

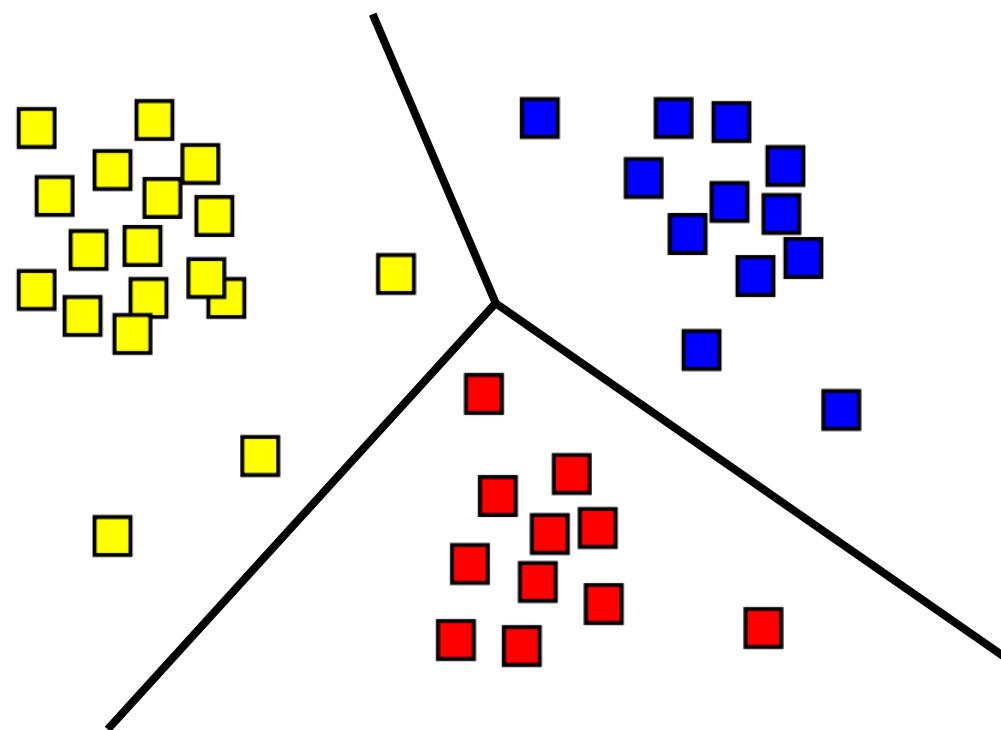
# COMP 6321 Machine Learning

## Multiclass Classification

Computer Science & Software Engineering  
Concordia University, Fall 2020



# Multiclass Classification



# Multiclass vs Multilabel vs Tagging

- **Multi-class classification:** given  $x$  predict one unique class (one “class label”) out of  $K$  classes
- **Multilabel classification:** given  $x$  predict the subset of  $K$  labels that should be associated
  - Essentially multiple-output binary classification
  - Terminology ambiguous with “labeling problems” which are essentially classification (assign one “label” to  $x$ )
- **Tagging:** multi-*label* classification, usually of images:



tags:  
“ice cream”  
“dessert”  
“bowl”  
“table”  
“cold”  
...

# Multiclass SVM strategies

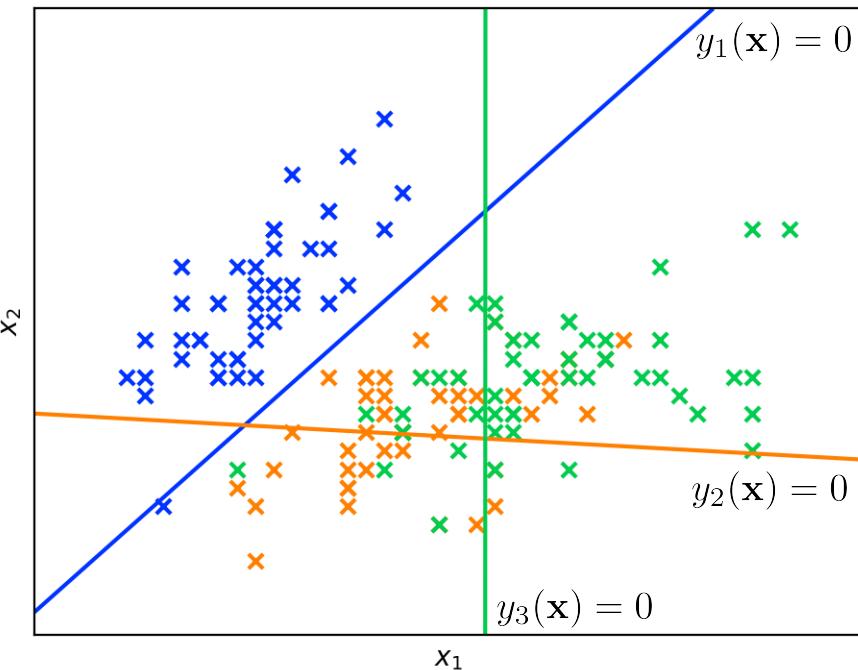
- For binary classification, the *maximum margin* principle is elegant to formulate and to solve. — quadratic program!
- For multiclass classification, the notion of “*maximum margin*” is harder to formulate.
- Can we reuse nice 2-class formulations for  $K$ -class?
  - **One-vs-Rest:** for each class  $k$ , train an SVM that is an expert at classifying  $k$  (one) versus non- $k$  (*the rest*).
  - **One-vs-One:** for each pair of classes  $(k, k')$ , train an SVM that is an expert at classifying  $k$  versus  $k'$ .
  - In either case, make multiclass prediction by combining individual binary predictions.

# One-vs-Rest (OvR)

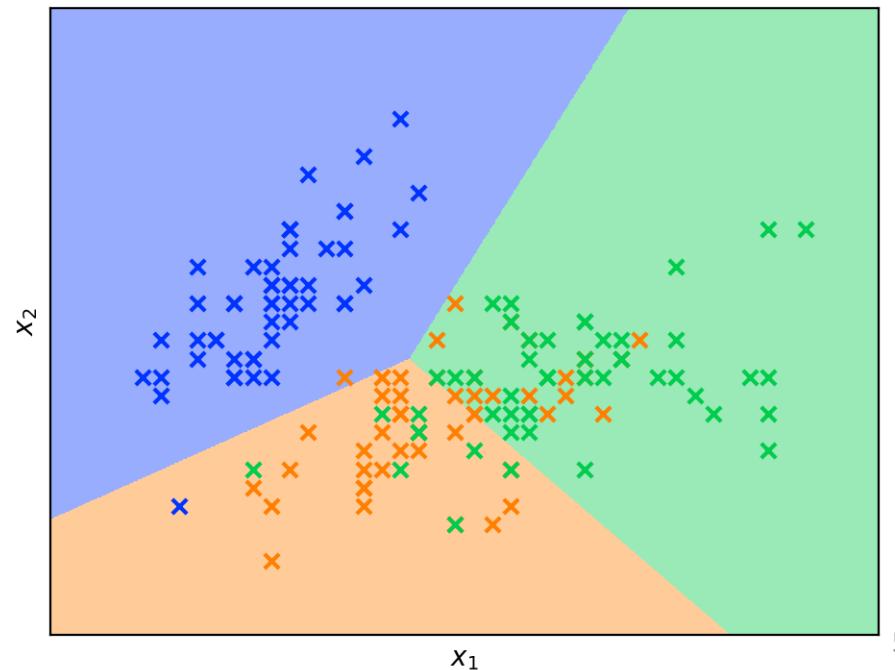
- For each class  $k$ , train an SVM that is an expert at classifying  $k$  (one) versus non- $k$  (*the rest*).
- Predict class of  $\mathbf{x}$  by choosing class  $k$  with highest decision function value  $k^* = \arg \max_{k=1,\dots,K} y_k(\mathbf{x})$

decision function  
of  $k^{\text{th}}$  binary SVM

One-vs-Rest (OvR) with linear SVM



One-vs-Rest (OvR) with linear SVM

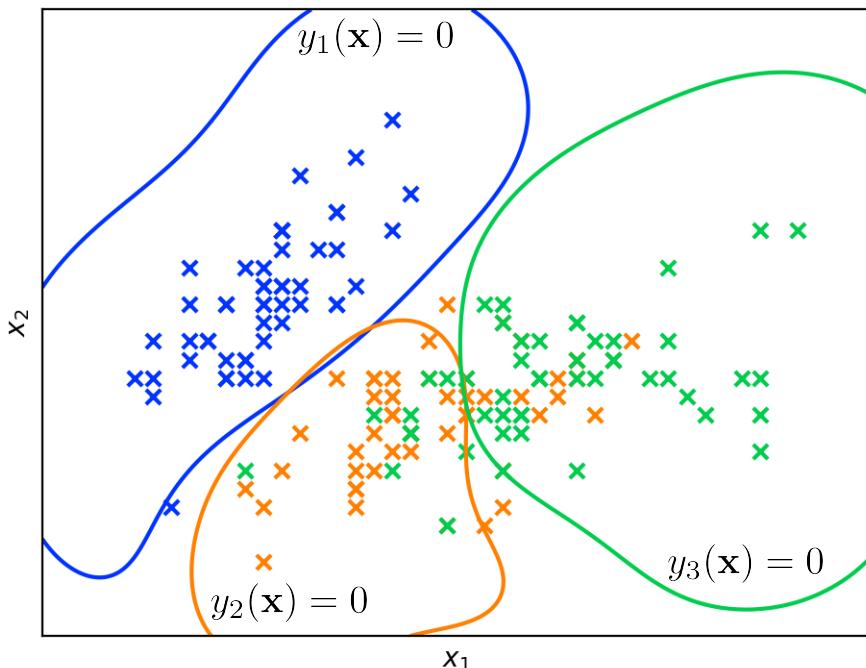


# One-vs-Rest (OvR)

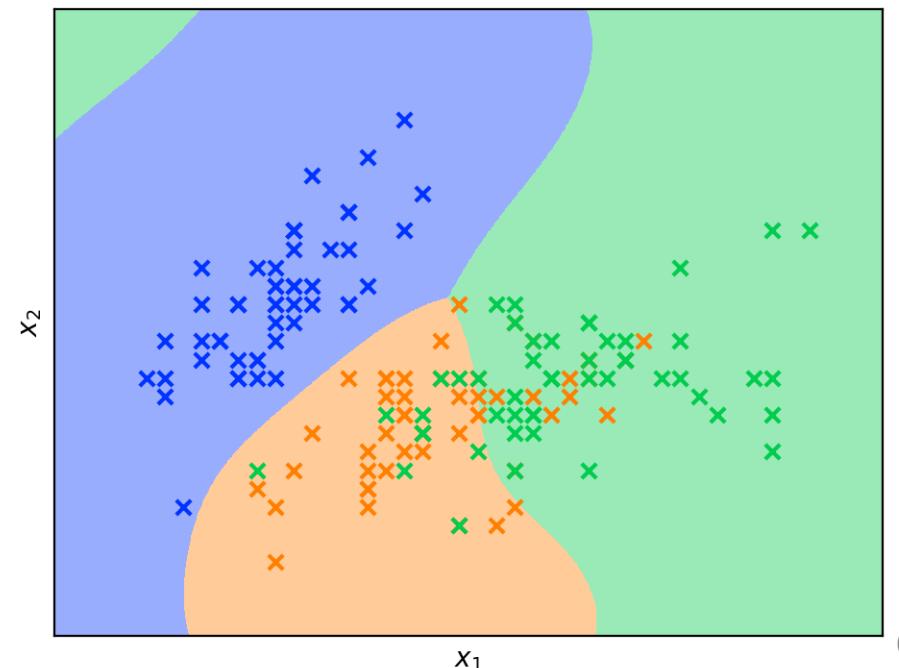
- For each class  $k$ , train an SVM that is an expert at classifying  $k$  (one) versus non- $k$  (*the rest*).
- Predict class of  $\mathbf{x}$  by choosing class  $k$  with highest decision function value  $k^* = \arg \max_{k=1,\dots,K} y_k(\mathbf{x})$

