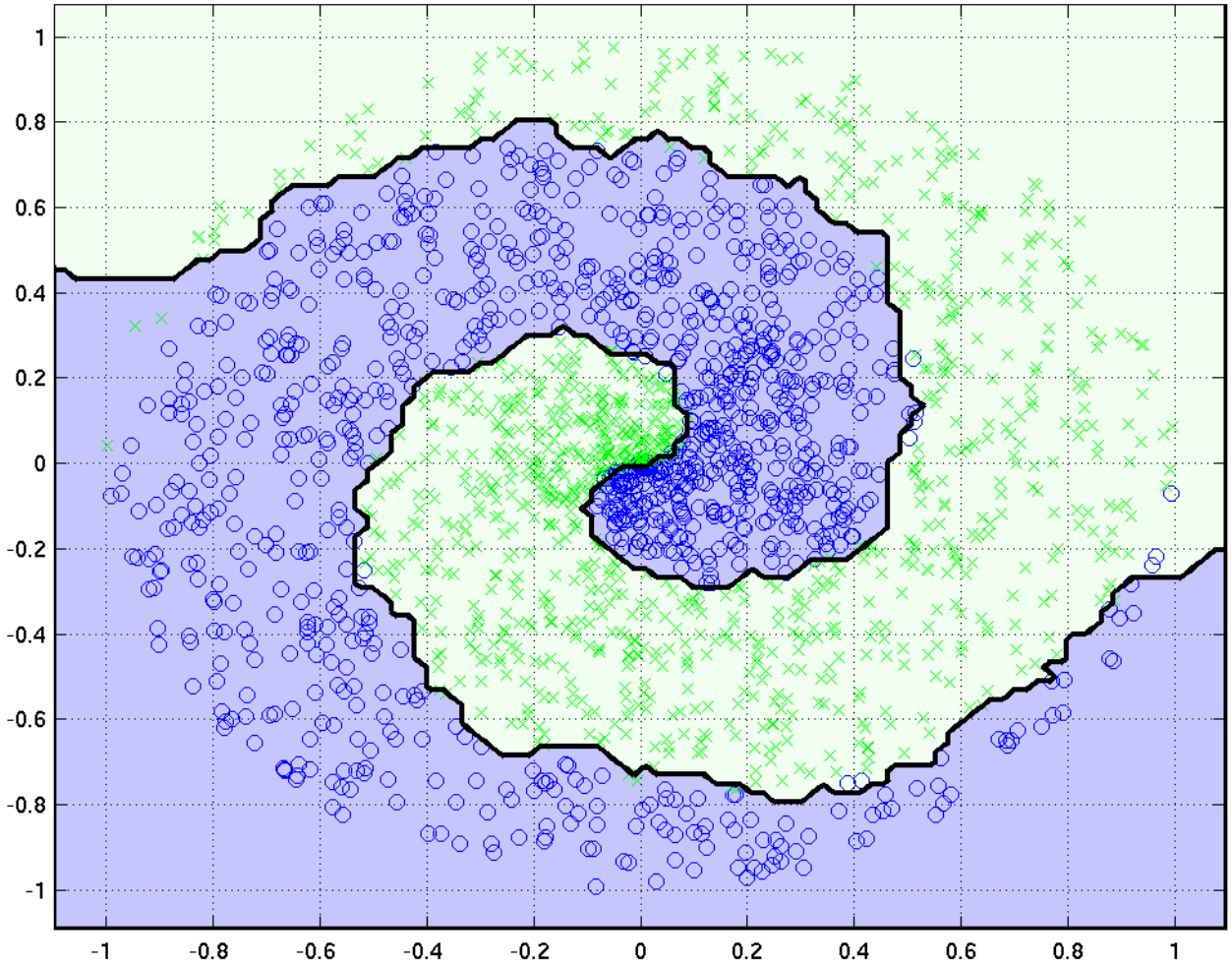
Decision boundaries $k = 1$



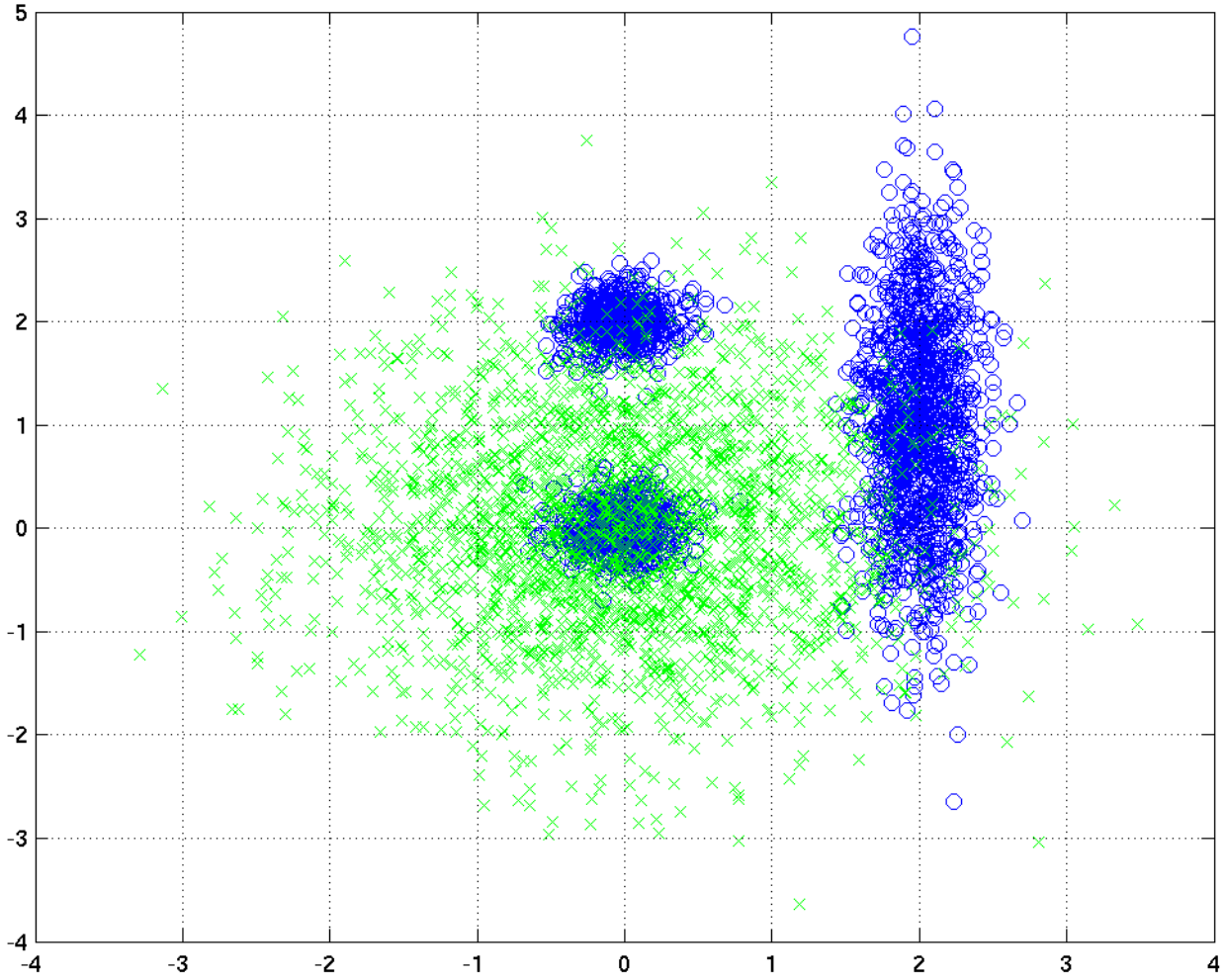
Decision boundaries $k = 5$



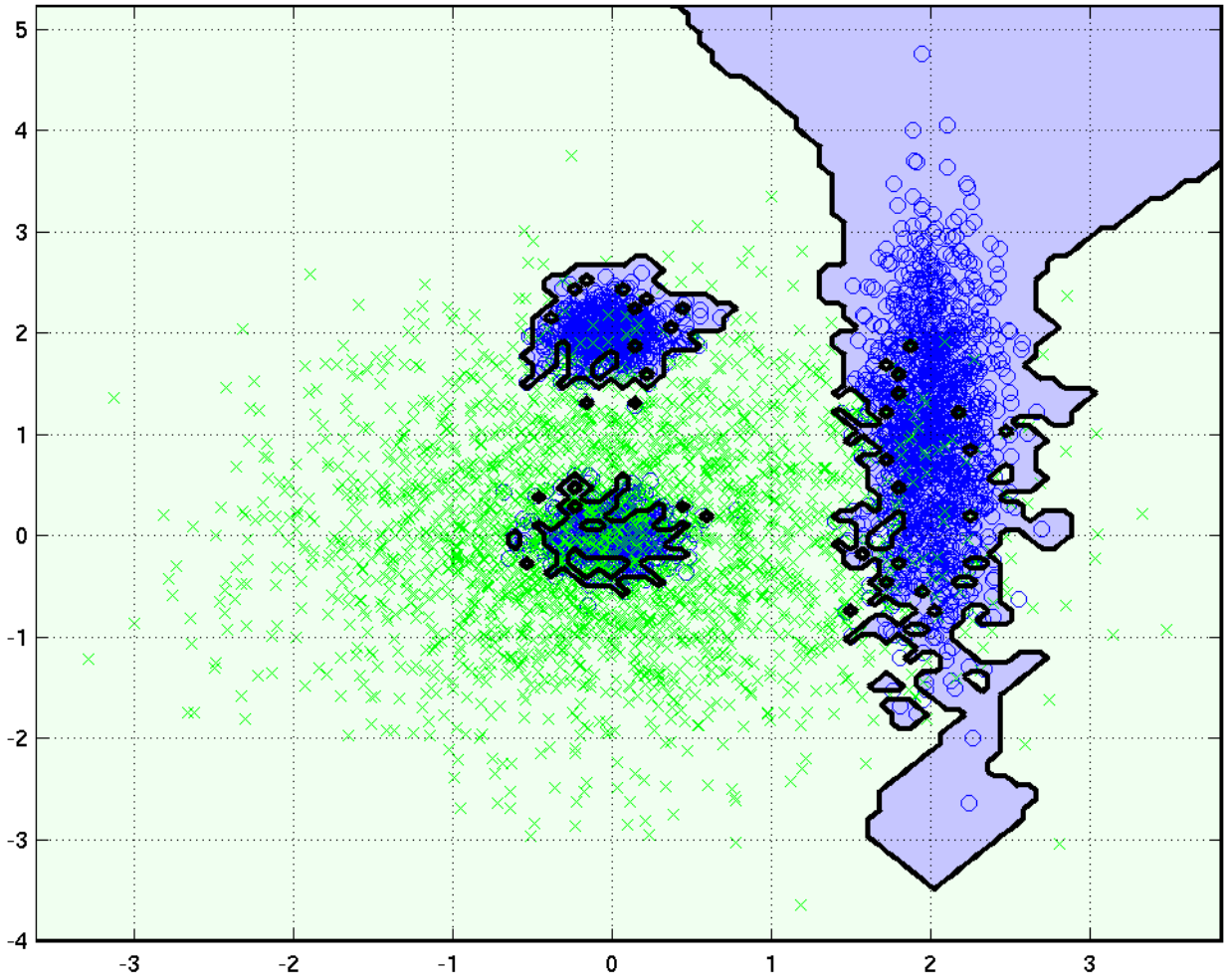
Decision boundaries $k = 10$



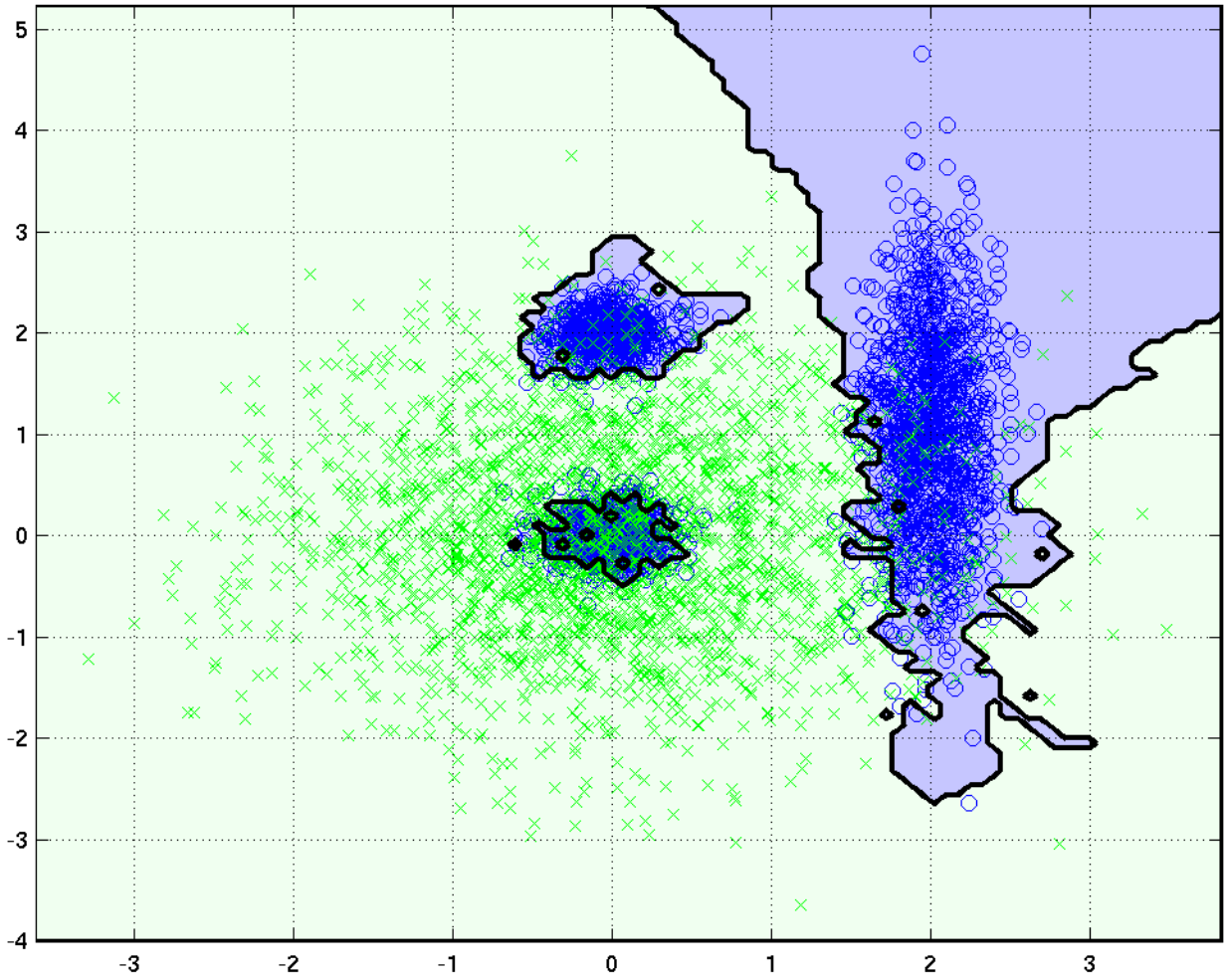
Decision boundaries



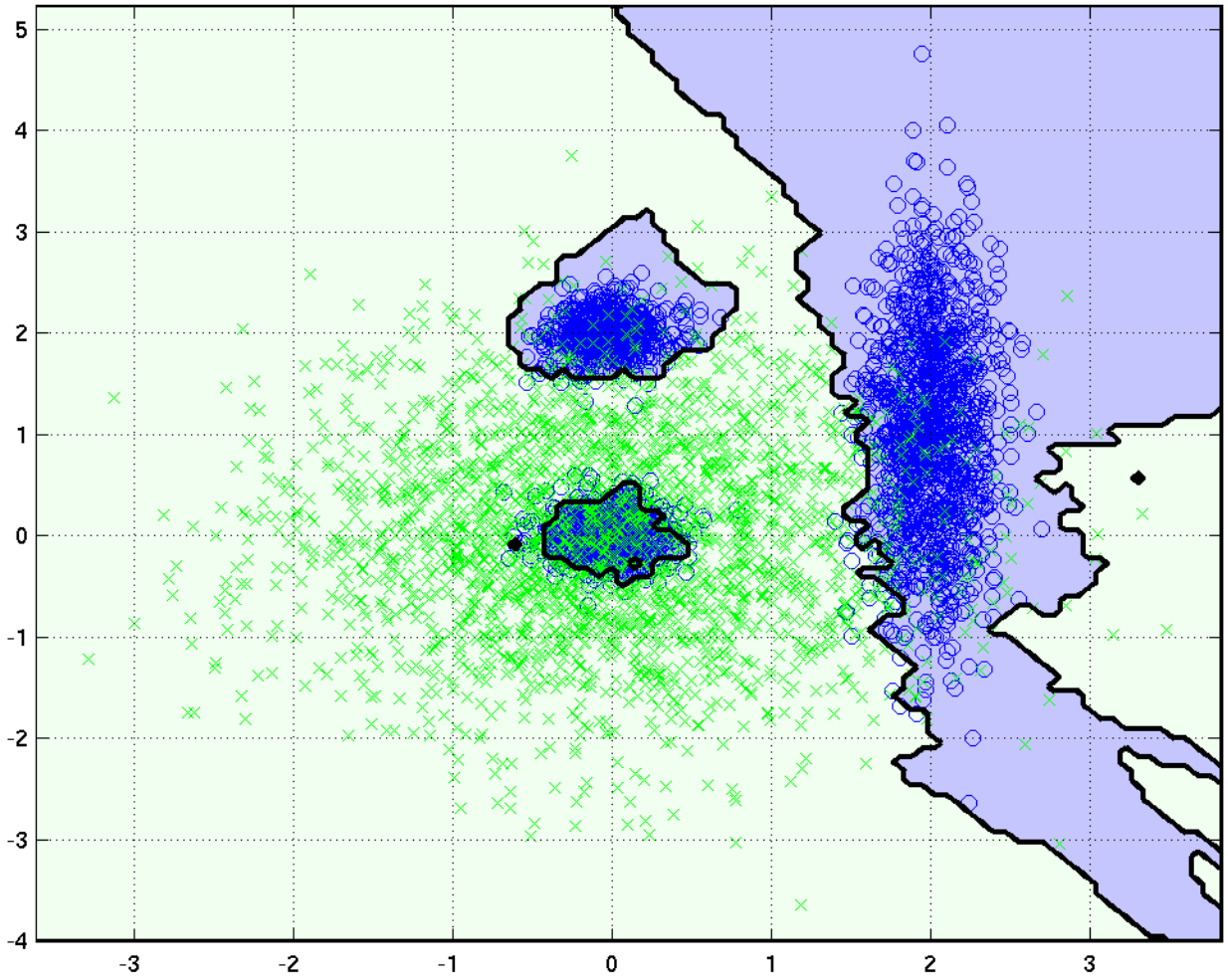
Decision boundaries $k = 1$



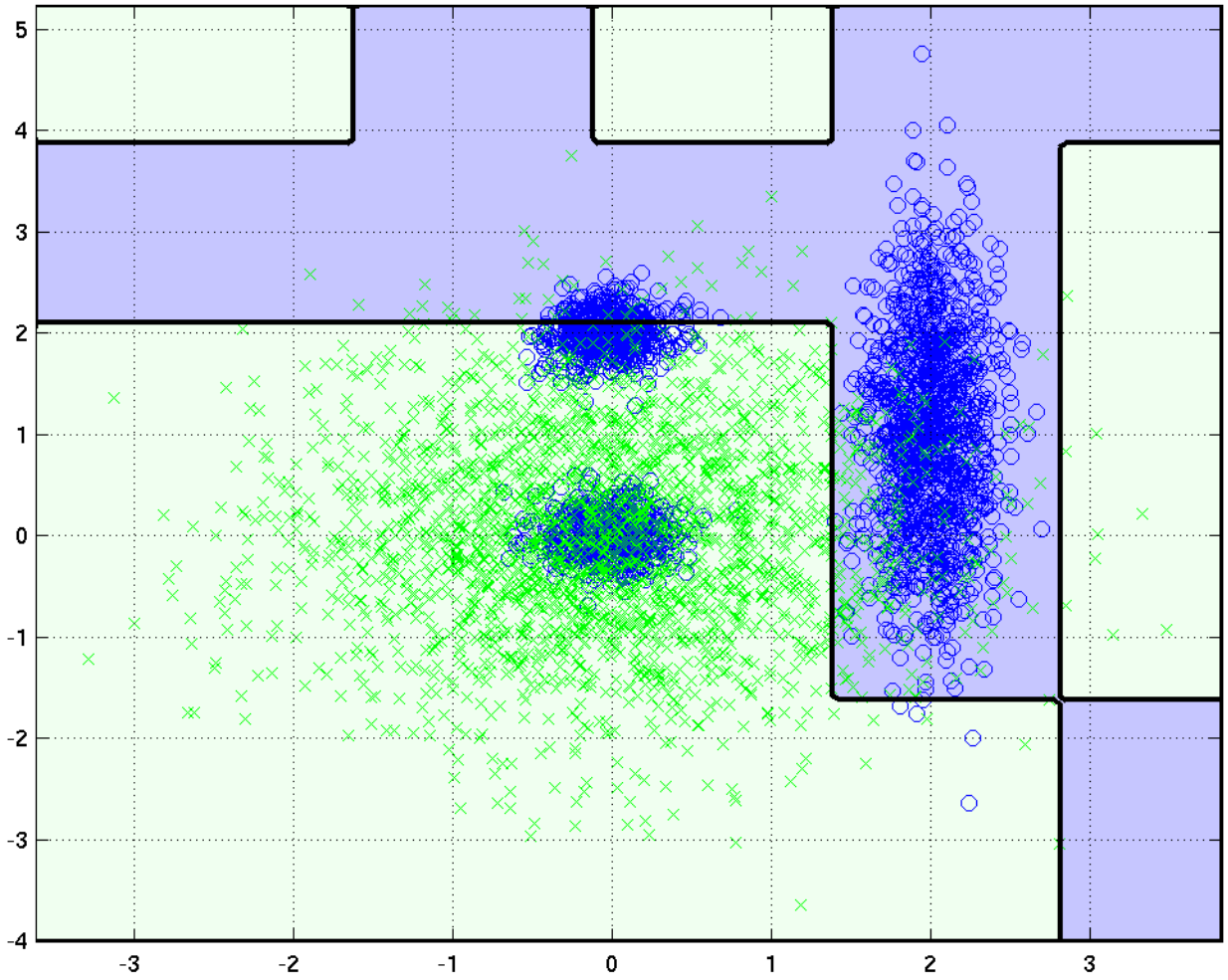
Decision boundaries $k = 5$



