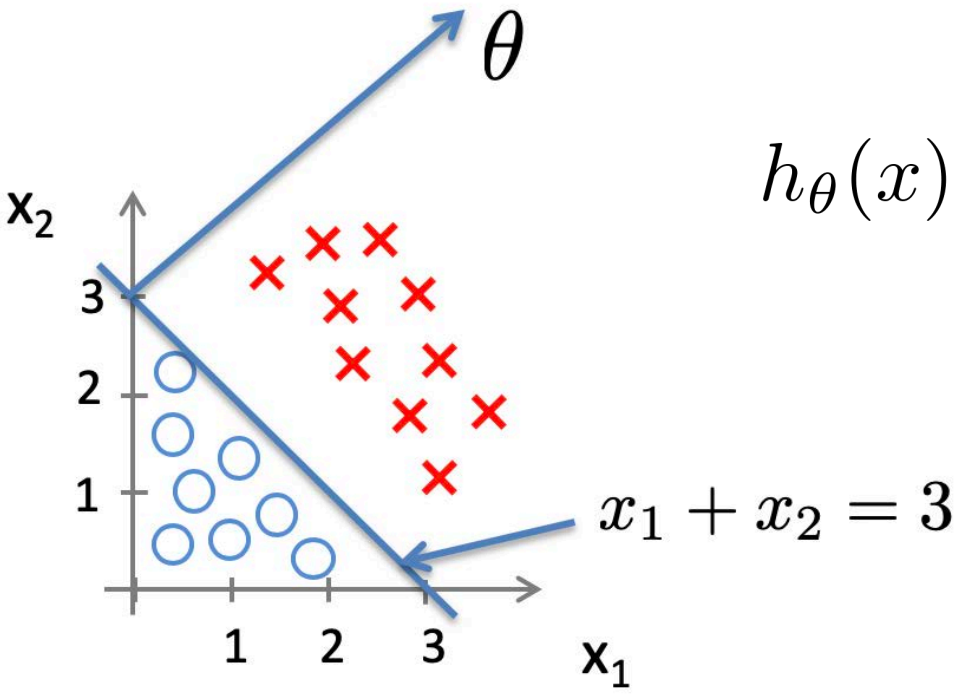
Support vector machines (SVM)

Andrew Ng

Support vector machines

Large margin intuition

Decision Boundary in Logistic Regression



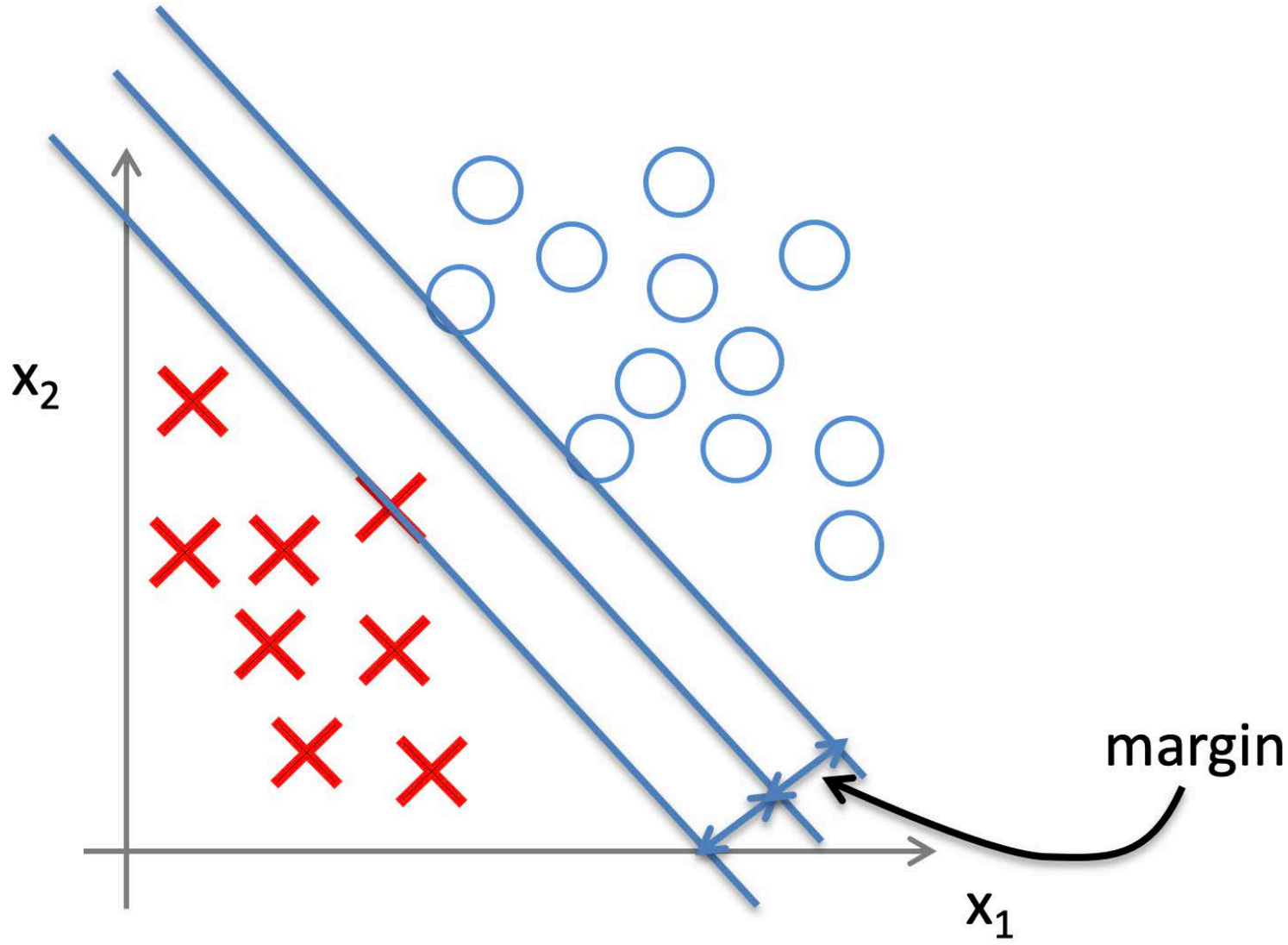
$$h_\theta(x) = g(\theta_0 + \theta_1 x_1 + \theta_2 x_2)$$

$$\theta = \begin{bmatrix} -3 \\ 1 \\ 1 \end{bmatrix}$$

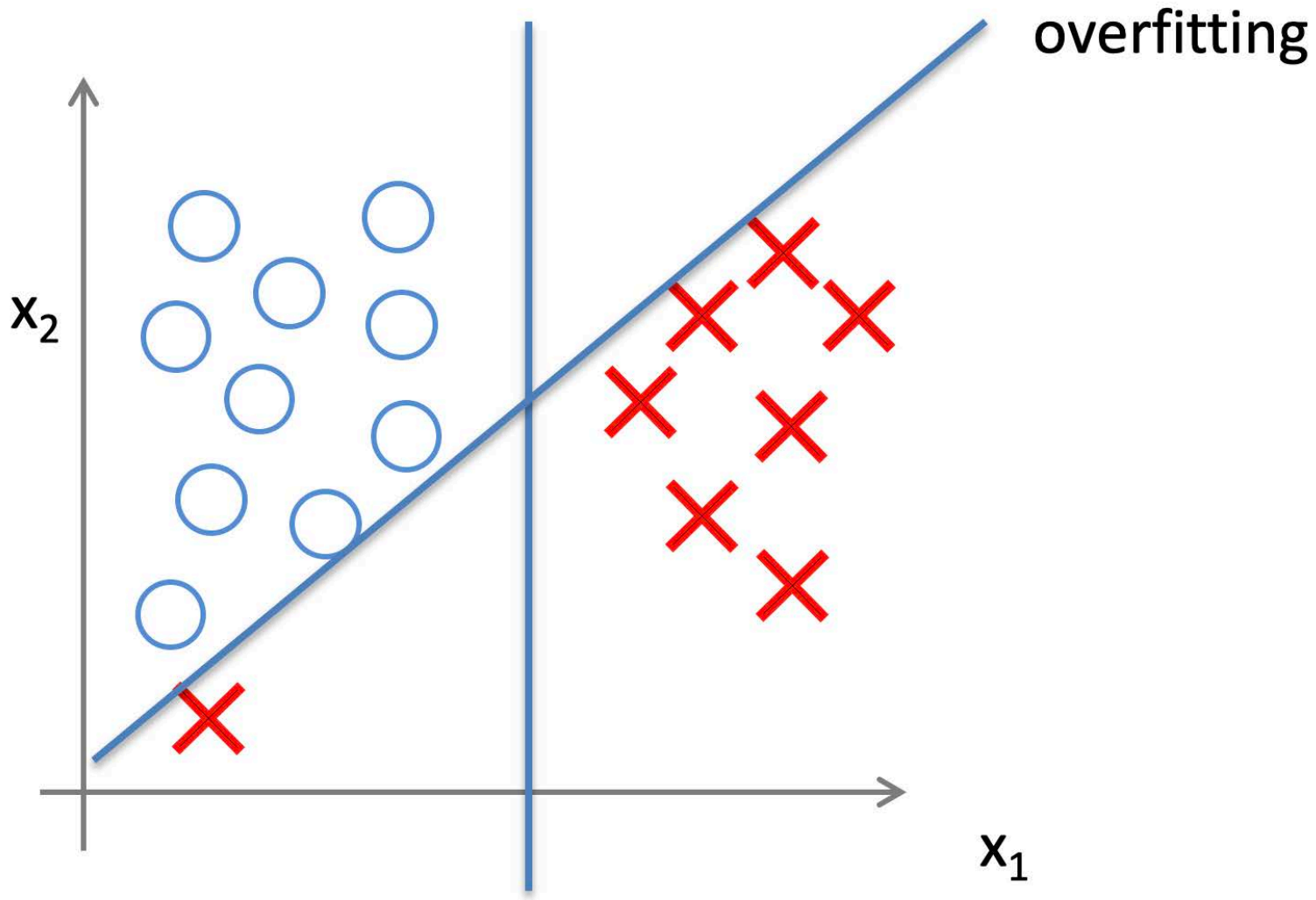
Predict "$y = 1$" if $-3 + x_1 + x_2 \geq 0$