decision function  
of  $k^{\text{th}}$  binary SVM

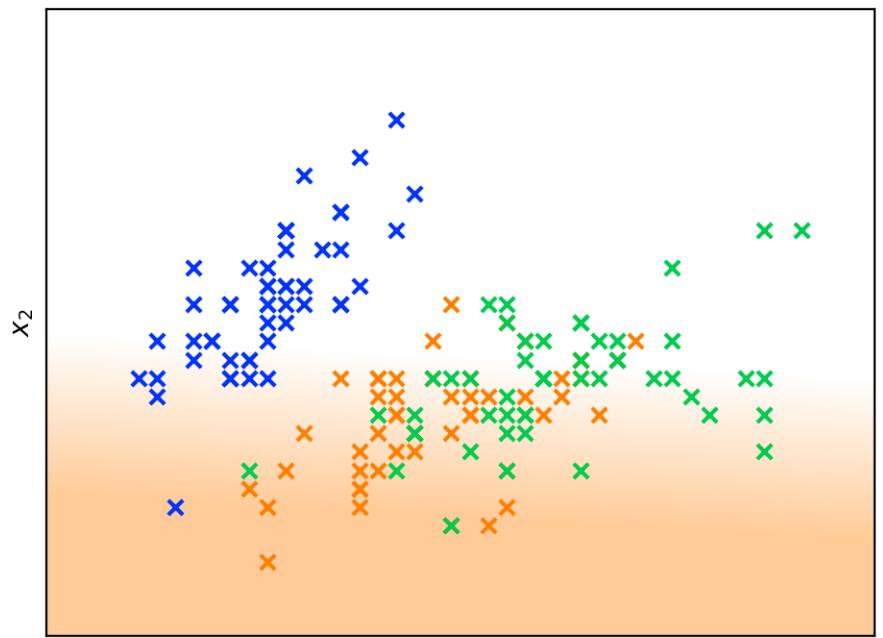
One-vs-Rest (OvR) with RBF SVM



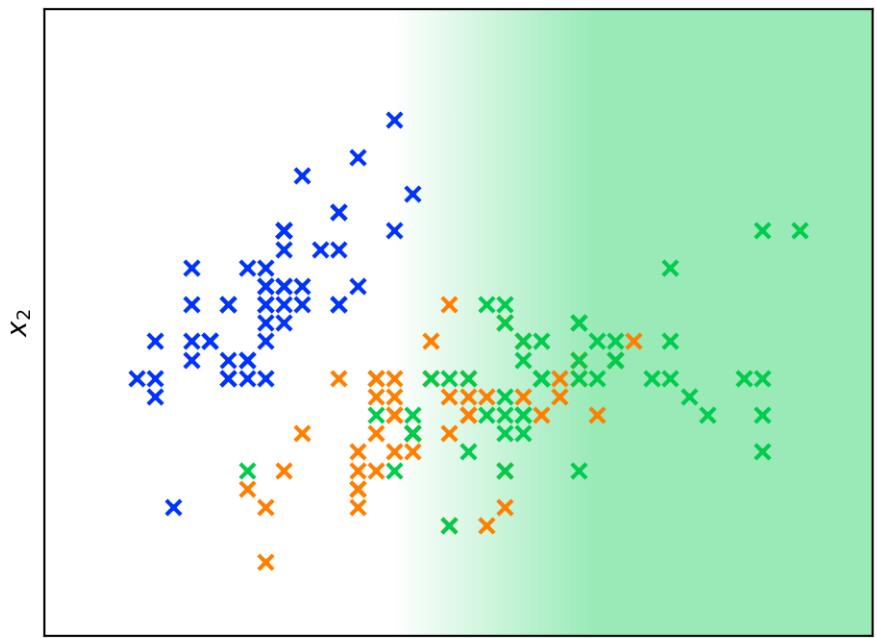
One-vs-Rest (OvR) with RBF SVM



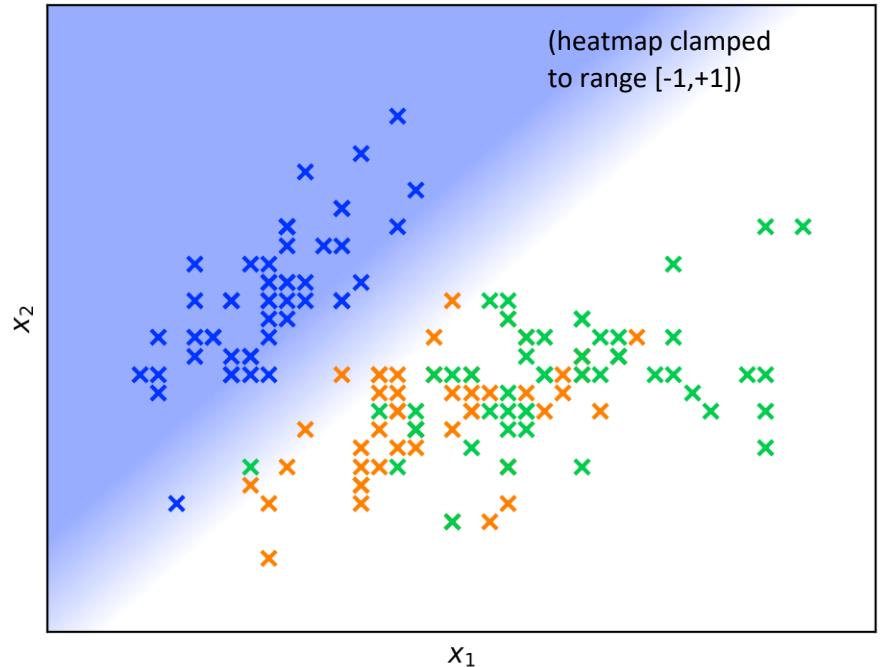
{2}-vs-{1, 3} with linear SVM



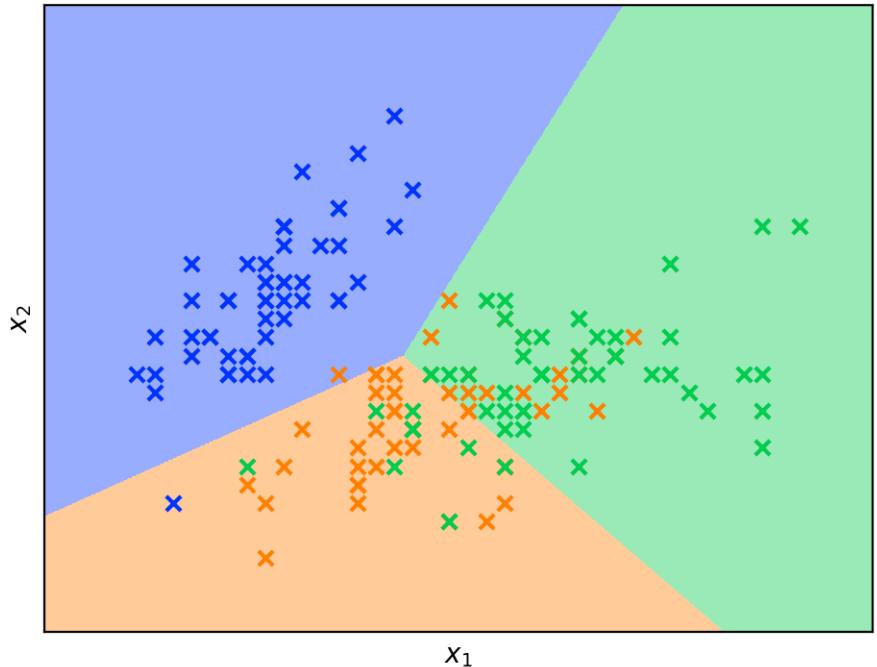
{3}-vs-{1, 2} with linear SVM



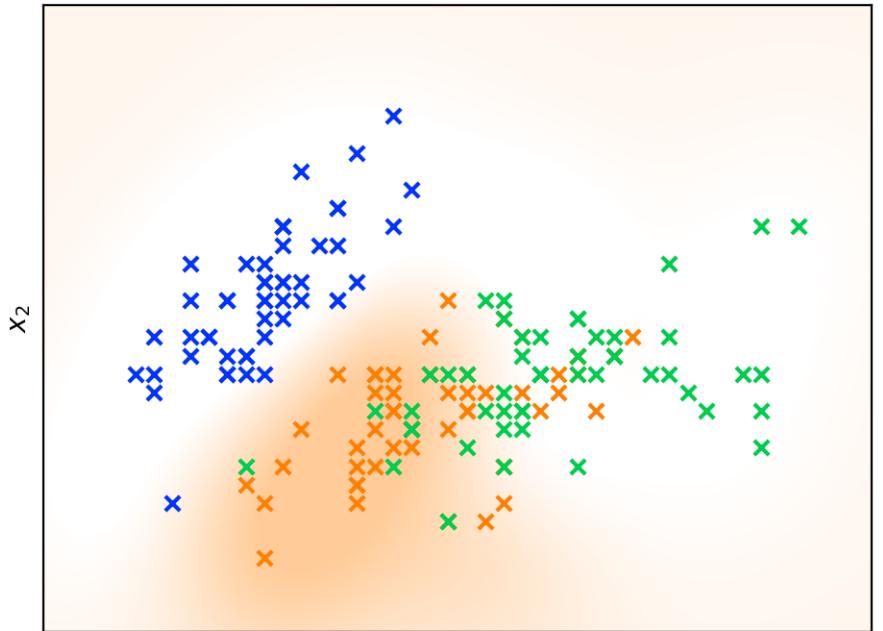
