SI 670: Applied Machine Learning

Multiclass Regression
Support Vector Machines

Grant Schoenebeck

Some slides (denoted "AM") adapted from material courtesy of Andrew W. Moore @ CMU. See http://www.cs.cmu.edu/~awm/tutorials.html

Outline

- SVMs
- SVMs with Kernals
- Data Leakage
- Imputing Missing Data
- Optimizing for Different Evaluation Metrics
- Multiclass Regression

Support Vector Machines

Linear classifiers: how would you separate these two groups of training examples with a straight line?

Feature vector

Class value

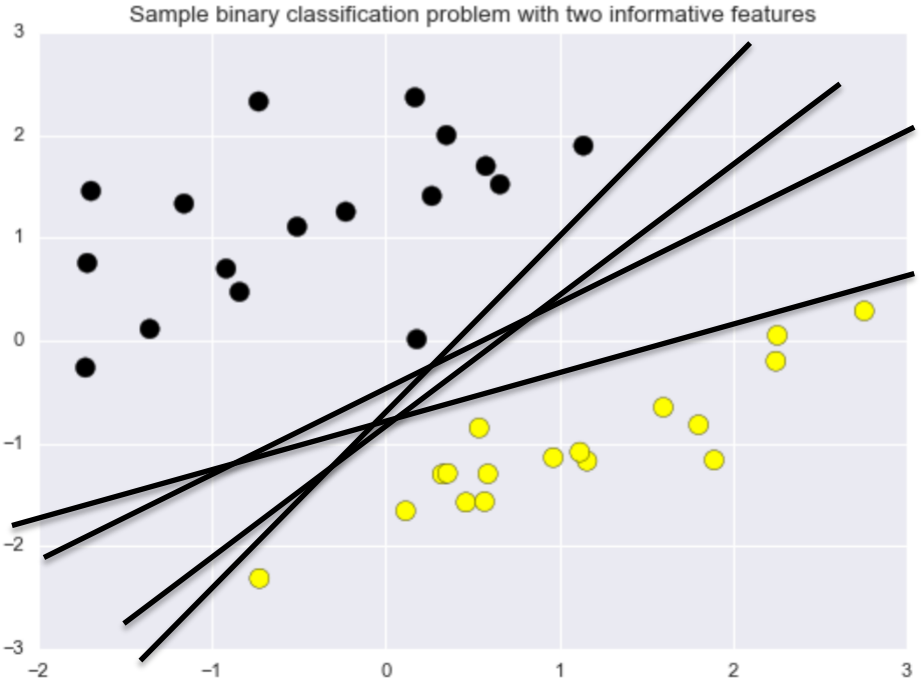
$$f(x, w, b) = sign(w \circ x + b)$$

$$= sign\left(\sum w[i]x[i] + b\right)$$

Linear Classifiers

$$f(x, w, b) = sign(w \circ x + b)$$

There are many possible linear classifiers that could separate the two classes.
Which one is best?



Maximum margin linear classifier: Linear Support Vector Machines

$$f(x, w, b) = sign(w \circ x + b)$$

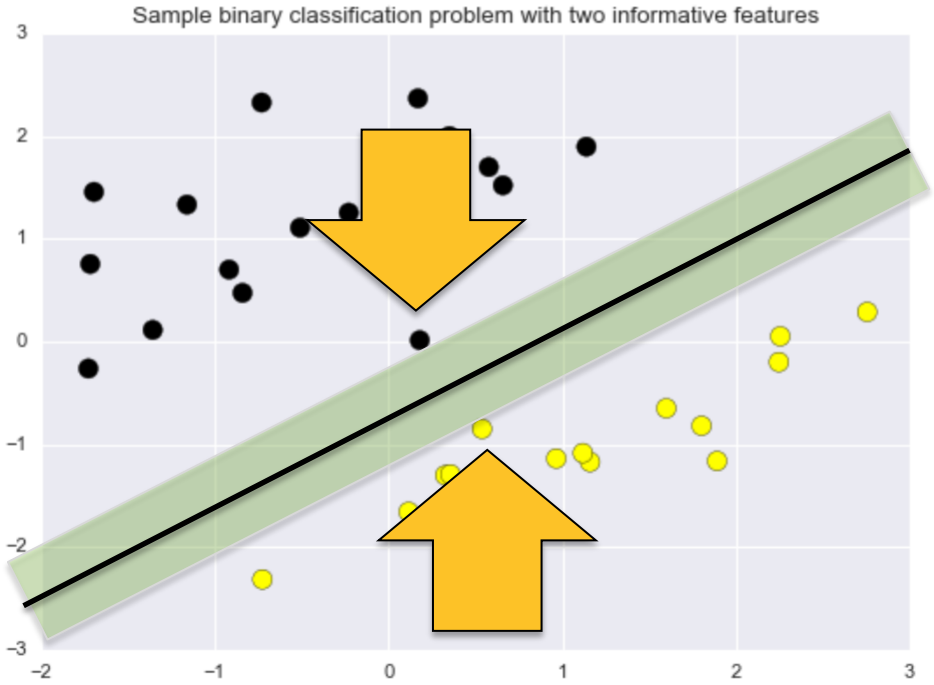
Margin: maximum decision boundary widgth before hitting a data point.

Linear Support Vector Machine (LSVM) is the maximum margin classifier.

Maximum margin linear classifier: Linear Support Vector Machines

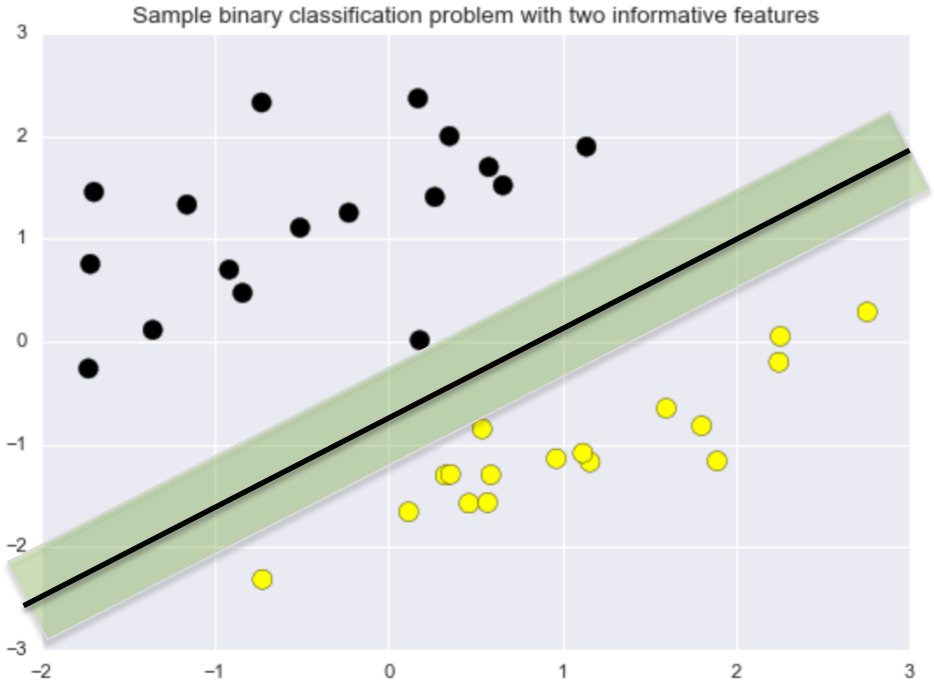
$$f(x, w, b) = sign(w \circ x + b)$$

Support Vectors: points that define maximum margin

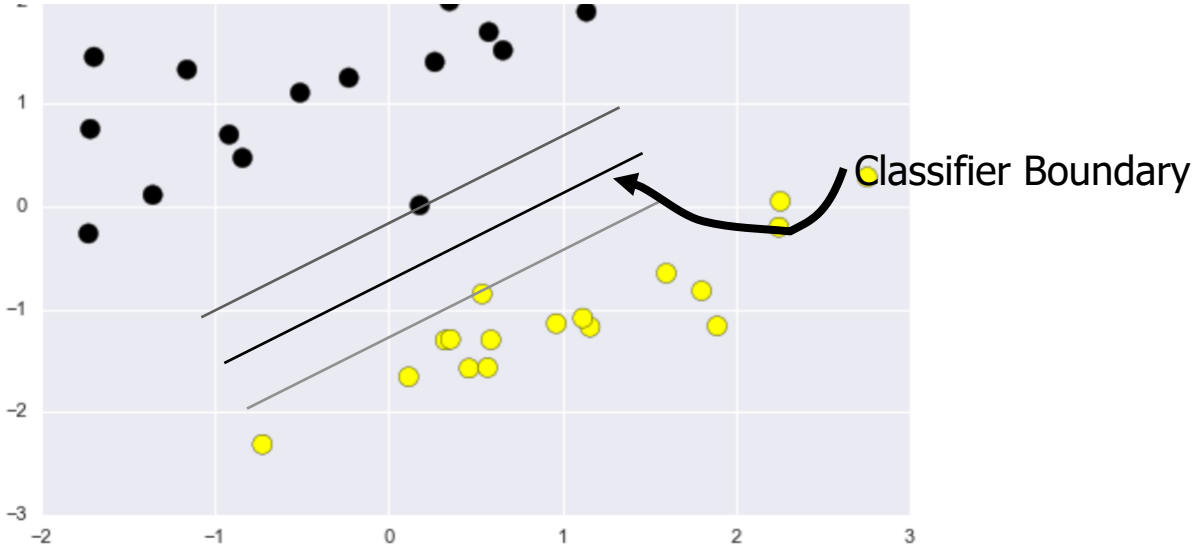


Why maximum margin?

1. Intuitively "safest"
   - If we've made a small error in the location of the boundary (it's been jolted in its perpendicular direction) this gives us least chance of causing a misclassification.
2. Empirically it works very well.
3. LOOCV is easy since the model is immune to removal of any non-support-vector datapoints.
4. Intuitively does dimension reduction

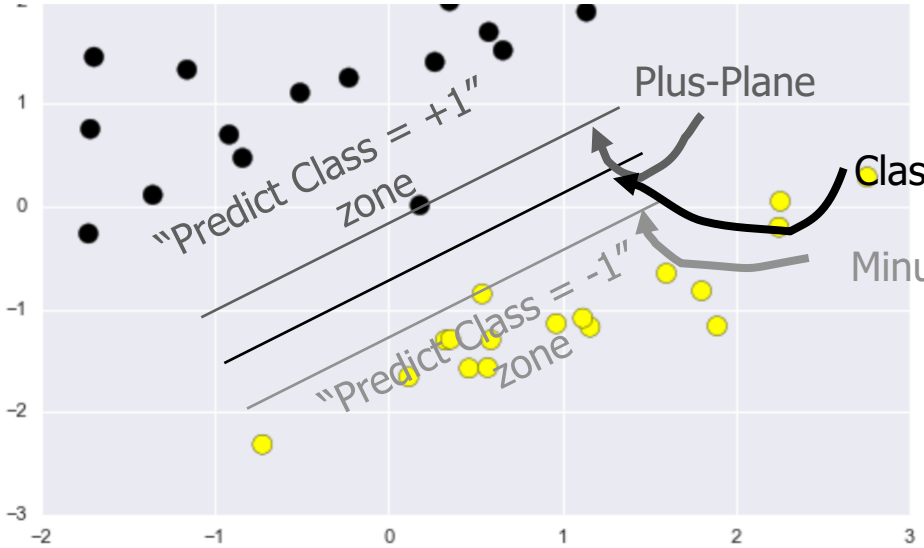


Specifying a line and margin



- How do we represent this mathematically?
- …in *m* input dimensions?

Specifying a line and margin

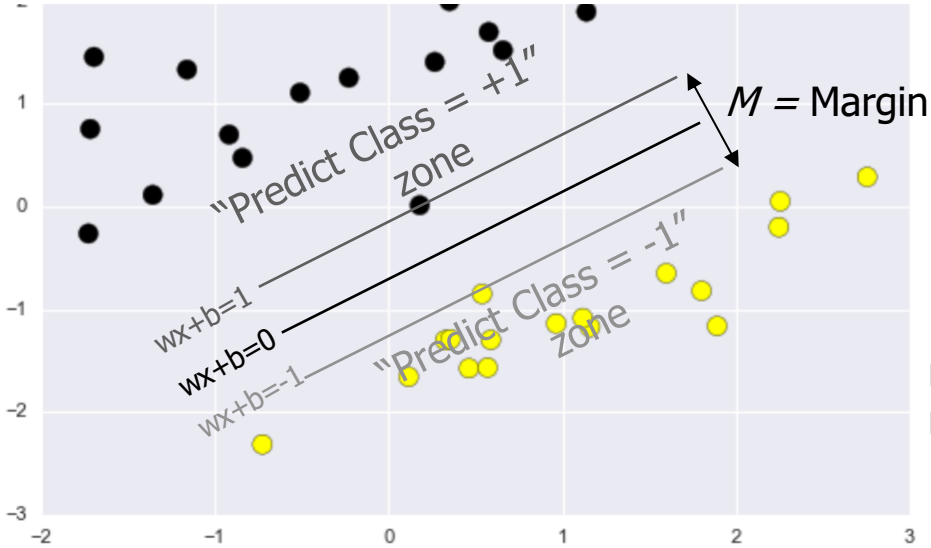


Plus-plane = $\{ x : w \cdot x + b = +1 \}$

Minus-plane = $\{ x : w \cdot x + b = -1 \}$

Classify as..	+1	if	$w \cdot x + b >= 1$
	-1	if	$w \cdot x + b <= -1$
	Whatever	if	$-1 < w \cdot x + b < 1$

Our goal is to find the line with maximum margin width M, so we must be able to compute M

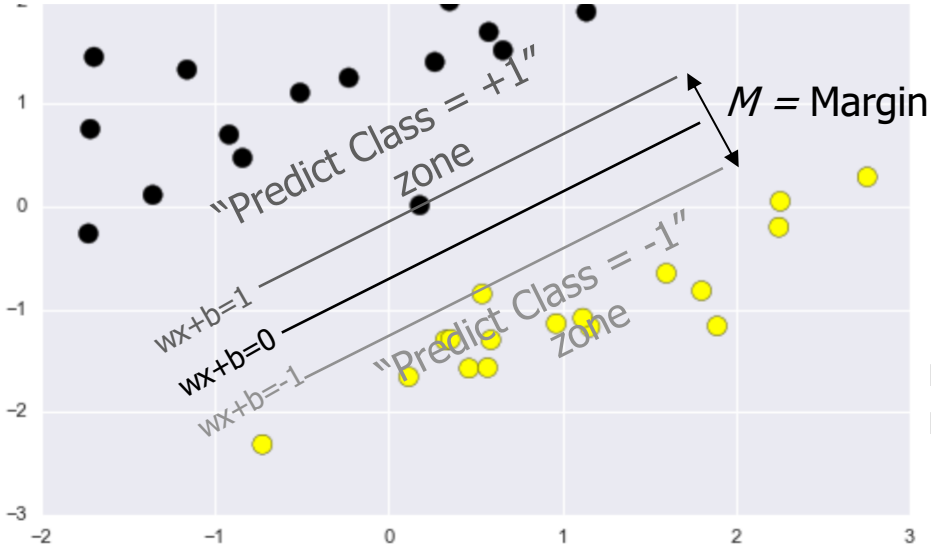


Plus-plane = $\{ x : w \cdot x + b = +1 \}$

Minus-plane = $\{ x : w \cdot x + b = -1 \}$

How do we compute M in terms of $\boldsymbol{w}$ and b?