$\theta^T x$

SVM Decision Boundary: Large margin classifier



Large margin classifier in presence of outliers

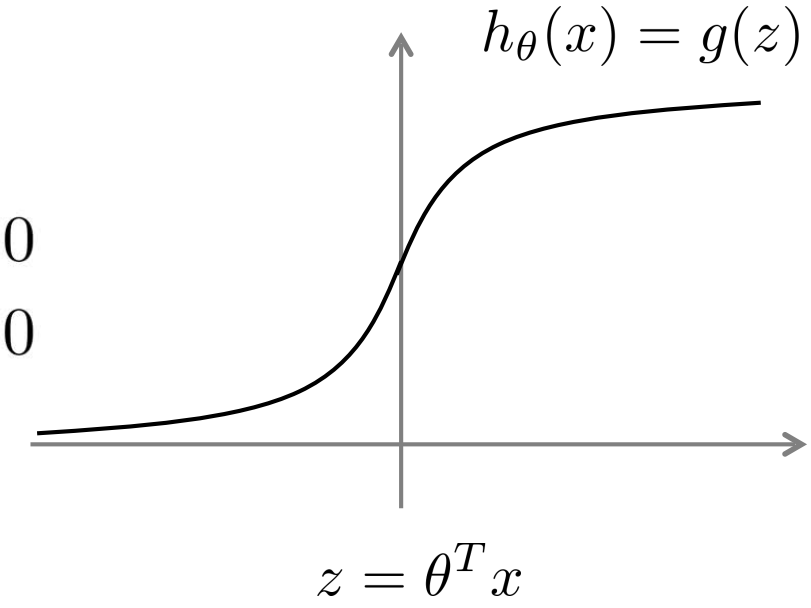


Support vector machines

Optimization objective

Logistic regression hypothesis

$$h_\theta(x) = \frac{1}{1 + e^{-\theta^T x}}$$



If $y = 1$, we want $h_\theta(x) \approx 1$, $\theta^T x \gg 0$

If $y = 0$, we want $h_\theta(x) \approx 0$, $\theta^T x \ll 0$

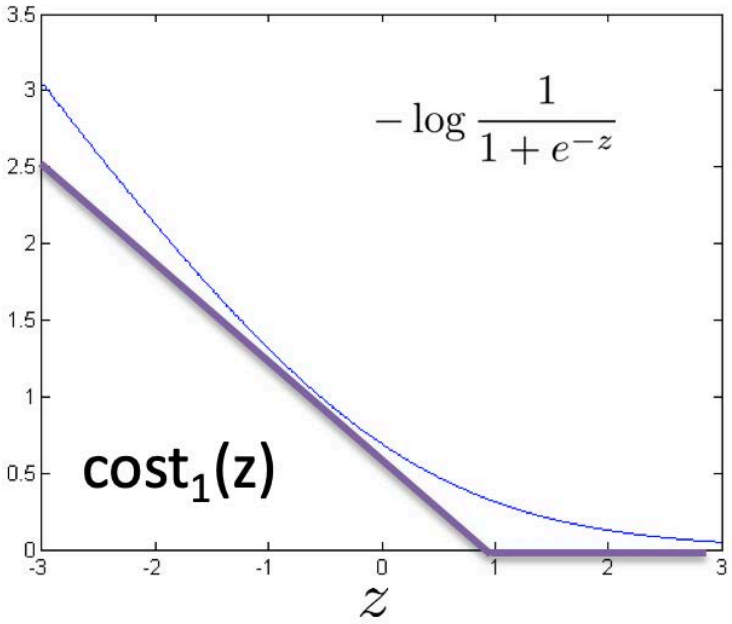
$$h_\theta(x) = \begin{cases} 1 & \text{if } \theta^T x \geq 0 \\ 0 & \text{otherwise} \end{cases}$$

SVM gets a similar effect with a different cost function

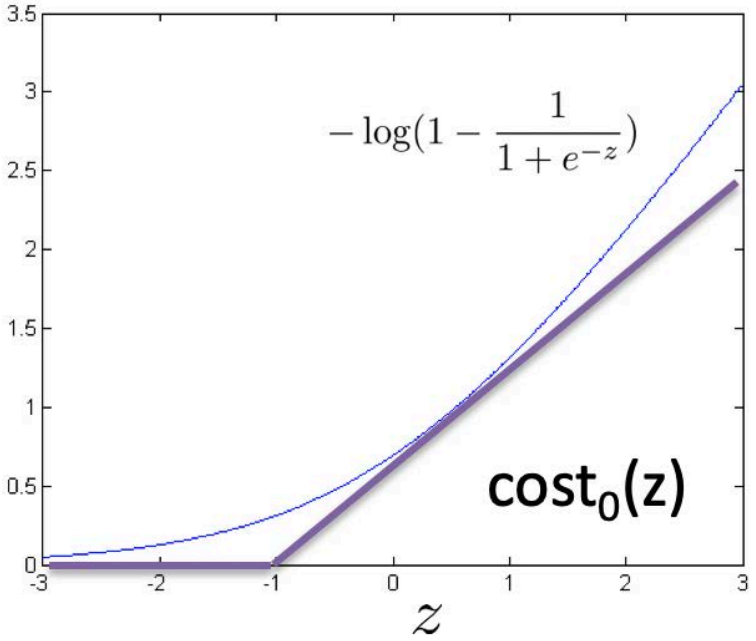
Cost of example: $-(y \log h_\theta(x) + (1-y)\log(1-h_\theta(x)))$

$$= -y \log \frac{1}{1+e^{-\theta^T x}} - (1-y)\log(1 - \frac{1}{1+e^{-\theta^T x}})$$

If $y = 1$ (want $\theta^T x \gg 0$):



If $y = 0$ (want $\theta^T x \ll 0$):



Support vector machine

Logistic regression:

$$\min_\theta \frac{1}{m}\left[\sum_{i=1}^{m} y^{(i)}\left(-\log h_\theta(x^{(i)})\right) + (1-y^{(i)})\left((-\log(1-h_\theta(x^{(i)}))\right)\right] + \frac{\lambda}{2m}\sum_{j=1}^{n}\theta_j^2$$

Support vector machine:

$$\min_\theta \frac{1}{m}\left[\sum_{i=1}^{m} y^{(i)} cost_1(\theta^T x^{(i)}) + (1-y^{(i)}) cost_0(\theta^T x^{(i)})\right] + \frac{\lambda}{2m}\sum_{j=1}^{n}\theta_j^2$$

$$\min_\theta C\sum_{i=1}^{m}\left[y^{(i)} cost_1(\theta^T x^{(i)}) + (1-y^{(i)}) cost_0(\theta^T x^{(i)})\right] + \frac{1}{2}\sum_{i=1}^{n}\theta_j^2$$

Support vector machines

The mathematics behind large margin classification

SVM Decision Boundary

$$\min_\theta C \sum_{i=1}^{m} \left[y^{(i)} cost_1(\theta^T x^{(i)}) + (1 - y^{(i)}) cost_0(\theta^T x^{(i)}) \right] + \frac{1}{2} \sum_{i=1}^{n} \theta_j^2$$

If C is very large then we choose θ s.t. the cost term is 0:

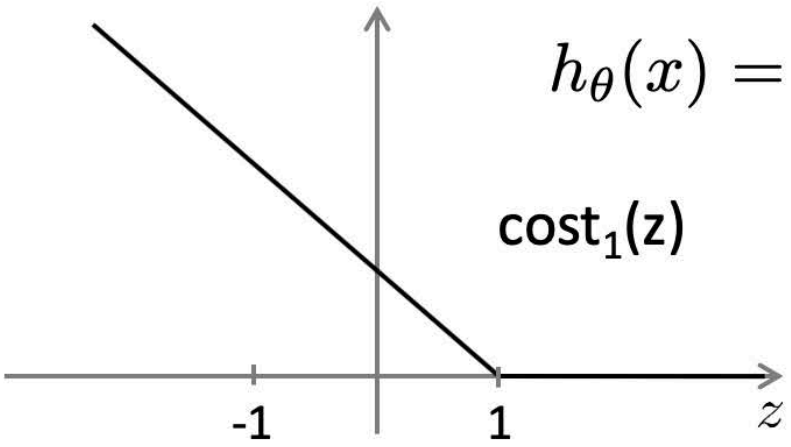
$$\min_\theta (C \times 0 + \frac{1}{2} \sum_{i=1}^{n} \theta_j^2)$$

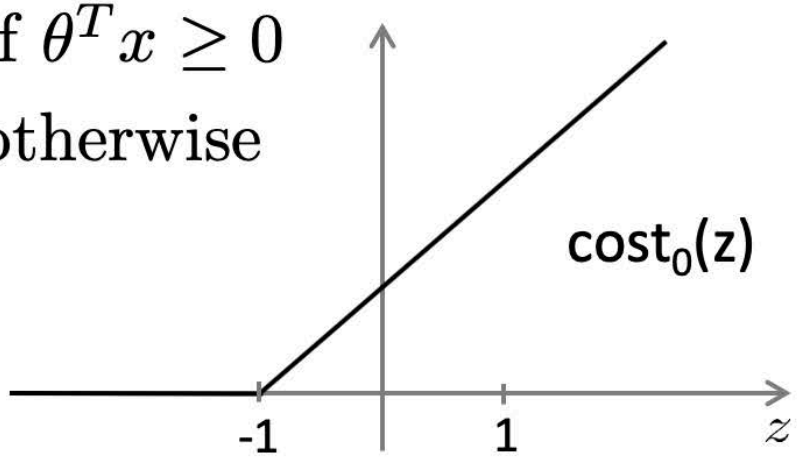
$$\begin{aligned} \text{s.t.} \quad & \theta^T x^{(i)} \geq 1 \quad && \text{if } y^{(i)} = 1 \\ & \theta^T x^{(i)} \leq -1 && \text{if } y^{(i)} = 0 \end{aligned}$$

Support Vector Machine

$$\min_{\theta} C \sum_{i=1}^{m} \left[y^{(i)} cost_1(\theta^T x^{(i)}) + (1 - y^{(i)}) cost_0(\theta^T x^{(i)}) \right] + \frac{1}{2} \sum_{i=1}^{n} \theta_j^2$$

$$h_\theta(x) = \begin{cases} 1 & \text{if } \theta^T x \geq 0 \\ 0 & \text{otherwise} \end{cases}$$

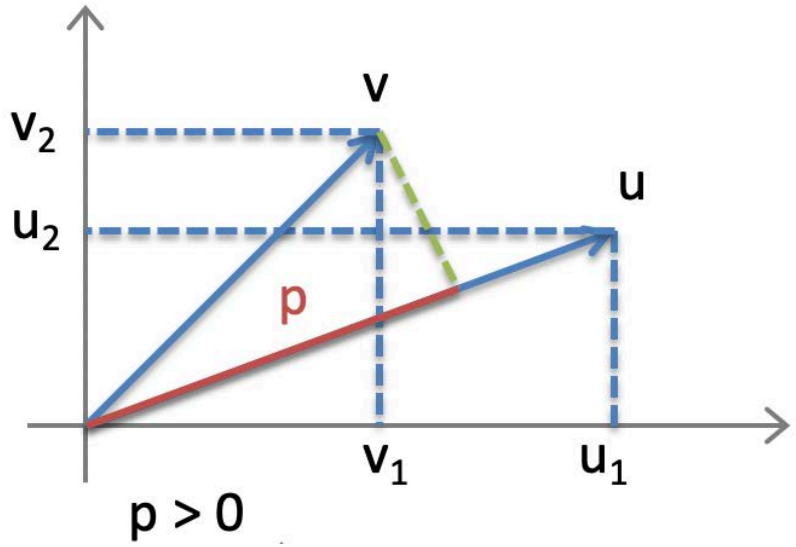




If $y = 1$, we want $\theta^T x \geq 1$ (not just ≥ 0)

If $y = 0$, we want $\theta^T x \leq -1$ (not just < 0)

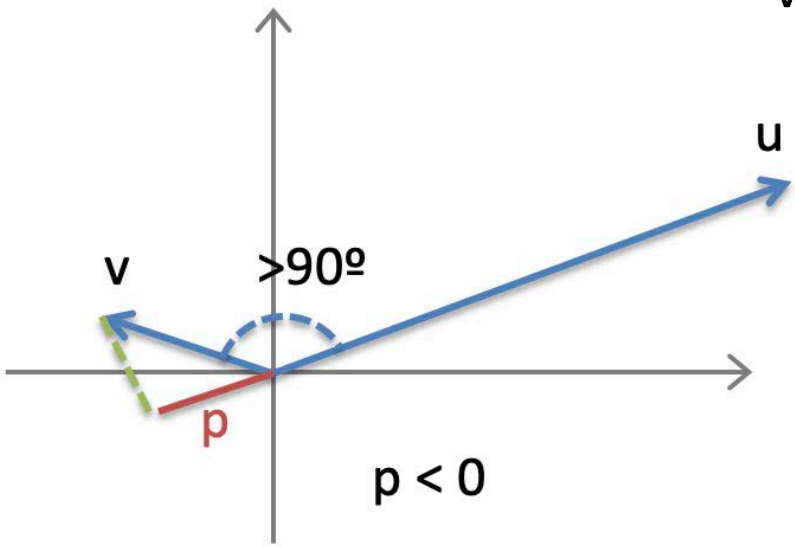
Vector Inner Product



$$u = \begin{bmatrix} u_1 \\ u_2 \end{bmatrix} \qquad v = \begin{bmatrix} v_1 \\ v_2 \end{bmatrix}$$

$$\|u\| = \sqrt{u_1^2 + u_2^2}$$

p: length of projection of v onto u



vector inner product: $u^T v$

$$u^T v = p \cdot \|u\| = u_1 v_1 + u_2 v_2$$

SVM Decision Boundary

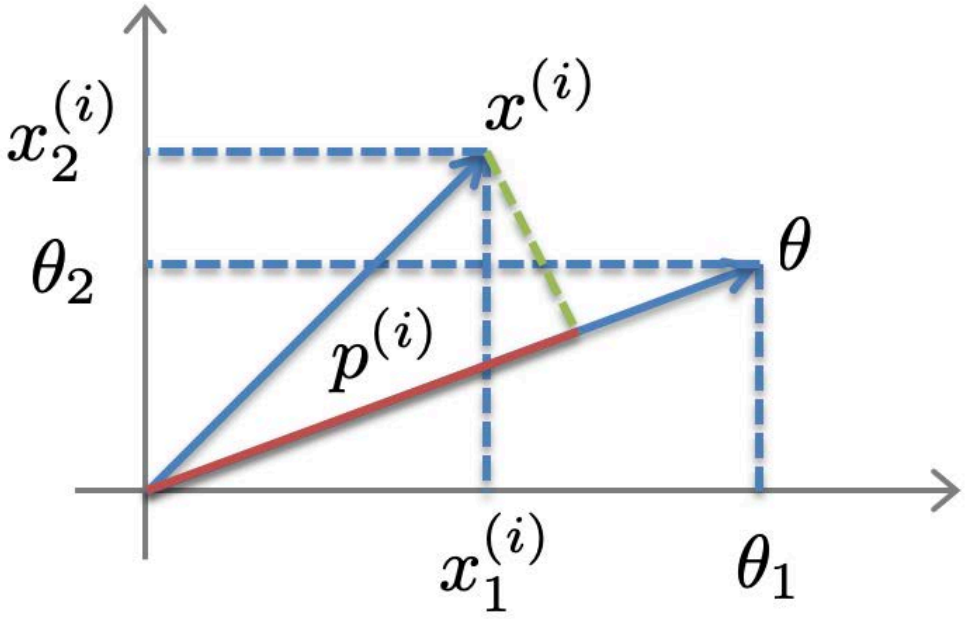
$$\min_\theta \frac{1}{2}\sum_{j=1}^{n}\theta_j^2 = \frac{1}{2}(\theta_1^2+\theta_2^2) = \frac{1}{2}\left(\sqrt{\theta_1^2+\theta_2^2}\right)^2 = \frac{1}{2}\|\theta\|^2$$