{1}-vs-{2, 3} with linear SVM



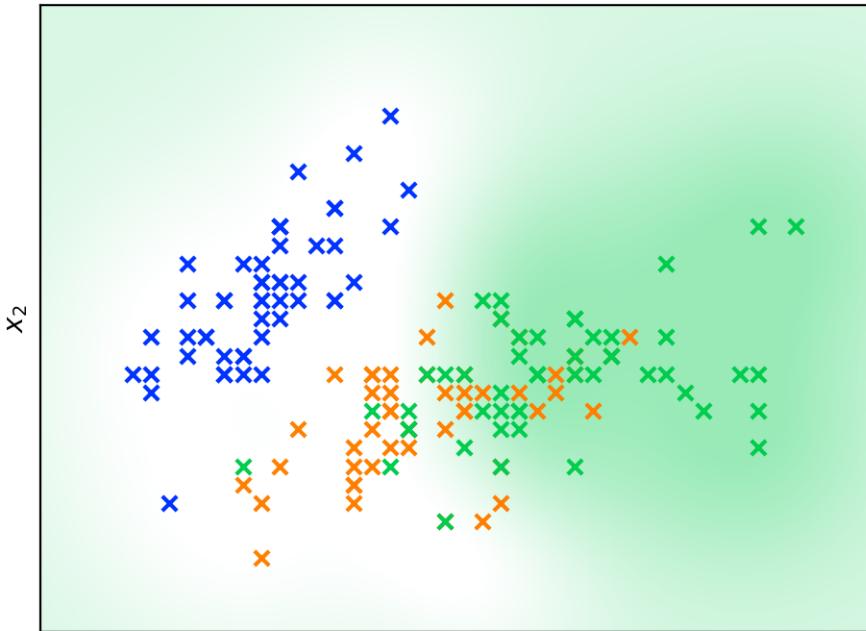
One-vs-Rest (OvR) with linear SVM



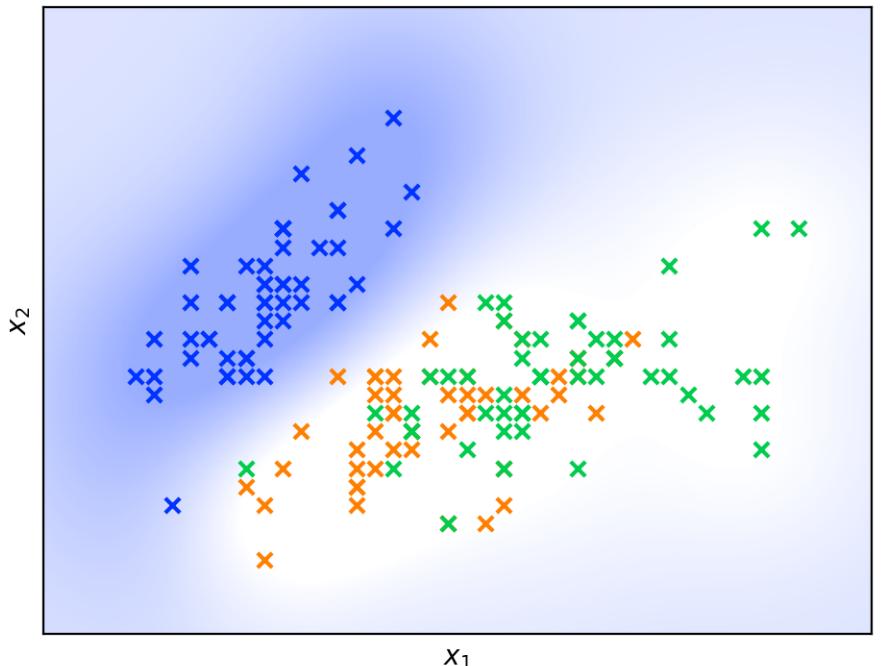
$\{2\}$ -vs- $\{1, 3\}$  with RBF SVM



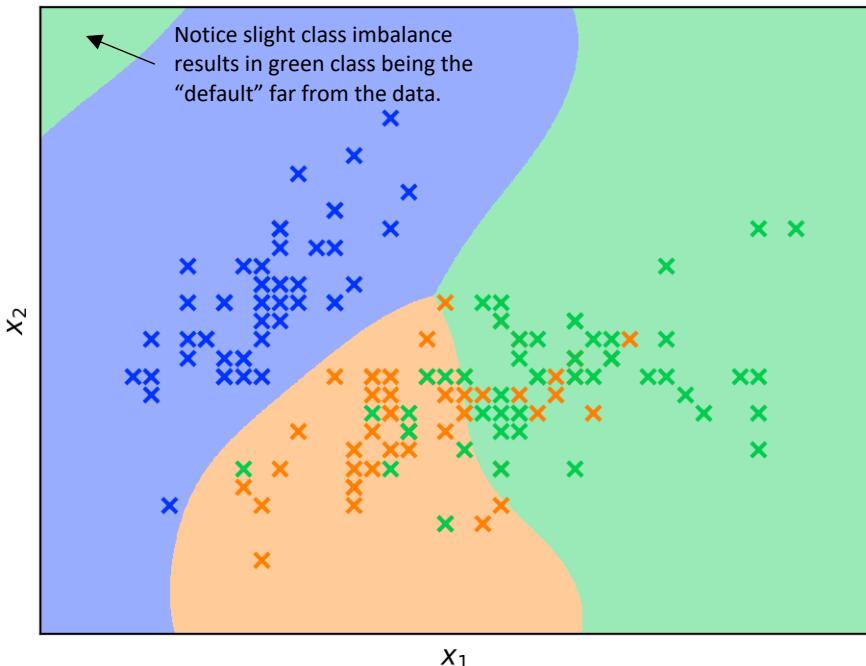
$\{3\}$ -vs- $\{1, 2\}$  with RBF SVM



$\{1\}$ -vs- $\{2, 3\}$  with RBF SVM

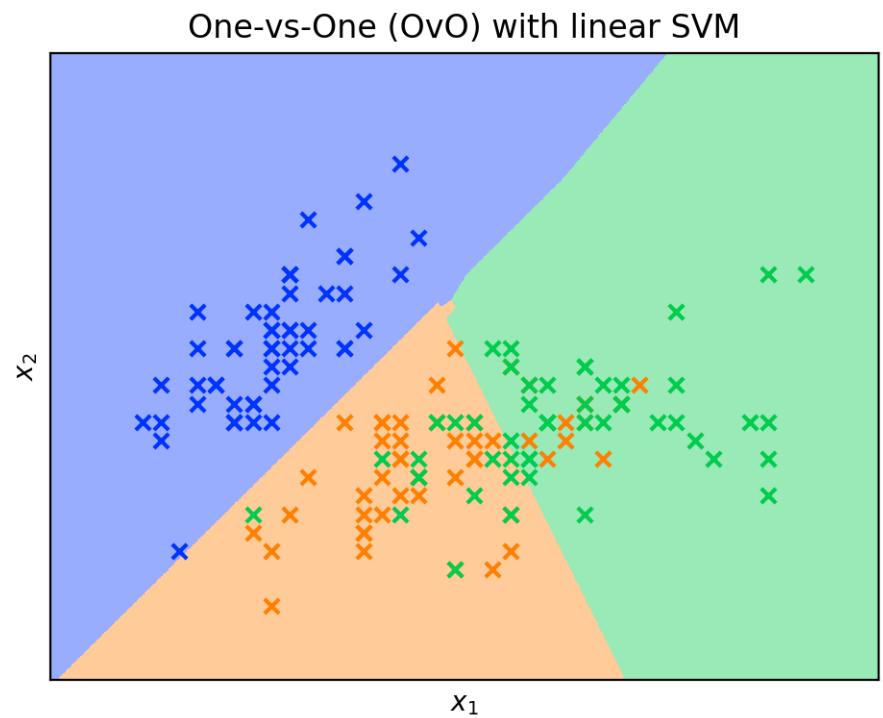
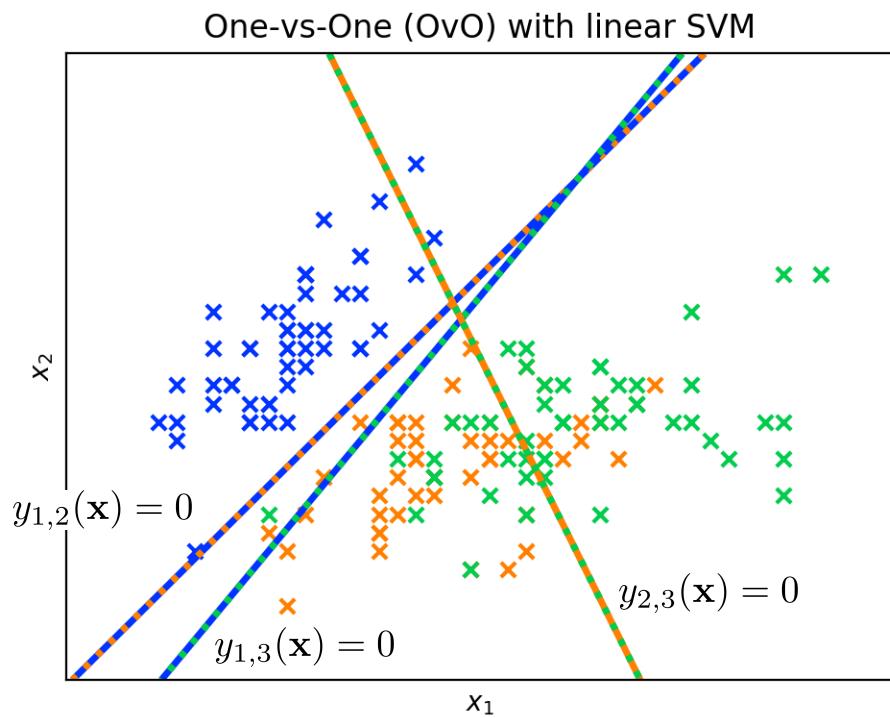


