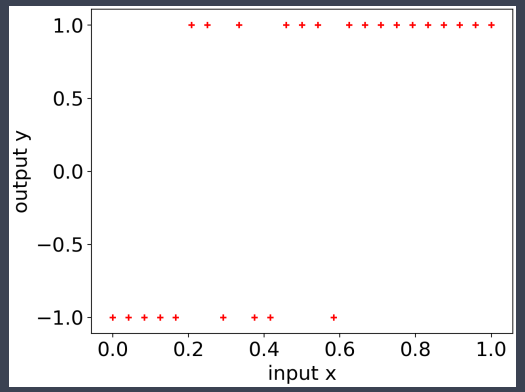
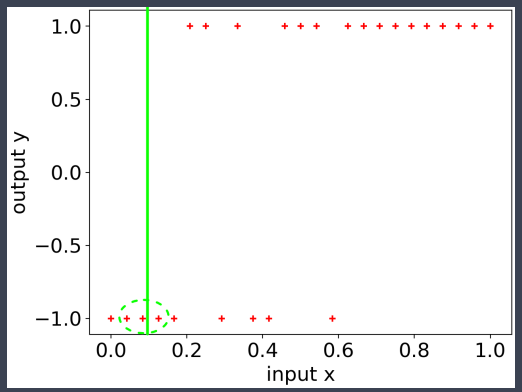
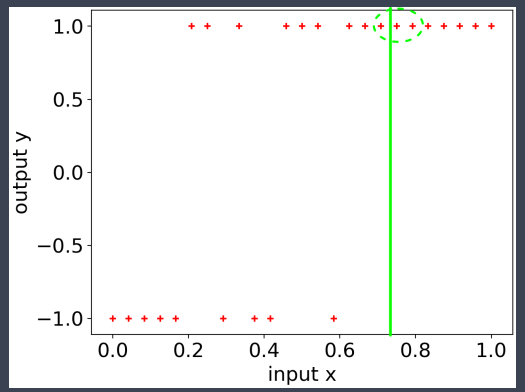
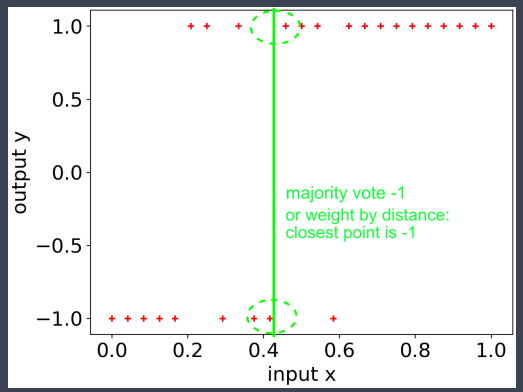
» Using Training Data As Features



training data



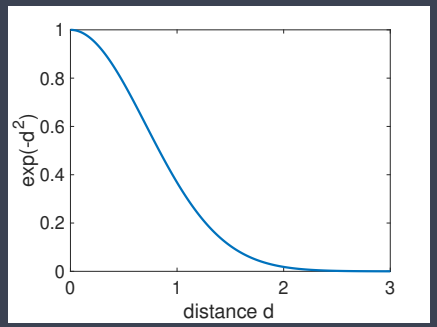




* We want to attach more weight to training data points that are close to input *x* and less weight to far away training points.

» k-Nearest Neighbour (kNN) Model

* Training data $(x^{(i)}, y^{(i)})$, $i = 1, 2, \ldots, m$
* Given feature vector x:
  1. For each training data point i calculate the distance $d(x^{(i)}, x)$ between feature vector $x^{(i)}$ and x
  2. Select the k training data points that are closest to x ... the k nearest neighbours
  3. Predict output y using the outputs $y^{(i)}$ for these k closest training points.
     * Weighting of neighbour points. E.g. *Gaussian* $K(x^{(i)}, x) = e^{-\gamma d(x^{(i)}, x)^2}$ (attach less weight to training points that are further away from query point x).