$$\text{s.t.} \quad \theta^T x^{(i)} \geq 1 \qquad \text{if } y^{(i)} = 1$$

$$\theta^T x^{(i)} \leq -1 \qquad \text{if } y^{(i)} = 0$$

simplification: $\theta_0 = 0$

n = 2



$$\theta^T x^{(i)} = p^{(i)} \cdot \|\theta|$$

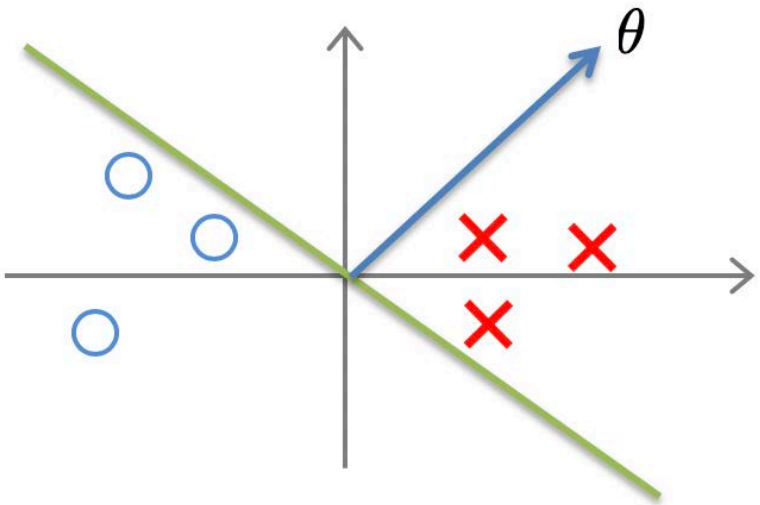
$$= \theta_1 x_1^{(i)} + \theta_2 x_2^{(i)}$$

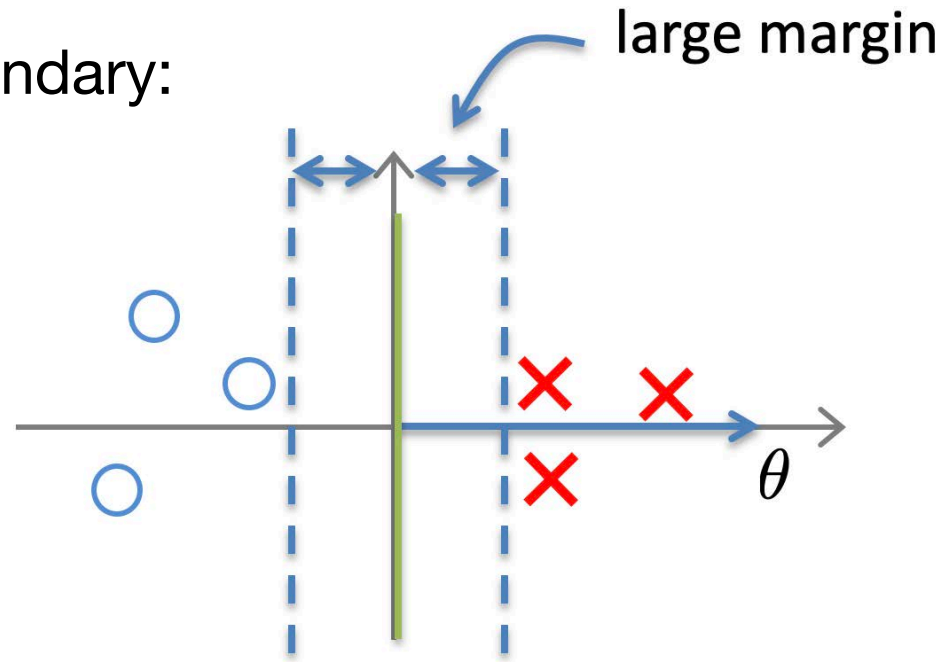
SVM Decision Boundary

$$\min_{\theta} \frac{1}{2}\sum_{j=1}^{n}\theta_j^2 = \frac{1}{2}\|\theta\|^2 \quad \text{s.t.} \quad p^{(i)} \cdot \|\theta\| \geq 1 \quad \text{if } y^{(i)} = 1$$
$$p^{(i)} \cdot \|\theta\| \leq -1 \quad \text{if } y^{(i)} = 0$$

where $p^{(i)}$ is the projection of $x^{(i)}$ onto the vector θ.
Simplification: $\theta_0 = 0$

θ is orthogonal to the decision boundary:



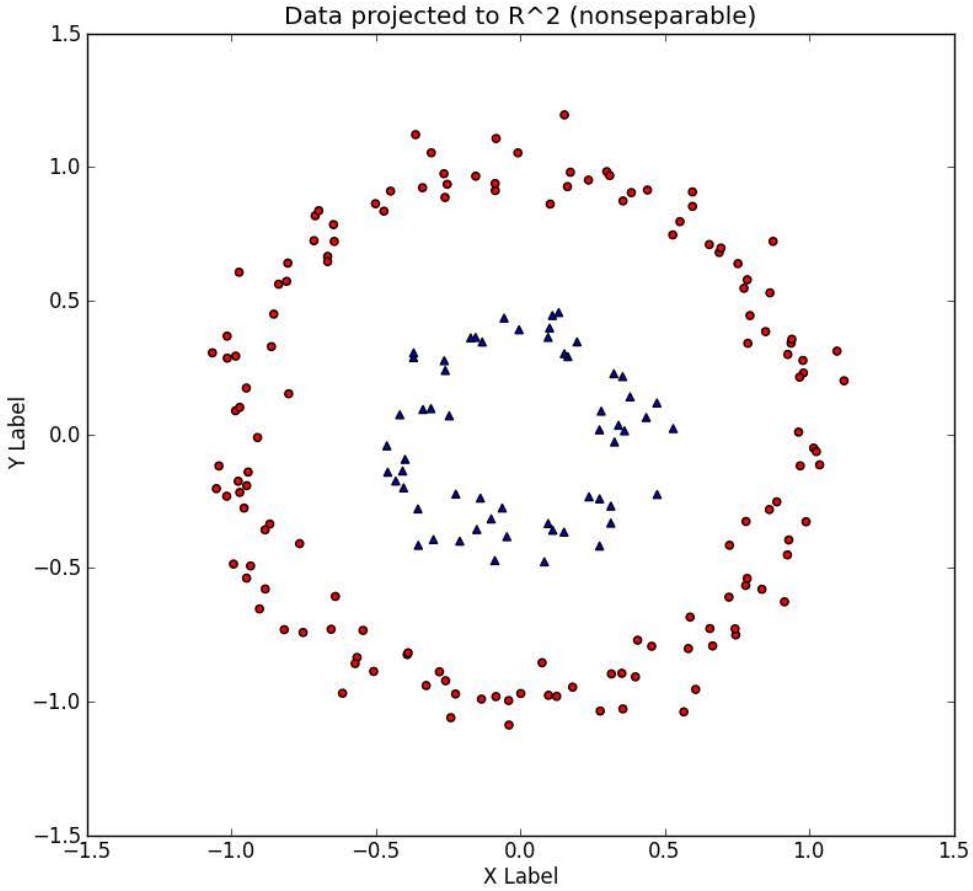


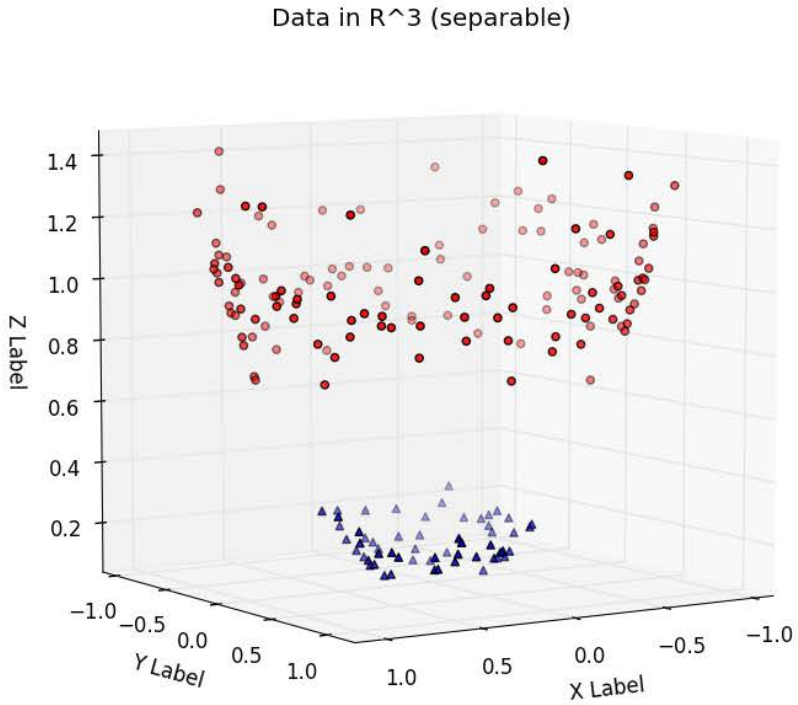
$$p^{(i)}\text{small} \implies \|\theta\| \text{ large}$$

