Data Mining

Tamás Budavári - budavari@jhu.edu
Class 5

- Regularization
- Principal Component Analysis
- Lagrange multipliers
- Explained variance

Linear Regression

- A linear combination of known $\phi_k(\cdot)$ **basis** functions

> $$f(t;\boldsymbol{\beta}) = \sum_{k=1}^{K} \beta_k\, \phi_k(t)$$
>
> It's a dot product with $\boldsymbol{\beta} = (\beta_1, \ldots, \beta_K)^T$

- Evaluated at all data points $x = (x_1, x_2, \ldots, x_N)$

> $$f(x;\boldsymbol{\beta}) = X\boldsymbol{\beta}$$
>
> where $X_{ik} = \phi_k(x_i)$

Method of Least Squares

- At the optimum

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

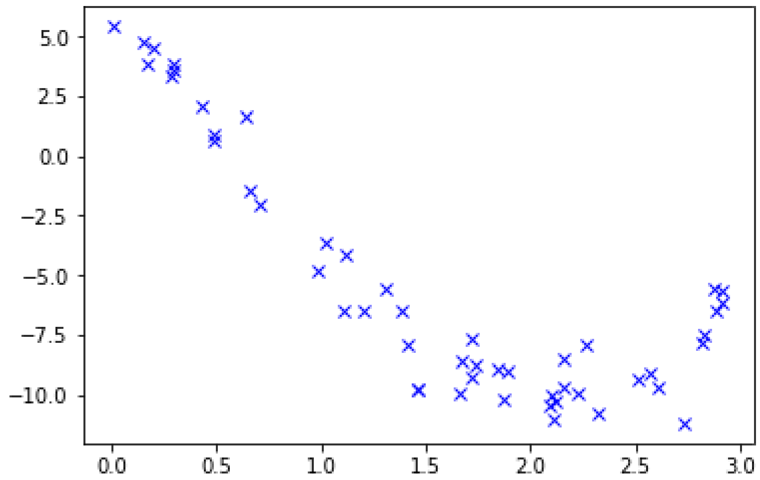
- Hat matrix

$$\hat{\boldsymbol{y}} = X\hat{\boldsymbol{\beta}} = H\boldsymbol{y}$$

In [1]:
```
%pylab inline
```

```
Populating the interactive namespace from numpy and matplotlib
```

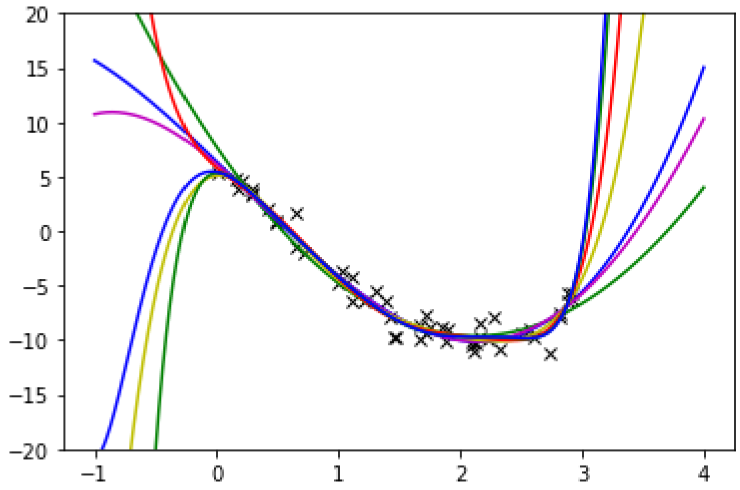
In [2]:
```
# generate a dataset with errors
x = 3 * random.rand(50) # uniform between 0 and 3
eps = 1 * random.randn(x.size) # normal noise
y = 10*cos(x+1) + eps;  plot(x,y,'bx');
```



In [3]:
```
def poly(x,n):
    X = np.zeros((x.size,n+1));
    for i in range(X.shape[1]):
        X[:,i] = x**i
    return X

# show data in black
plot(x,y,'kx'); ylim(-20,20);

xx = np.linspace(-1,4,500) # grid on x
color = 'yrgbm' * 5 # color sequence
for n in range(2,9):
    X = poly(x,n) # design matrix for fitting
    bHat = linalg.pinv(X).dot(y)
    yy = poly(xx,n).dot(bHat) # prediction
    plot(xx,yy,'-',c=color[n]);
```