     * Regression: weighted mean

$$\hat{y} = \frac{\sum_{i \in N_k} K(x^{(i)}, x) y^{(i)}}{\sum_{i \in N_k} K(x^{(i)}, x)} = \sum_{i \in N_k} K(x^{(i)}, x) y^{(i)} \alpha^{(i)}, \text{ with}$$

$$\alpha^{(i)} = 1 / \sum_{i \in N_k} K(x^{(i)}, x)$$

     * Classification:

$$\hat{y} = sign\left(\frac{\sum_{i \in N_k} K(x^{(i)}, x) y^{(i)}}{\sum_{i \in N_k} K(x^{(i)}, x)}\right) = sign\left(\sum_{i \in N_k} K(x^{(i)}, x) y^{(i)} \alpha^{(i)}\right).$$

» Kernalising Linear Models: Using Training Data As Features

kNN:

* Regression: weighted mean $\hat{y} = \sum_{i \in N_k} K(x^{(i)}, x) y^{(i)} \alpha^{(i)}$
* Classification: $\hat{y} = sign(\sum_{i \in N_k} K(x^{(i)}, x) y^{(i)} \alpha^{(i)})$.

Idea: (i) use all data points (choose $k = m$ in kNN) and (ii) make coefficients $y^{(i)} \alpha^{(i)}$ into model parameters

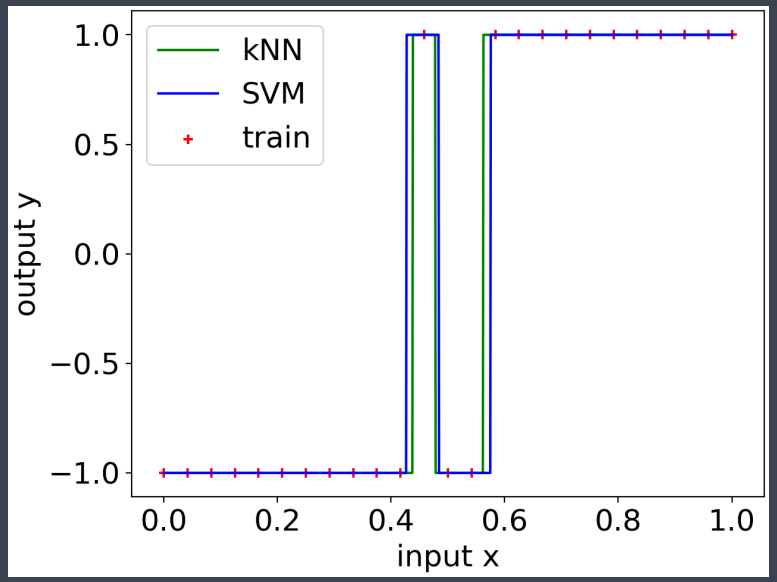
* Model:
  $\hat{y} = sign\left(\theta_0 + \theta_1 K(x^{(1)}, x) + \theta_2 K(x^{(2)}, x) + \cdots + \theta_m K(x^{(m)}, x)\right)$
* Now can learn parameters $\theta_0, \theta_1, \ldots$ by selecting them to minimise a cost function e.g. logistic regression or SVM cost function.
* Can do same thing for regression problems, model is then
  $\hat{y} = \theta_0 + \theta_1 K(x^{(1)}, x) + \theta_2 K(x^{(2)}, x) + \cdots + \theta_m K(x^{(m)}, x)$
* $K(x^{(i)}, x)$ is referred to as a *kernel*

» Kernalising Linear Models: Using Training Data As Features

Note:

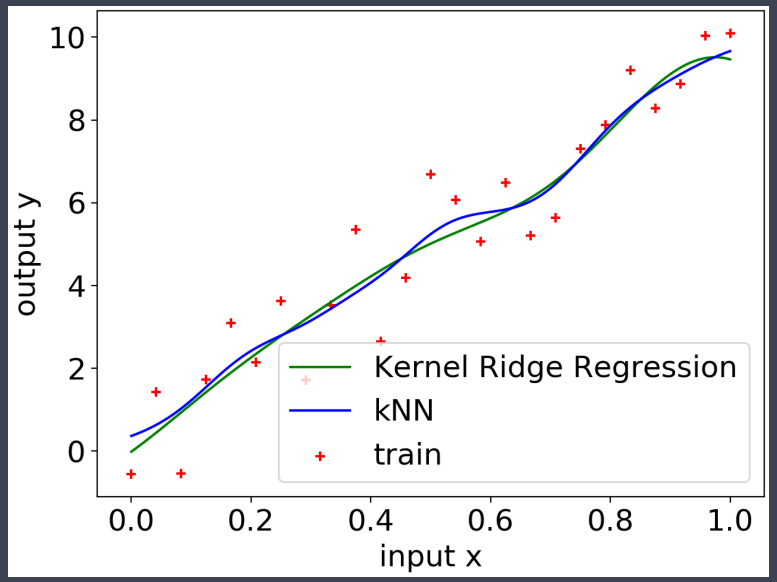
- Sometimes write model as
  $\hat{y} = \theta_0 + \theta_1 y^{(1)} K(x^{(1)}, x) + \theta_2 y^{(2)} K(x^{(2)}, x) + \cdots + \theta_m y^{(m)} K(x^{(m)}, x)$
  for consistency with original SVM formulation ... will come back to this later
- Since we can freely choose the θ_i's, this makes no difference.

» Classification example



- Kernalised SVM: 1) $\gamma = 50$, L_2 penalty weight $C = 1$
- kNN model: 1) Euclidean distance, 2) (i) $k = m$, 3) gaussian weights, 4) sign(weighted average)
- SVM and kNN predictions are not identical, but much the same.
- *Note: No kernalised version of logistic regression available in sklearn currently. Its easy to implement one e.g. see https://people.cs.umass.edu/~sheldon/teaching/cs335/lec/12-demo.html.*

» Regression example



- Kernalised Ridge Regression: 1) $\gamma = 10$, L_2 penalty weight $C = 10$
- *k*NN model: 1) Euclidean distance, 2) $k = m$, 3) gaussian weights, 4) weighted average
- kernel and *k*NN predictions are not identical, but much the same.

» Classification Example Python Code

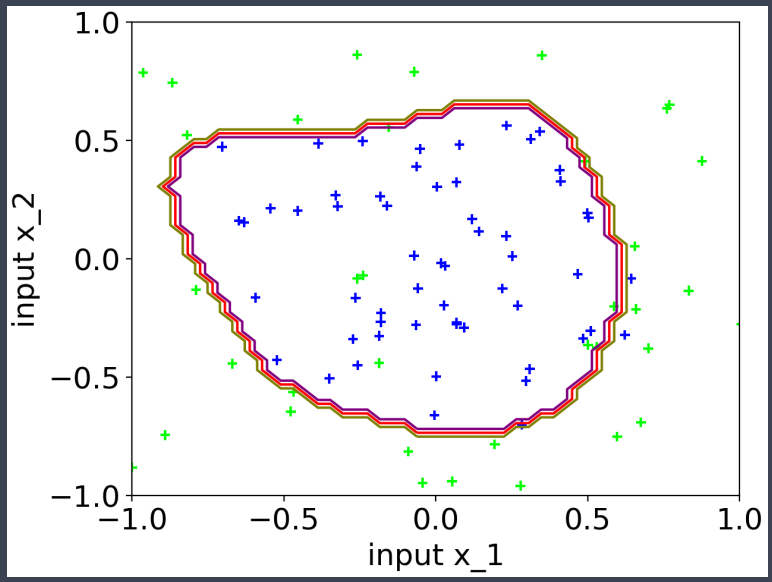
```
import numpy as np
m = 25
Xtrain = np.linspace(0.0,1.0,num=m)
ytrain = np.sign(Xtrain-0.5+np.random.normal(0,0.2,m))
Xtrain = Xtrain.reshape(-1, 1)

def gaussian_kernel(distances):
        weights = np.exp(-100*(distances**2))
        return weights

from sklearn.neighbors import KNeighborsClassifier
model = KNeighborsClassifier(n_neighbors=25,weights=gaussian_kernel).fit(Xtrain, ytrain)

Xtest=np.linspace(0.0,1.0,num=1000).reshape(-1, 1)
ypred = model.predict(Xtest)
import matplotlib.pyplot as plt
plt.rc('font', size=18); plt.rcParams['figure.constrained_layout.use'] = True
plt.scatter(Xtrain, ytrain, color='red', marker='+')
plt.plot(Xtest, ypred, color='green')

from sklearn.svm import SVC
model = SVC(C=1000, kernel='rbf', gamma=50).fit(Xtrain, ytrain)
ypred = model.predict(Xtest)
plt.plot(Xtest, ypred, color='blue')

plt.xlabel("input x"); plt.ylabel("output y")
plt.legend(["kNN","SVM","train"])
plt.show()
```