Support vector machines

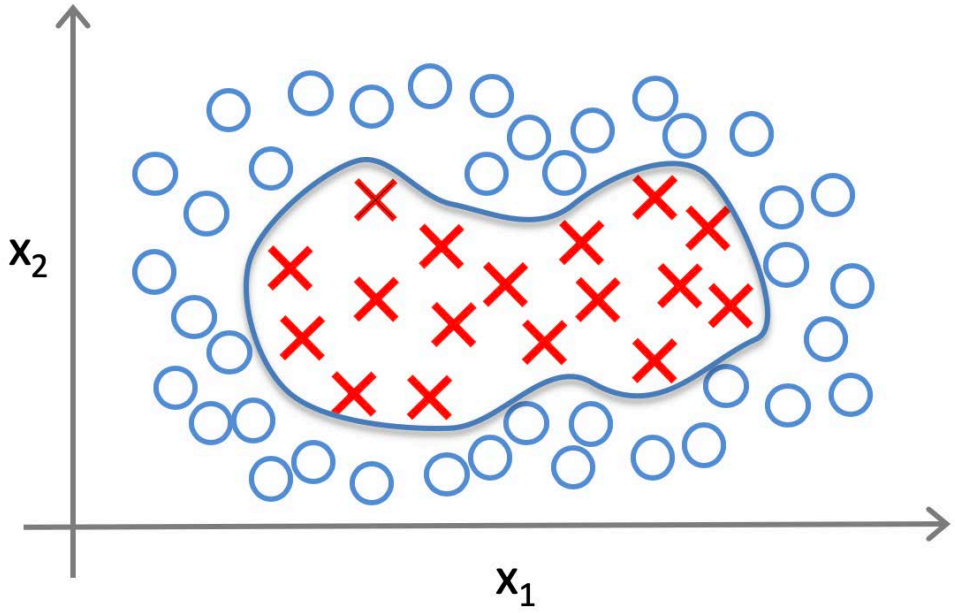
Kernels I

Non-linear Decision Boundary





Non-linear Decision Boundary



$$\text{Predict } y = 1 \text{ if}$$

$$\theta_0 + \theta_1 x_1 + \theta_2 x_2 + \theta_3 x_1 x_2 + \theta_4 x_1^2 + \theta_5 x_2^2 + \cdots \geq 0$$

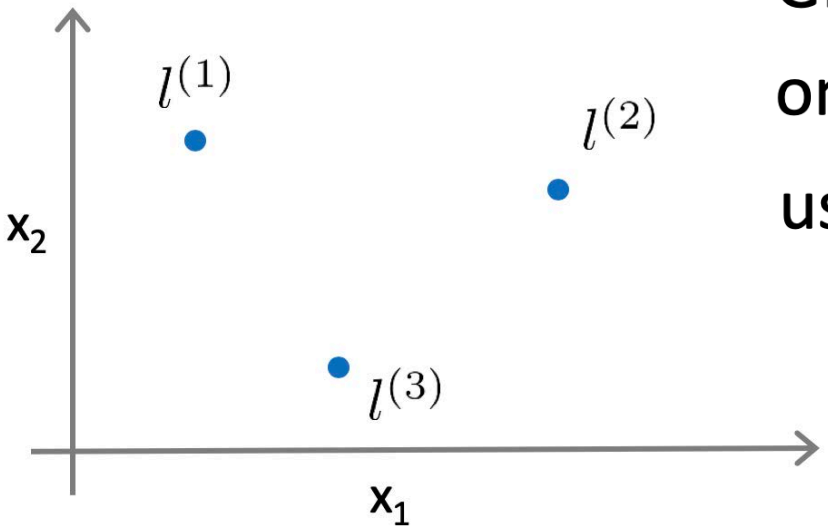
$$h_\theta(x) = \begin{cases} 1 & \text{if } \theta_0 + \theta_1 x_1 + \cdots \geq 0 \\ 0 & \text{otherwise} \end{cases}$$

$$\theta_0 + \theta_1 f_1 + \theta_2 f_2 + \theta_3 f_3 + \ldots$$

$$f_1 = x_1, f_2 = x_2, f_3 = x_1 x_2, f_4 = x_1^2, f_5 = x_2^2, \ldots$$

Is there a different / better choice of the features $f_1, f_2, f_3, \ldots$?

Kernel



Given x, compute new feature depending on proximity to landmarks $l^{(1)}, l^{(2)}, l^{(3)}$ using for example a gaussian function:

$$f_1 = similarity(x, l^{(1)}) = exp\left(-\frac{\|x - l^{(1)}\|^2}{2\sigma^2}\right)$$

$$f_2 = similarity(x, l^{(2)}) = exp\left(-\frac{\|x - l^{(2)}\|^2}{2\sigma^2}\right)$$

$$f_3 = similarity(x, l^{(3)}) = exp\left(-\frac{\|x - l^{(3)}\|^2}{2\sigma^2}\right)$$

similarity or “kernel” function: $k(x, l^{(i)})$

Kernels and Similarity

$$f_1 = \text{similarity}(x, l^{(1)}) = \exp\left(-\frac{\|x - l^{(1)}\|^2}{2\sigma^2}\right) = \exp\left(-\frac{\sum_{j=1}^{n}(x_j - l_j^{(1)})^2}{2\sigma^2}\right)$$

If $x \approx l^{(1)}$:

$$f_1 \approx exp\left(-\frac{0^2}{2\sigma^2}\right) \approx 1$$

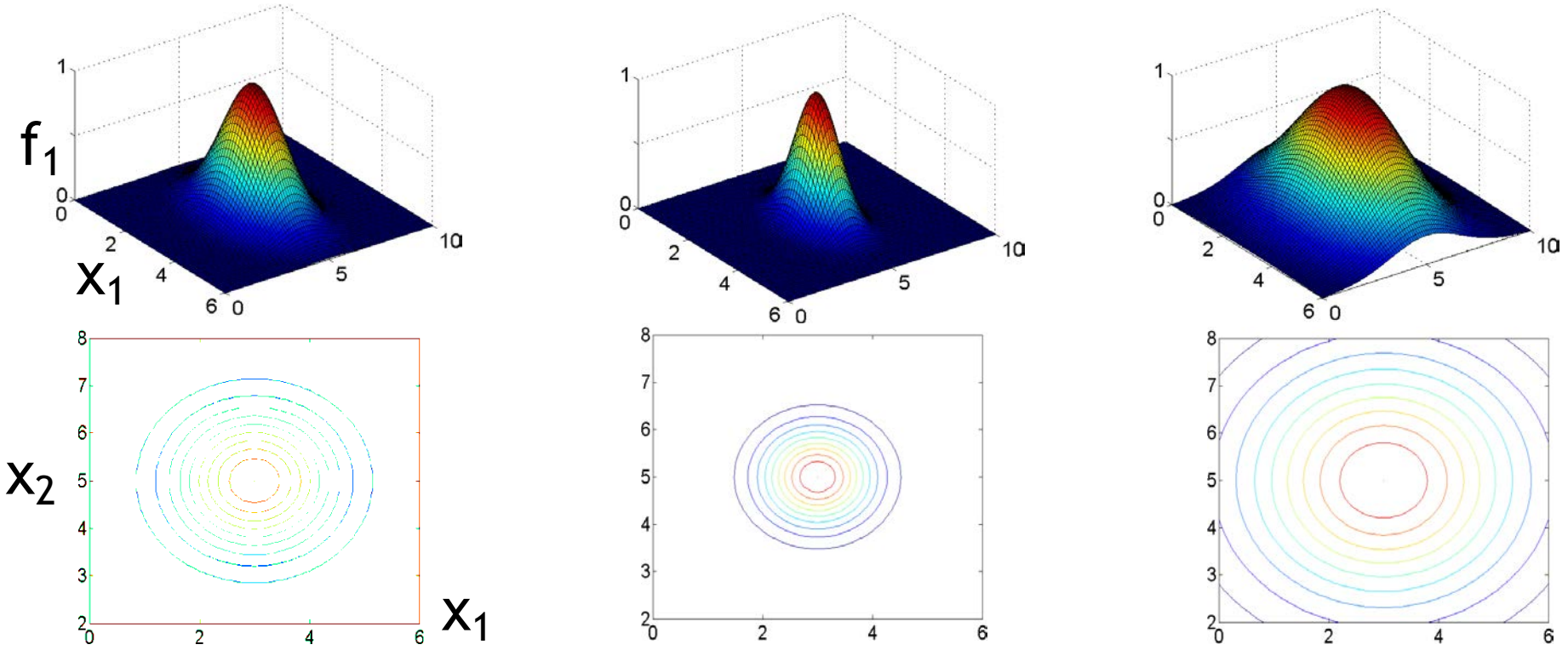
If x if far from $l^{(1)}$:

$$f_1 \approx exp\left(-\frac{(\text{large number})^2}{2\sigma^2}\right) \approx 0$$