One-vs-Rest (OvR) with RBF SVM



# One-vs-One (OvO)

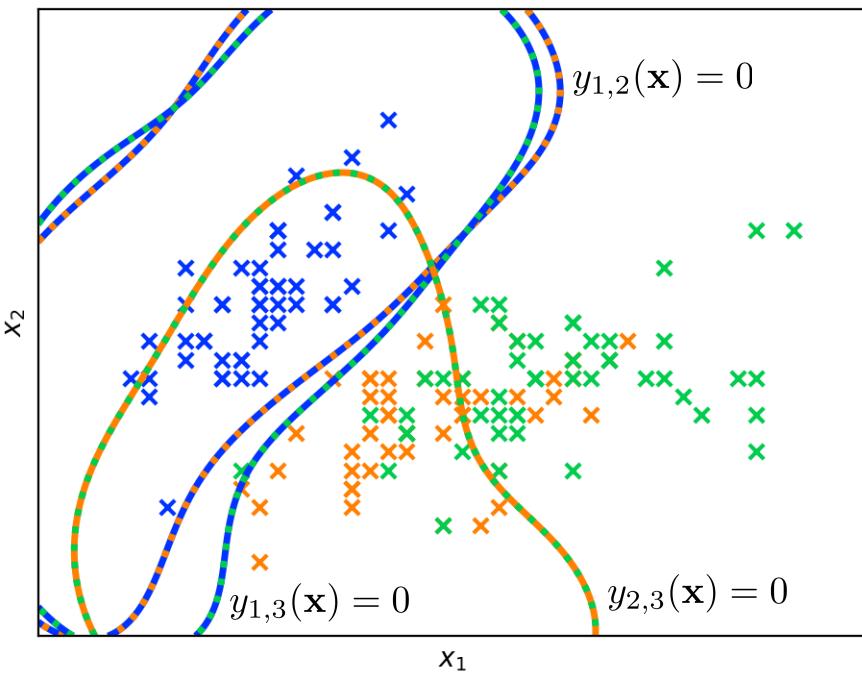
- For each pair of classes  $(k, k')$ , train an SVM that is classifies  $k$  versus  $k'$ . A total of  $K(K - 1)/2$  classifiers.
- Predict class of  $\mathbf{x}$  by choosing  $k$  with highest number of “votes” where  $y_{k,k'}(\mathbf{x}) > 0$  over all  $k'$ .



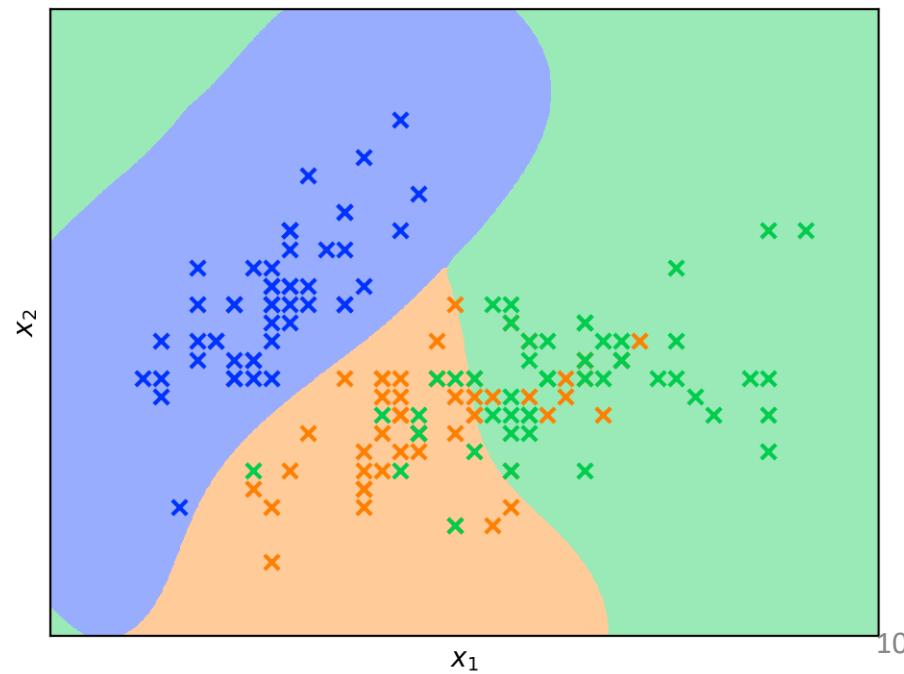
# One-vs-One (OvO)

- For each pair of classes  $(k, k')$ , train an SVM that is classifies  $k$  versus  $k'$ . A total of  $K(K - 1)/2$  classifiers.
- Predict class of  $\mathbf{x}$  by choosing  $k$  with highest number of “votes” where  $y_{k,k'}(\mathbf{x}) > 0$  over all  $k'$ .