» Regression Example Python Code

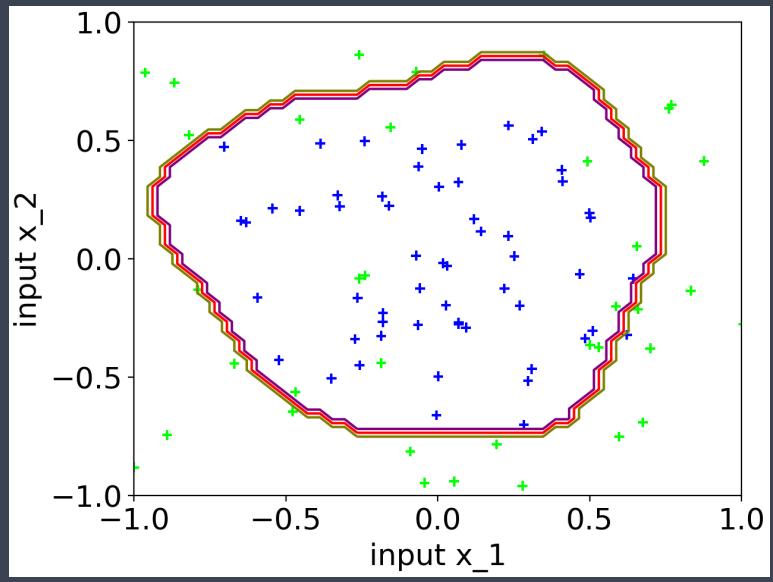
```
import numpy as np
m = 25
Xtrain = np.linspace(0.0,1.0,num=m)
ytrain = 10*Xtrain + np.random.normal(0.0,1.0,m)
Xtrain = Xtrain.reshape(-1, 1)
from sklearn.kernel_ridge import KernelRidge
C=10;
model = KernelRidge(alpha=1.0/C, kernel='rbf', gamma=10).fit(Xtrain, ytrain)

Xtest=np.linspace(0.0,1.0,num=1000).reshape(-1, 1)
ypred = model.predict(Xtest)

def gaussian_kernel(distances):
        weights = np.exp(-100*(distances**2))
        return weights

from sklearn.neighbors import KNeighborsRegressor
model2 = KNeighborsRegressor(n_neighbors=m,weights=gaussian_kernel).fit(Xtrain, ytrain)
ypred2 = model2.predict(Xtest)

import matplotlib.pyplot as plt
plt.rc('font', size=18); plt.rcParams['figure.constrained_layout.use'] = True
plt.scatter(Xtrain, ytrain, color='red', marker='+')
plt.plot(Xtest, ypred, color='green')
plt.plot(Xtest, ypred2, color='blue')
plt.xlabel("input x"); plt.ylabel("output y")
plt.legend(["Kernel Ridge Regression","kNN","train"])
plt.show()
```

