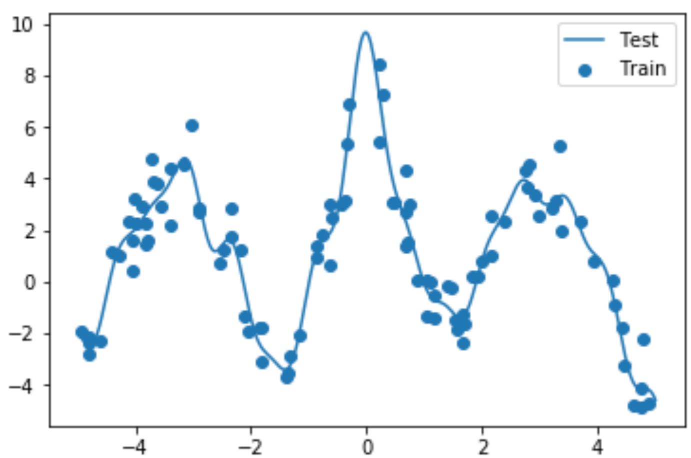
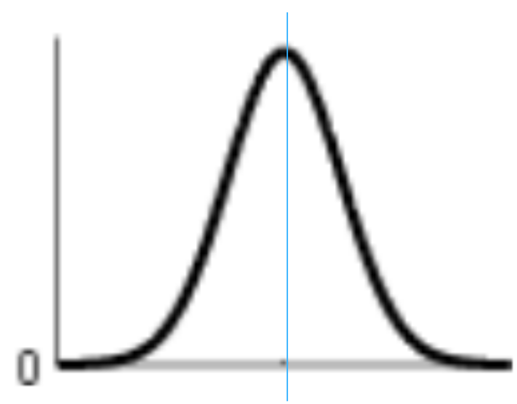
Tufts COMP 135: Introduction to Machine Learning

https://www.cs.tufts.edu/comp/135/2019s/

Kernel Methods

for regression and classification





Prof. Mike Hughes

Many ideas/slides attributable to:
Dan Sheldon (U.Mass.)
James, Witten, Hastie, Tibshirani (ISL/ESL books)

Objectives for Day 19: Kernels

Big idea: Use kernel functions (similarity function with special properties) to obtain flexible high-dimensional feature transformations without explicit features

- From linear regression (LR) to kernelized LR
- What is a kernel function?
  - Basic properties
  - Example: Polynomial kernel
  - Example: Squared Exponential kernel
- Kernels for classification
  - Logistic Regression
  - SVMs

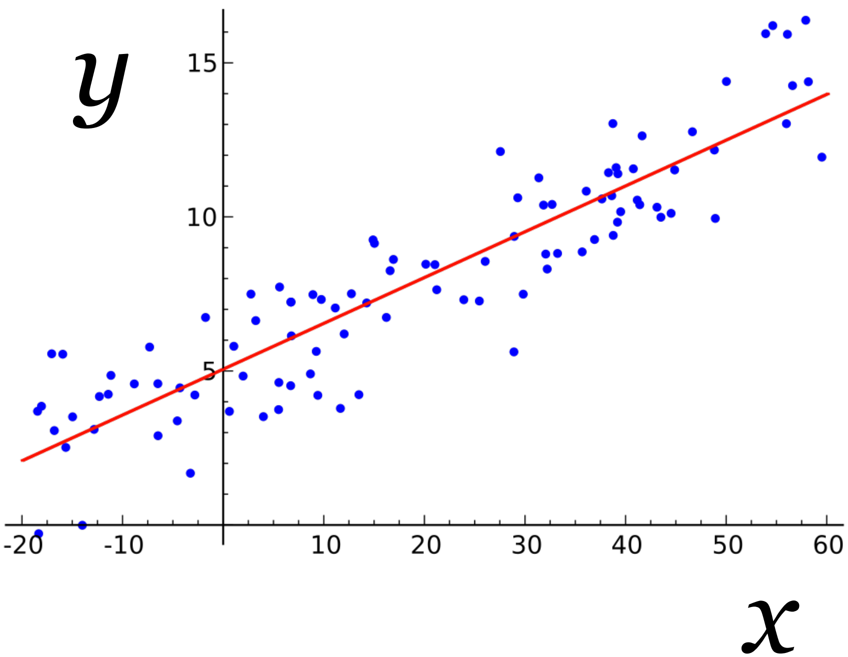
Task: **Regression** & Classification

Supervised Learning

Unsupervised Learning

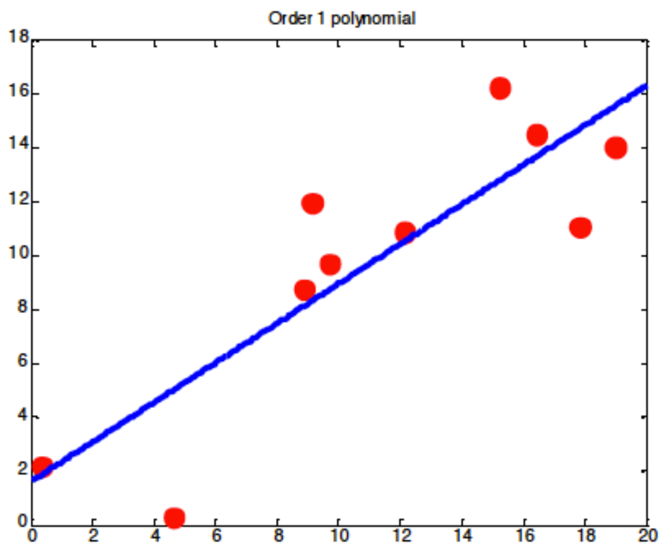
Reinforcement Learning

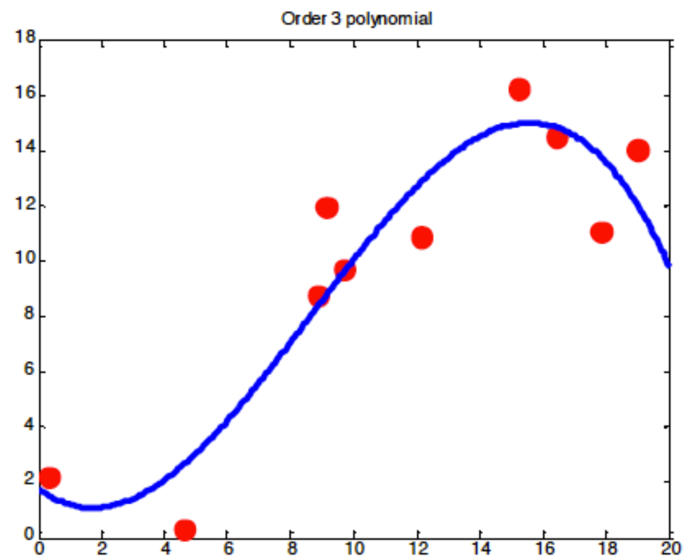
y is a numeric variable
e.g. sales in $$