Our goal is to find the line with maximum margin width M, so we must be able to compute M



Plus-plane = $\{ x : w \cdot x + b = +1 \}$

Minus-plane = $\{ x : w \cdot x + b = -1 \}$

After some linear algebra using our constraints

$$M = \text{Margin Width} = \frac{2}{\sqrt{w \cdot w}}$$

Optimization

Find w and b efficiently using Quadratic Programming

- Maximizing a quadratic function of some real-valued variables subject to linear constraints.

Objective:

- Width of the margin $\frac{2}{\sqrt{w \cdot w}}$

Constraints:

- Whether all data points are in the correct half-planes

Quadratic criterion

Find $\arg\max_{\mathbf{u}} \quad c + \mathbf{d}^T\mathbf{u} + \frac{\mathbf{u}^T R\mathbf{u}}{2}$

Subject to

$$a_{11}u_1 + a_{12}u_2 + \ldots + a_{1m}u_m \le b_1$$
$$a_{21}u_1 + a_{22}u_2 + \ldots + a_{2m}u_m \le b_2$$
$$\vdots$$
$$a_{n1}u_1 + a_{n2}u_2 + \ldots + a_{nm}u_m \le b_n$$

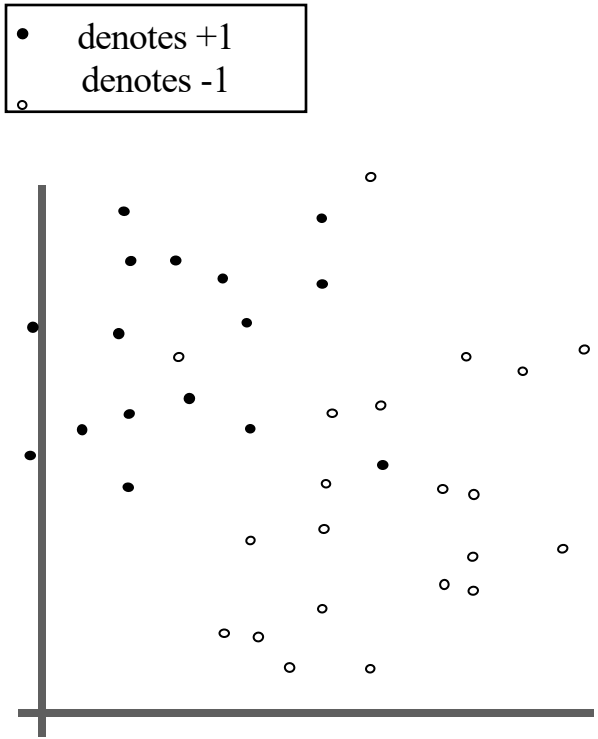
n additional linear inequality constraints

And subject to

$$a_{(n+1)1}u_1 + a_{(n+1)2}u_2 + \ldots + a_{(n+1)m}u_m = b_{(n+1)}$$
$$a_{(n+2)1}u_1 + a_{(n+2)2}u_2 + \ldots + a_{(n+2)m}u_m = b_{(n+2)}$$
$$\vdots$$
$$a_{(n+e)1}u_1 + a_{(n+e)2}u_2 + \ldots + a_{(n+e)m}u_m = b_{(n+e)}$$

additional linear equality constraints

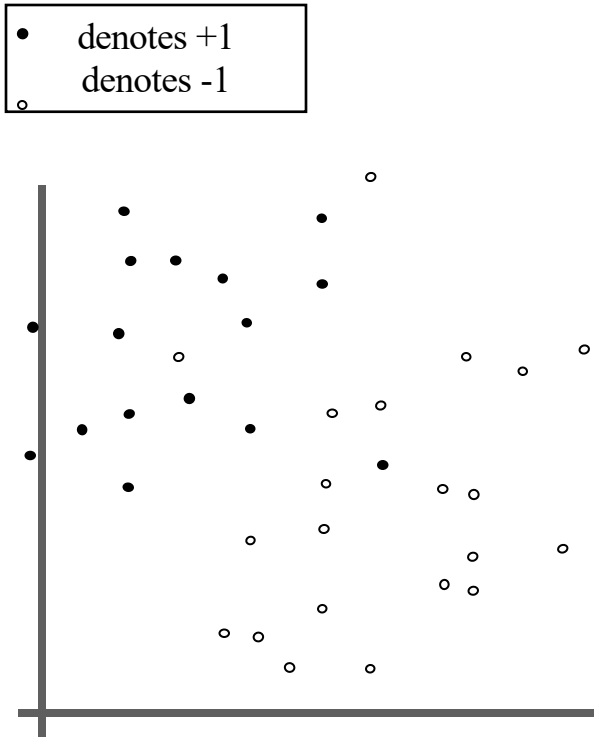
What quadratic objective should we really minimize?



Minimize

$w \cdot w$?

What quadratic objective should we really minimize?



Minimize

$\boldsymbol{w}\cdot\boldsymbol{w}$ + C *(distance of error points to their correct place)*

Regularization for SVMs: the C parameter

- C: strength of regularization
- Larger C: less regularization
  - *Prioritize pointwise classification correctness over large margin*
- Smaller C: more regularization
  - *Prioritize large margin over pointwise classification correctness*

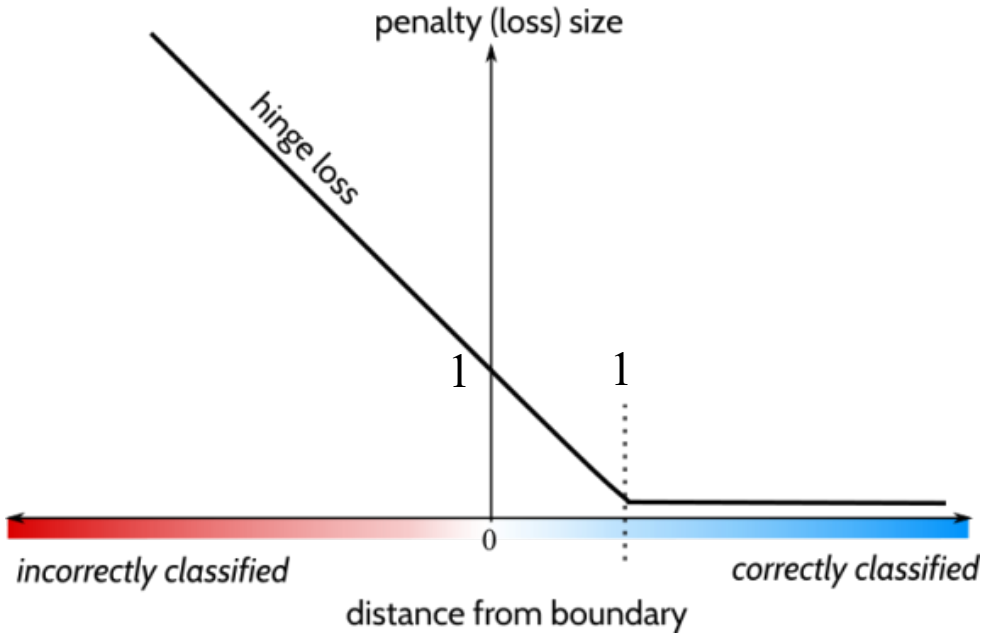
Actually…

- SVMs can be seen as Empirical Risk Minimization
- Use Hinge Loss with L2 regularization

$$\left[\frac{1}{n}\sum_{i=1}^{n}\max\left(0, 1 - y_i(w \cdot x_i - b)\right)\right] + \lambda\|w\|^2$$

- Here $\lambda = \frac{1}{c}$

Hinge Loss


penalty (loss) size
hinge loss
1
1
0
incorrectly classified
correctly classified
distance from boundary

Review

- **L2 regularization with**
  - *Square-loss*

$$\ell_{sq}(y, z) = (y - z)^2$$

  - *Log loss*

$$\ell_{\log}(y, z) = \ln(1 + e^{-yz})$$

  - *Hinge loss*

$$\ell(y) = \max(0, 1 - t \cdot y)$$

Ridge Regression, Logistic Regression, SVM

Think, Pair, Share

The SVM's boundary will change if you:

A) Add many items that are clearly correctly classified

B) Add many items that are clearly misclassified

C) Both of the Above

D) None of the Above

Linear SVMs

- **Features**
  - *Must relate linearly*
  - *Must normalize*
- **Parameters**
  - *Regularization*
- **Evaluation**
  - *Usual*
  - *Coefficients are interpretable*

Linear Models: Pros and Cons

Pros:

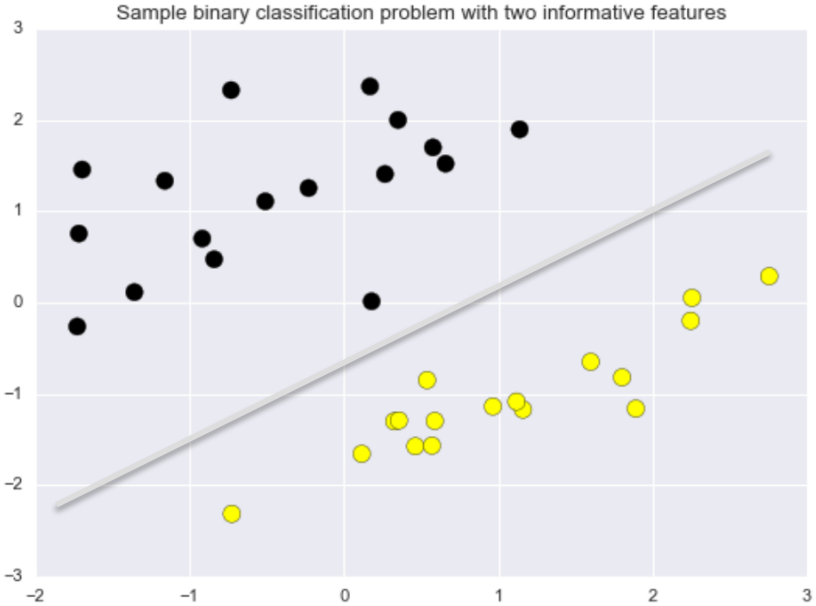
- Simple and easy to train.
- Fast prediction.
- Scales well to very large datasets.
- Works well with sparse data.
- Reasons for prediction are relatively easy to interpret.

Cons:

- For lower-dimensional data, other models may have superior generalization performance.
- For classification, data may not be linearly separable (more on this in SVMs with non-linear kernels)

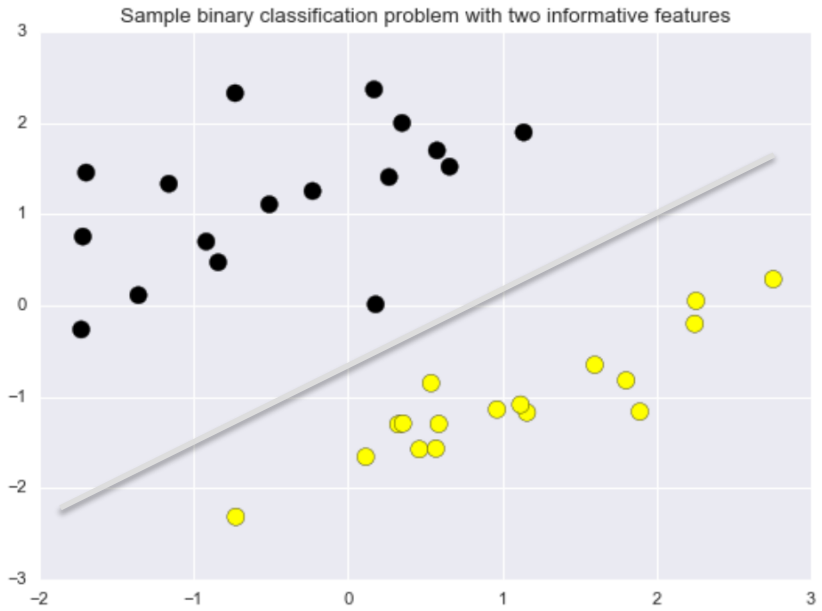
Kernelized Support Vector Machines

We saw how linear support vector classifiers could effectively find a decision boundary with maximum margin

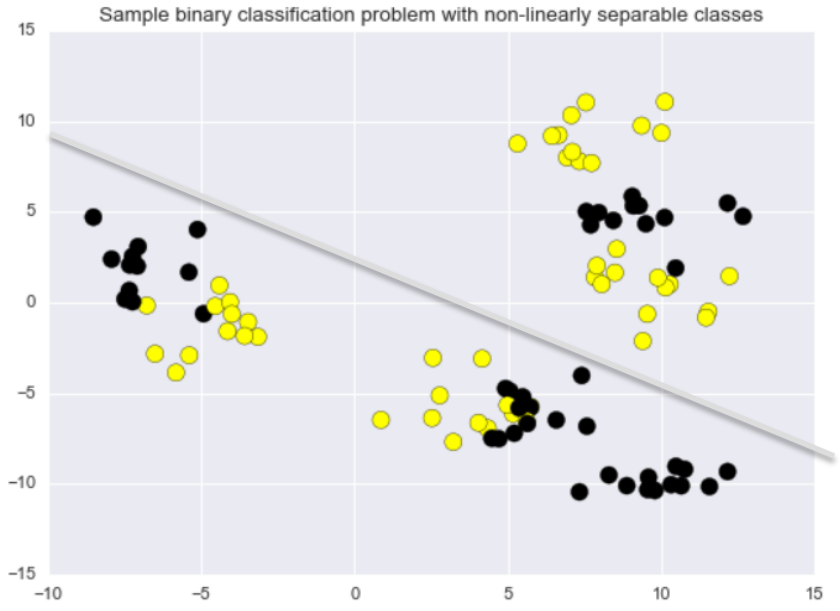


Easy for a linear classifier

But what about more complex binary classification problems?