Regularization

Penalize large coefficients in β

- **Ridge regression** uses L_2

> $$\hat{\beta} = \underset{\beta}{\operatorname{argmin}} |y - X\beta|_2^2 \ + \lambda\, |\beta|_2^2$$
>
> or even with a constant matrix Γ
>
> $$\hat{\beta} = \underset{\beta}{\operatorname{argmin}} |y - X\beta|_2^2 + \lambda\, |\Gamma\beta|_2^2$$

- **Lasso regression** uses L_1

> $$\hat{\beta} = \underset{\beta}{\operatorname{argmin}} |y - X\beta|_2^2 + \lambda\, |\beta|_1$$
>
> L_1 yields sparse results

Different geometric meanings!

Linear Combinations

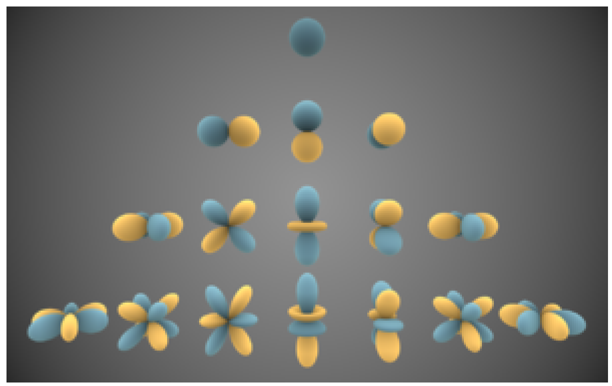
- Coefficients mix a given set of basis vectors, functions, images, shapes, ...

$$f(x;\beta) = \sum_k \beta_k \phi_k(x)$$

Fourier series

Discrete Cosine Transform (JPEG)

Spherical Harmonics



- What is a good basis like?

Principal Component Analysis

Statistical Learning

	Supervised	Unsupervised
Discrete	Classification	Clustering
Continuous	Regression	Dimensionality Reduction

Directions of Maximum Variance

- Let $X \in \mathbb{R}^N$ be a continuous random variable with $\mathbb{E}[X] = 0$ mean and covariance matrix C. What is the direction of maximum variance?

For any vector $a \in \mathbb{R}^N$

$$\mathbb{Var}[a^T X] = \mathbb{E}\left[(a^T X)(X^T a)\right] = \mathbb{E}\left[a^T (XX^T)\, a\right]$$

so

$$\mathbb{Var}[a^T X] = a^T\, \mathbb{E}\left[XX^T\right]\; a = a^T C\, a$$

We have to maximize this such that $a^2 = 1$

Constrained Optimization

- **Lagrange multiplier**: extra term with new parameter λ

$$\hat{a} = \arg\max_{a \in \mathbb{R}^N} \left[a^T C\, a - \lambda\, (a^2 - 1) \right]$$

- Partial derivatives vanish at optimum

$$\frac{\partial}{\partial \lambda} \rightarrow \quad \hat{a}^2 - 1 = 0 \quad \text{(duh!)}$$

$$\frac{\partial}{\partial a_k} \rightarrow \quad ?$$

With indices

$$\max_{a \in \mathbb{R}^N} \left[\sum_{i,j} a_i C_{ij} a_j - \lambda \left(\sum_i a_i^2 - 1 \right) \right]$$

- Partial derivatives $\partial \big/ \partial a_k$ vanish at optimum

$$\sum_{i,j} \frac{\partial a_i}{\partial a_k} C_{ij} a_j + \sum_{i,j} a_i C_{ij} \frac{\partial a_j}{\partial a_k} - 2\lambda \left(\sum_i a_i \frac{\partial a_i}{\partial a_k} \right) =$$

$$= \sum_{i,j} \delta_{ik} C_{ij} a_j + \sum_{i,j} a_i C_{ij} \delta_{jk} - 2\lambda \left(\sum_i a_i \delta_{ik} \right) =$$

$$= \sum_j C_{kj} a_j + \sum_i a_i C_{ik} - 2\lambda\, a_k$$

And back again...

- With vectors and matrices

> $$C\hat{a} + C^T\hat{a} - 2\lambda\hat{a} = 0$$
>
> but C is symmetric
>
> $$C\,\hat{a} = \lambda\,\hat{a}$$

- Eigenproblem !!