Keys to Regression Success

- Feature transformation + linear model
- Penalized weights to avoid overfitting





(c) Alexander Ihler

Can fit **linear** functions to **nonlinear** features

A nonlinear function of x:

$$\hat{y}(x_i) = \theta_0 + \theta_1 x_i + \theta_2 x_i^2 + \theta_3 x_i^3$$

Can be written as a linear function of $\phi(x_i) = [1 \ x_i \ x_i^2 \ x_i^3]$

$$\hat{y}(x_i) = \sum_{g=1}^{4} \theta_g \phi_g(x_i) = \theta^T \phi(x_i)$$

"Linear regression" means linear in the parameters (weights, biases)

Features can be arbitrary transforms of raw data

What feature transform to use?

- Anything that works for your data!
  - sin / cos for periodic data
  - polynomials for high-order dependencies

$$\phi(x_i) = [1 \ x_i \ x_i^2 \ldots]$$

  - interactions between feature dimensions

$$\phi(x_i) = [1 \quad x_{i1}x_{i2} \quad x_{i3}x_{i4} \ldots]$$

  - Many other choices possible

Review: Linear Regression

Prediction: Linear transform of G-dim features

$$\hat{y}(x_i, \theta) = \theta^T \phi(x_i) = \sum_{g=1}^{G} \theta_g \cdot \phi(x_i)_g$$

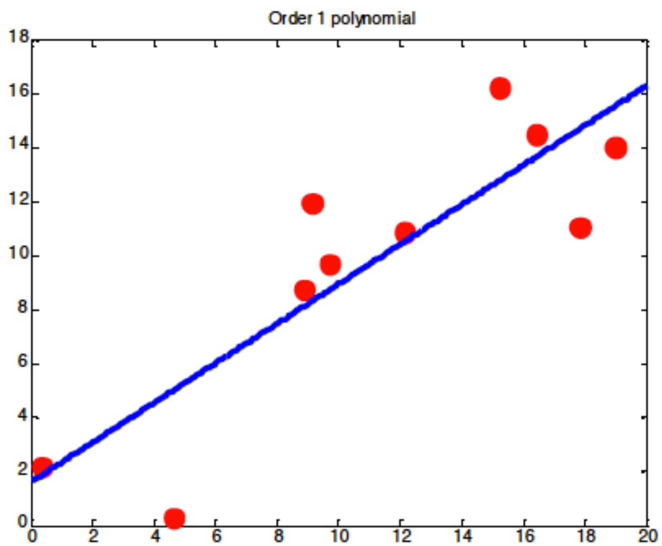
Training: Solve optimization problem

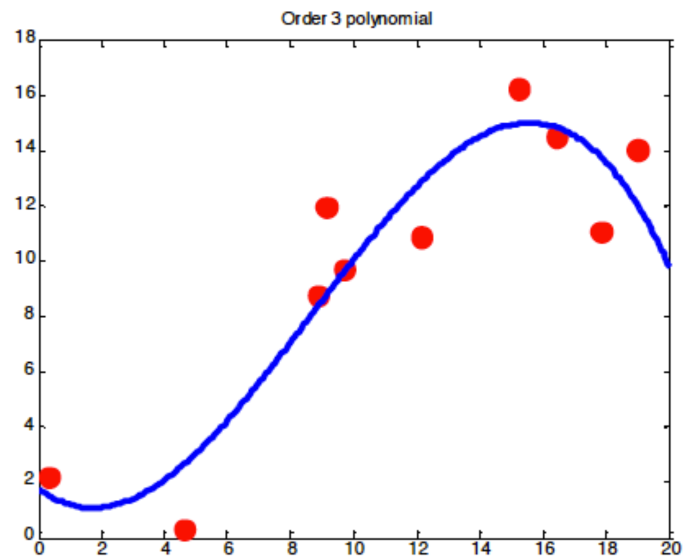
$$\min_{\theta} \sum_{n=1}^{N} \left(y_n - \hat{y}(x_n, \theta)\right)^2$$

+ L2 penalty (optional)

Problems with high-dim features

- Feature transformation + linear model





(c) Alexander Ihler

How expensive is this transformation?
(Runtime and storage)

Thought Experiment

- Suppose that the optimal weight vector can be exactly constructed via a *linear combination* of the training set feature vectors

$$\theta^* = \alpha_1 \phi(x_1) + \alpha_2 \phi(x_2) + \ldots + \alpha_N \phi(x_N)$$

Each alpha is a **scalar**

Each feature vector is a **vector of size G**

Justification?

Is optimal theta a linear combo of feature vectors?

Stochastic gradient descent, with 1 example per batch, can be seen as creating optimal weight vector of this form

- Starting with all zero vector
- In each step, adding a weight * feature vector

Each update step:

$$\theta_t \leftarrow \theta_{t-1} - \eta \cdot \frac{d}{d\theta} \mathrm{loss}(y_n, \theta^T \phi(x_n))$$

Let's simplify this via chain rule!

Justification?

Stochastic gradient descent, with 1 example per batch, can be seen as creating optimal weight vector of this form

- Starting with all zero vector
- In each step, adding a weight * feature vector

Each update step:

$$\theta_t \leftarrow \theta_{t-1} - \eta \cdot \frac{d}{da}\text{loss}(y_n, a) \cdot \frac{d}{d\theta}\theta^T \phi(x_n)$$

scalar *scalar* *Vector of size G*

Justification?