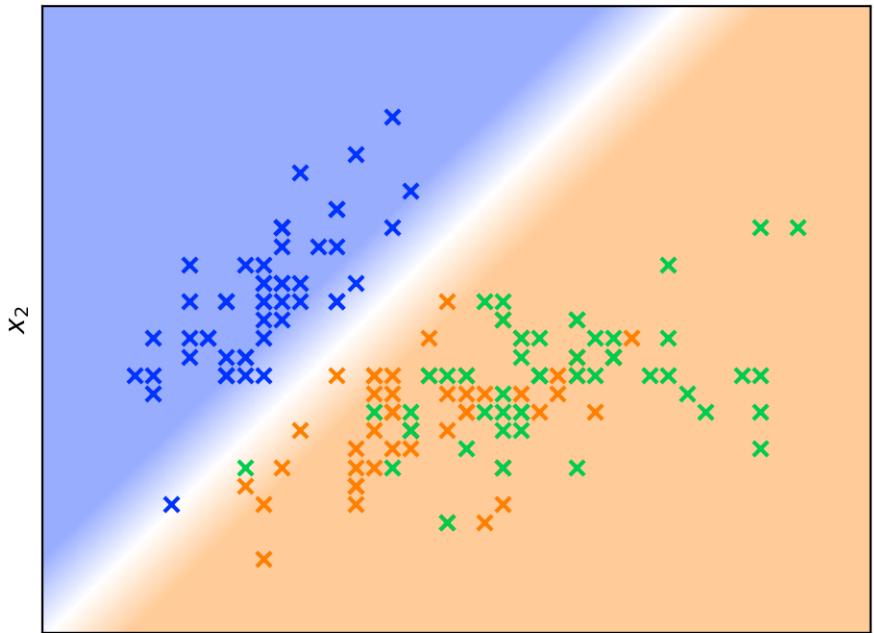
One-vs-One (OvO) with RBF SVM



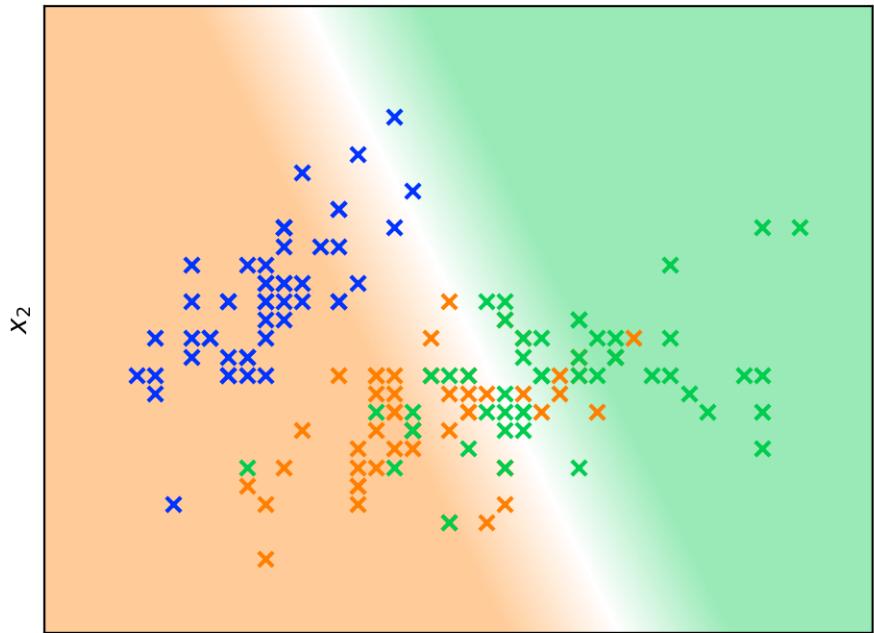
One-vs-One (OvO) with RBF SVM



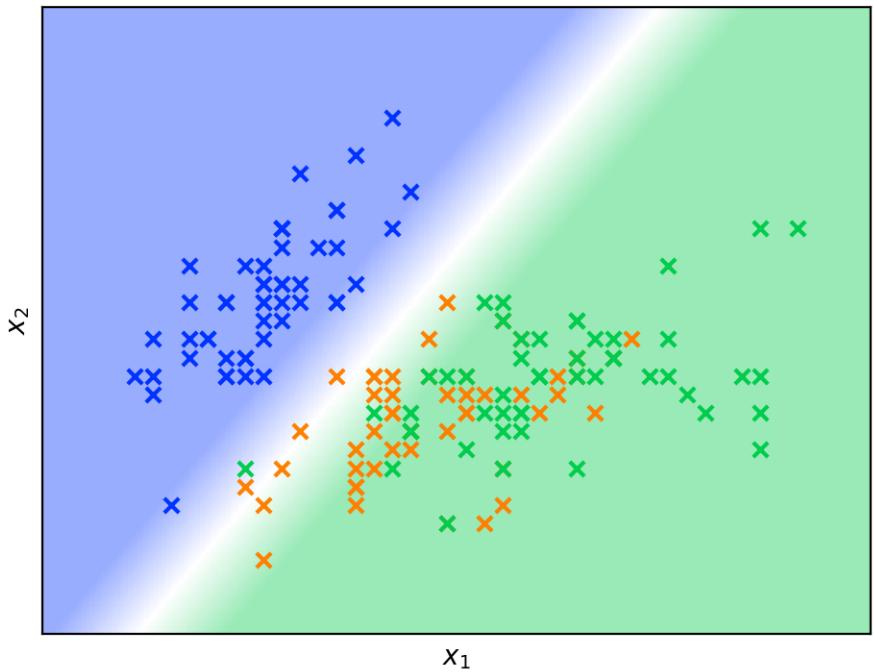
1-vs-2 with linear SVM



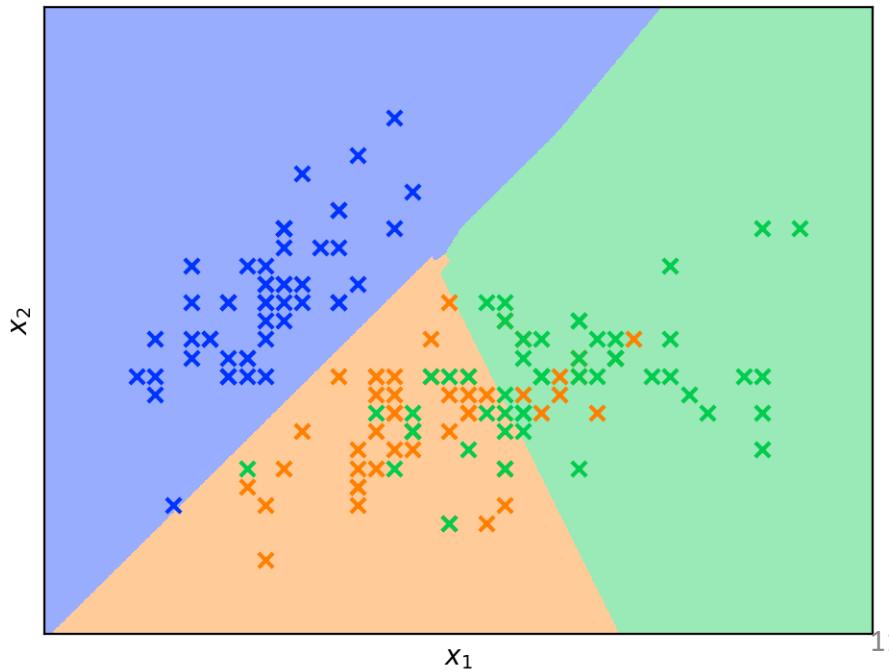
2-vs-3 with linear SVM



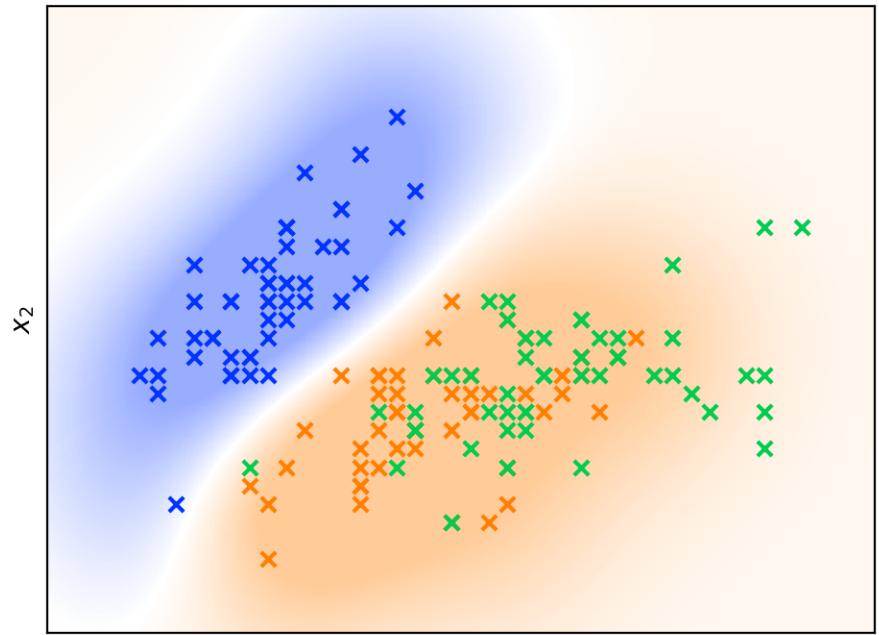
1-vs-3 with linear SVM



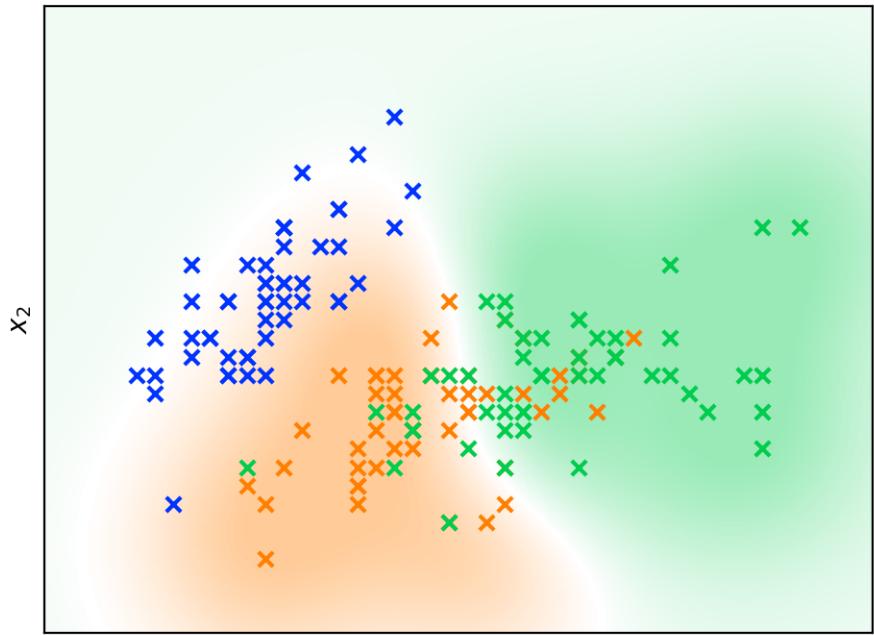
One-vs-One (OvO) with linear SVM



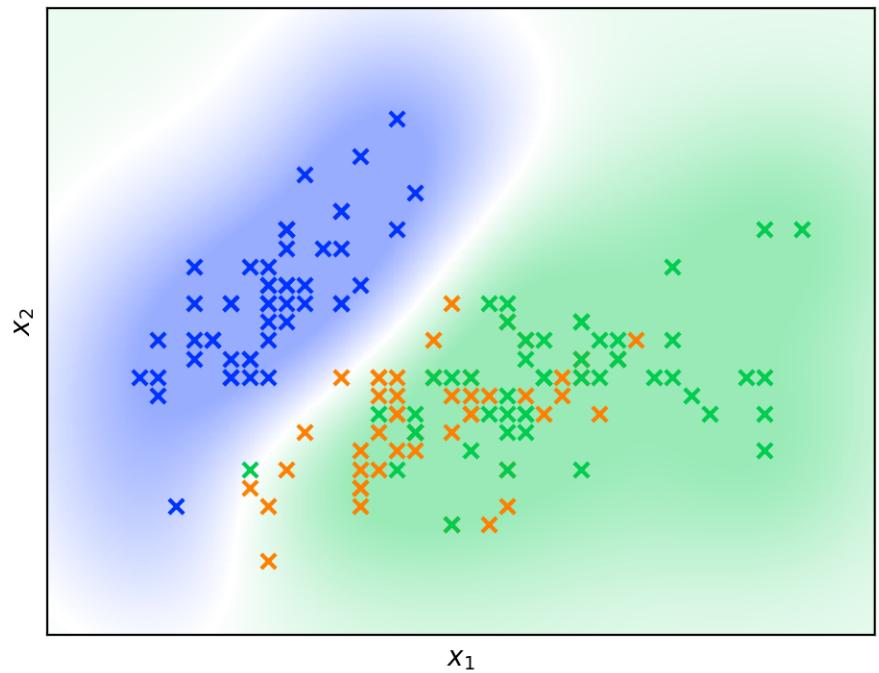
1-vs-2 with RBF SVM



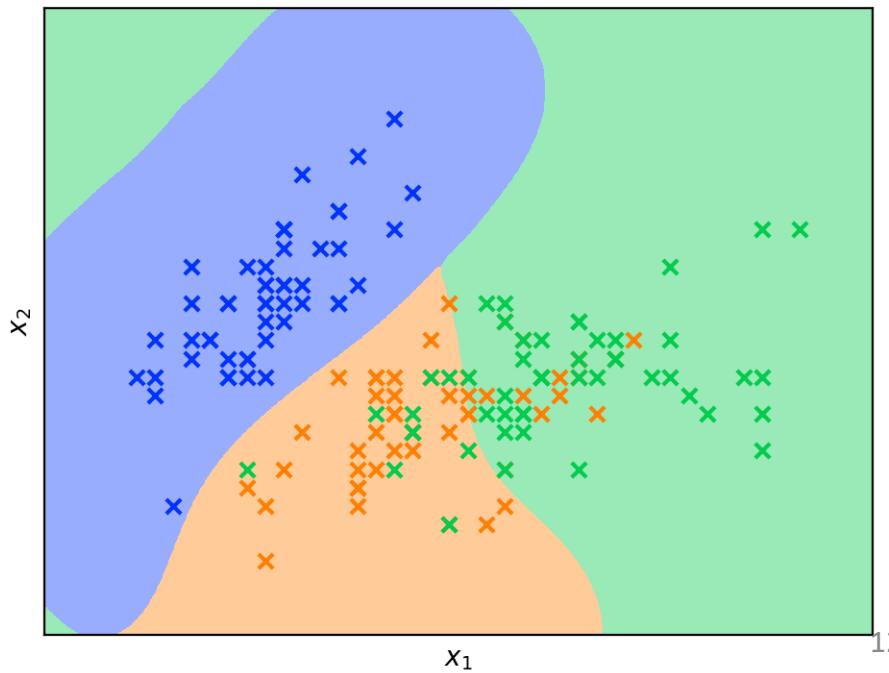
2-vs-3 with RBF SVM



1-vs-3 with RBF SVM



One-vs-One (OvO) with RBF SVM



# 1.12. Multiclass and multilabel algorithms

**Warning:** All classifiers in scikit-learn do multiclass classification out-of-the-box. You don't need to use the `sklearn.multiclass` module unless you want to experiment with different multiclass strategies.

## 1.12.2.1. Multiclass learning

Below is an example of multiclass learning using OvR:

```
>>> from sklearn import datasets
>>> from sklearn.multiclass import OneVsRestClassifier
>>> from sklearn.svm import LinearSVC
>>> iris = datasets.load_iris()
>>> X, y = iris.data, iris.target
>>> OneVsRestClassifier(LinearSVC(random_state=0)).fit(X, y).predict(X)
array([0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 2, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 2, 2, 2, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2,
       2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 1, 2, 2, 2, 2, 1, 2, 2, 2,
       2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2])
```

# Problems with these approaches

- Scale of SVM decision function  $y(\mathbf{x})$  is arbitrary
  - Strictly speaking they are *not comparable* values
  - But we compare anyway *as a heuristic*; often works
- Can be slow to train
  - One-vs-Rest trains  $K$  SVMs, each on *full* dataset
  - One-vs-One trains  $O(K^2)$  SVMs, each on *fraction* of data
- One-vs-Rest SVMs trained on imbalanced data
  - Data for the “rest” is all  $K-1$  classes; more data than class  $k$ .