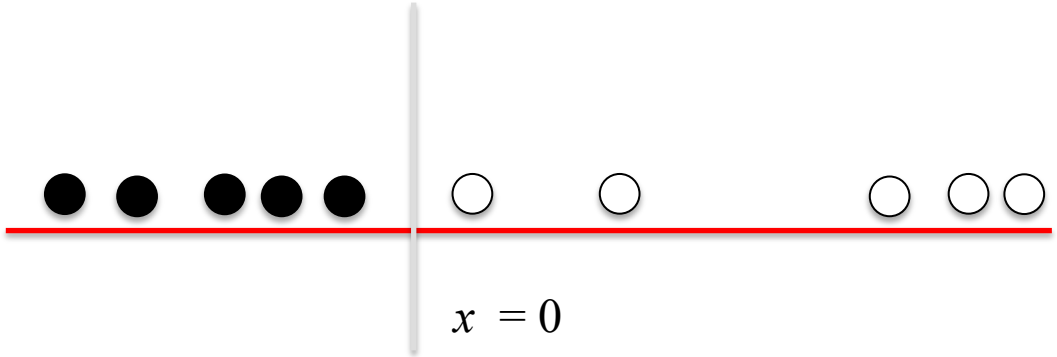


Easy for a linear classifier



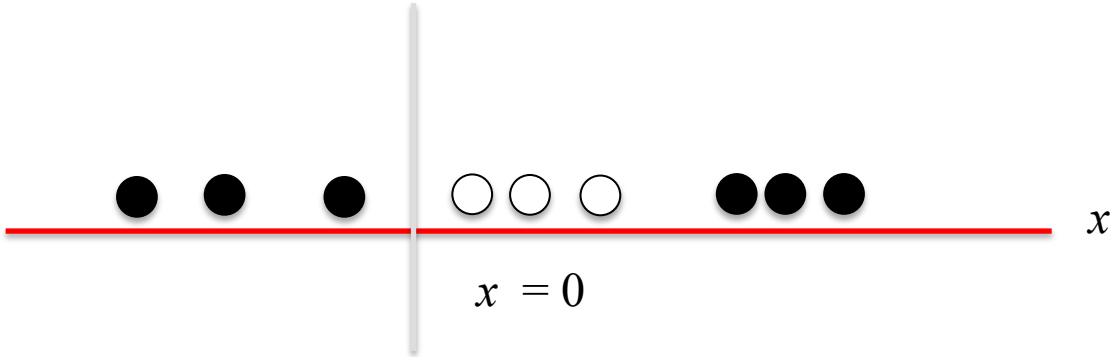
Difficult/impossible for a linear classifier

A simple 1-dimensional classification problem for a linear classifier

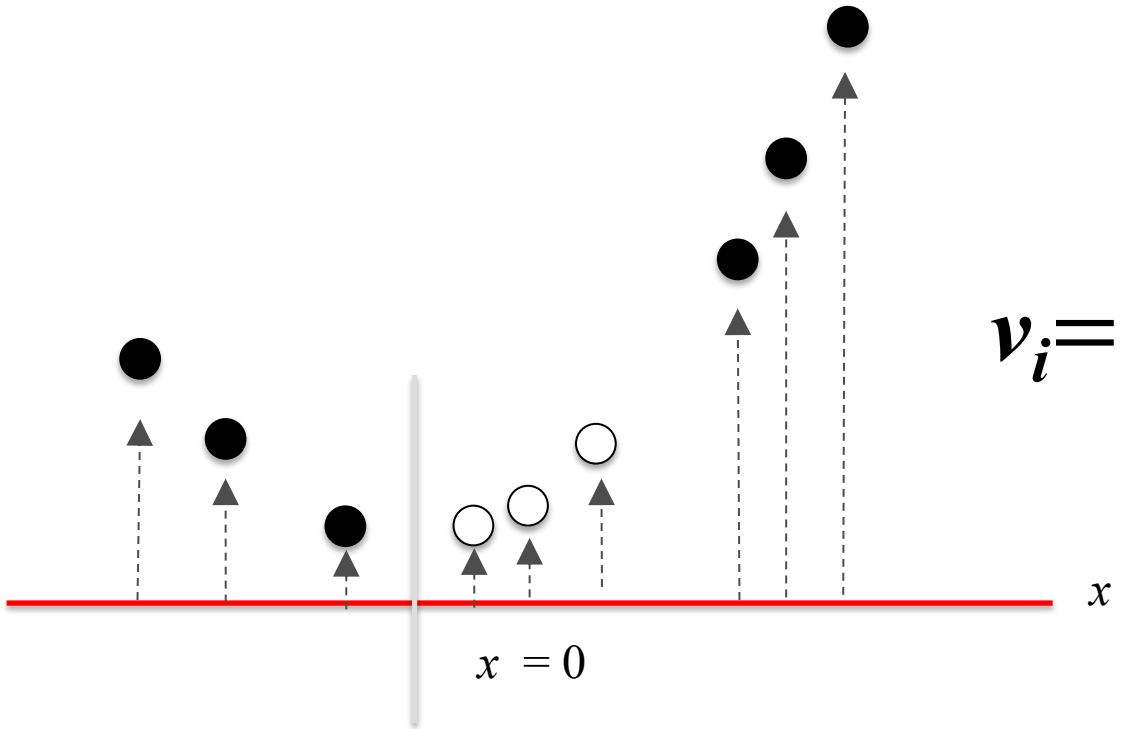


A more perplexing 1-d classification problem for a linear classifier

A more perplexing 1-d classification problem for a linear classifier

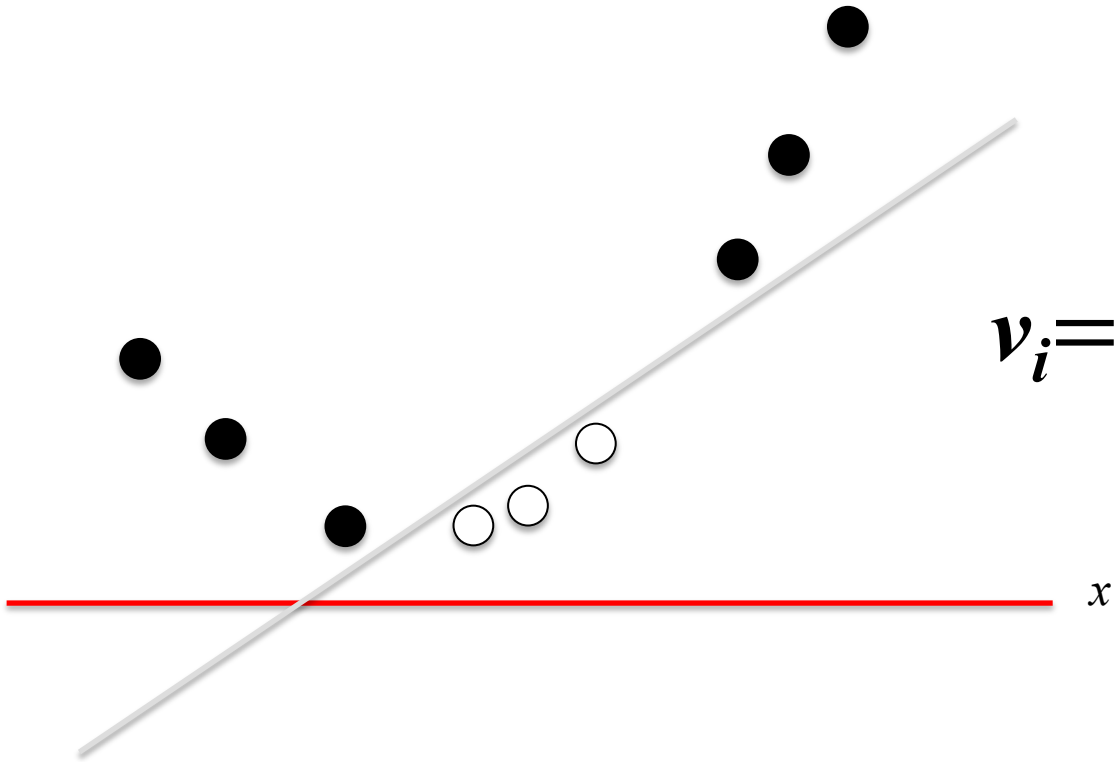


Let's transform the data by adding a second dimension/feature (set to the squared value of the first feature)



$$v_i = (x_i, x_i^2)$$

The data transformation makes it possible to solve this with a linear classifier

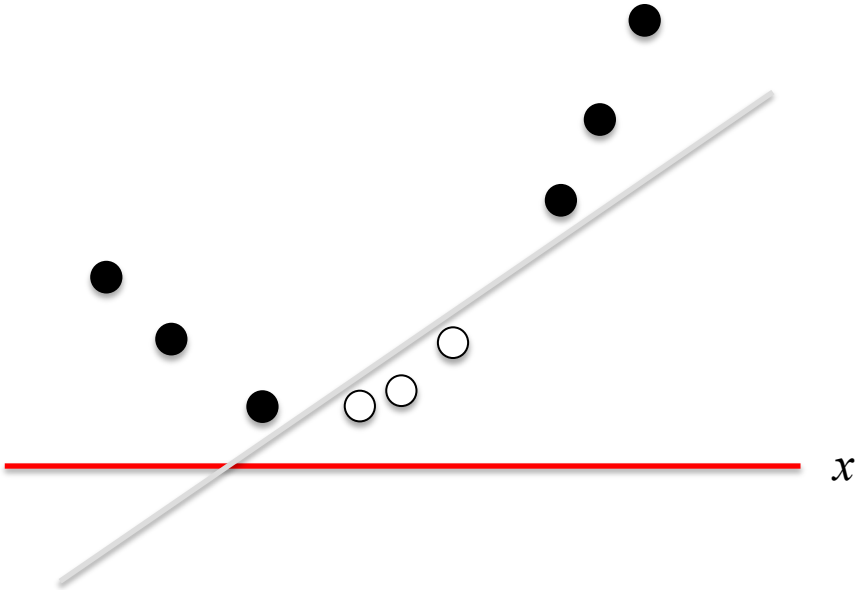


$$v_i = (x_i, x_i^2)$$

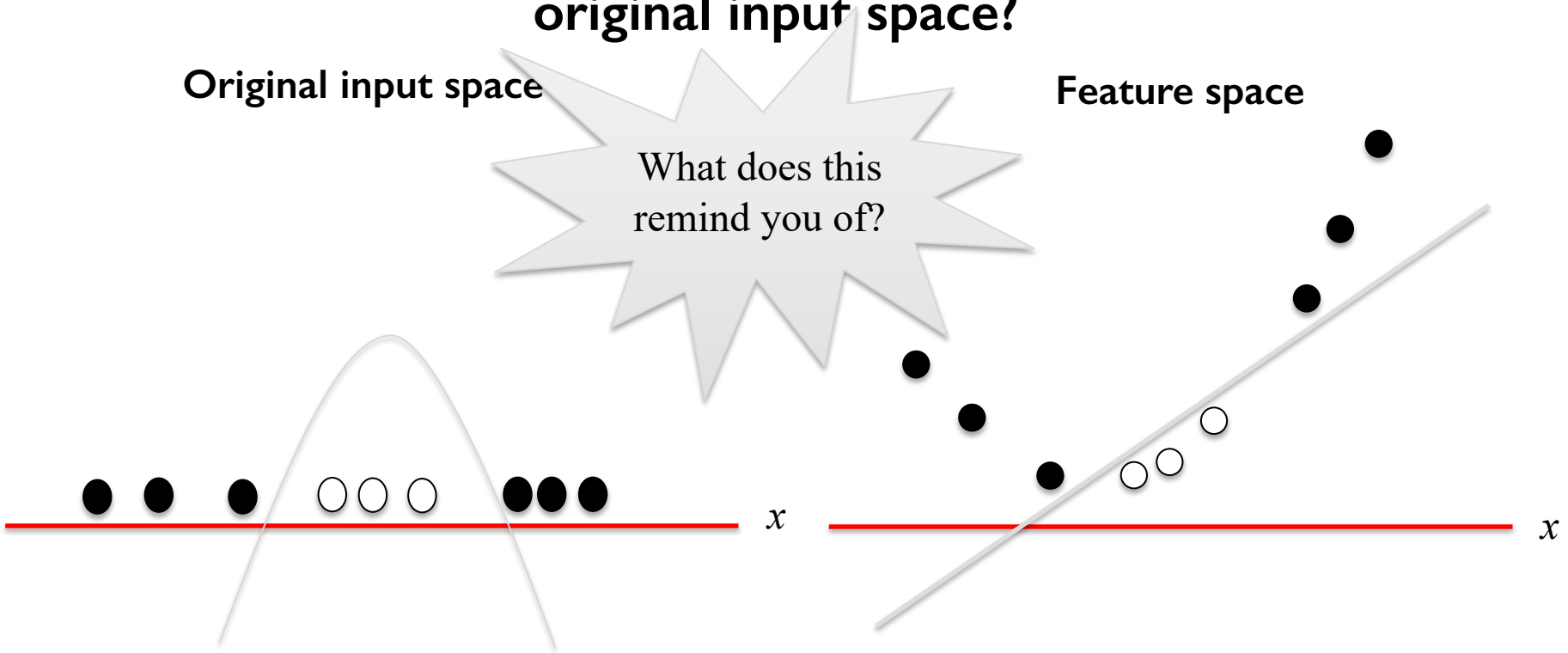
What does the linear decision boundary in feature space correspond to in the original input space?

Original input space

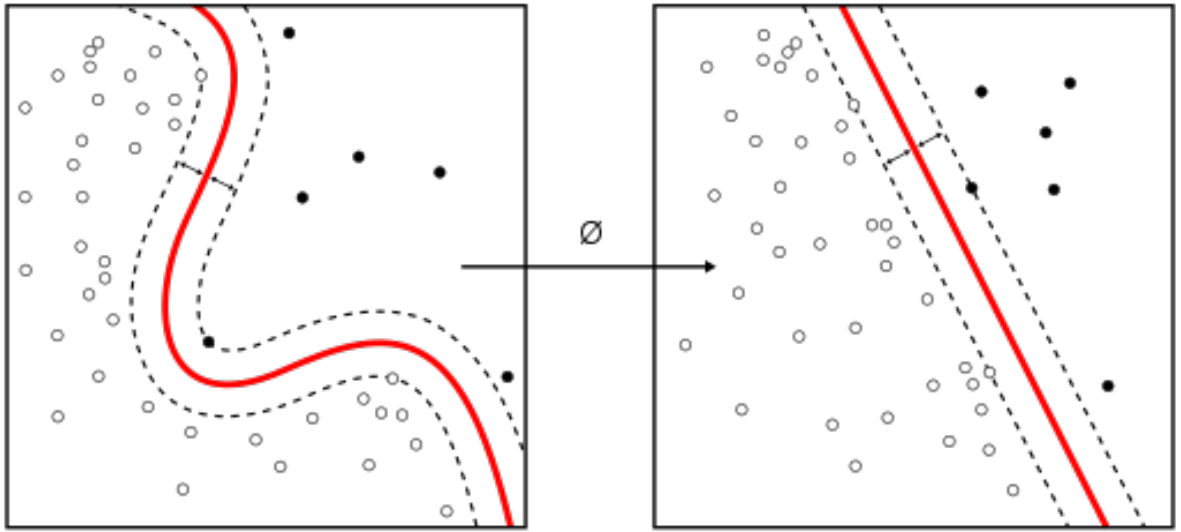
Feature space



What does the linear decision boundary correspond to in the original input space?