Decision boundaries $k = 10$



Decision boundaries ID3



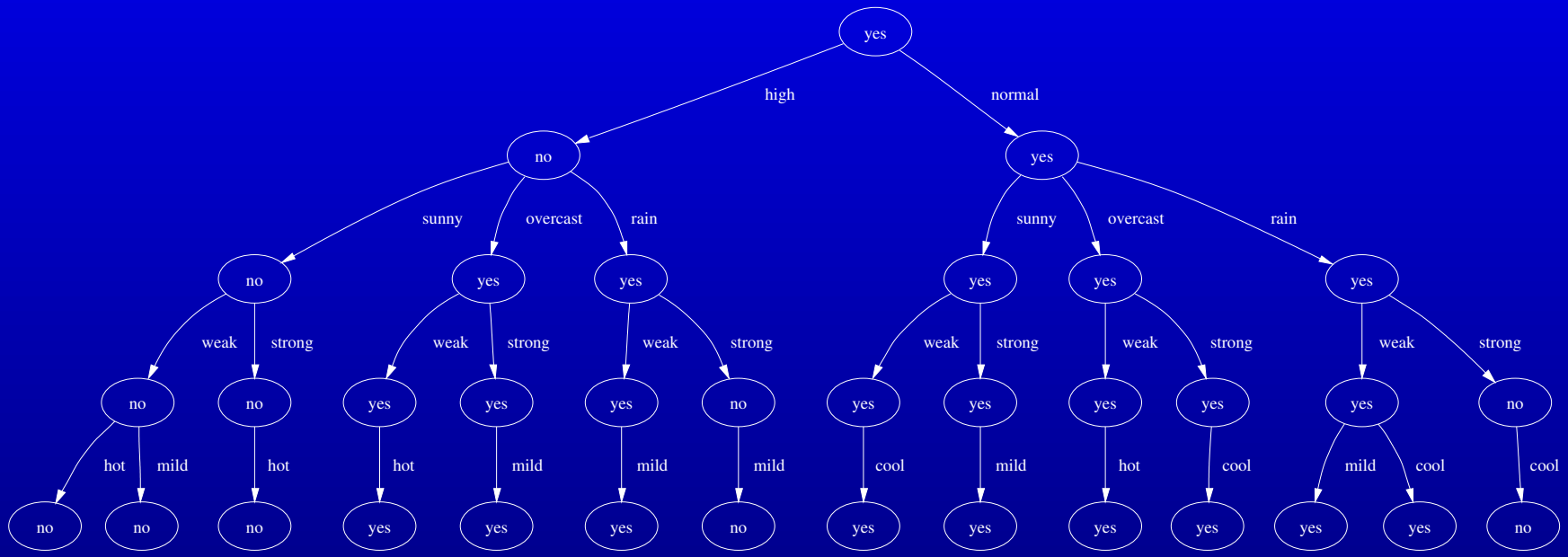
Instance indexing

- The computational cost of the training phase in IBL is minimal
- Classification can be expensive, if the training set is very large
- Careful indexing can help
- Database of instances can be organized into a *tree,* ordered by gain ratio – this gives compact storage and quick access to similar examples