» Another Classification example

When points from one class are clumped together then using training data as features can work nicely:



SVM

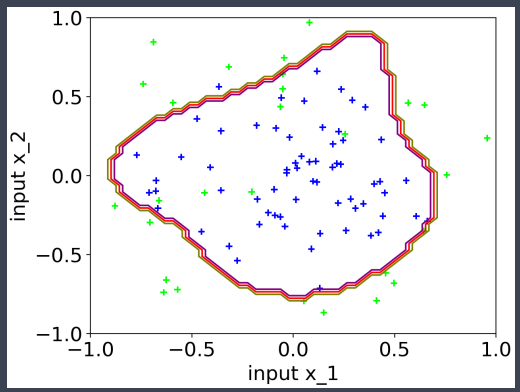


*k*NN

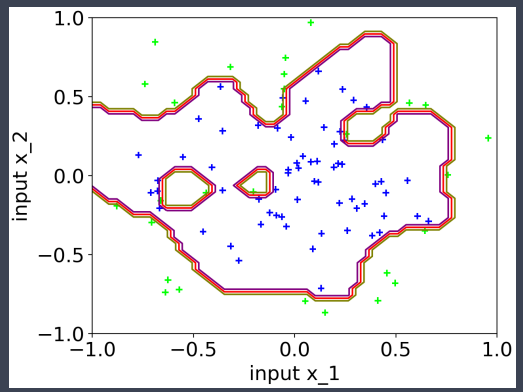
- Kernalised SVM: 1) $\gamma = 1$, L_2 penalty weight $C = 1$
- *k*NN model: 1) Euclidean distance, 2) (i) $k = m$, 3) gaussian weights, 4) sign(weighted average)

» Another Classification example (cont)

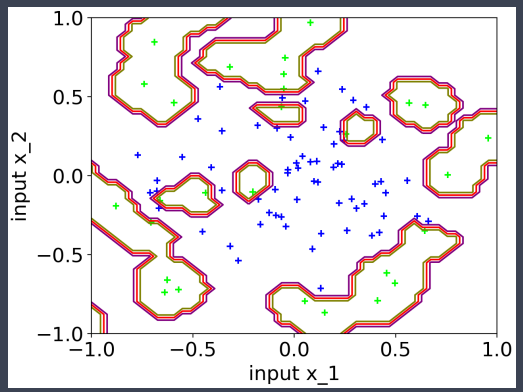
Impact of Gaussian kernel parameter γ:



$\gamma = 2$

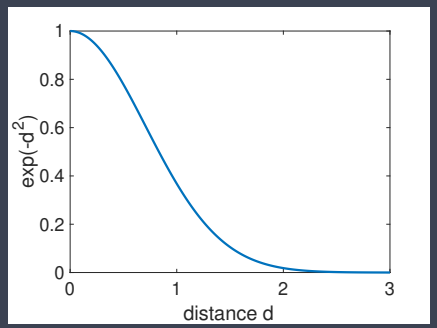


$\gamma = 10$

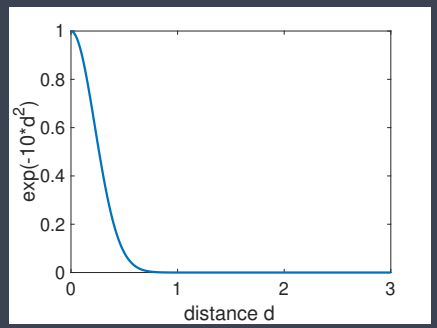


$\gamma = 100$

* As γ increases the kernel decreases more quickly with distance. This makes the predictions tend to be less smooth and to just snap to the nearest training point