Stochastic gradient descent, with 1 example per batch, can be seen as creating optimal weight vector of this form

- Starting with all zero vector
- In each step, adding a weight * feature vector

Each update step:

$$\theta_t \leftarrow \theta_{t-1} - \eta \cdot \frac{d}{da} \text{loss}(y_n, a) \cdot \phi(x_n)$$

scalar *scalar* *Vector of size G (simplified)*

How to Predict in this thought experiment

$$\theta^* = \alpha_1 \phi(x_1) + \alpha_2 \phi(x_2) + \ldots + \alpha_N \phi(x_N)$$

Prediction:

$$\hat{y}(x_i, \theta) = \theta^T \phi(x_i)$$

$$\hat{y}(x_i, \theta^*) = \left(\sum_{n=1}^{N} \alpha_n \phi(x_n) \right)^T \phi(x_i)$$

How to Predict in this thought experiment

$$\theta^* = \alpha_1 \phi(x_1) + \alpha_2 \phi(x_2) + \ldots + \alpha_N \phi(x_N)$$

Prediction:

$$\hat{y}(x_i, \theta) = \theta^T \phi(x_i)$$

$$\hat{y}(x_i, \theta^*) = \sum_{n=1}^{N} \alpha_n \boxed{\phi(x_n)^T \phi(x_i)}$$

Inner product of test feature vector with each training feature!

Kernel Function

$$k(x_i, x_j) = \phi(x_i)^T \phi(x_j)$$

Input: any two vectors x_i and x_j
Output: scalar real

Interpretation: similarity function for x_i and x_j

Properties:

- Larger output values mean i and j are more similar
- Symmetric

Kernelized Linear Regression

- **Prediction**:

$$\hat{y}(x_i, \alpha, \underbrace{\{x_n\}_{n=1}^{N}}_{= X}) = \sum_{n=1}^{N} \alpha_n k(x_n, x_i)$$

- **Training**

$$\min_{\alpha} \sum_{n=1}^{N} \left(y_n - \hat{y}(x_n, \alpha, X)\right)^2$$

Can do all needed operations with only access to kernel (no feature vectors)

Compare: Linear Regression

Prediction: Linear transform of G-dim features

$$\hat{y}(x_i, \theta) = \theta^T \phi(x_i) = \sum_{g=1}^{G} \theta_g \cdot \phi(x_i)_g$$

Training: Solve optimization problem

$$\min_{\theta} \sum_{n=1}^{N} (y_n - \hat{y}(x_n, \theta))^2$$

+ L2 penalty (optional)

Why is kernel trick good idea?

Before:

Training problem optimized vector of size G

Prediction cost:

scales linearly with G (num. high-dim features)

After:

Training problem optimized vector of size N

Prediction cost:

scales linearly with N (num. train examples)

requires N evaluations of kernel

So we get some saving in runtime/storage if

G is bigger than N

AND we can compute k faster than inner product

Example: From Features to Kernels

$$x = \begin{bmatrix} x_1 & x_2 \end{bmatrix} \qquad z = \begin{bmatrix} z_1 & z_2 \end{bmatrix}$$

$$\phi(x) = \begin{bmatrix} 1 & x_1^2 & x_2^2 & \sqrt{2}x_1 & \sqrt{2}x_2 & \sqrt{2}x_1 x_2 \end{bmatrix}$$

$$k(x, z) = (1 + x_1 z_1 + x_2 z_2)^2$$

Compare:

What is relationship between these two functions defined above?

$$k(x, z) \qquad \phi(x)^T \phi(z)$$

Example: From Features to Kernels

$$x = \begin{bmatrix} x_1 & x_2 \end{bmatrix} \qquad z = \begin{bmatrix} z_1 & z_2 \end{bmatrix}$$

$$\phi(x) = \begin{bmatrix} 1 & x_1^2 & x_2^2 & \sqrt{2}x_1 & \sqrt{2}x_2 & \sqrt{2}x_1 x_2 \end{bmatrix}$$

$$k(x, z) = (1 + x_1 z_1 + x_2 z_2)^2$$

Compare:

What is relationship between these two functions defined above?

$$k(x, z) \quad = \quad \phi(x)^T \phi(z)$$

Punchline: Can sometimes find **faster** ways to compute high-dim. inner product

Cost comparison

$$x = \begin{bmatrix} x_1 & x_2 \end{bmatrix} \qquad z = \begin{bmatrix} z_1 & z_2 \end{bmatrix}$$

$$\phi(x) = \begin{bmatrix} 1 & x_1^2 & x_2^2 & \sqrt{2}x_1 & \sqrt{2}x_2 & \sqrt{2}x_1 x_2 \end{bmatrix}$$

$$k(x, z) = (1 + x_1 z_1 + x_2 z_2)^2$$

Compare:

Number of add and multiply ops to compute $\phi(x)^T \phi(z)$

Number of add and multiply ops to compute $k(x, z)$

Example:
Kernel cheaper than inner product

$$x = \begin{bmatrix} x_1 & x_2 \end{bmatrix}$$

$$\phi(x) = \begin{bmatrix} 1 & x_1^2 & x_2^2 & \sqrt{2}x_1 & \sqrt{2}x_2 & \sqrt{2}x_1 x_2 \end{bmatrix}$$

$$k(x, z) = (1 + x_1 z_1 + x_2 z_2)^2$$

$$z = \begin{bmatrix} z_1 & z_2 \end{bmatrix}$$

Compare:

Number of add and multiply ops to compute $\phi(x)^T \phi(z)$

6 multiply and 5 add

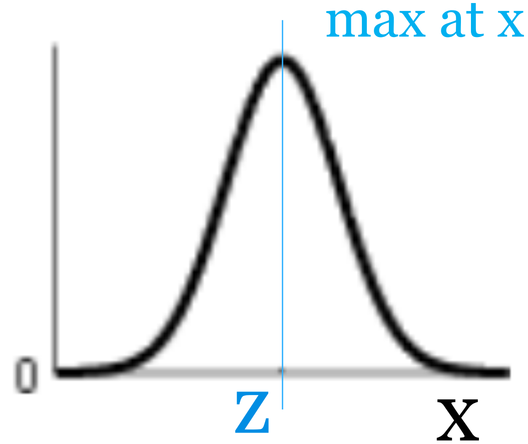
Number of add and multiply ops to compute $k(x, z)$