Result

- The value of maximum variance is

> $$\hat{a}^T C\,\hat{a} = \hat{a}^T\lambda\,\hat{a} = \lambda\,\hat{a}^T\hat{a} = \lambda$$
>
> the largest eigenvalue λ_1

- The direction of maximum variance is the corresponding eigenvector a_1

> $$Ca_1 = \lambda_1 a_1$$

- This is the **1st Principal Component**

2nd Principal Component

- Direction of largest variance uncorrelated to 1st PC

$$\hat{a} = \arg\max_{a \in \mathbb{R}^N} \left[a^T C\, a - \lambda\,(a^2 - 1) - \lambda'(a^T C\, a_1) \right]$$

- Partial derivatives vanish at optimum

$$2C\,\hat{a} - 2\lambda\,\hat{a} - \lambda' C a_1 \;= 0$$

Result

- Multiply by $a_1^T \cdot$

$$2a_1^T C\hat{a} - 2a_1^T \lambda \hat{a} - a_1^T \lambda' C a_1 \;= 0$$

$$0 - 0 - \lambda' \lambda_1 = 0 \quad \rightarrow \quad \lambda' = 0$$

- Still just an eigenproblem

$$C\,\hat{a} = \lambda\,\hat{a}$$

- Solution λ_2 and a_2

PCA

- Spectral decomposition or eigenvalue decomposition or eigendecomposition

> Let $\lambda_1 \geq \lambda_2 \geq \cdots \geq \lambda_N \geq 0$ be the eigenvalues of C and $e_1, \ldots, e_N$ the corresponding eigenvectors
>
> $$C = \sum_{k=1}^{N} \lambda_k \left(e_k\, e_k^T\right)$$
>
> Consider $C\, e_l = \sum_k \lambda_k\, e_k \left(e_k^T e_l\right) = \lambda_l\, e_l$ for any l

- Matrix form

> With diagonal Λ matrix of the eigenvalues and an E matrix of $[e_1, \ldots, e_N]$
>
> $$C = E\, \Lambda\, E^T$$

- The eigenvectors of largest eigenvalues capture the most variance

> If keeping only $K < N$ eigenvectors, the best approximation is taking the first K PCs
>
> $$C \approx \sum_{k=1}^{K} \lambda_k \left(e_k\, e_k^T\right) = E_K \Lambda_K E_K^T$$

New Coordiante System

- The E matrix of eigenvectors is a rotation, $E\,E^T = I$

$$Z = E^T\,X$$

- A truncated set of eigenvectors E_K defines a projection

$$Z_K = E_K^T\,X$$

and

$$X_K = E_K Z_K = E_K E_K^T\,X = P_K\,X$$

Detour: Projections

- If the square of a matrix is equal to itself

$$P^2 = P$$

- For example, projecting on the e unit vector

Scalar times vector

$$r' = e\,\left(e^T r\right) = e\,\beta_r$$

Or projection of vector r

$$r' = \left(e\,e^T\right)\,r = P\,r$$

Again

- The eigenvectors of largest eigenvalues capture the most variance

$$C \approx C_K = \sum_{k=1}^{K} \lambda_k \left(e_k \, e_k^T \right) = \sum_{k=1}^{K} \lambda_k \, P_k$$

- And the remaining eigenvectors span the subspace with the least variance

$$C - C_K = \sum_{l=K+1}^{N} \lambda_l \, P_l$$

Samples

- Set of N-vectors arranged in matrix $X = [x_1, x_2, \ldots, x_n]$ with average of 0
  This is NOT the random variable we talked about previously but the data matrix!

> Sample covariance matrix is
>
> $$C = \frac{1}{n-1} XX^T = \frac{1}{n-1} \sum_i x_i x_i^T$$

- Singular Value Decomposition (SVD)

> $X = UWV^T$
>
> where $U^T U = I$, W is diagonal, and $V^T V = I$

- Hence

> $$C = \frac{1}{n-1} UWV^T \, VWU^T = \frac{1}{n-1} UW^2 U^T$$
>
> So, if $C = E \Lambda E^T$ then $E = U$ and $\Lambda = \frac{1}{n-1} W^2$