$\gamma = 1$



$\gamma = 10$

* Use γ to manage trade-off between under-fitting and over-fitting

» Circle Example Python Code

```
import numpy as np
m = 100
Xtrain = 0.5*np.random.randn(m,2)
ytrain = np.sign((Xtrain[:,0]**2+Xtrain[:,1]**2)-0.5+np.random.normal(0,0.2,m))

import matplotlib.pyplot as plt
plt.rc('font', size=18); plt.rcParams['figure.constrained_layout.use'] = True

xx,yy = np.meshgrid(np.linspace(-1, 1, 50),np.linspace(-1, 1, 50))
Xtest = np.c_[xx.ravel(), yy.ravel()]
ytest = np.sign((xx**2+yy**2)-0.5)

from sklearn.svm import SVC
model = SVC(C=1000, kernel='rbf', gamma=1).fit(Xtrain, ytrain)
ypred = model.predict(Xtest)
plt.contour(xx,yy, ypred.reshape(xx.shape), c=ypred,cmap=plt.cm.brg, levels=2)
#plt.scatter(xx,yy,marker='.',c=ypred.reshape(xx.shape),cmap=plt.cm.brg)
plt.scatter(Xtrain[:,0],Xtrain[:,1],marker='+',c=ytrain,cmap=plt.cm.brg)
plt.xlim((-1,1)); plt.ylim((-1,1))
plt.xlabel("input x_1"); plt.ylabel("input x_2")
plt.show()

def gaussian_kernel(distances):
        weights = np.exp(-10*(distances**2))
        return weights

from sklearn.neighbors import KNeighborsClassifier
model = KNeighborsClassifier(n_neighbors=m,weights=gaussian_kernel).fit(Xtrain, ytrain)
ypred = model.predict(Xtest)
plt.contour(xx,yy, ypred.reshape(xx.shape), c=ypred,cmap=plt.cm.brg, levels=2)
plt.scatter(Xtrain[:,0],Xtrain[:,1],marker='+',c=ytrain,cmap=plt.cm.brg)
plt.xlim((-1,1)); plt.ylim((-1,1))
plt.xlabel("input x_1"); plt.ylabel("input x_2")
plt.show()
```

» Kernel Logistic Regression

* Hypothesis: $sign(\sum_{j=1}^{m} \theta_j K(x, x^{(j)}))$
* Cost: $\frac{1}{m} \sum_{i=1}^{m} \log(1 + e^{-y^{(i)} \sum_{j=1}^{m} \theta_j K(x^{(i)}, x^{(j)})})$
* Select θ to minimise cost function. Select γ (kernel parameter) using cross-validation.

» Kernalised Regression

- Hypothesis: $\hat{y} = \sum_{j=1}^{m} \theta_j K(x, x^{(j)})$
- Cost: $\frac{1}{m} \sum_{i=1}^{m} (y^{(i)} - \sum_{j=1}^{m} \theta_j K(x^{(i)}, x^{(j)}))^2$
- Select θ to minimise cost function. Select γ (kernel parameter) using cross-validation.

» How To Add $L2$ Regularisation?

For regression could use:

* $\frac{1}{m}\sum_{i=1}^{m}(y^{(i)} - \sum_{j=1}^{m}\theta_j K(x^{(i)}, x^{(j)}))^2 + \lambda\theta^T\theta$

but that's not the normal way. Instead we use

* $\frac{1}{m}\sum_{i=1}^{m}(y^{(i)} - \sum_{j=1}^{m}\theta_j K(x^{(i)}, x^{(j)}))^2 + \lambda\sum_{i=1}^{m}\sum_{j=1}^{m}\theta_i K(x^{(i)}, x^{(j)})\theta_j$

This is called kernel ridge regression. Similarly for kernelised logistic regression with regularisation.