3 multiply (include square) and 2 add

Squared Exponential Kernel

Assume x is a scalar

$$k(x, z) = e^{-(x-z)^2}$$



Also called "radial basis function (RBF)" kernel

Squared Exponential Kernel

Assume *x* is a scalar

$$
\begin{aligned}
k(x, z) &= e^{-(x-z)^2} \\
&= e^{-x^2 - z^2 + 2xz} \\
&= e^{-x^2} e^{-z^2} e^{2xz}
\end{aligned}
$$

Recall: Taylor series for *e^x*

$$e^x = \sum_{k=0}^{\infty} \frac{1}{k!} x^k = 1 + x + \frac{1}{2} x^2 + \dots$$

$$e^{2xz} = \sum_{k=0}^{\infty} \frac{2^k}{k!} x^k z^k$$

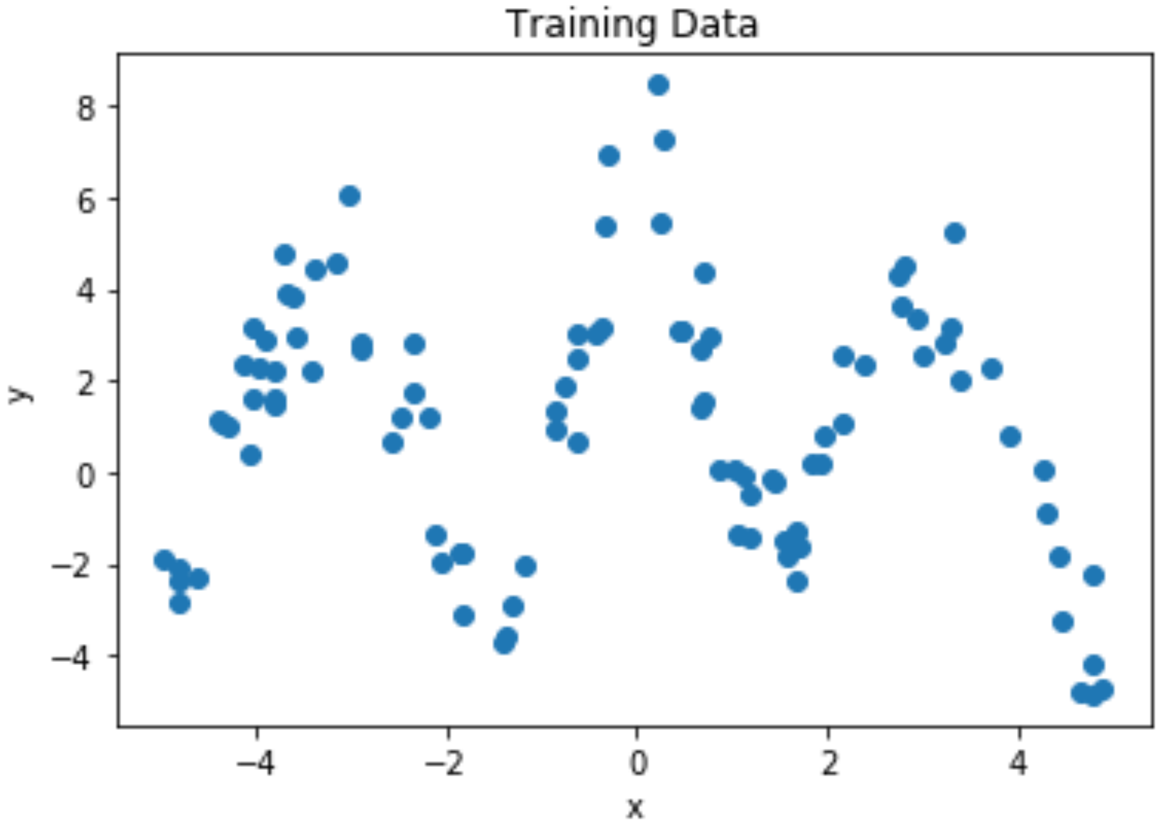
Squared Exponential Kernel

$$
\begin{aligned}
k(x, z) &= e^{-(x-z)^2} \\
&= e^{-x^2 - z^2 + 2xz} \\
&= e^{-x^2} e^{-z^2} \left(\sum_{k=0}^{\infty} \sqrt{\frac{2^k}{k!}} x^k \right) \left(\sum_{k=0}^{\infty} \sqrt{\frac{2^k}{k!}} z^k \right) \\
&= \phi(x)^T \phi(z)
\end{aligned}
$$

Corresponds to an INFINITE DIMENSIONAL feature vector

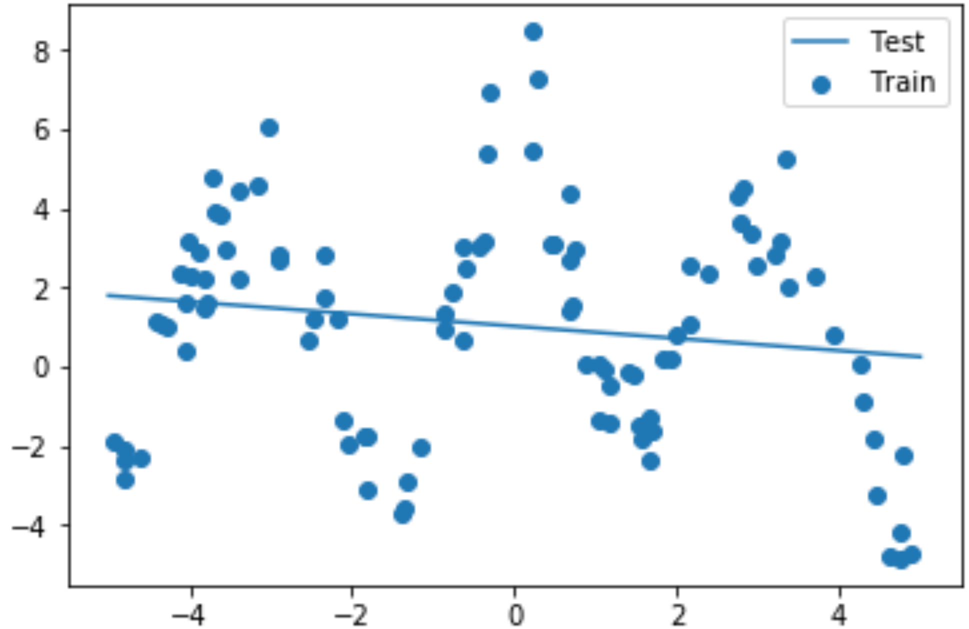
$$
\phi(x) = [\sqrt{\frac{2^0}{0!}} x^0 e^{-x^2} \quad \sqrt{\frac{2^1}{1!}} x^1 e^{-x^2} \quad \ldots \quad \sqrt{\frac{2^k}{k!}} x^k e^{-x^2} \quad \ldots \quad]
$$