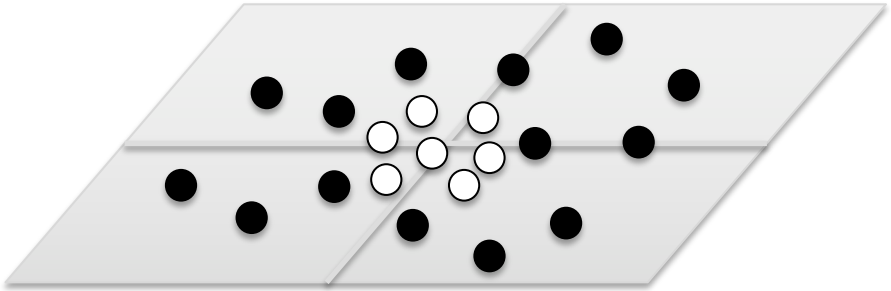
Transforming the data can make it much easier for a linear classifier.



Source: Wikipedia "Kernel Machine" article.
https://commons.wikimedia.org/w/index.php?curid=47868867

Example of mapping a 2D classification problem to a 3D feature space to make it linearly separable

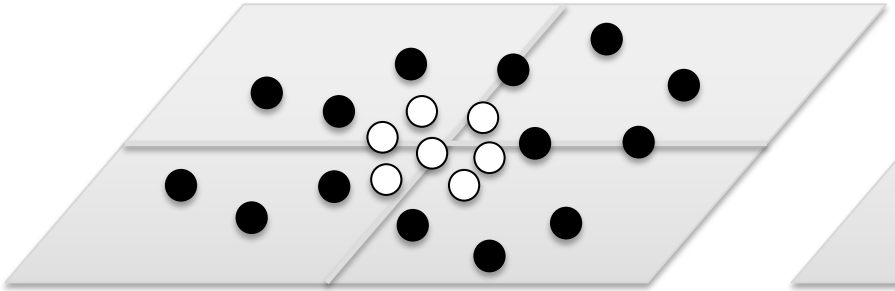
Original input space



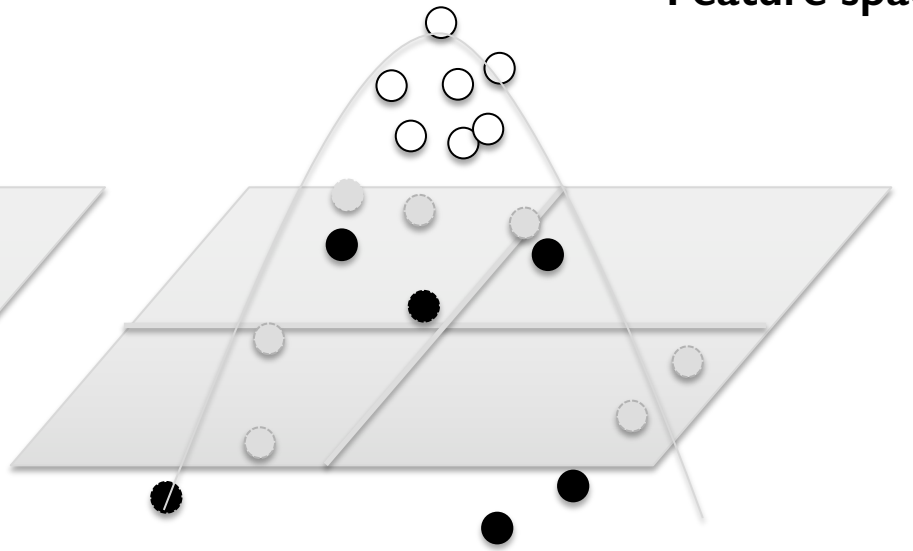
$$x_i = (x_0, x_1)$$

Example of mapping a 2D classification problem to a 3D feature space to make it linearly separable

Original input space



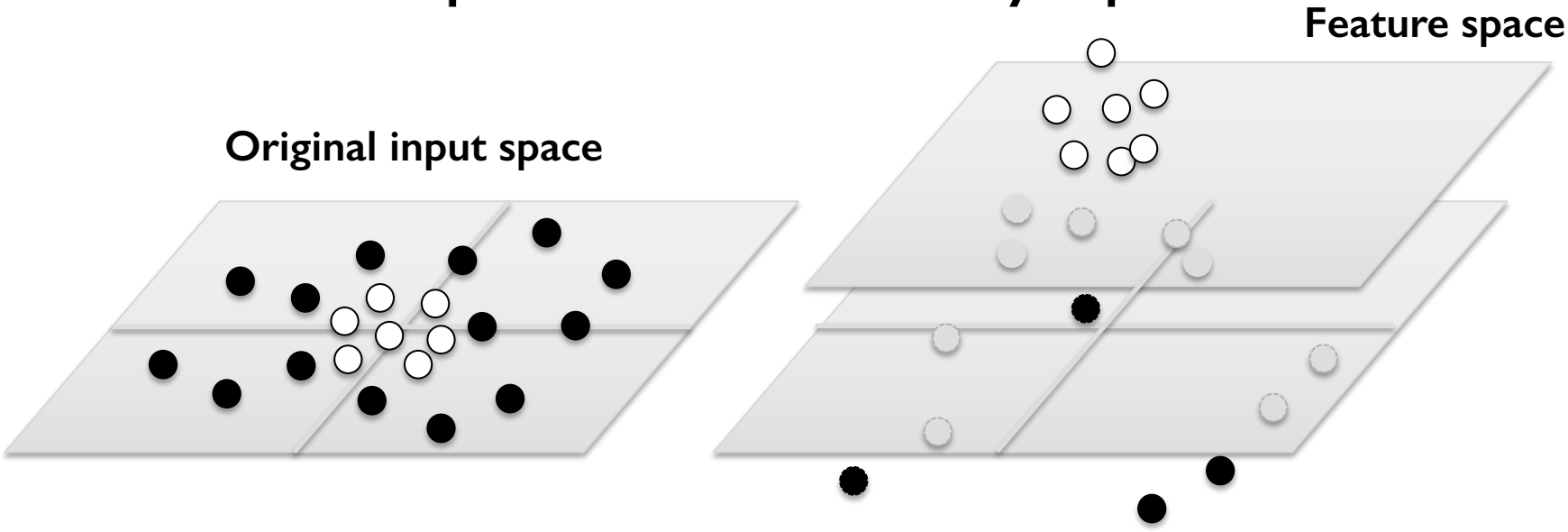
Feature space



$$x_i = (x_0, x_1)$$

$$v_i = (x_0, x_1, 1 - (x_0^2 + x_1^2))$$

Example of mapping a 2D classification problem to a 3D feature space to make it linearly separable

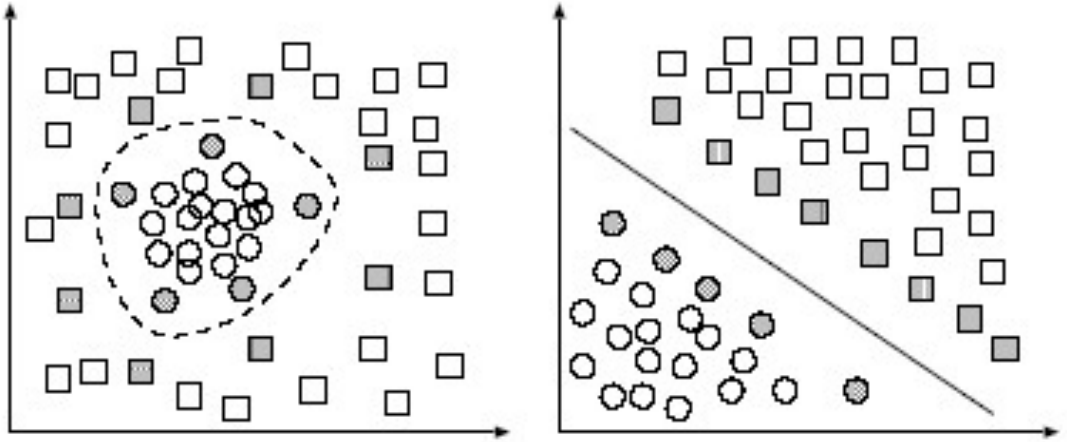


$$x_i = (x_0, x_1)$$

$$v_i = (x_0, x_1, 1 - (x_0^2 + x_1^2))$$

Radial Basis Function Kernel

$$K(\boldsymbol{x}, \boldsymbol{x}') = \exp\ [-\gamma \cdot \|\boldsymbol{x} - \boldsymbol{x}'\|^2]$$



Original input space

Feature space

A kernel is a similarity measure (modified dot product) between data points

Common SVM basis functions

$\boldsymbol{z}_k$ = (polynomial terms of $\boldsymbol{x}_k$ of degree 1 to q)

$\boldsymbol{z}_k$ = (radial basis functions of $\boldsymbol{x}_k$)

$$\mathbf{z}_k[j] = \varphi_j(\mathbf{x}_k) = \text{KernelFn}\left(\frac{|\mathbf{x}_k - \mathbf{c}_j|}{\text{KW}}\right)$$

$\boldsymbol{z}_k$ = (sigmoid functions of $\boldsymbol{x}_k$)

SVM Kernel Functions

- $K(a,b)=(a \cdot b + 1)^d$ is an example of an SVM Kernel Function
- Beyond polynomials there are other very high dimensional basis functions that can be made practical by finding the right Kernel Function
  - *Radial-Basis-style Kernel Function:*

$$K(\mathbf{a},\mathbf{b}) = \exp\left(-\frac{(\mathbf{a}-\mathbf{b})^2}{2\sigma^2}\right)$$

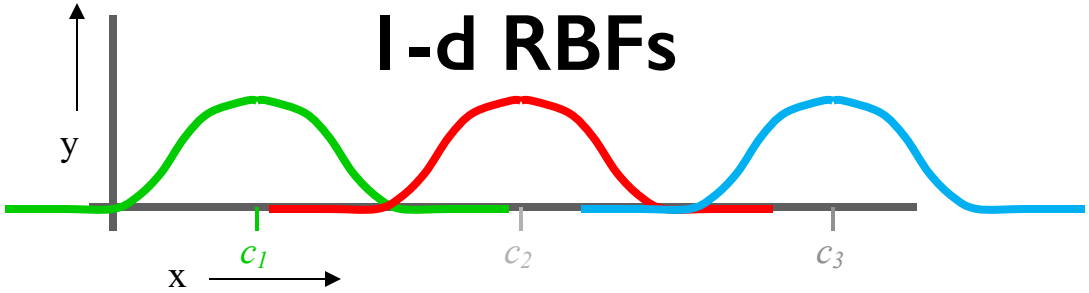
  - *Neural-net-style Kernel Function:*

$$K(\mathbf{a},\mathbf{b}) = \tanh(\kappa \mathbf{a}.\mathbf{b} - \delta)$$

σ, κ and δ are magic parameters that must be chosen by a model selection method such as CV

Radial basis function

- A real-valued function $\phi_c(x)$ whose value depends on the distance of x from some center point *c*.



$y^{est} = \beta_1 \phi_1(x) + \beta_2 \phi_2(x) + \beta_3 \phi_3(x)$

where

$\phi_i(x) = KernelFunction(| x - c_i | / KW)$

Informally, the representation of point x is transformed to a set of distances from center points

Radial Basis Functions (RBFs)

X_1	X_2	Y
3	2	7
1	1	3
:	:	:

$\mathbf{X}=$

3	2
1	1
:	:

$\mathbf{y}=$

7
3
:

$\mathbf{Z}=$

…	…	…	…	…	…
…	…	…	…	…	…
…	…	…	…	…	…

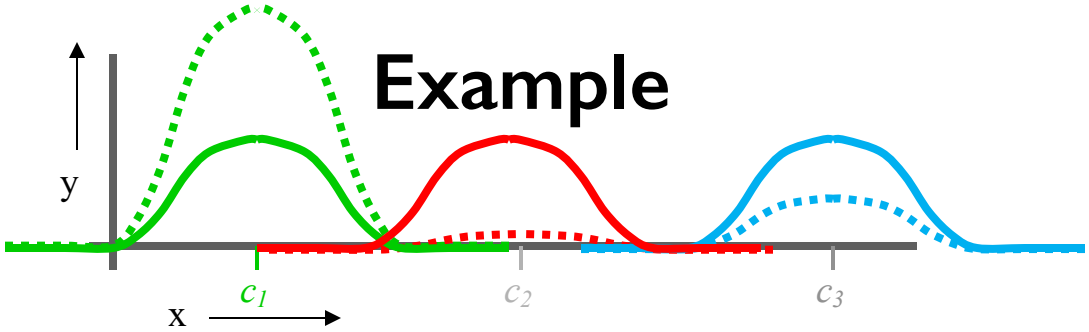
$\mathbf{y}=$

7
3
:

z=(list of radial basis function evaluations)

$$\boldsymbol{\beta} = (\boldsymbol{Z}^T\boldsymbol{Z})^{-1}(\boldsymbol{Z}^T\boldsymbol{y})$$

