Training data \((x^{(i)}, y^{(i)}), \ i = 1, 2, \ldots, m\)

Idea: associate a feature with each training data point and then use linear model...

Feature \(i\): function \(y^{(i)}K(x^{(i)}, x)\) outputting a real number for input \(x\). \(K(x^{(i)}, x)\) measures the distance between input \(x\) and training point \(x^{(i)}\). \(K(x^{(i)}, x)\) is referred to as a *kernel*

Model: \(\hat{y} = \text{sign} (\theta_0 + \theta_1 y^{(1)}K(x^{(1)}, x) + \theta_2 y^{(2)}K(x^{(2)}, x) + \cdots + \theta_m y^{(m)}K(x^{(m)}, x))\)

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 \(\hat{y} = \theta_0 + \theta_1 y^{(1)}K(x^{(1)}, x) + \theta_2 y^{(2)}K(x^{(2)}, x) + \cdots + \theta_m y^{(m)}K(x^{(m)}, x)\)
Remind you of a \( k \)NN Model? I hope so ...

- We want to attach more weight to training data points that are close to input \( x \) and less weight to far away training points.
- So \( K(x^{(i)}, x) \) should be about 1 when distance between \( x^{(i)} \) and \( x \) is small, falling to 0 as distance grows.
Using Training Data As Features

- Model: \( \hat{y} = \text{sign}(\theta_0 + \theta_1 y^{(1)} K(x^{(1)}, x) + \theta_2 y^{(2)} K(x^{(2)}, x) + \cdots + \theta_m y^{(m)} K(x^{(m)}, x) \)

- Gaussian kernel \( K(x^{(i)}, x) = e^{-\gamma d(x^{(i)}, x)^2} \)

- Parameter \( \gamma \) controls how quickly \( K(x^{(i)}, x) \) decreases as distance between \( x^{(i)} \) and \( x \) grows.

Choose \( \gamma \) using cross-validation.

- Now train parameters \( \theta \) to improve on basic fit to training data provided by kernel.

  - This is like a \( k \)NN with \( k = m \) (all of training data) and enhanced by addition of parameters \( \theta \) that provide extra flexibility to tune model.
  
  - Another way to think about it is that in \( k \)NN model the parameters \( \theta \) change with the input \( x \), i.e. \( \theta_i = 1 \) for training points \( i \in N_k \) and \( \theta_i = 0 \) for \( i \notin N_k \) (\( N_k \) is the set of \( k \) points closest to \( x \)).
Classification example

- Kernalised SVM: 1) $\gamma = 50$, $L_2$ penalty weight $C = 1$
- $k$NN model: 1) Euclidean distance, 2) (i) $k = m$, 3) gaussian weights, 4) sign(weighted average)
- SVM and $k$NN predictions are not identical, but much the same.
- Note: No kernalised version of logistic regression available in sklearn currently. Its certainly possible to implement one but SVM lends itself to more efficient kernelised implementation than logistic regression.
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.
```python
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/np.sum(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)

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

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

- 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 $\gamma$:

As $\gamma$ 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.

Use $\gamma$ to manage trade-off between under-fitting and over-fitting.
```python
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/np.sum(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()
```
Note:

* $L_2$ regularisation $\sum_{i=1}^{m} \theta_i^2 = \theta^T\theta$
* With kernelised models number of parameters $\theta$ is same as size of training data, so often quite large
* SVM implementation used weighted penalty $\theta^T M \theta$ where $M$ is a weighting matrix → improves computational performance

In more detail ...
Kernel SVMs

Linear model: $\text{sign}(\theta^T x)$. First try at kernelising:

* We’ll refer to parameters in kernelised model as $\alpha_i$ rather than $\theta_i$.

* Replace $\theta^T x$ with $\sum_{j=1}^{m} \alpha_j y^{(j)} K(x, x^{(j)})$

* Hypothesis: $\text{sign}(\sum_{j=1}^{m} \alpha_j y^{(j)} K(x, x^{(j)}))$

* Cost: $\frac{1}{m} \sum_{i=1}^{m} \max(0, 1 - y^{(i)} \sum_{j=1}^{m} \alpha_j y^{(j)} K(x^{(i)}, x^{(j)})) + \lambda \theta^T \theta$

* What about $\theta^T \theta$ term? We’d like cost to be only in terms of $\alpha$

---

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

Second try at kernalising:

* 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 $\theta = \sum_{j=1}^m \alpha_j y^{(j)} \phi(x^{(j)})$
* Then

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

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

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

where $M$ is matrix with $M_{ij} = y^{(j)} K(x^{(j)}, x^{(i)}) y^{(i)}$ and $\alpha$ is parameter vector.

* Cost: $\frac{1}{m} \sum_{i=1}^m \max(0, 1 - y^{(i)} \sum_{j=1}^m \alpha_j y^{(j)} K(x^{(i)}, x^{(j)})) + \lambda \alpha^T M \alpha$
* Now everything is in terms of the new parameters $\alpha$. 

[13/16]
Kernel Logistic Regression [Optional]

* Replace $\theta^T x$ with $\sum_{j=1}^m \alpha_j y^{(j)} K(x, x^{(j)})$
* Hypothesis: $\text{sign}(\sum_{j=1}^m \alpha_j y^{(j)} K(x, x^{(j)}))$
* Cost: $\frac{1}{m} \sum_{i=1}^m \log(1 + e^{-y^{(i)}} \sum_{j=1}^m \alpha_j y^{(j)} K(x^{(i)}, x^{(j)}))$
Replace $\theta^T x$ with $\sum_{j=1}^{m} \alpha_j y^{(i)} K(x, x^{(j)})$

Use $\theta^T \theta = \alpha^T M \alpha$ where $M$ is matrix with $M_{ij} = y^{(j)} K(x^{(j)}, x^{(i)}) y^{(i)}$ and $\alpha$ is parameter vector.

Hypothesis: $\sum_{j=1}^{m} \alpha_j y^{(i)} K(x, x^{(j)})$

Cost: $\frac{1}{m} \sum_{i=1}^{m} (y^{(i)} - \sum_{j=1}^{m} \alpha_j y^{(i)} K(x^{(i)}, x^{(j)}))^2 + \lambda \alpha^T M \alpha$
Kernel Summary

* Easy to use → hyperparameters are kernel parameter $\gamma$ and $L_2$ penalty weight $C$. Also need to choose kernel, but Gaussian usually works well.
* Essentially an enhanced form of $k$NN model, so shares many of the same characteristics
* Small data only → as training data increases kernel approaches tend to become expensive/slow.
* Efficient kernel SVM implementations:
  * 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