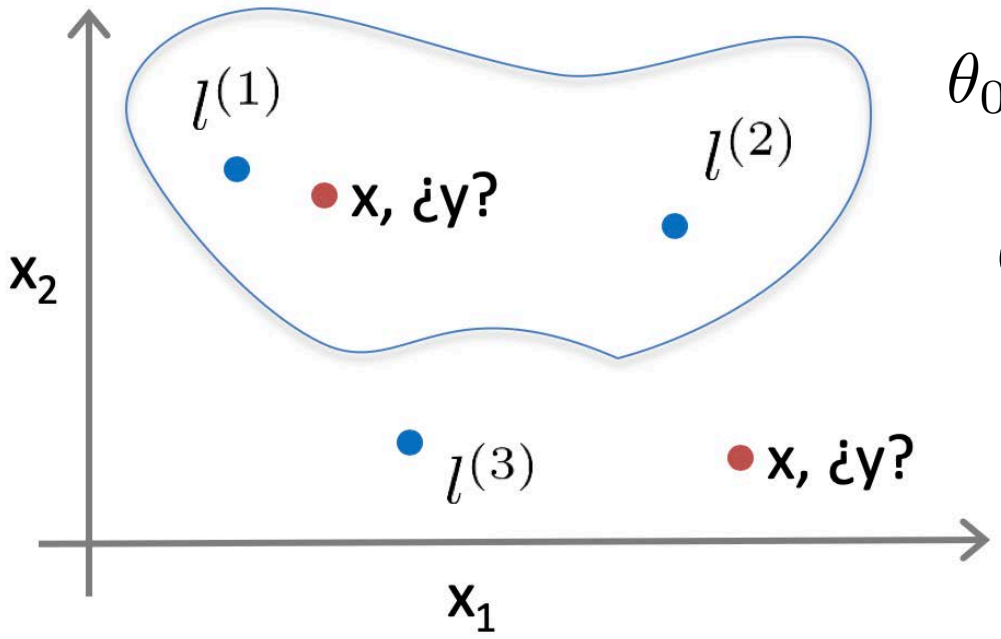
Example

$$l^{(1)} = \begin{bmatrix} 3 \\ 5 \end{bmatrix}, \quad f_1 = \exp\left(-\frac{\|x - l^{(1)}\|^2}{2\sigma^2}\right)$$

$\sigma^2 = 1$ $\sigma^2 = 0.5$ $\sigma^2 = 3$



Thus we get SVMs with non-linear decision boundaries



Predict "1" when

$$\theta_0 + \theta_1 f_1 + \theta_2 f_2 + \theta_3 f_3 \geq 0$$

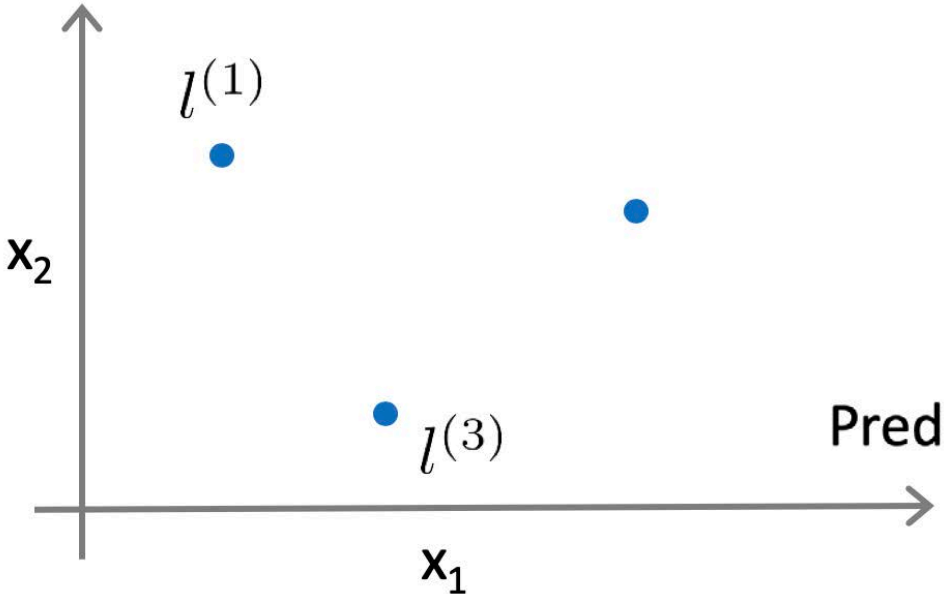
$$\theta_0 = -0.5, \theta_1 = 1, \theta_2 = 1, \theta_3 = 0$$

Predict 1 when x close to $l^{(1)}$ or $l^{(2)}$

Support vector machines

Kernels II

Choosing the landmarks

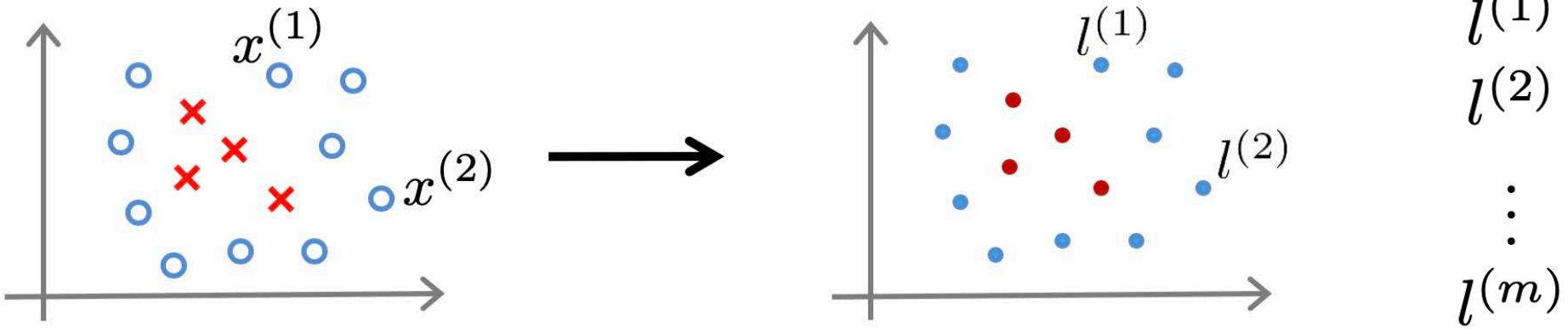


Given x:

$$f_i = \text{similarity}(x, l^{(i)})$$
$$= \exp\left(-\frac{||x - l^{(i)}||^2}{2\sigma^2}\right)$$

Predict $y = 1$ if $\theta_0 + \theta_1 f_1 + \theta_2 f_2 + \theta_3 f_3 \geq 0$

Where to get $l^{(1)}, l^{(2)}, l^{(3)}, \ldots$?



SVM with Kernels

Given $(x^{(1)}, y^{(1)}), (x^{(2)}, y^{(2)}), \ldots, (x^{(m)}, y^{(m)})$,
choose $l^{(1)} = x^{(1)}, l^{(2)} = x^{(2)}, \ldots, l^{(m)} = x^{(m)}$.