$$y^{est} = \beta_0 + \beta_1 x_1 + \ldots$$

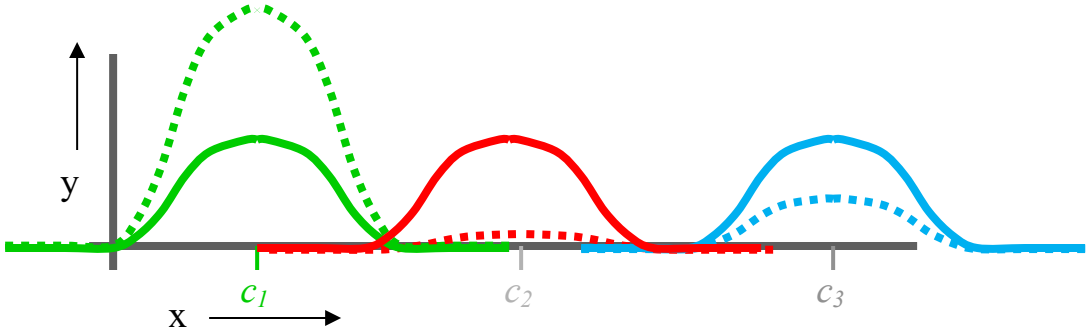


$y^{est} = 2\phi_1(x) + 0.05\,\phi_2(x) + 0.5\,\phi_3(x)$

where

$\phi_i(x) = KernelFunction(|x - c_i| / KW)$

RBFs with Linear Regression

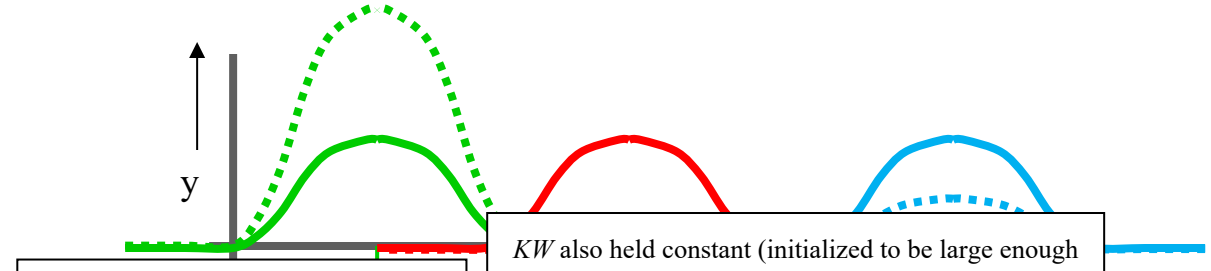


$y^{est} = 2\phi_1(x) + 0.05\,\phi_2(x) + 0.5\,\phi_3(x)$

where

$\phi_i(x) = KernelFunction(|x - c_i| / KW)$

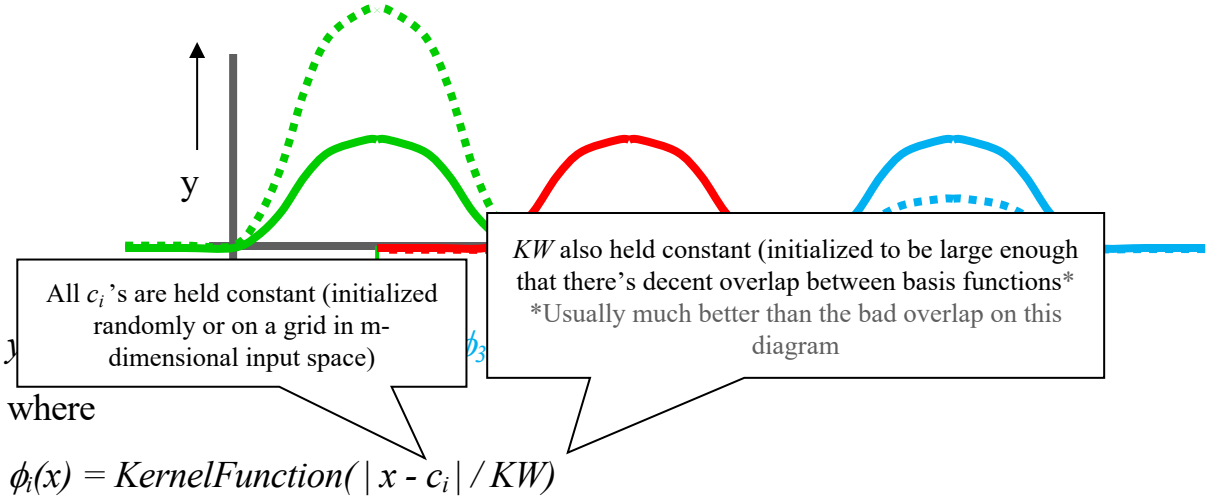
RBFs with Linear Regression



where

$\phi_i(x) = KernelFunction(| x - c_i | / KW)$

RBFs with Linear Regression



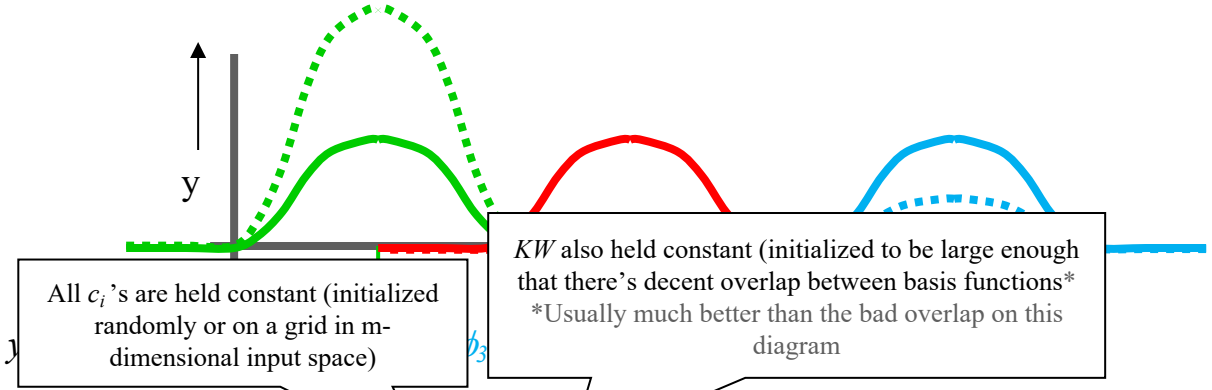
Then given Q basis functions, define the matrix Z such that

$$Z_{kj} = KernelFunction(|x_k - c_i| / KW)$$

where x_k is the k-th vector of inputs

And as before, $\boldsymbol{\beta} = (\boldsymbol{Z}^T\boldsymbol{Z})^{-1}(\boldsymbol{Z}^T\boldsymbol{y})$

RBFs with Linear Regression

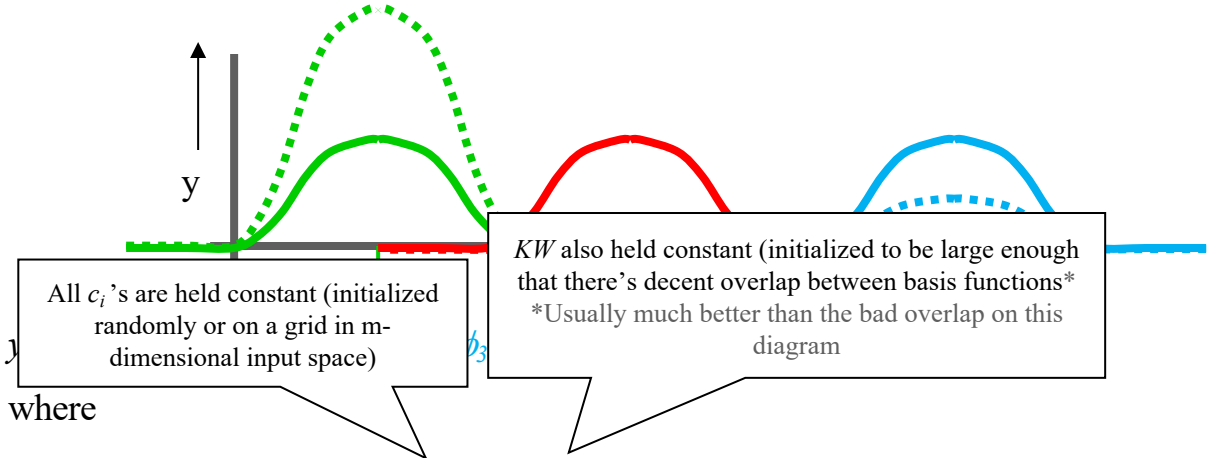


where

$\phi_i(x) = KernelFunction(| x - c_i | / KW)$

But how do we now find all the β_j's, c_i 's and *KW* ?

RBFs with Linear Regression



where

$\phi_i(x) = KernelFunction(| x - c_i | / KW)$

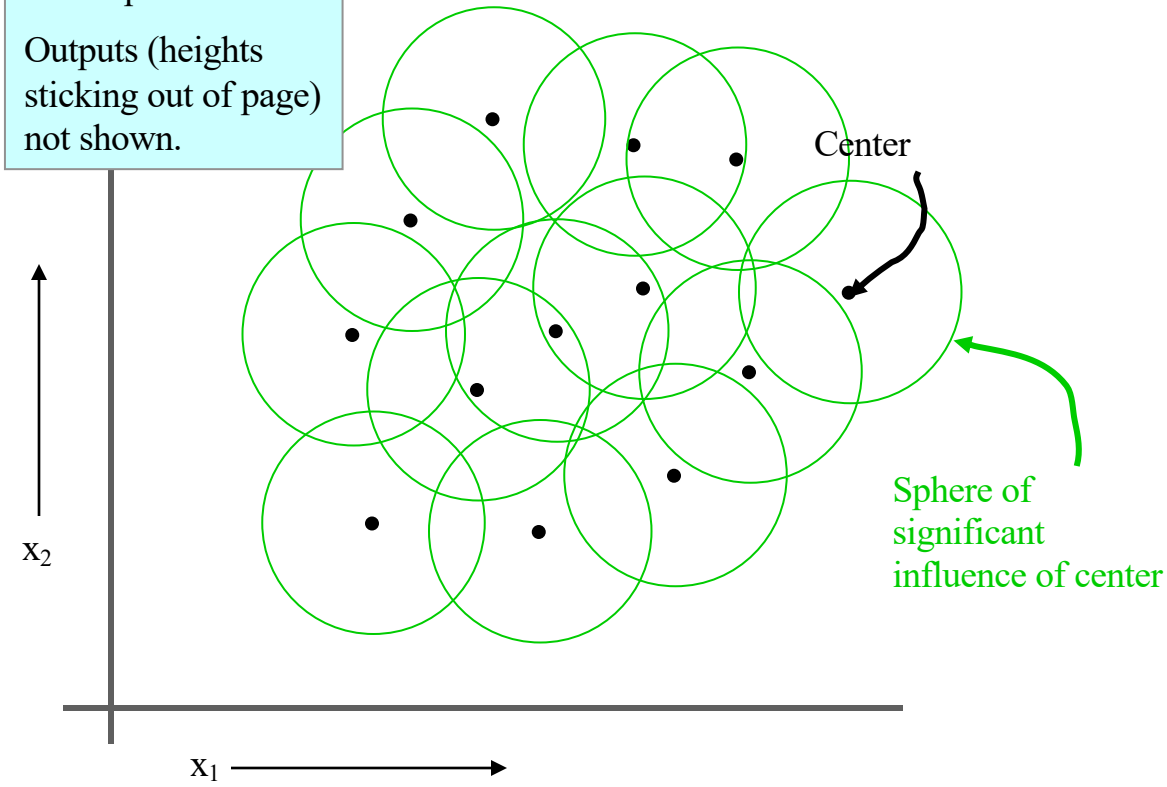
But how do we now find all the β_j's, c_i 's and *KW* ?

Answer: Gradient descent

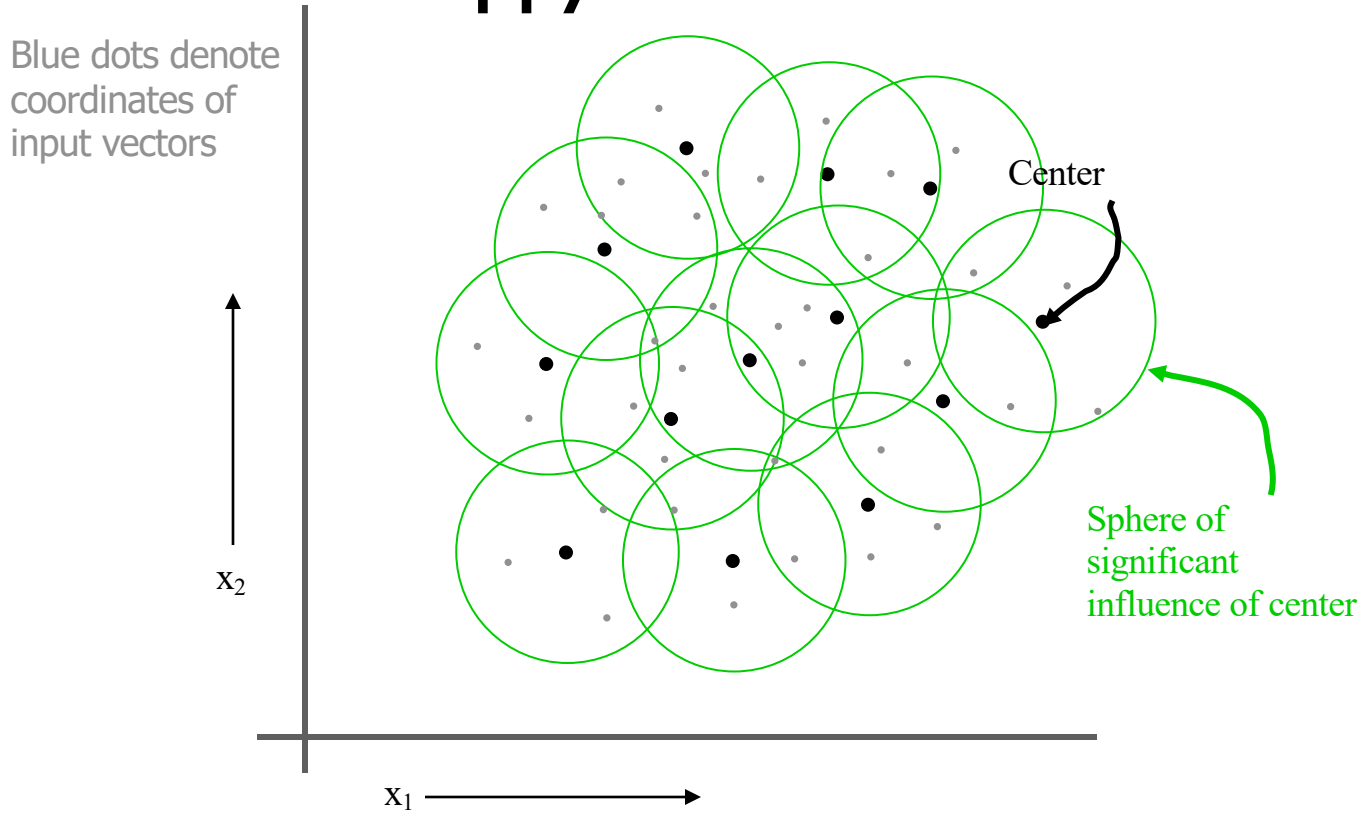
Radial Basis Functions in 2-d

Two inputs.

Outputs (heights sticking out of page) not shown.