- An *inverted index* lists training instances which are relevant by feature – this allows quick access to relevant examples

Example

Day	*Outlook*	*Temp*	*Humid*	*Wind*	*Play?*
d1	sunny	hot	high	weak	no
d2	sunny	hot	high	strong	no
d3	overcast	hot	high	weak	yes
d4	rain	mild	high	weak	yes
d5	rain	cool	normal	weak	yes
d6	rain	cool	normal	strong	no
d7	overcast	cool	normal	strong	yes
d8	sunny	mild	high	weak	no
d9	sunny	cool	normal	weak	yes
d10	rain	mild	normal	weak	yes
d11	sunny	mild	normal	strong	yes
d12	overcast	mild	high	strong	yes
d13	overcast	hot	normal	weak	yes
d14	rain	mild	high	strong	no

Instance indexing



IGTree

- The tree used to store the training examples is a lot like a decision tree (though it's not used as one)
- But, it could be, if we marked each node in the tree with the majority class in its descendants: IGTREE
- Daelemans, et al. describe this as an 'approximation' to IBL, but really it's a decision tree
- As a decision tree, it's limited to discrete features and target functions
- And, pruning can sometimes help

IGTree


yes
sunny
rain
no
yes
normal
high
normal
yes
yes
yes
strong
strong
no
no

TRIBL

- Classification with IGTREE is *much* faster
- The accuracy is usually comparable to IB1, though not always
- IGTREE goes wrong when a conspiracy of low-ranked features overthrows a higher-ranked feature
- TRIBL combines IGTREE and IB1: use a decision tree for features with above average gain ratio, then apply IB1 to the remaining subset of the training examples
- TRIBL2 uses IGTREE until no exact match is found, then applies IB1
- These methods give much of the speed up of IGTREE with generally a lower decrease in accuracy

Instance-based learning

- Like decision trees, IBL is a simple, useful, general purpose machine learning method applicable to a wide range of NLP problems
- Almost all of the computational cost occurs when the model is applied, which is often useful in research situations
- A high quality implementation of a number of IBL algorithms is available from the University of Tilburg: Timbl
- Often easier to use and better than C4.5, but hasn't caught on as well