Given example x:

$$f_1 = \text{similarity}(x, l^{(1)})$$
$$f_2 = \text{similarity}(x, l^{(2)})$$
$$\ldots$$

For training example $(x^{(i)}, y^{(i)})$:

$x^{(i)}$

$$f_1^{(i)} = sim(x^{(i)}, l^{(1)})$$
$$f_2^{(i)} = sim(x^{(i)}, l^{(2)})$$
$$\vdots$$
$$f_i^{(i)} = sim(x^{(i)}, l^{(i)}) = exp\left(-\frac{0}{2\sigma^2}\right) = 1$$
$$\vdots$$
$$f_m^{(i)} = sim(x^{(i)}, l^{(m)})$$

$$f^{(i)} = \begin{bmatrix} f_0^{(i)} \\ f_1^{(i)} \\ f_2^{(i)} \\ \vdots \\ f_m^{(i)} \end{bmatrix}$$

$$f_0^{(i)} = 1$$

SVM with Kernels

Hypothesis: Given x, compute features $f \in \mathbb{R}^{m+1}$

Predict "y=1" if $\theta^T f \geq 0$

Training: *n=m*

$$\min_\theta C \sum_{i=1}^{m} y^{(i)} cost_1(\theta^T f^{(i)}) + (1 - y^{(i)}) cost_0(\theta^T f^{(i)}) + \frac{1}{2} \sum_{j=1}^{n} \theta_j^2$$

For efficiency reasons ('m' can be very big) most software packages minimize a modified version of this:

$$\sum_{j=1}^{n} \theta_j^2 = \theta^T \theta \quad \longrightarrow \quad \theta^T M \theta$$

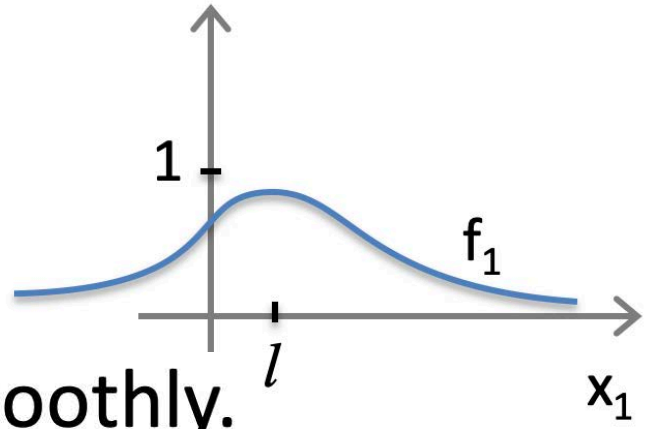
and do not return θ but a *model* that can be used to make predictions

SVM parameters

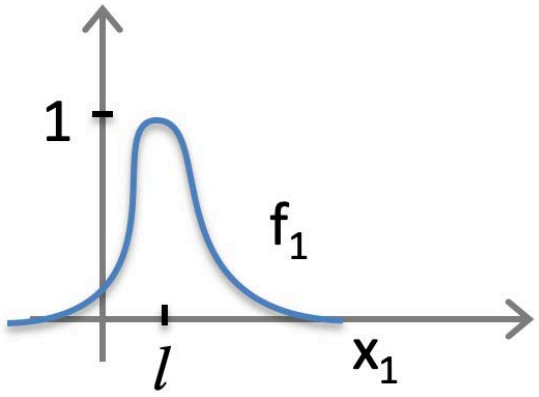
C ($= \frac{1}{\lambda}$). Large C (small λ): Lower bias, high variance.
Small C (large λ): Higher bias, low variance.

σ^2 Large σ^2: Features f_i vary more smoothly.
Higher bias, lower variance.

$$exp\left(-\frac{\|x - l^{(i)}\|^2}{2\sigma^2}\right)$$



Small σ^2: Features f_i vary less smoothly.
Lower bias, higher variance.



Support vector machines

Using an SVM

Use SVM software package (e.g. liblinear, libsvm, …) to solve for parameters θ

Need to specify:

Choice of parameter C.

Choice of kernel (similarity function):

E.g. No kernel (“linear kernel”) (useful when ‘n’ is large and ‘m’ is small)

Predict “y = 1” if $\theta^T x \geq 0$

Gaussian kernel:

$$f_i = \exp\left(-\frac{||x - l^{(i)}||^2}{2\sigma^2}\right)$$, where $l^{(i)} = x^{(i)}$

Need to choose σ^2

Some packages may ask you to provide the kernel function

```
def gaussianKernel(x1, x2, sigma):
```

$$f = \exp\left(-\frac{||\texttt{x1} - \texttt{x2}||^2}{2\sigma^2}\right)$$

```
return f
```

f: $f^{(i)}$

x1: $x^{(i)}$

x2: $l^{(j)} = x^{(j)}$

Note: Do perform feature scaling (if needed) before using the Gaussian kernel

$$\|x - l\|^2 = (x_1 - l_1)^2 + (x_2 - l_2)^2 + \cdots + (x_n - l_n)^2$$

without feature scaling large attributes will subsume smaller ones

Other choices of kernel

Note: Not all similarity functions $\mathrm{similarity}(x, l)$ make valid kernels.

(Need to satisfy technical condition called "Mercer's Theorem" to make sure SVM packages' optimizations run correctly, and do not diverge).

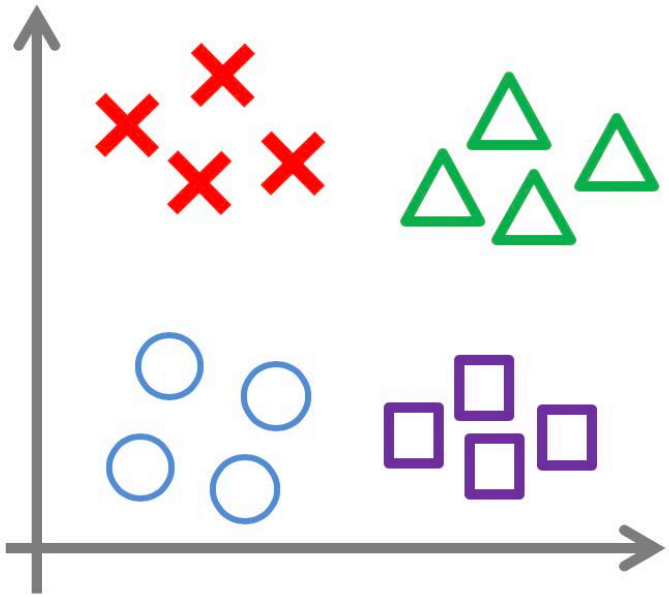
Many off-the-shelf kernels available:

- Polynomial kernel:

$$k(x, l) = (x^T l + \mathrm{constant})^{\mathrm{degree}}$$

- More esoteric: String kernel, chi-square kernel, histogram intersection kernel, …

Multi-class classification



$$y \in \{1, 2, 3, \ldots, K\}$$

Many SVM packages already have built-in multi-class classification functionality

Otherwise, use one-vs.-all method. (Train K SVMs, one to distinguish $y = i$ from the rest, for $i = 1, 2, \ldots, K$), get $\theta^{(1)}, \theta^{(2)}, \ldots, \theta^{(K)}$

Pick class i with largest $(\theta^{(i)})^T x$

Logistic regression vs. SVMs

$n =$number of features ($x \in \mathbb{R}^{n+1}$), $m =$ number of training examples

If n is large relative to m (e.g., n =10.000, m = 1.000):

Use logistic regression, or SVM without a kernel (“linear kernel”)

If n is small, m is intermediate (e.g., n=1 .. 1000, m=10 .. 10.000):

Use SVM with Gaussian kernel

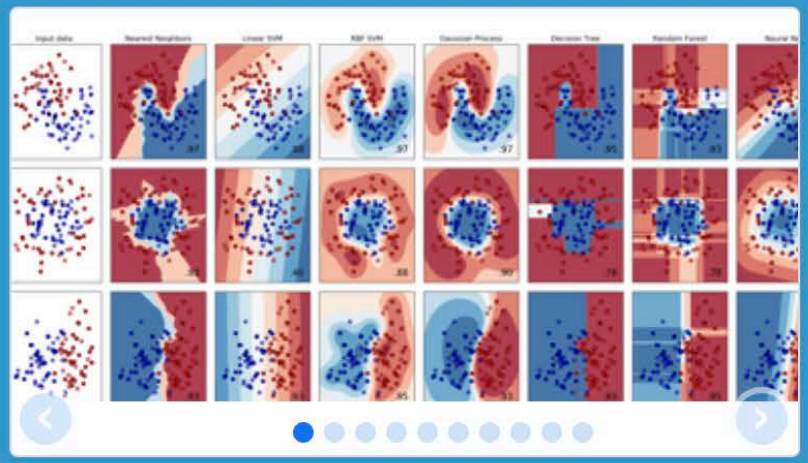
If n is small, m is large (e.g., n=1 .. 1000, m=50.000+):

Create/add more features, then use logistic regression or SVM without a kernel (Gaussian kernel too slow with m so large)

Neural network likely to work well for most of these settings, but may be slower to train

Support vector machines

scikit-learn



scikit-learn

Machine Learning in Python

- Simple and efficient tools for data mining and data analysis
- Accessible to everybody, and reusable in various contexts
- Built on NumPy, SciPy, and matplotlib
- Open source, commercially usable - BSD license

Classification

Identifying to which category an object belongs to.

Applications: Spam detection, Image recognition.
Algorithms: SVM, nearest neighbors, random forest, … — Examples

Regression

Predicting a continuous-valued attribute associated with an object.