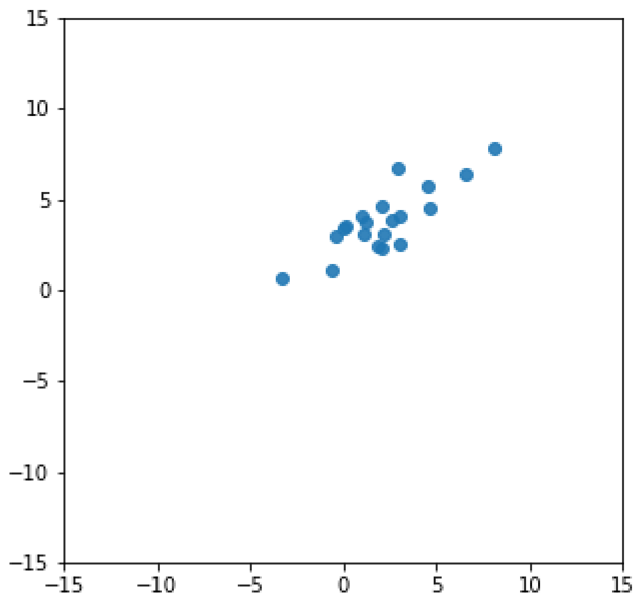
Random Sample from Bivariate Normal

- See previous lecture

In [4]:
```
from scipy.stats import norm
# generate multiple 2-D (column) vectors
S = norm.rvs(0,1,(2,20))
S[0,:] *= 4  # scale axis 0
f = +pi/4    # rotate by 45 degrees
R = array([[cos(f), -sin(f)],
           [sin(f),  cos(f)]])
X = R.dot(S)
X += np.array([[1],[3]]) # shift

figure(figsize=(5,5)); xlim(-15,15); ylim(-15,15);
plot(X[0,:],X[1,:],'o',alpha=0.9)
```

Out[4]:
```
[<matplotlib.lines.Line2D at 0x1a13f9c7f0>]
```



In [5]:
```
# subtract sample mean
avg = mean(X, axis=1).reshape(X[:,1].size,1)
X -= avg
# sample covariance matrix
C = X.dot(X.T) / (X[0,:].size-1)
print ("Average\n", avg)
print ("Covariance\n", C)
```

```
Average
 [[ 2.13266854]
 [ 3.81741732]]
Covariance
 [[ 6.56960563  3.80359671]
 [ 3.80359671  3.21387998]]
```

In [6]:
```
L, E = np.linalg.eig(C)
E, L
```

Out[6]:
```
(array([[ 0.83773535, -0.54607644],
        [ 0.54607644,  0.83773535]]), array([ 9.04897404,  0.73451157]))
```

In [7]:
```
E, L, E_same = np.linalg.svd(C)
E, L
```

Out[7]:
```
(array([[-0.83773535, -0.54607644],
        [-0.54607644,  0.83773535]]), array([ 9.04897404,  0.73451157]))
```

In [8]:
```
E.dot(E.T)
```

Out[8]:
```
array([[  1.00000000e+00,   5.55111512e-17],
       [  5.55111512e-17,   1.00000000e+00]])
```

In [9]:
```
np.allclose( E.T, np.linalg.inv(E) )
```

Out[9]:
```
True
```

In [10]:
```
U, W, V = np.linalg.svd(X)
U, W**2 / (X[0,:].size-1)
```

Out[10]:
```
(array([[-0.83773535, -0.54607644],
        [-0.54607644,  0.83773535]]), array([ 9.04897404,  0.73451157]))
```

In [11]:
```python
# alternatively
U, W**2 / (X.shape[1]-1)
```

Out[11]:
```
(array([[-0.83773535, -0.54607644],
        [-0.54607644,  0.83773535]]), array([ 9.04897404,  0.73451157]))
```

In [12]:
```python
[ np.allclose( U.dot(U.T), np.eye(U.shape[0]) ),
  np.allclose( V.dot(V.T), np.eye(V.shape[0]) )  ]
```

Out[12]:
```
[True, True]
```

In [13]:
```python
from sklearn import decomposition
pca = decomposition.PCA(n_components=X.shape[0])
pca.fit(X.T) # different convention: row vs col !!!
pca.components_.T, pca.explained_variance_
```

Out[13]:
```
(array([[ 0.83773535, -0.54607644],
        [ 0.54607644,  0.83773535]]), array([ 9.04897404,  0.73451157]))
```