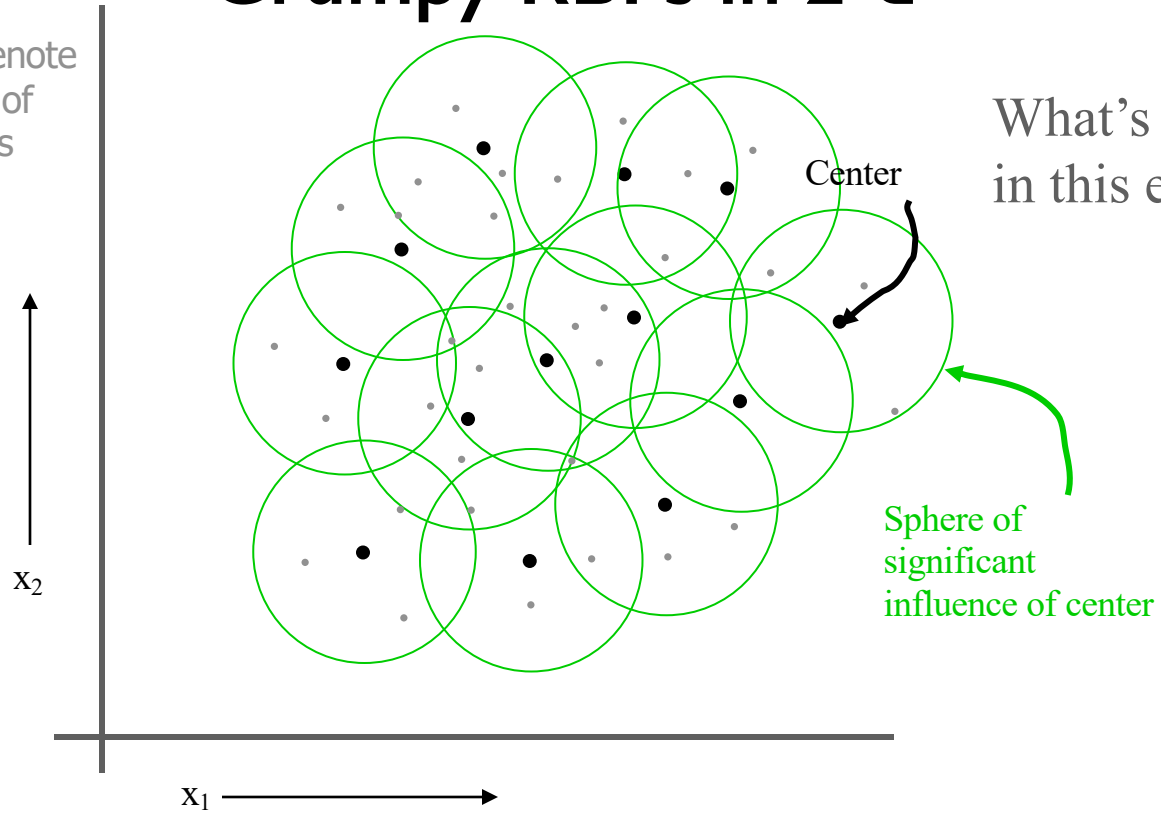


Happy RBFs in 2-d



Grumpy RBFs in 2-d

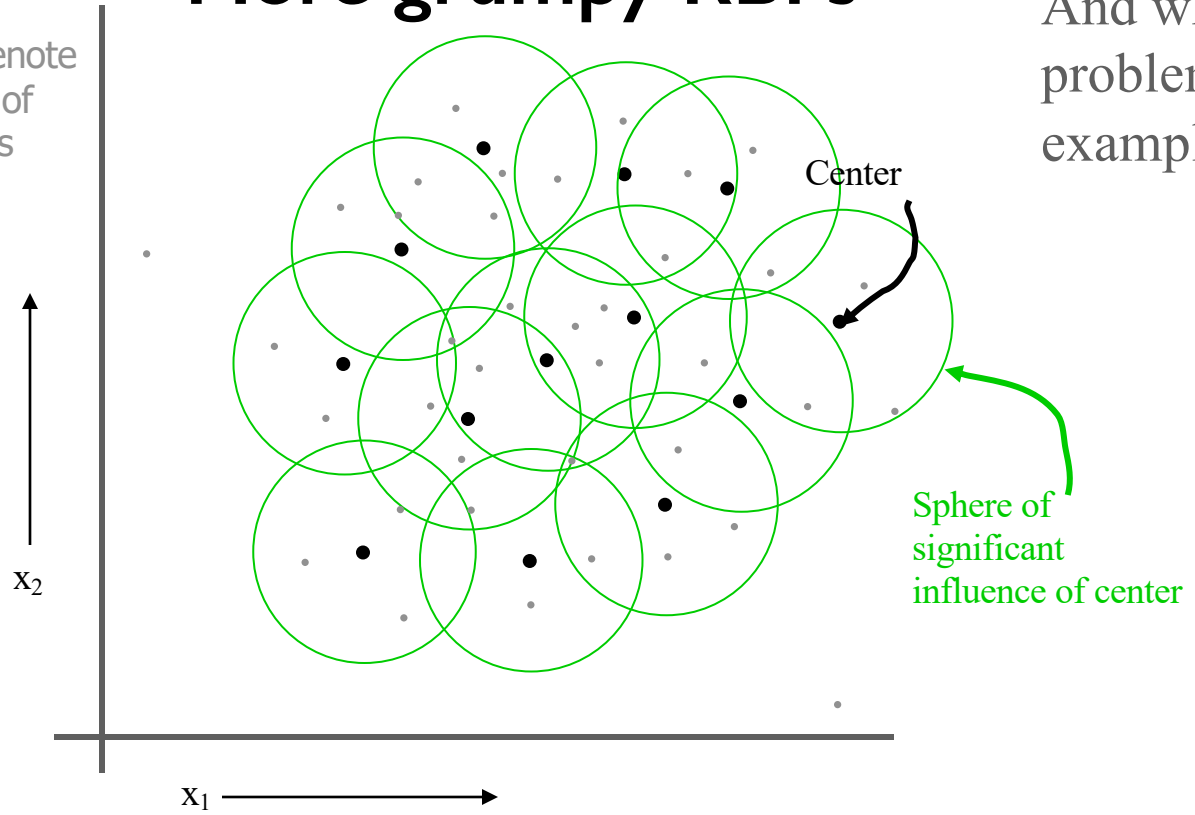
Blue dots denote coordinates of input vectors



What's the problem in this example?

More grumpy RBFs

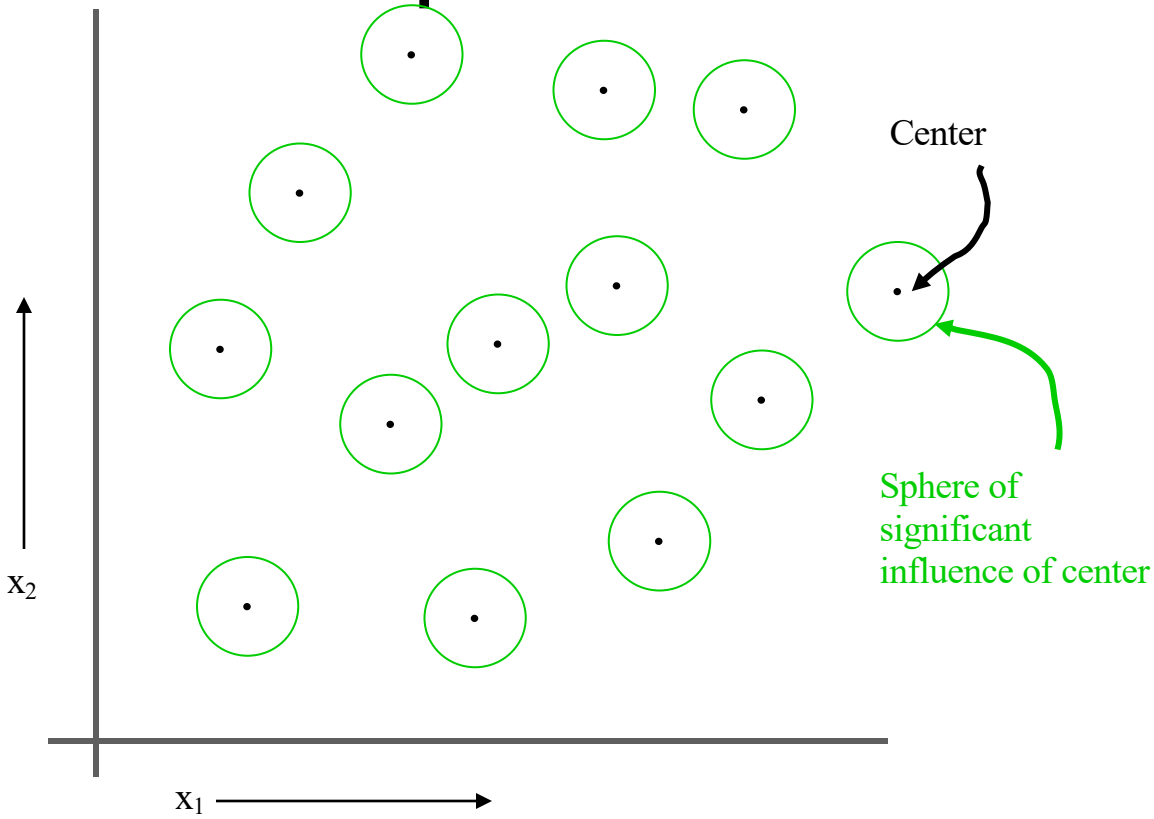
Blue dots denote coordinates of input vectors



And what's the problem in this example?

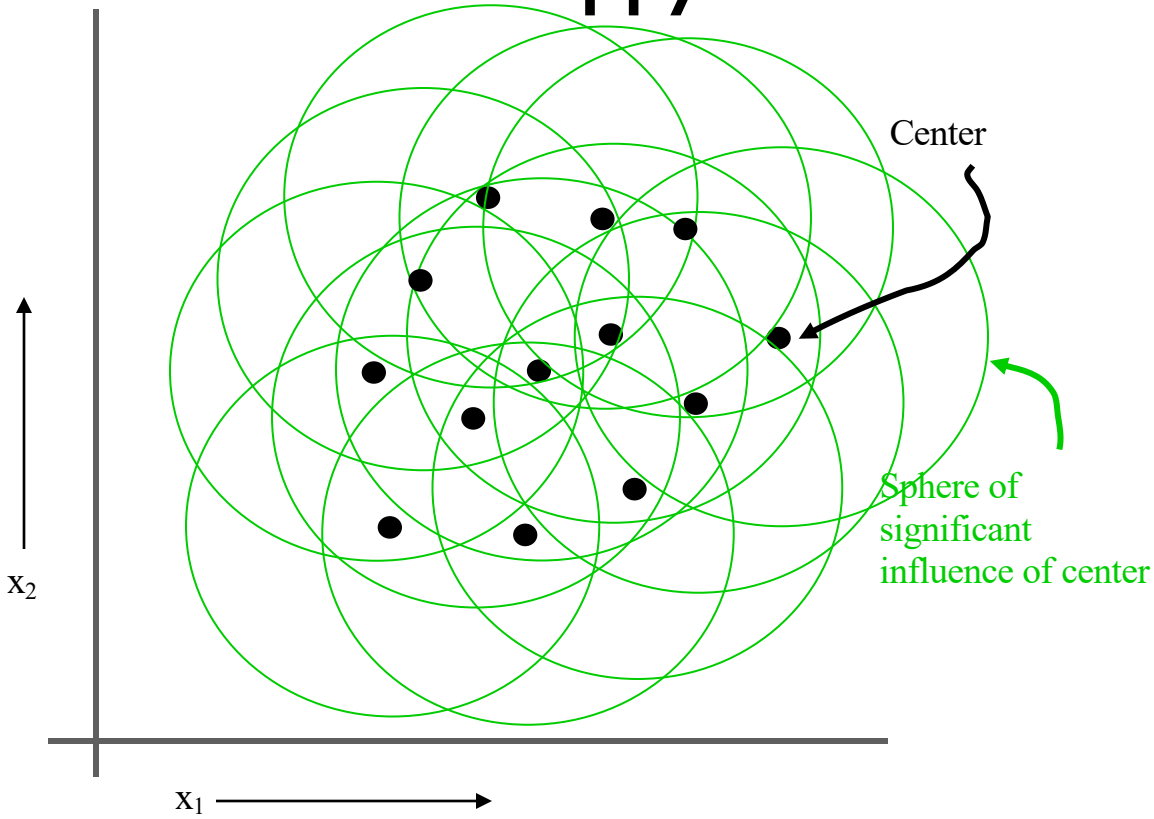
Hopeless!

Even before seeing the data, you should understand that this is a disaster!

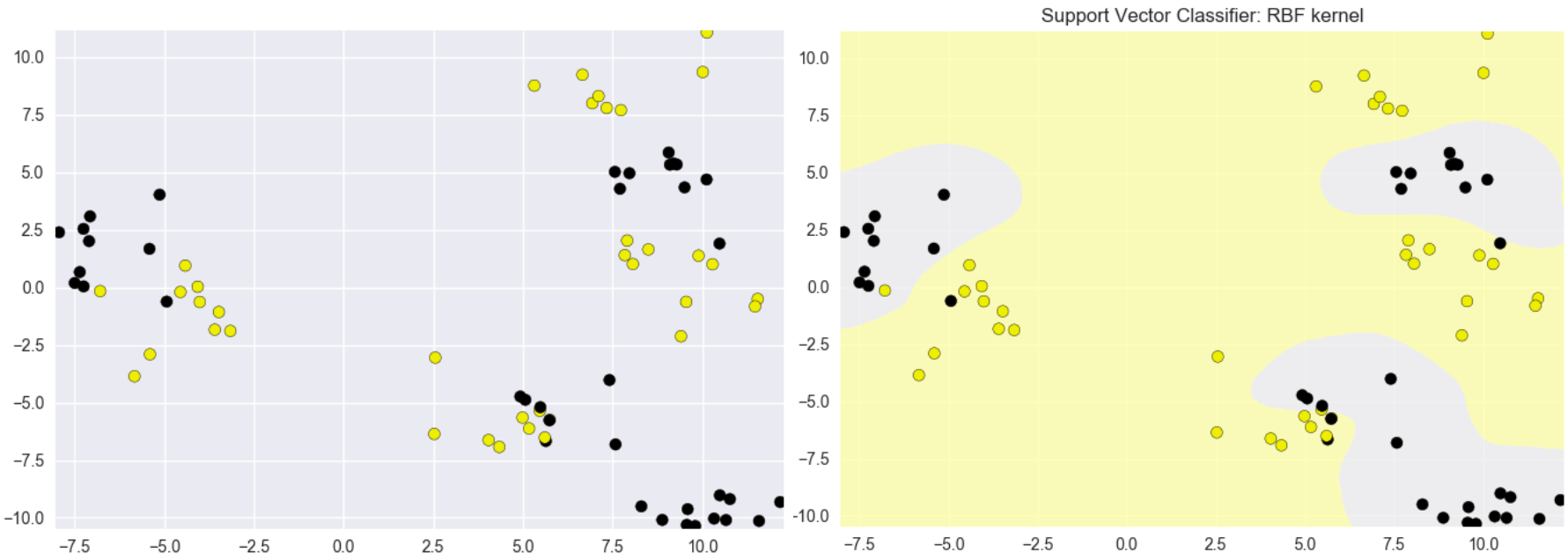


Unhappy

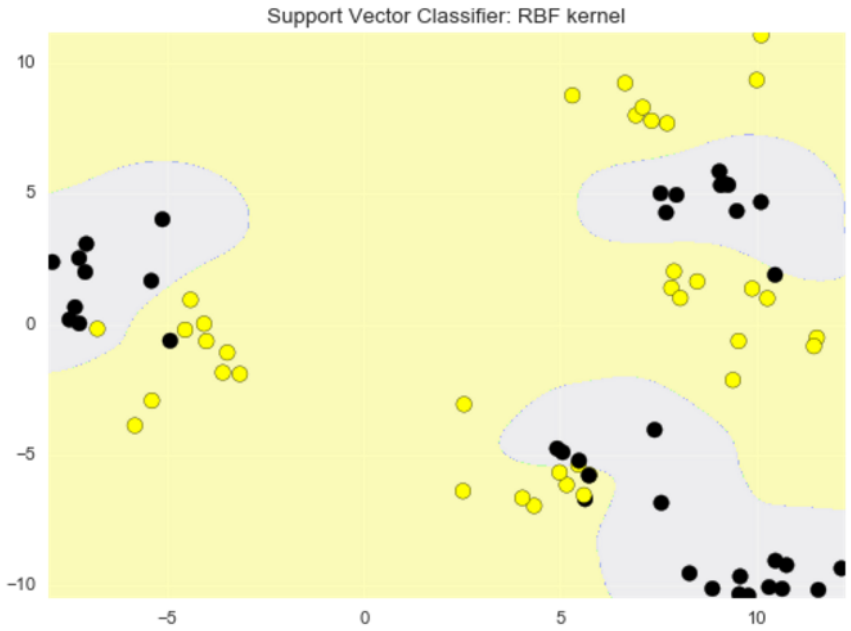
Even before seeing the data, you should understand that this isn't good either..