Kernelized Regression Demo



Linear Regression

```
clf = sklearn.linear_model.LinearRegression()
clf.fit(x_train, y_train)
plot_model(x_test, clf)
```



Kernel Matrix for training set

- **K : N x N symmetric matrix**

$$K = \begin{bmatrix} k(x_1, x_1) & k(x_1, x_2) \ldots k(x_1, x_N) \\ k(x_2, x_1) & k(x_2, x_2) \ldots k(x_2, x_N) \\ \vdots & \\ k(x_N, x_1) & k(x_N, x_2) \ldots k(x_N, x_N) \end{bmatrix}$$

Linear Regression with Kernel

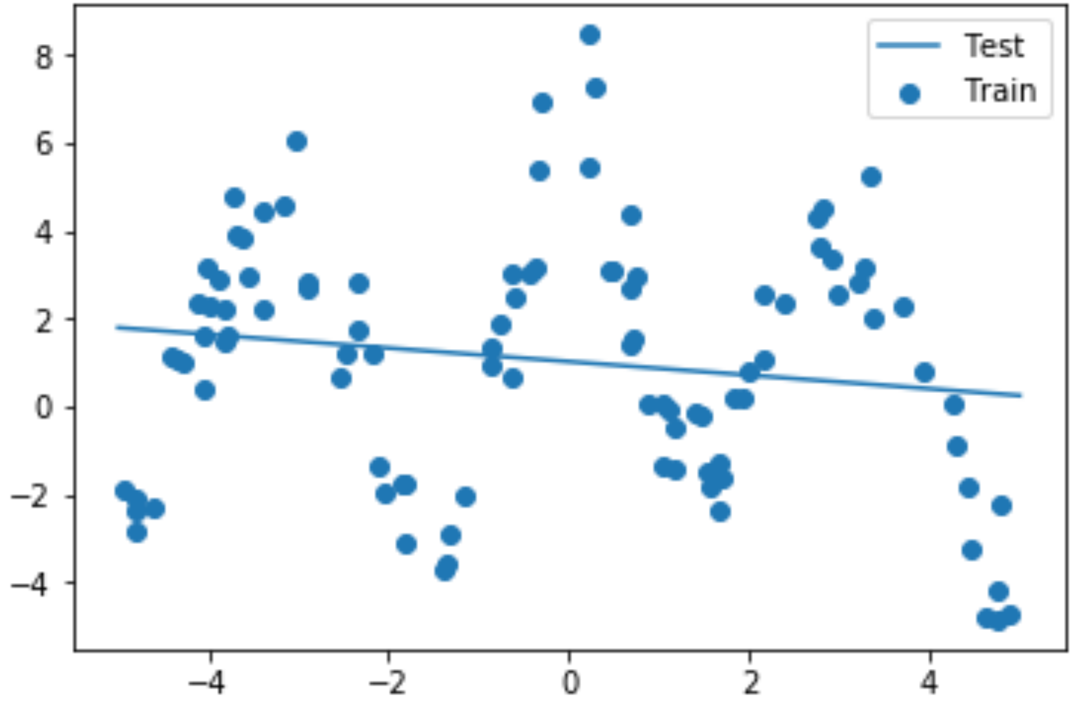
100 training examples in x_train
505 test examples in x_test

```
def linear_kernel(X, Z):
    '''
    Compute dot product between each row of X and each row of Z
    '''
    m1,_ = X.shape
    m2,_ = Z.shape
    K = np.zeros((m1, m2))
    for i in range(m1):
        for j in range(m2):
            K[i,j] = np.dot(X[i,:], Z[j,:])
    return K

K_train = linear_kernel(x_train, x_train) + 1e-10 * np.eye(N)  # see note belo
K_test  = linear_kernel(x_test,  x_train)

print("Shape of K_train: %s" % str(K_train.shape))
print("Shape of K_test: %s" % str(K_test.shape))
```