Other approaches to multiclass?

# What about Logistic Regression?

- Logistic regression has similar decision function  $y(\mathbf{x})$ , but different training principle
  - maximum likelihood vs maximum margin
- Logistic regression's decision function can also be thresholded (e.g. at  $y = 0$ ) to give binary classifier
  - So, can apply OvR and OvO to logistic regression
- However, logistic regression turns out to have a *natural multiclass maximum-likelihood formulation!* 
- We'll build up to multiclass LR in 3 steps:
  1. Extend linear regression to multiple outputs
  2. Extend logistic regression to multiple outputs (trivial)
  3. Normalize those outputs with **softmax** function

# Multi-output Linear Regression

Given supervised training set with multiple outputs  
 $\mathcal{D} = \{(\mathbf{x}_1, \mathbf{y}_1), \dots, (\mathbf{x}_N, \mathbf{y}_N)\}$  where data has form

$$\mathbf{x} = [1 \quad x_1 \quad x_2 \quad \cdots \quad x_D]^T \quad (D+1)\text{-dimensional inputs}$$
$$\mathbf{y} = [y_1 \quad y_2 \quad \cdots \quad y_K]^T \quad K\text{-dimensional outputs}$$

We can define a multivariate model as  $\hat{\mathbf{y}}(\mathbf{x}, \mathbf{W}) = \mathbf{W}\mathbf{x}$   
where

$$\mathbf{W} = \begin{bmatrix} w_{1,0} & w_{1,1} & \cdots & w_{1,D} \\ w_{2,0} & w_{2,1} & \cdots & w_{2,D} \\ \vdots & \vdots & \ddots & \vdots \\ w_{K,0} & w_{K,1} & \cdots & w_{K,D} \end{bmatrix}$$

Row  $k$  is a parameter vector  
 $\mathbf{w}_k = [w_{k,0} \quad w_{k,1} \quad \cdots \quad w_{k,D}]^T$   
of a single independent linear model,  
where row  $i$  determines output  $y_i$

$\hat{y}_{ik} \equiv \hat{y}_k(\mathbf{x}_i, \mathbf{w}_k)$

And can of course define a loss  $\ell(\mathbf{W}) = \frac{1}{2} \sum_{i=1}^N \sum_{k=1}^K (y_{ik} - \hat{y}_{ik})^2$

# Multi-output Logistic Regression

Slap an element-wise sigmoid onto the output

$$\hat{\mathbf{y}}(\mathbf{x}, \mathbf{W}) = \sigma(\mathbf{W}\mathbf{x})$$

Since this is just regressing multiple standard logistic models, one per output, can define multivariate logistic regression loss in terms of NLL as before:

$$\begin{aligned} \ell(\mathbf{W}) &= -\ln p(\mathbf{Y} \mid \mathbf{X}, \mathbf{W}) \quad (\text{negative log likelihood}) \\ &= -\sum_{i=1}^N \sum_{k=1}^K y_{ik} \ln \hat{y}_{ik} + (1 - y_{ik}) \ln(1 - \hat{y}_{ik}) \end{aligned}$$

thereby maximizing the likelihood of all training targets  $\mathbf{Y} = \begin{bmatrix} \mathbf{y}_1^T \\ \mathbf{y}_2^T \\ \vdots \\ \mathbf{y}_N^T \end{bmatrix}$

Using  $\hat{y}_{ik}$  for *multilabel* is obvious, but is this *multiclass*?

# Toward regression for multiclass

A regression formulation for *multiclass* must involve *competition* between class labels, *i.e.* one class must ultimately “win out” among all the possible labels.

Example:  $\mathbf{y} = [1.0 \quad 0.0 \quad 0.0 \quad 0.0]^T$  (training target, 1-of- $K$ )

$$\hat{\mathbf{y}} = [0.9 \quad 0.0 \quad 0.3 \quad 0.8]^T \quad (K \text{ independent predictions})$$

What if we just train using regression, and for test-time just predict the class label with maximum predicted probability?

$$\hat{\mathbf{y}}(\mathbf{x}, \mathbf{W}) = \text{hardmax} (\sigma(\mathbf{W}\mathbf{x}))$$

where  $\text{hardmax} \begin{pmatrix} 0.9 \\ 0.0 \\ 0.3 \\ 0.8 \end{pmatrix} = \begin{pmatrix} 1 \\ 0 \\ 0 \\ 0 \end{pmatrix}$

We can, but not ideal.



1. Forces other classes to predict “zero” despite already “losing”
2. Not differentiable, so not ideal for gradient-based training

# Toward regression for multiclass

**Key:** For classification, the actual probabilities of  $\mathbf{x}$  belonging each class *cannot* be predicted independently.

Must satisfy:

$$\sum_{k=1}^K p(y_k = 1 \mid \mathbf{x}, \mathbf{W}) = 1$$

If our confidence in  $p(y_k = 1 \mid \mathbf{x}, \mathbf{W})$  goes up, then our confidence in  $p(y_{k'} = 1 \mid \mathbf{x}, \mathbf{W})$  should automatically go down for one or more  $k' \neq k$ .

**Idea:** Rather than predict  $K$  independent binary class probabilities  $\hat{y}_k = p(y_k = 1 \mid \mathbf{x}, \mathbf{w}_k)$ , predict a **joint** probability  $\hat{\mathbf{y}} = p(\mathbf{y} \mid \mathbf{x}, \mathbf{W})$  over class membership, where the model itself ensures  $\hat{y}_k \geq 0$ ,  $\sum_{k=1}^K \hat{y}_k = 1$ . <sub>19</sub>

# Multinomial Logistic Regression

Multi-output logistic regression outputs the probabilities of  $K$  independent binomial variables (flip of a 2-sided coin) so model each  $p(y = 1 \mid \mathbf{x}, \mathbf{w}_k)$  using a separate logistic model  $\mathbf{w}_k$ .

We *want* a model that outputs the probability of a single 1-of- $K$  outcome (roll of “ $K$ -sided dice”), which corresponds to multinomial random variable over values  $K$  possible outcomes, so model  $p(y = k \mid \mathbf{x}, \mathbf{W})$  using a single model with normalized output. We output a vector  $\hat{\mathbf{y}}$  of probabilities  $\hat{y}_k = p(y = k \mid \mathbf{x}, \mathbf{W})$

# Softmax function

Inputs called 'activations'  
(have *nothing* to do with  
SVM dual variables  $\mathbf{a}$ )

$$\text{softmax}(\mathbf{a}) = \frac{\exp(\mathbf{a})}{\sum_{j=1}^K \exp(a_j)}$$

**Intuition:** Amplify the largest value(s) in vector  $\mathbf{a}$ ,  
then normalize to ensure  $\sum_{k=1}^K \text{softmax}(\mathbf{a})_k = 1$ .

$$\lim_{\gamma \rightarrow \infty} \text{softmax}(\gamma \mathbf{a}) = \text{hardmax}(\mathbf{a}) \quad \frac{\exp(a_1)}{\exp(a_1) + \exp(a_2)}$$

**Example:**  $\text{softmax} \left( \begin{bmatrix} -2 \\ 2 \end{bmatrix} \right) = \begin{bmatrix} 0.018 \\ 0.982 \end{bmatrix}$

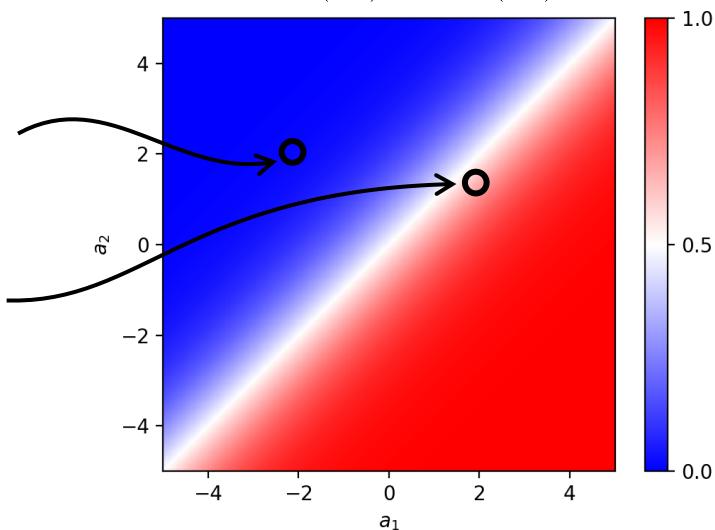
Notice connection  
to logistic sigmoid...

$$\sigma(2 - \frac{3}{2}) = 0.622$$

$$\sigma(20 - 15) = 0.993$$

$$\text{softmax} \left( \begin{bmatrix} 2 \\ \frac{3}{2} \end{bmatrix} \right) = \begin{bmatrix} 0.622 \\ 0.378 \end{bmatrix}$$

$$\text{softmax} \left( \begin{bmatrix} 20 \\ 15 \end{bmatrix} \right) = \begin{bmatrix} 0.993 \\ 0.007 \end{bmatrix}$$



# Softmax vs Logistic Sigmoid

Consider 2-class softmax of linear function

$$\begin{bmatrix} \hat{y}_1 \\ \hat{y}_2 \end{bmatrix} = \text{softmax} \left( \begin{bmatrix} a_1 \\ a_2 \end{bmatrix} \right) = \frac{1}{e^{a_1} + e^{a_2}} \begin{bmatrix} e^{a_1} \\ e^{a_2} \end{bmatrix}$$

Focus on just  $\hat{y}_1$ :  $\hat{y}_1 = \frac{e^{a_1}}{e^{a_1} + e^{a_2}} = \frac{1}{1 + e^{a_2 - a_1}}$

Assume softmax inputs are linear function  $\mathbf{a} = \mathbf{Wx}$   $\mathbf{w} = \begin{bmatrix} \mathbf{w}_1^T \\ \mathbf{w}_2^T \end{bmatrix}$

$$\hat{y}_1 = \frac{1}{1 + e^{(\mathbf{w}_2^T - \mathbf{w}_1^T)\mathbf{x}}} = \boxed{\sigma(\mathbf{w}^T \mathbf{x})} \text{ where we let } \boxed{\mathbf{w} = \mathbf{w}_1 - \mathbf{w}_2}$$

Similarly  $\hat{y}_2 = 1 - \hat{y}_1$ , so 2-class softmax regression is just **over-parameterized logistic regression!**

(“Softmax regression”)

# Multinomial Logistic Regression

**Prediction:**  $\hat{y}(\mathbf{x}, \mathbf{W}) = \text{softmax}(\mathbf{W}\mathbf{x})$        $\mathbf{W} = \begin{bmatrix} \mathbf{w}_1^T \\ \vdots \\ \mathbf{w}_K^T \end{bmatrix}$

In other words, each component  $\hat{y}_k$  of the output (the predicted class probabilities) depends on *all K* linear decision functions fed into the softmax.