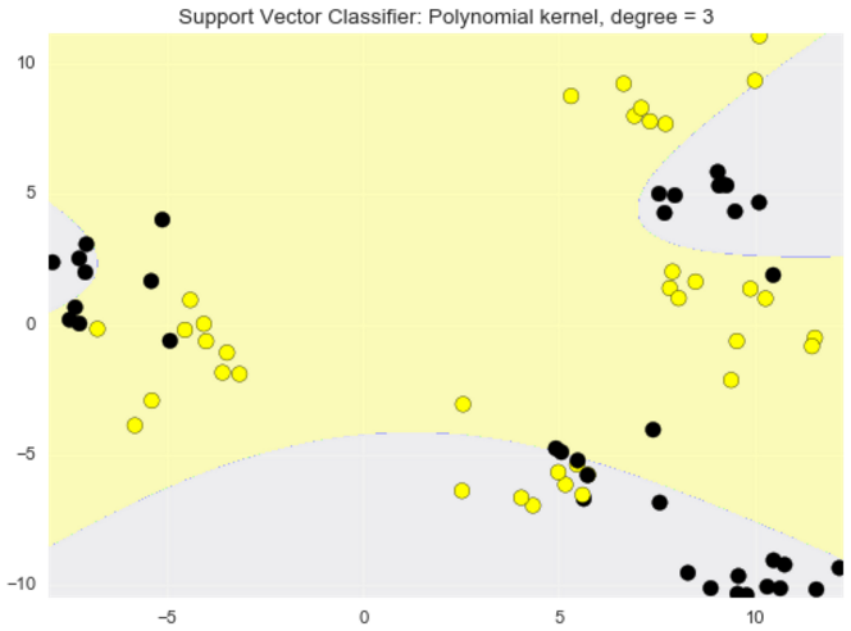


Applying the SVM with RBF kernel



Radial Basis Kernel vs Polynomial Kernel



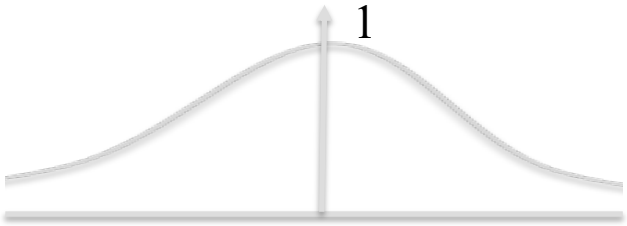


Radial Basis Function kernel: Gamma Parameter

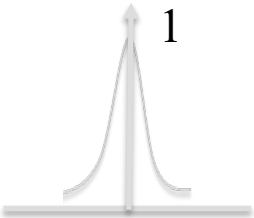
$$K(\boldsymbol{x}, \boldsymbol{x}') = \exp\ [-\gamma \cdot \|\boldsymbol{x} - \boldsymbol{x}'\|^2]$$

gamma (γ): kernel width parameter

small gamma (0.01)



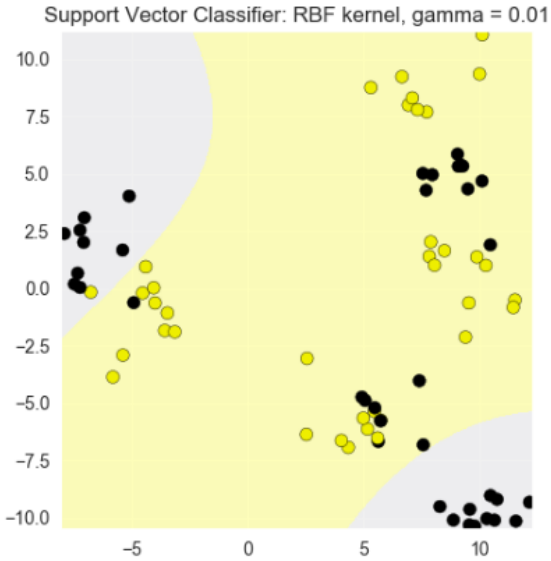
large gamma (10)



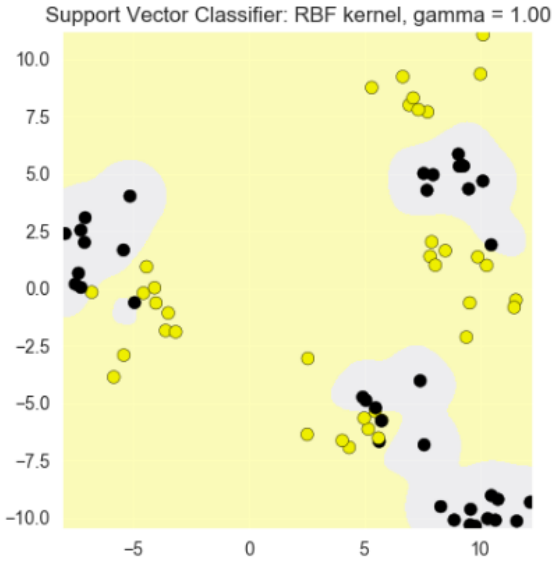
$\|\boldsymbol{x} - \boldsymbol{x}'\|^2$

Squared distance between x and x'

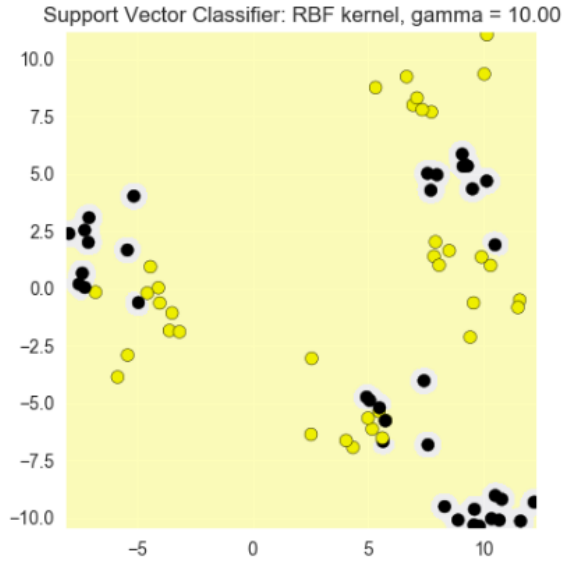
The effect of the RBF gamma parameter on decision boundaries



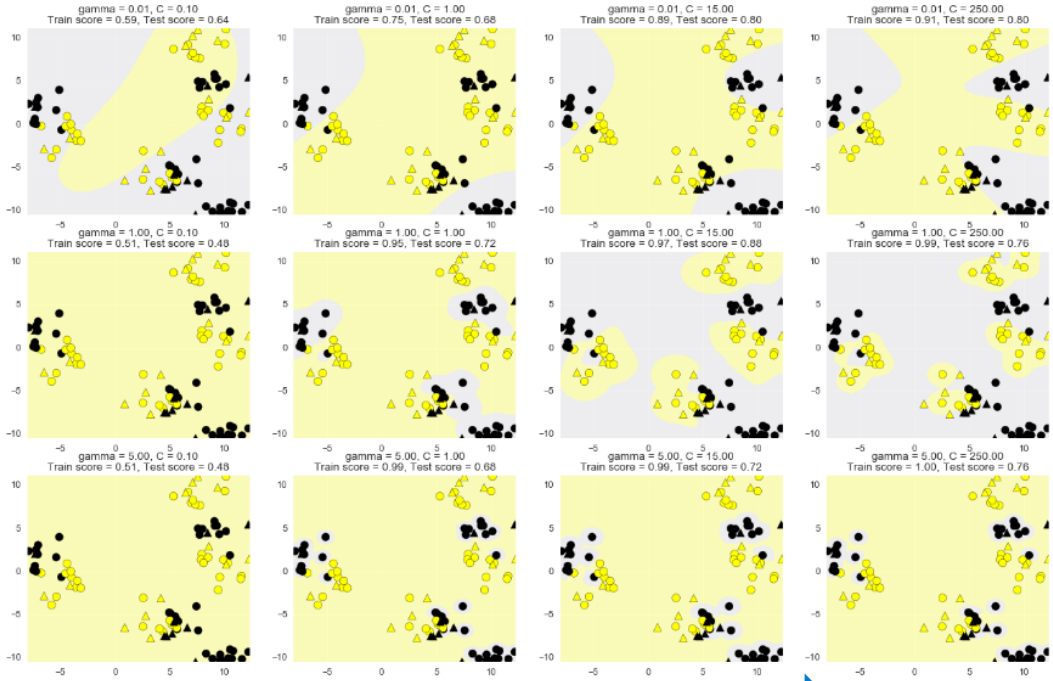
gamma = 0.01



gamma = 1.0



gamma = 10


gamma = 0.01, C = 0.10
Train score = 0.59, Test score = 0.64
gamma = 0.01, C = 1.00
Train score = 0.75, Test score = 0.68
gamma = 0.01, C = 15.00
Train score = 0.89, Test score = 0.80
gamma = 0.01, C = 250.00
Train score = 0.91, Test score = 0.80
gamma = 1.00, C = 0.10
Train score = 0.51, Test score = 0.48
gamma = 1.00, C = 1.00
Train score = 0.95, Test score = 0.72
gamma = 1.00, C = 15.00
Train score = 0.97, Test score = 0.88
gamma = 1.00, C = 250.00
Train score = 0.99, Test score = 0.76
gamma = 5.00, C = 0.10
Train score = 0.51, Test score = 0.48
gamma = 5.00, C = 1.00
Train score = 0.99, Test score = 0.68
gamma = 5.00, C = 15.00
Train score = 0.99, Test score = 0.72
gamma = 5.00, C = 250.00
Train score = 1.00, Test score = 0.76
Increasing gamma
Increasing C

Kernelized Support Vector Machines: pros and cons

Pros:

- Can perform well on a range of datasets.
- Versatile: different kernel functions can be specified, or custom kernels can be defined for specific data types.
- Works well for both low- and high-dimensional data.

Cons:

- Efficiency (runtime speed and memory usage) decreases as training set size increases (e.g. over 50000 samples).
- Needs careful normalization of input data and parameter tuning.
- Does not provide direct probability estimates (but can be estimated using e.g. Platt scaling).
- Difficult to interpret why a prediction was made.

Kernalized SVMs

- **Features**
  - *Must relate locally*
  - *Must normalize*
- **Parameters**
  - *Next slides*
- **Evaluation**
  - *Usual*
  - *Coefficients are not very interpretable*

Kernelized Support Vector Machines (SVC): Important parameters

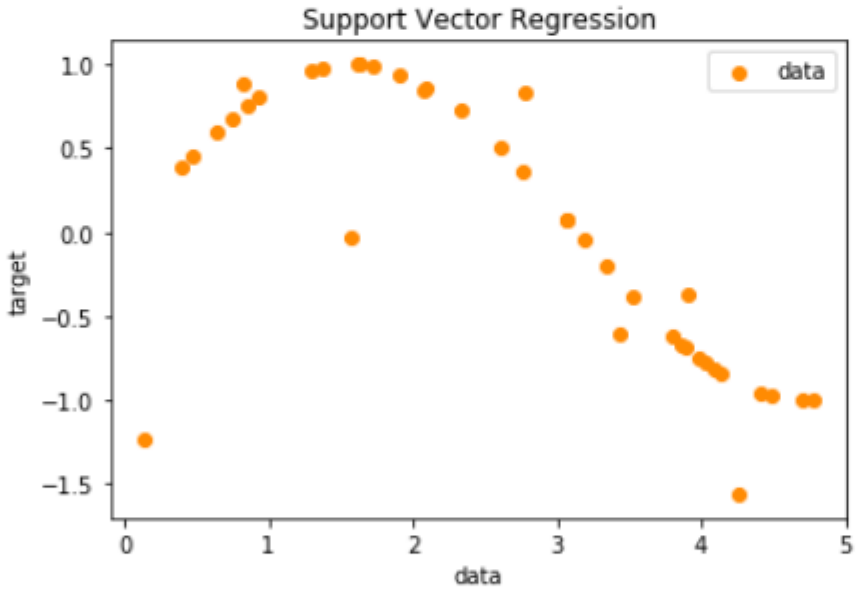
Model complexity

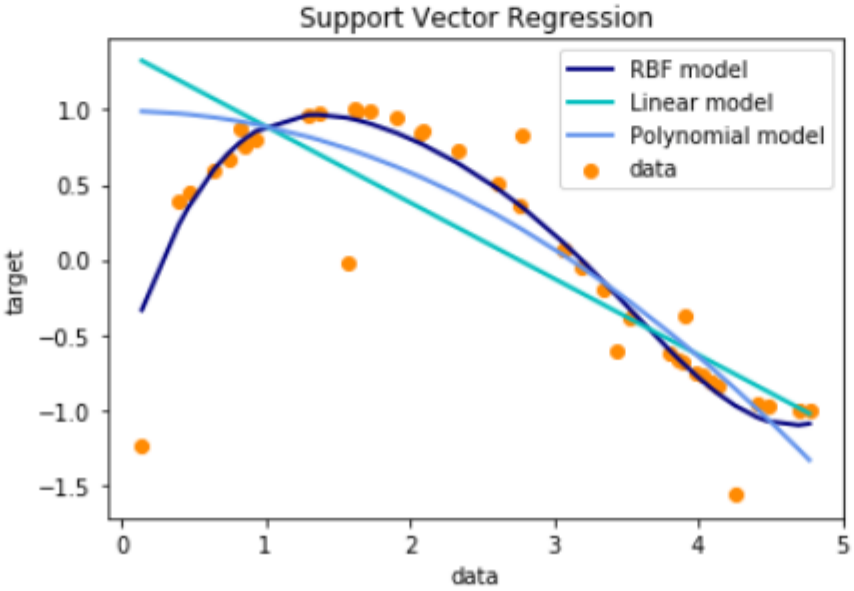
- **`kernel`: Type of kernel function to be used**
  - *Default = 'rbf' for radial basis function*
  - *Other types include 'polynomial'*
- **kernel parameters**
  - `gamma` *(γ): RBF kernel width*
- **C: regularization parameter: tolerance for training errors**
- **Typically `C` and `gamma` are tuned at the same time.**

Support Vector Regression

Can the maximum margin principle be used for regression?