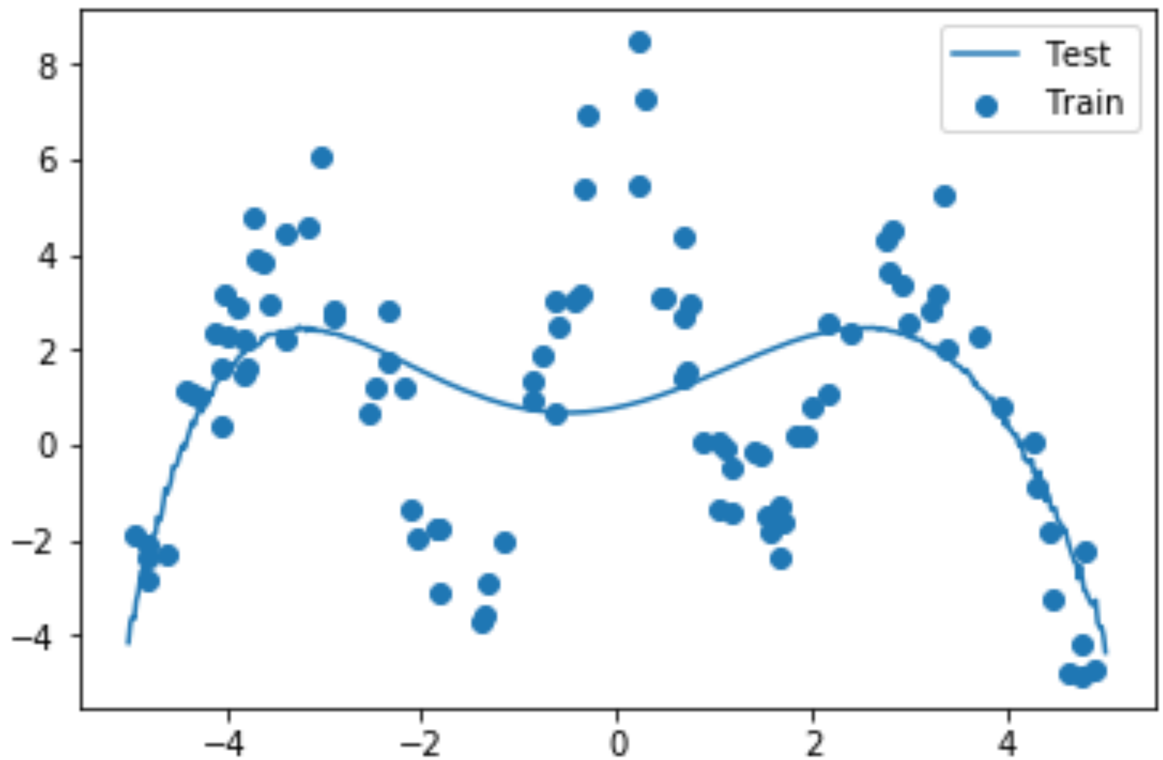
```
Shape of K_train: (100, 100)
Shape of K_test: (505, 100)
```

Linear Regression with Kernel

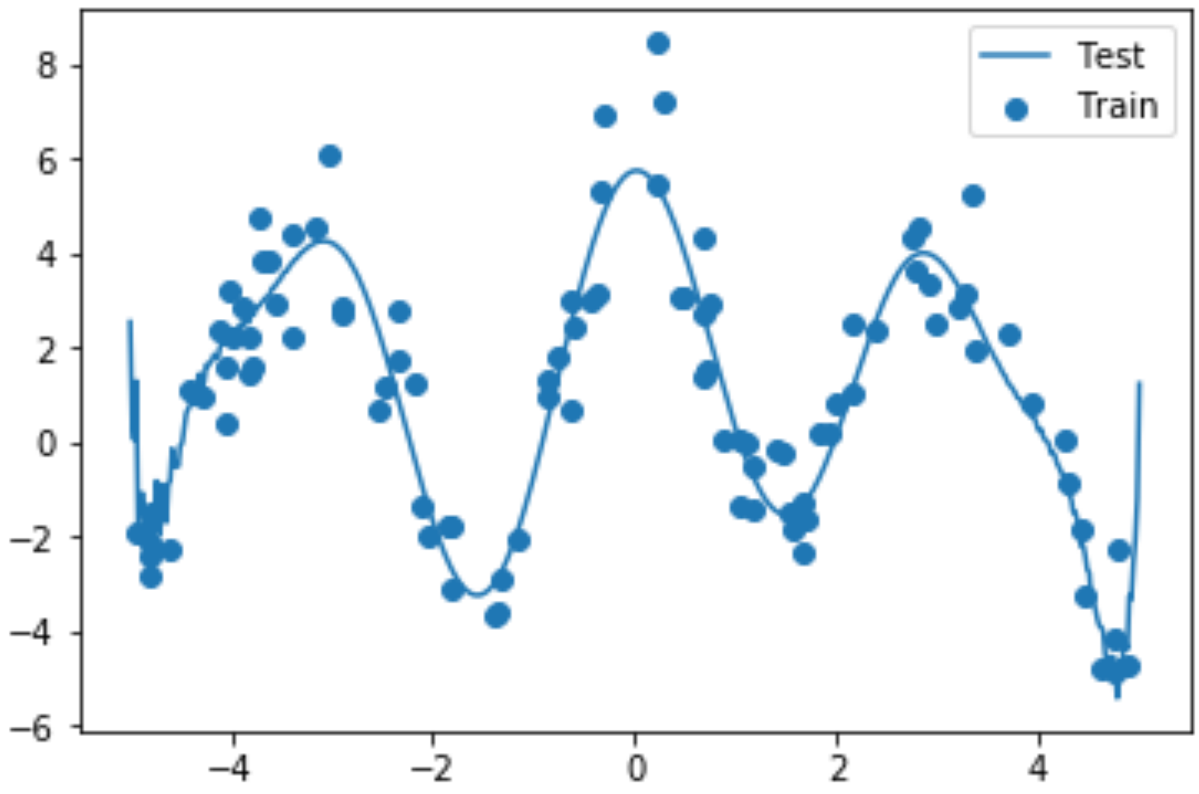
```
clf = sklearn.linear_model.LinearRegression()
clf.fit(K_train, y_train)
plot_model(K_test, clf)
```