Directly models *class probabilities*  $\hat{y}_k = p(y = k \mid \mathbf{x}, \mathbf{W})$

**Likelihood:**  $p(\mathbf{Y} \mid \mathbf{X}, \mathbf{W}) = \prod_{i=1}^N \prod_{k=1}^K \hat{y}_{ik}^{y_{ik}}$        $\mathbf{Y} = \begin{bmatrix} \mathbf{y}_1^T \\ \vdots \\ \mathbf{y}_N^T \end{bmatrix}$

“Only the one  $\hat{y}_{ik}$  for which  $y_{ik} = 1$  matters in this product.

Maximizing likelihood means increase the  $\hat{y}_{ik}$  of the ‘correct’ class.

(“Softmax regression”)

# Multinomial Logistic Regression

**Negative log likelihood for multinomial LR:**

$$\ell(\mathbf{W}) = -\ln p(\mathbf{Y} \mid \mathbf{X}, \mathbf{W}) = -\sum_{i=1}^N \sum_{k=1}^K y_{ik} \ln \hat{y}_{ik}$$

$$\hat{y}_{ik} \equiv \hat{y}_k(\mathbf{x}_i, \mathbf{W})$$

Compare to NLL of 2-class logistic regression below:

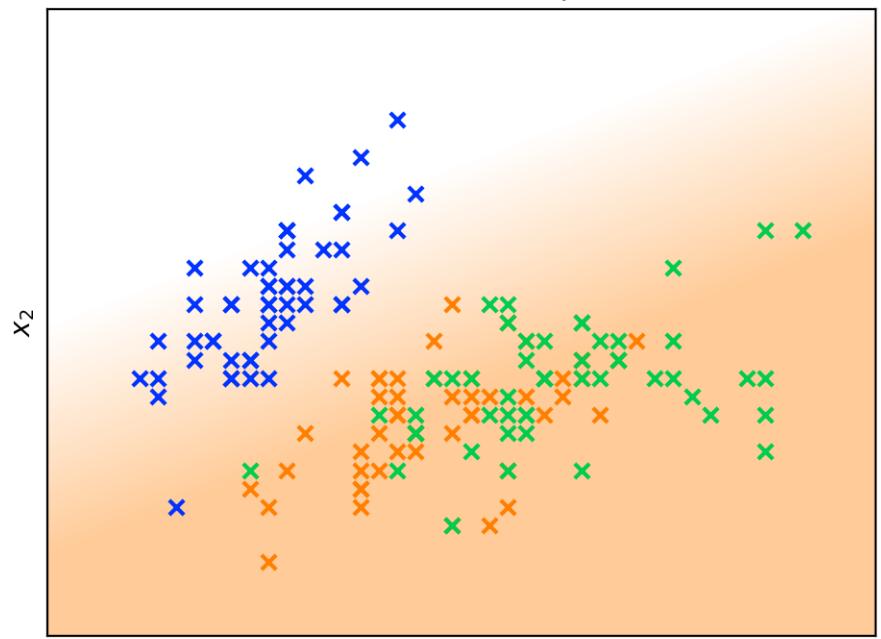
$$-\sum_{i=1}^N y_i \underbrace{\ln \hat{y}_i + (1 - y_i) \ln(1 - \hat{y}_i)}_{}$$

2-class NLL took the same form  
as this version! Logistic regression  
implicitly sums over  $K=2$  classes,  
so the subscript for  $k$  is omitted.

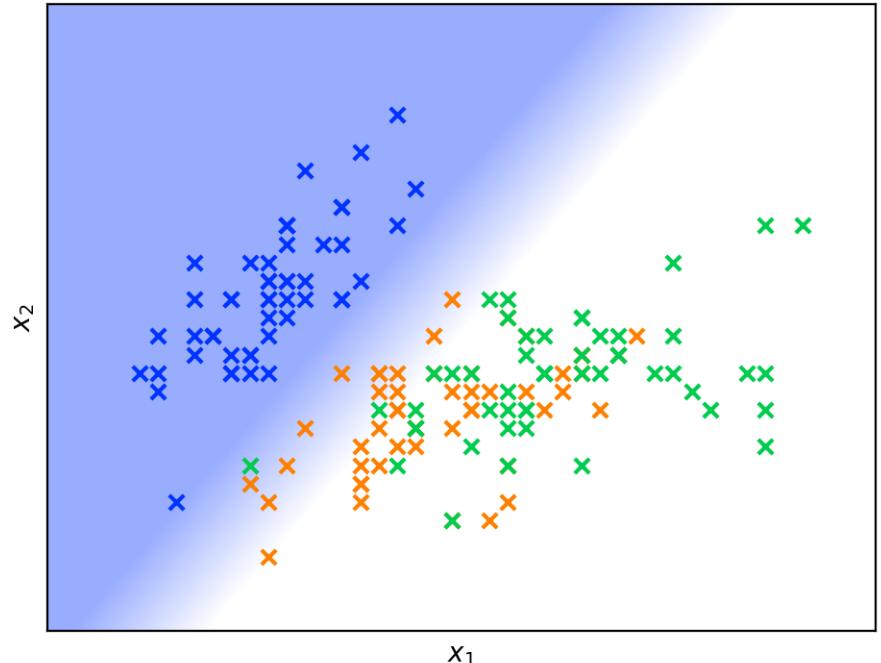
$$= \sum_{i=1}^N y_{i1} \ln \hat{y}_{i1} + y_{i2} \ln \hat{y}_{i2}$$

with  $\hat{y}_2 = 1 - \hat{y}_1$

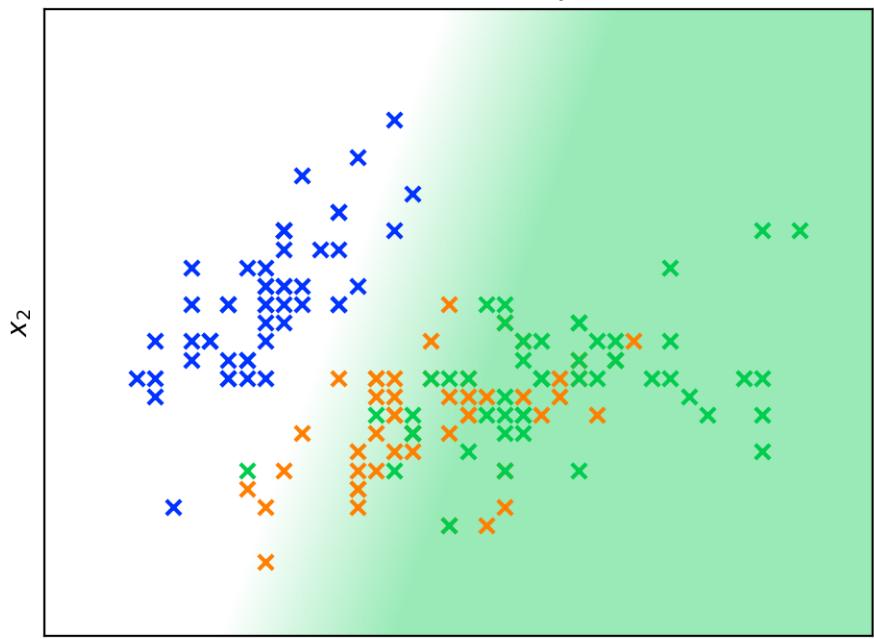
Activation for class 2 clamped to [-1,+1]



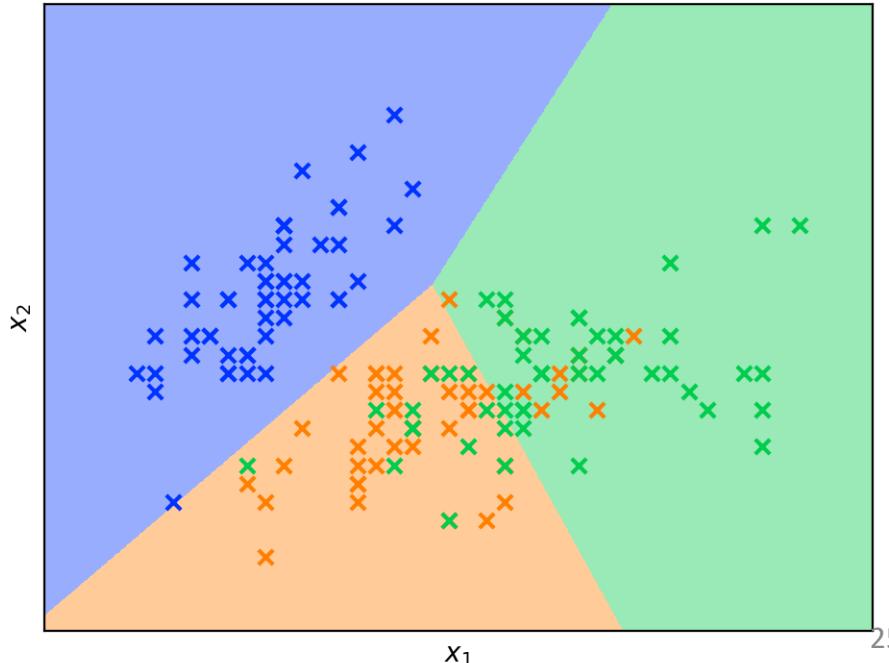
Activation for class 1 clamped to [-1,+1]



Activation for class 3 clamped to [-1,+1]



Classifications from softmax regression

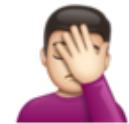


# $k$ -Nearest Neighbours

- Softmax regression (multinomial logistic regression) is a parametric model.
- SVM is a non-parametric model. Are there other *non-parametric* multiclass classification methods?
- Suppose given training set  $\mathcal{D} = \{(\mathbf{x}_1, y_1), \dots, (\mathbf{x}_N, y_N)\}$  where each  $y_i \in \{1, \dots, C\}$  is a class label.
- **Idea:** given a new sample  $\mathbf{x}$ , predict its class label to be the same as the  $y_i$  of the “most similar”  $\mathbf{x}_i$ 
  - The 1-nearest neighbour (1-NN) classification algorithm!