Notation: recall $\theta^T\theta = \sum_{i=1}^{m}\sum_{j=1}^{m}\theta_i\theta_j$ so all we've done is add a weighting $K(x^{(i)}, x^{(j)})$. In matrix notation this is

* $\frac{1}{m}\sum_{i=1}^{m}(y^{(i)} - \sum_{j=1}^{m}\theta_j K(x^{(i)}, x^{(j)}))^2 + \lambda\theta^T M\theta$

where the i,j'th element of matrix M is $K(x^{(i)}, x^{(j)})$.

» How To Add $L2$ Regularisation? [Optional]

Why use weighted regularisation? Here's one reason:

- Start with linear model $\theta^T x$
- Write $K(x, x^{(j)}) = \phi(x^{(j)})^T \phi(x) \rightarrow$ can't do this for all weight functions K, need to restrict ourselves to ones where we can.
- Replace x by $\phi(x)$ and define $\theta = \sum_{j=1}^{m} \alpha_j \phi(x^{(j)})$
- Then

$$\theta^T \phi(x) = \sum_{j=1}^{m} \alpha_j \phi(x^{(j)})^T \phi(x) = \sum_{j=1}^{m} \alpha_j K(x, x^{(j)})$$

$$\theta^T \theta = \sum_{j=1}^{m} \alpha_j \phi(x^{(j)})^T \sum_{i=1}^{m} \alpha_i \phi(x^{(i)})$$

$$= \sum_{i=1}^{m} \sum_{j=1}^{m} \alpha_j K(x^{(j)}, x^{(i)}) \alpha_i = \alpha^T M \alpha$$

- Note: refer here to parameters in kernelised model as α_j rather than θ_j.

So can view kernel model as replacing features x by new features $\phi(x)$ and modifying the parameters.

» How To Add $L2$ Regularisation? [Optional]

Why use weighted regularisation? Here's another reason:

- For weights that can be written as $K(x, x^{(j)}) = \phi(x^{(j)})^T \phi(x)$ then functions of the form $\sum_{j=1}^{m} \theta_j K(x^{(j)}, x)$ belong to a Reproducing Kernel Hilbert Space (RKHS)
- The "size" of a function in an RKHS is measured the Hilbert norm $\theta^T M \theta$ and so this is a natural penalty to use for regularising
- You'll see lots of mentioins of RKHS's if you search internet for kernel methods, but we won't go further into this here.

» Kernel SVMs[1]

- Write $K(x, x^{(j)}) = \phi(x^{(j)})^T \phi(x)$
- Replace x by $\phi(x)$ and define $\theta = \sum_{j=1}^{m} \alpha_j y^{(j)} \phi(x^{(j)})$ (note use of $y^{(j)}$)
- Hypothesis: $sign(\sum_{j=1}^{m} \alpha_j y^{(j)} K(x, x^{(j)}))$
- Cost: $\frac{1}{m} \sum_{i=1}^{m} \max(0, 1 - y^{(i)} \sum_{j=1}^{m} \alpha_j y^{(j)} K(x^{(i)}, x^{(j)})) + \lambda \alpha^T M \alpha$ where M is matrix with $M_{ij} = y^{(j)} K(x^{(j)}, x^{(i)}) y^{(i)}$

[1] Training a Support Vector Machine in the Primal. Olivier Chapelle, Neural Computation 2007

» Kernel Summary

- Easy to use $\rightarrow$ hyperparameters are kernel parameter γ and L_2 penalty weight C. Also need to choose kernel, but Gaussian usually works well.
- Essentially an enhanced form of kNN model, so shares many of the same characteristics
- Small data only $\rightarrow$ as training data increases kernel approaches tend to become expensive/slow.
- SVMs:
  - Often online "SVM" is used to mean "kernel SVM", so can get confusing. Often you'll also be told that SVM is better than logistic regression etc without further explanation
  - Its important to keep clearly in mind that *two* tools are usually being conflated here: (i) use of kernels and (ii) use of SVMs. Its use of kernels that's key – its a powerful approach but kernels can be applied with any linear model not just SVMs. Kernel logistic regression and kernel SVMs typically have similar performance