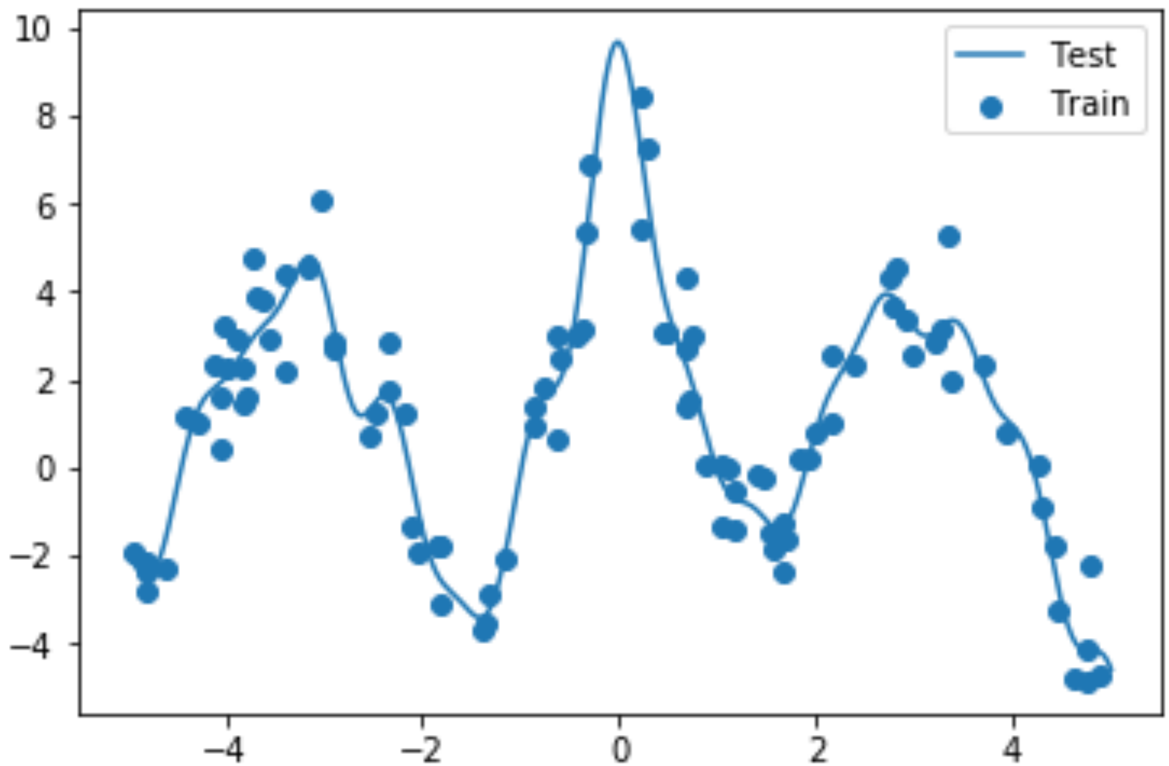
Polynomial Kernel, deg. 5



Polynomial Kernel, deg. 12



Gaussian kernel (aka sq. exp.)



Kernel Regression in sklearn

sklearn.kernel_ridge.KernelRidge

class `sklearn.kernel_ridge.` **`KernelRidge`** (*alpha=1*, *kernel='linear'*, *gamma=None*, *degree=3*, *coef0=1*, *kernel_params=None*) [source]

`fit` (*X*, *y=None*, *sample_weight=None*) [source]

Demo will use

kernel='precomputed'

Fit Kernel Ridge regression model

Parameters:	**X : *{array-like, sparse matrix}, shape = [n_samples, n_features]*** Training data. If kernel == "precomputed" this is instead a precomputed kernel matrix, shape = [n_samples, n_samples]. **y : *array-like, shape = [n_samples] or [n_samples, n_targets]*** Target values **sample_weight : *float or array-like of shape [n_samples]*** Individual weights for each sample, ignored if None is passed.
Returns:	**self : *returns an instance of self.***

Can kernelize any linear model

Regression: Prediction

$$\hat{y}(x_i, \alpha, \{x_n\}_{n=1}^N) = \sum_{n=1}^N \alpha_n k(x_n, x_i)$$

Logistic Regression: Prediction

$$p(Y_i = 1 | x_i) = \sigma(\hat{y}(x_i, \alpha, X))$$

Training for kernelized versions of
* Linear Regression
* Logistic Regression

$$\min_{\alpha} \sum_{n=1}^{N} \left(y_n - \hat{y}(x_n, \alpha, X)\right)^2$$

$$\min_{\alpha} \sum_{n=1}^{N} \text{log\_loss}(y_n, \sigma(\hat{y}(x_n, \alpha, X)))$$

SVMs: Prediction

$$\hat{y}(x_i) = w^T x_i + b$$

Make binary prediction via hard threshold

$$\begin{cases} 1 & \text{if } \hat{y}(x_i) \geq 0 \\ 0 & \text{otherwise} \end{cases}$$

SVMs and Kernels: Prediction

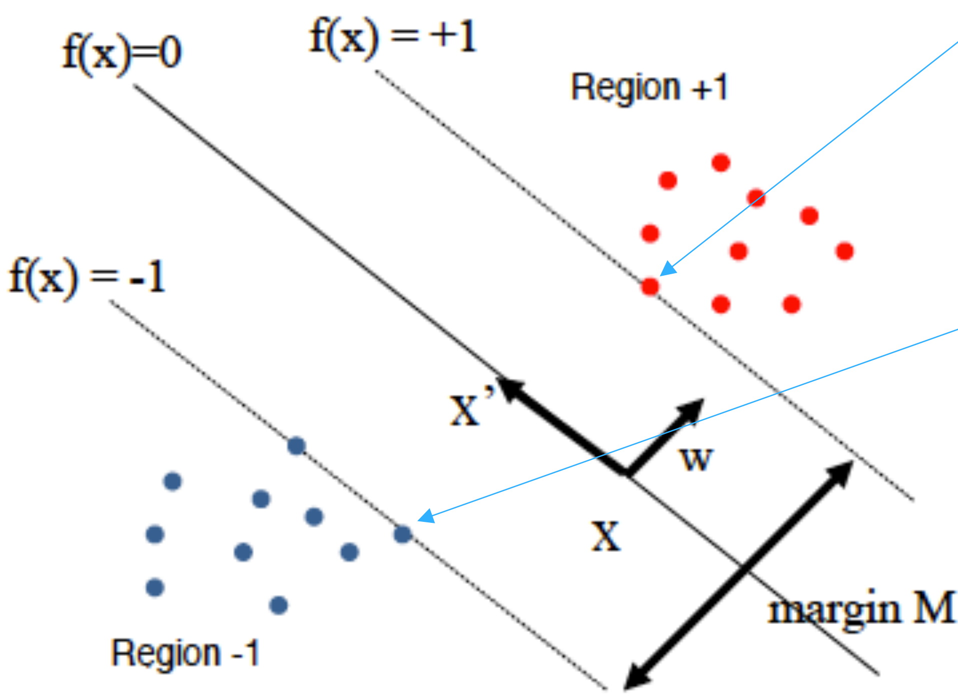
$$\hat{y}(x_i) = \sum_{n=1}^{N} \alpha_n k(x_n, x_i)$$