# $k$ -Nearest Neighbours

This  $k$  has nothing to do with the number  $K$  of classes!



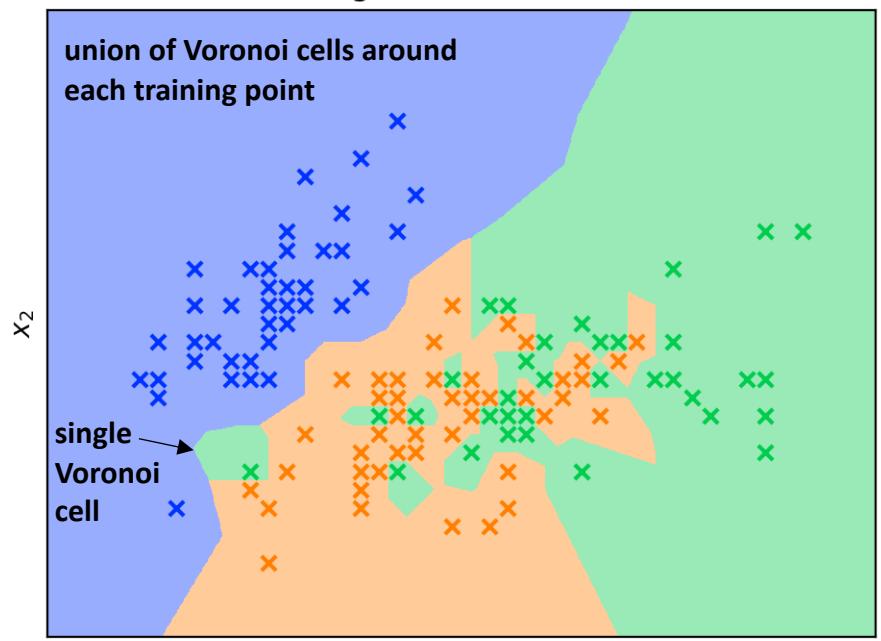
**Idea:** Given a new sample  $\mathbf{x}$  and a number  $k$ , we...

1. Find the  $k$  “nearest neighbours” of  $\mathbf{x}$ , i.e. the  $k$  training instances  $\mathbf{x}_i$  that are closest to  $\mathbf{x}$  by some metric (e.g.  $\|\mathbf{x} - \mathbf{x}_i\|^2$ ) .
2. Predict the class of  $\mathbf{x}$  to be the class that wins the majority vote of the  $y_i$  corresponding to the nearest neighbours.

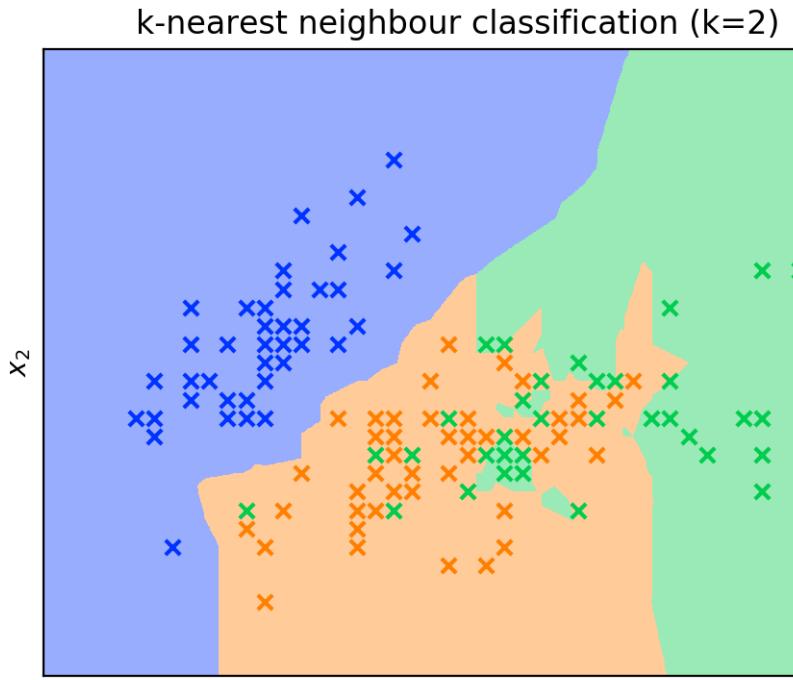
Non-parametric since the data *is* the model.

Like kernel density, prediction can be slow, so often build efficient  $D$ -dimensional search tree structure over the training data.

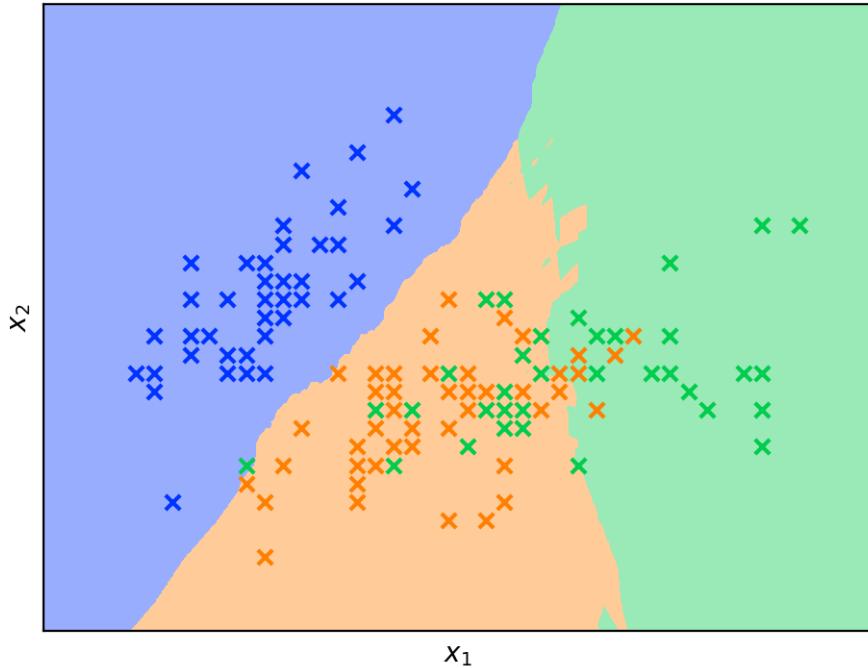
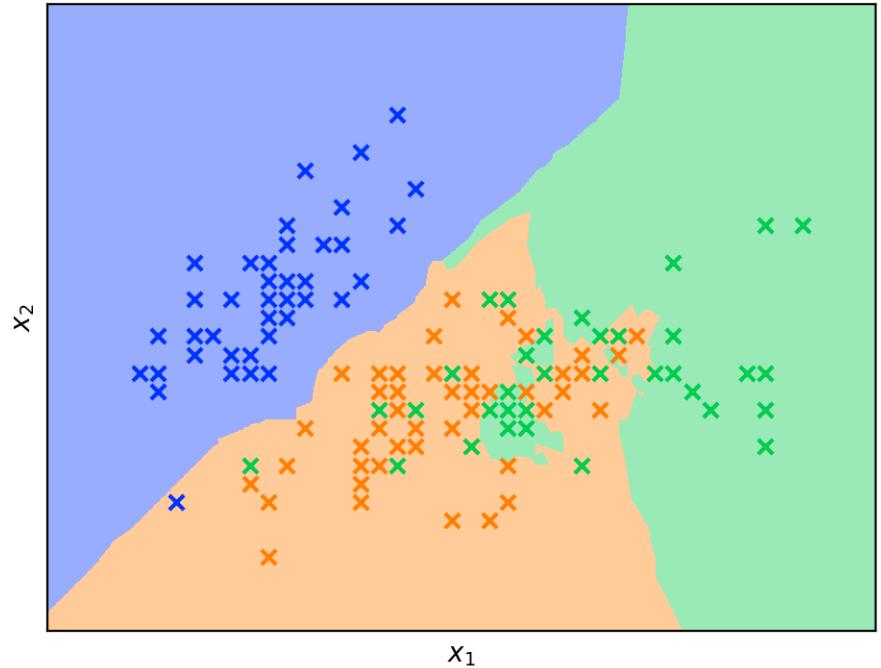
k-nearest neighbour classification ( $k=1$ )



k-nearest neighbour classification ( $k=2$ )



k-nearest neighbour classification ( $k=4$ )



# sklearn.neighbors.KNeighborsClassifier

```
class sklearn.neighbors.KNeighborsClassifier(n_neighbors=5, weights='uniform', algorithm='auto',
leaf_size=30, p=2, metric='minkowski', metric_params=None, n_jobs=None, **kwargs) [source]
```

Classifier implementing the k-nearest neighbors vote.

Read more in the User Guide.

## Parameters:

### n\_neighbors : int, optional (default = 5)

Number of neighbors to use by default for `kneighbors` queries.

### weights : str or callable, optional (default = ‘uniform’)

weight function used in prediction. Possible values:

- ‘uniform’ : uniform weights. All points in each neighborhood are weighted equally.
- ‘distance’ : weight points by the inverse of their distance. in this case, closer neighbors of a query point will have a greater influence than neighbors which are further away.
- [callable] : a user-defined function which accepts an array of distances, and returns an array of the same shape containing the weights.

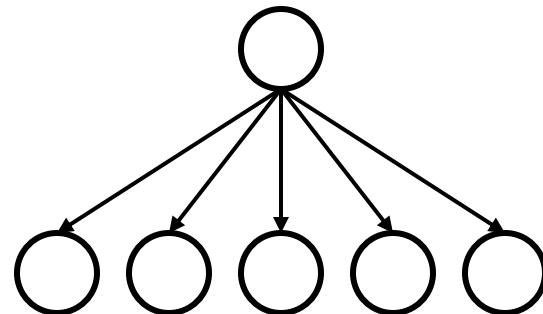
### algorithm : {‘auto’, ‘ball\_tree’, ‘kd\_tree’, ‘brute’}, optional

Algorithm used to compute the nearest neighbors:

- ‘ball\_tree’ will use `BallTree` OK
- ‘kd\_tree’ will use `KDTree` OK
- ‘brute’ will use a brute-force search. super slow
- ‘auto’ will attempt to decide the most appropriate algorithm based on the values passed to `fit` method.

*Naïve Bayes is also an important multiclass classifier...*

# Naive Bayes



*However, instead of introducing this now, we'll introduce it in the context of “generative versus discriminative models” later on in the course!*

# PRML Readings

§2.5.2 Nearest-neighbour methods

§4.1.2 Multiple classes

§4.3.4 Multiclass logistic regression

- See also equation 4.6.2 in section §4.2.0