Applications: Drug response, Stock prices.
Algorithms: SVR, ridge regression, Lasso, … — Examples

Clustering

Automatic grouping of similar objects into sets.

Applications: Customer segmentation, Grouping experiment outcomes
Algorithms: k-Means, spectral clustering, mean-shift, … — Examples

Dimensionality reduction

Reducing the number of random variables to consider.

Applications: Visualization, Increased efficiency
Algorithms: PCA, feature selection, non-negative matrix factorization. — Examples

Model selection

Comparing, validating and choosing parameters and models.

Goal: Improved accuracy via parameter tuning
Modules: grid search, cross validation, metrics. — Examples

Preprocessing

Feature extraction and normalization.

Application: Transforming input data such as text for use with machine learning algorithms.
Modules: preprocessing, feature extraction. — Examples

scikit-learn comes with a few standard datasets, for instance the **iris** and **digits** datasets for classification and the **boston house prices** dataset for regression

```
$ python
>>> from sklearn import datasets
>>> iris = datasets.load_iris()
>>> digits = datasets.load_digits()
```

```
>>> print(digits.data)
[[ 0.   0.   5. ...   0.   0.   0.]
 [ 0.   0.   0. ...  10.   0.   0.]
 [ 0.   0.   0. ...  16.   9.   0.]
 ...
 [ 0.   0.   1. ...   6.   0.   0.]
 [ 0.   0.   2. ...  12.   0.   0.]
 [ 0.   0.  10. ...  12.   1.   0.]]
```

```
>>> digits.target
array([0, 1, 2, ..., 8, 9, 8])
```

In scikit-learn, an estimator for classification is a Python object that implements the methods *fit(X, y)* and *predict(T)*

An example of an estimator is the class *sklearn.svm.SVC*, which implements support vector classification. The estimator's constructor takes as arguments the model's parameters.

```
>>> from sklearn import svm
>>> clf = svm.SVC(gamma=0.001, C=100.)
```

The *clf* (for classifier) estimator instance is first fitted to the model; that is, it must learn from the model. This is done by passing our training set to the fit method. For the training set, we'll use all the images from our dataset, except for the last image, which we'll reserve for our predicting

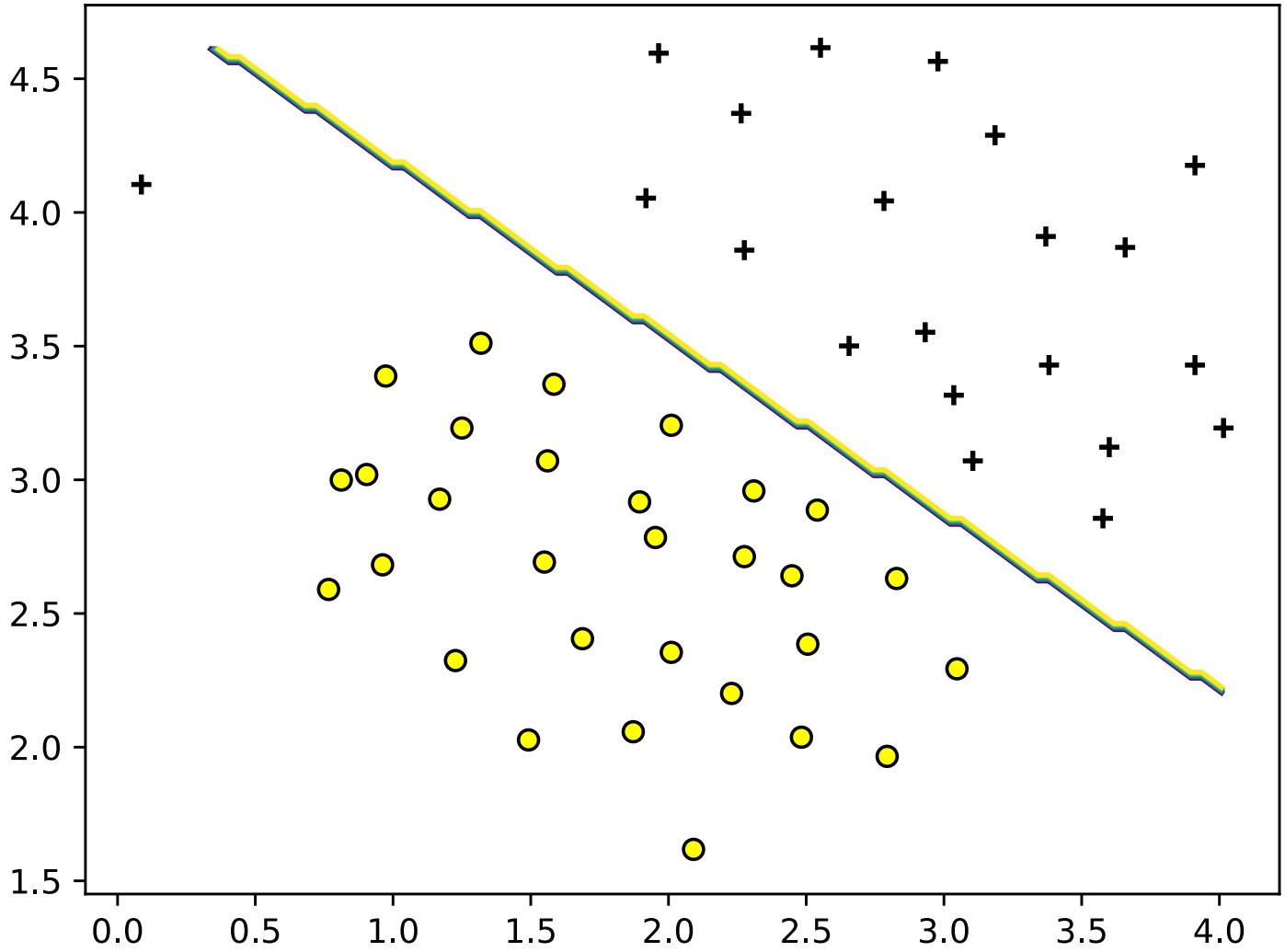
```
>>> clf.fit(digits.data[:-1], digits.target[:-1])
SVC(C=100.0, cache_size=200, class_weight=None, coef0=0.0,
  decision_function_shape='ovr', degree=3, gamma=0.001, kernel='rbf',
  max_iter=-1, probability=False, random_state=None, shrinking=True,
  tol=0.001, verbose=False)

>>> clf.predict(digits.data[-1:])
array([8])
```

```
svm = SVC(kernel='linear', C=100)
svm.fit(X, y.ravel())
x1 = np.linspace(X[:, 0].min(), X[:, 0].max(), 100)
x2 = np.linspace(X[:, 1].min(), X[:, 1].max(), 100)
x1, x2 = np.meshgrid(x1, x2)
yp = svm.predict(np.array([x1.ravel(),x2.ravel()]).T).reshape(x1.shape)

pos = (y == 1).ravel()
neg = (y == 0).ravel()

plt.figure()
plt.scatter(X[pos, 0], X[pos, 1], color='black', marker='+')
plt.scatter(X[neg, 0], X[neg, 1], color='yellow',
            edgecolors='black', marker='o')
plt.contour(x1, x2, yp)
```


4.5
4.0
3.5
3.0
2.5
2.0
1.5
0.0
0.5
1.0
1.5
2.0
2.5
3.0
3.5
4.0

class `sklearn.svm.` **SVC** (*C=1.0*, *kernel='rbf'*, *degree=3*, *gamma='auto_deprecated'*, *coef0=0.0*, *shrinking=True*, *probability=False*, *tol=0.001*, *cache_size=200*, *class_weight=None*, *verbose=False*, *max_iter=-1*, *decision_function_shape='ovr'*, *random_state=None*) ¶

Parameters:

C : *float, optional (default=1.0)*

Penalty parameter C of the error term.

kernel : *string, optional (default='rbf')*

Specifies the kernel type to be used in the algorithm. It must be one of 'linear', 'poly', 'rbf', 'sigmoid', 'precomputed' or a callable. If none is given, 'rbf' will be used. If a callable is given it is used to pre-compute the kernel matrix from data matrices; that matrix should be an array of shape `(n_samples, n_samples)`.

degree : *int, optional (default=3)*

Degree of the polynomial kernel function ('poly'). Ignored by all other kernels.

gamma : *float, optional (default='auto')*

Kernel coefficient for 'rbf', 'poly' and 'sigmoid'.

Current default is 'auto' which uses 1 / n_features, if `gamma='scale'` is passed then it uses 1 / (n_features * X.std()) as value of gamma. The current default of gamma, 'auto', will change to 'scale' in version 0.22. 'auto_deprecated', a deprecated version of 'auto' is used as a default indicating that no explicit value of gamma was passed.