Make binary prediction via hard threshold

$$\begin{cases} 1 & \text{if } \hat{y}(x_i) \geq 0 \\ 0 & \text{otherwise} \end{cases}$$

Efficient training algorithms using modern quadratic programming solve the dual optimization problem of SVM soft margin problem

Support vectors are often **small** fraction of all examples



Nearest positive example
x_+

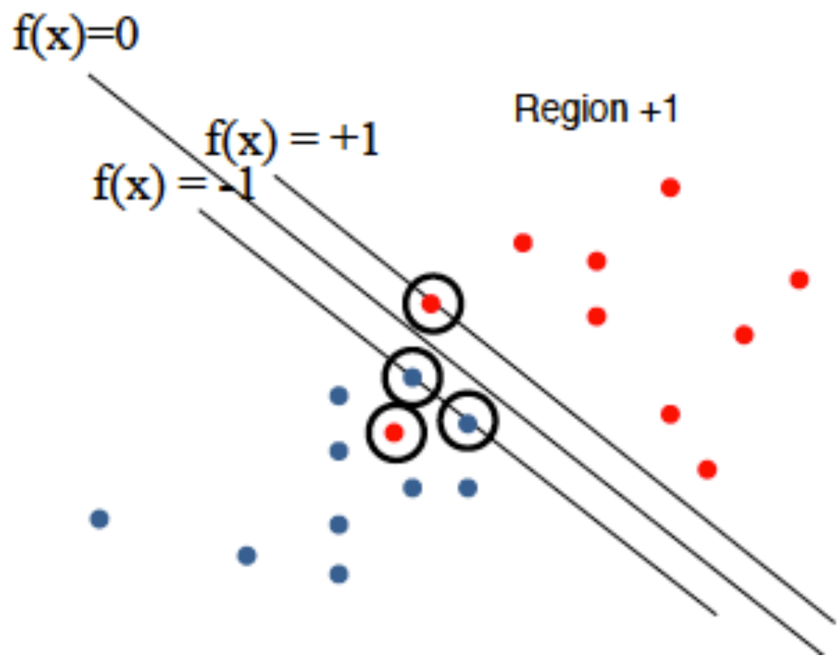
Nearest negative example
x_-

Support vectors defined by **non-zero alpha** in kernel view

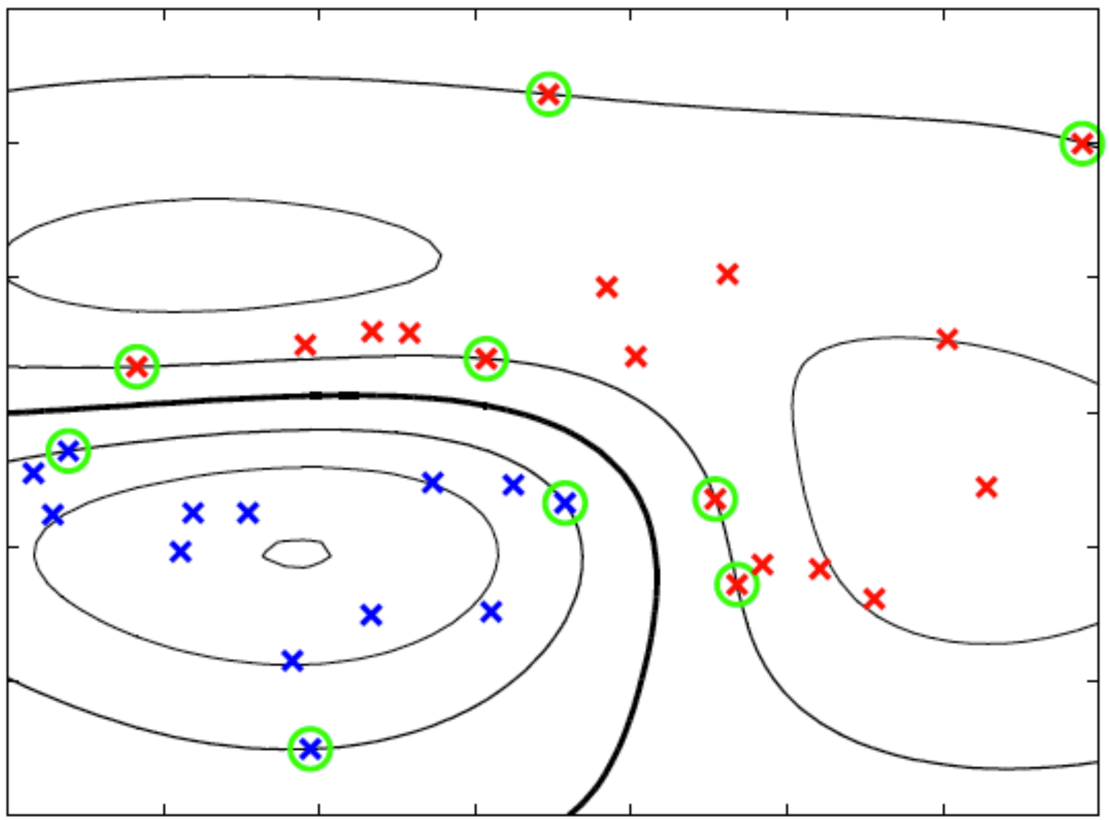
Data points i with non-zero weight α_i:

- Points with minimum margin (on optimized boundary)
- Points which violate margin constraint, but are still correctly classified
- Points which are misclassified

For all other training data, features have *no impact* on learned weight vector



SVM + Squared Exponential Kernel



Support vectors (green) for data separable by radial basis function kernels, and non-linear margin boundaries

Kernel Unit Objectives

Big idea: Use kernel functions (similarity function with special properties) to obtain flexible high-dimensional feature transformations without explicit features

- From linear regression (LR) to kernelized LR
- What is a kernel function?
  - Basic properties
  - Example: Polynomial kernel
  - Example: Squared Exponential kernel
- Kernels for classification
  - Logistic Regression
  - SVMs