Yes! Support Vector Regression

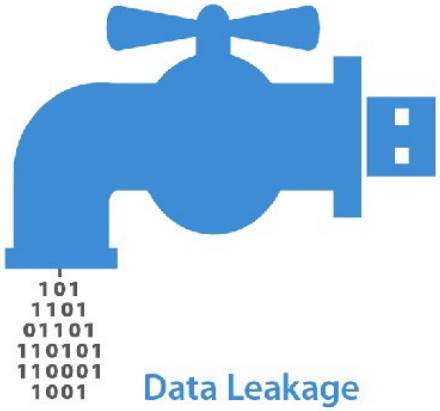



Support Vector Regression
RBF model
Linear model
Polynomial model
data
target
data
1.0
0.5
0.0
−0.5
−1.0
−1.5
0
1
2
3
4
5

SVM Regression

- **Ideal:**
  - *Minimize band so that all data within band.*
- **Actual:**
  - *Add some error for points outside band.*

Data Leakage



Data Leakage

- **Features contains illicit information about the target.**
  - *not available during actual use.*
  - *Obvious example: label to be predicted included as a feature*
- **Training data contains illicit information about test data**
  - *Including test data with training data*
- **If your model performance is too good to be true, it probably is and likely due to "giveaway" features.**

More subtle examples of illicit features

- Target: will user stay on a site, or leave?
  - *Giveaway feature: total session length*
  - *based on information about future page visits*
- Target: will user open an account?
  - *Giveaway feature: account number field*
  - *only filled in once the user does open an account.*
- Diagnostic medical test
  - *Giveaway feature: whether they had surgery for tested condition.*
  - *The patient ID could contain information about specific diagnosis paths (e.g. for routine visit vs specialist).*
- Giveaway features are highly predictive, but not available for the prediction

More subtle examples of illicit features

Leakage in features:

- Removing illicit variables but leaving others that encode related information (e.g. diagnosis info may still exist in patient ID).
- Implicitly reversing intentional randomization / anonymization
- Above problems when extra data is added:
  - *external data joined to the training set.*
- Time-series datasets: using records from the future when computing features for the current prediction.
- Errors in data values/gathering or missing variable indicators (e.g. the special value 999) can encode information about missing data that reveals information about the future.

Illicit training

- Data preprocessing using parameters or results from analyzing the entire dataset:
  - *Normalizing and rescaling,*
  - *detecting and removing outliers,*
  - *estimating missing values,*
  - *feature selection,*
  - *cross-validation.*
- Perform data preparation and hyper-parameter tuning within each cross-validation fold separately

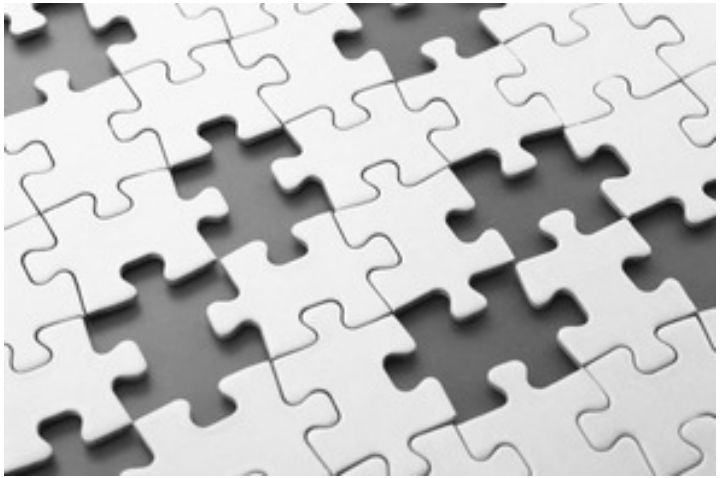
Detecting data leakage

- Think!
- Before building the model
  - *Exploratory data analysis to find surprises in the data*
  - *Are unexpected features highly correlated with the target?*
- After building the model
  - *Look for surprising feature behavior in the fitted model.*
    - *high weights, (or high information gain)*
  - *Is overall model performance surprisingly good?*
    - *compare to known results on the same/similar datasets*
    - *Example: Coupon Detection*
- (Limited) real-world deployment of the trained model
  - *Do results generalize?*

Minimizing Data Leakage

- Perform data preparation and hyper-parameter tuning within each cross-validation fold separately
- With time series data, use a timestamp cutoff
- Before any work with a new dataset, split off a final test dataset
  - *… if you have enough data*
  - *Use this final test dataset as the very last step in your validation*
  - *Helps to check the true generalization performance of any trained models*

Missing Data



Well the two men took to fighting
And when they pulled them off the floor
Leroy looked like a jigsaw puzzle
With a couple of pieces gone.

Imputing missing data

- How to handle missing data?
- Data will be missing!
- No Silver Bullet
- For categories, create "unknown" category.
  - *Can create missing dummy variable for numerical values.*
- Otherwise
  - *can discard data*
  - *can "learn" missing value.*

"Learning" Missing values

- Can use a Dummy Regressor to learn:
  - *Replace with average*
  - *Replace with constant*
- More sophisticated single imputation:
  - *Less common*
  - *Ordinary Least Squares*
  - *K-NN*
    - *choose randomly?*
  - *From a posterior distribution*
- Multiple imputation:
  - *Impute all incomplete features at the same time.*

More on Missing Features

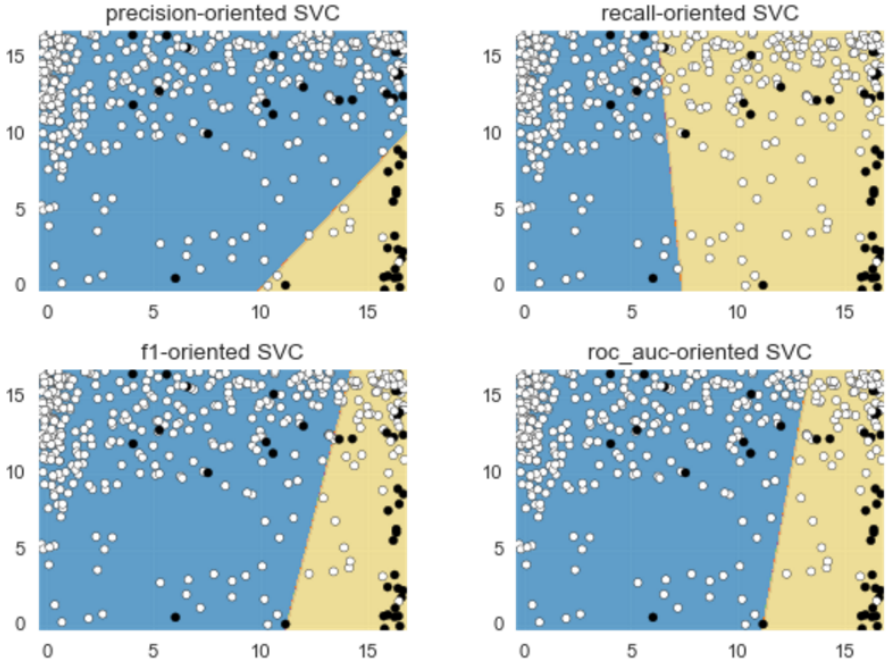
- **Censorship: Missing data depends on values**
  - *Cannot just remove data*
  - *Replacing with average not great*
- **Time series data:**
  - *Often other approaches to take*
- **Less important in ML than statistics**
  - *Better imputation will improve prediction*
  - *Do not have to worry about confidences*
  - *Still implications for fairness*

Optimizing Classifiers for Different Evaluation Metrics

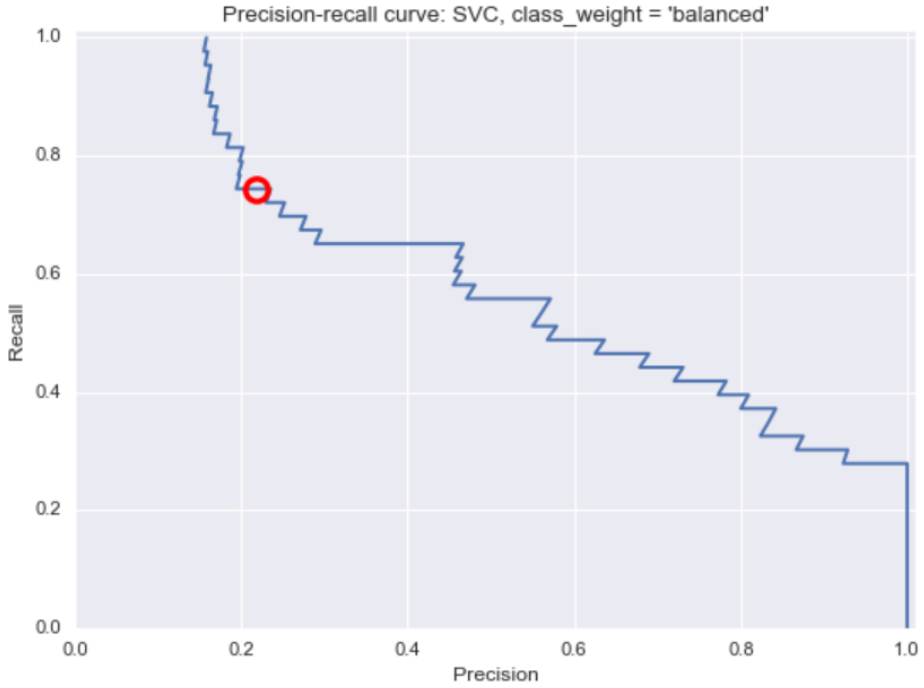
Certain functions (parameter search) can take an evaluation metric as a parameter

```
clf = Logistic(C=100.0).fit(X_twovar_train, y_train)
grid_values = {'class_weight':['balanced',
{1:2},{1:3},{1:4},{1:5},{1:10},{1:20},{1:50}]}
plt.figure(figsize=(10,7))
for i, eval_metric in enumerate(('precision','recall', 'f1','roc_auc')):
    grid_clf_custom = GridSearchCV(clf, param_grid=grid_values,
                                   scoring=eval_metric)
    grid_clf_custom.fit(X_twovar_train, y_train)
        ... etc
```

Example: Optimizing a Classifier Using Different Evaluation Metrics



Example: Precision-Recall Curve of Default Support Vector Classifier



Regression to the Mean

Thought Experiment I

- Each person in this class gets a die.
- Each die:
  - *Has six sides.*
  - *Each side contains some number 1-6.*
  - *Each number may appear more than once.*
- We all role the dice 10 times.
- The winning die is the die with the greatest sum of its rolls.
- Let H be this sum on the winning die.
- Roll the winning die 10 more bonus times, let B be their sum.
- Do we expect:
- A) It is more likely that H > B
- B) It is more likely that B < H
- C) The above are equally likely.

Thought Experiment II

- Each person in this class gets a die.
- Each die:
  - *Has six sides with the numbers 1-6 appearing, one per side. We all role the dice 10 times.*
- The winning die is the die with the greatest sum of its rolls.
- Let H be this sum on the winning die.
- Roll the winning die 10 more bonus times, let B be their sum.
- Do we expect:
- A) It is more likely that H > B
- B) It is more likely that B < H
- C) The above are equally likely.

Thought Experiment III

- Each person in this class gets a die.
- Each die:
  - *Has six sides.*
  - *Sum of all the sides is a number between 1 and 120*
  - *Each die has as different sum*
- The winning die is the die with the greatest sum of its rolls.
- Let H be this sum on the winning die.
- Roll the winning die 10 more bonus times, let B be their sum.
- Do we expect:
- A) It is more likely that H > B
- B) It is more likely that B < H
- C) The above are equally likely.

Thought Experiment III

- Each person in this class comes submits hyper parameters for SVM with RBF.
- We use cross validation to train each model.
- The winning model is the one with the highest average test accuracy.
- Let H be this accuracy.
- We test the winning parameters on a completely new set of data and find it has accuracy B.
- Do we expect:
- A) It is more likely that H > B
- B) It is more likely that B < H
- C) The above are equally likely.

Regression to the Mean

- After selecting for an extreme event:
  - *Likely detecting noise AND signal.*
  - *Part of the reason for selection is signal.*
  - *Part of the reason for selection is noise.*
  - *When it is rerun, new noise is drawn.*
- Selecting Faculty/Employees for Promotion
- Punishing Poor Performance
- Selecting over Hyper-parameters!

Training, Validation, and Test Framework for Model Selection and Evaluation

- Using only cross-validation or a test set to do model selection may lead to more subtle overfitting / optimistic generalization estimates

How Many Splits Do You Need?

Training, Validation, and Test Framework for Model Selection and Evaluation

- **Three data splits:**
  1. *Training set (model building)*
  2. *Validation set (model selection—tune parameters, but never fit models)*
  3. *Test set (final evaluation)*

Training, Validation, and Test Framework for Model Selection and Evaluation

- **In practice:**
  - *Create an initial training/test split*
  - *Do cross-validation on the training data for model/parameter selection*
  - *Save the held-out test set for final model evaluation*

Case 1

- You have 10 different algorithms to test. You want to discover the best algorithm. You use your labeled data to run cross validation on each algorithm. You determine that algorithm with the highest average score is most likely to be the best.
- Did you do anything wrong?
  - *A) Yes, you should have divided your training data in to training and test data.*
  - *B) No, what could have gone wrong?*

Case 2

- You have 10 different algorithms to test. You want to discover the *accuracy* of the best algorithm. You use your labeled data to run cross validation on each algorithm. You return the accuracy of the highest algorithm.
- Did you do anything wrong?
  - *A) Yes, you should have divided your training data in to training and test data.*
  - *B) No, what could have gone wrong?*