Lecture 12-15: Support Vector Machines

SVM is a discriminative method for classification. It devises a hyper-plane in a high dimensional space (defined implicitly or explicitly) as the decision boundary.

Outline

1, Hyper-plane and linear classification
2, Perceptron learning
3, Kernel induced feature space
4, SVM -- maximal margin classifier
5, SVM -- soft margin classifier

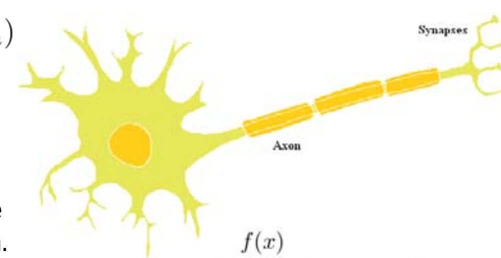
6, Structured SVM
7, Rank SVM

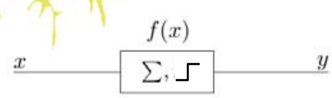
The note is based on a book by N. Cristianini and J. Shawe-Taylor, "Introduction to Support Vector Machines", Cambridge University Press, 2000.

1, Hyper-plane and linear classification

Again, we are dealing with a 2-class classification problem in a n-dimensional feature space.

$$x \in \Re^n, \quad x = (x_1, x_2, ..., x_n)$$
$$y \in \{-1, 1\} \text{ or } \{0, 1\}$$





We start with a simple model for the neurons in the central nerve system. It accumulates the input from the synapses at its dendrite and outputs 0 or 1 with a threshold function.

Hyper-plane as a classifier

A linear classifier is a simple model defined by:

$$g(x) = \langle w, x \rangle + b = \sum_{i=1}^{n} w_i x_i + b$$

Then, $g(x; w, b) = 0$

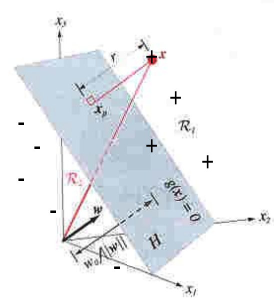
is a hyper-plane, H, in $\Re^n$. Then we define:

$$f(x) = sign\{g(x; w, b)\}$$

as a linear classifier.

Points above the plane is classified as positive and points below as negative.

W is the normal of hyper-plane H.
- b is the distance from the origin to the plane.



Hyper-plane as a classifier

Note that *H* has two degrees of freedom in its definition.

1) The normal has an arbitrary length, and thus needs to be normalized.

$$W \leftarrow \frac{W}{||W||_2} \qquad b \leftarrow \frac{b}{||W||_2}$$

2) The data points come from arbitrary scales/ranges, and needs to be normalized. We define the range as,

$$R^2 = \max_i ||x_i||^2$$

2, Perceptron: learning the hyper-plane

Given a set of training samples:

$$D = \{(x_i, y_i) : i = 1, 2, ..., m\}$$

Rosenblatt (1956) introduced an on-line "mistake-driven" algorithm for learning the optimal hyper-plane iteratively.

If the data are "linearly separable", then the algorithm is guaranteed to converge.

Definition 1: For a hyperplane $H = (\frac{W}{||W||}, \frac{b}{||W||})$, the "functional" margin of an example (x_i, y_i) is:

$$\gamma_i = y_i \cdot g(x_i) = y_i(\langle w, x_i\rangle + b_i)$$

(*We saw the margin yf(x) in boosting*)

$\gamma_i > 0$ if $g(x_i)$ and y_i have the same sign (i.e. are correctly classified)

$\gamma_i < 0$ if incorrectly classified

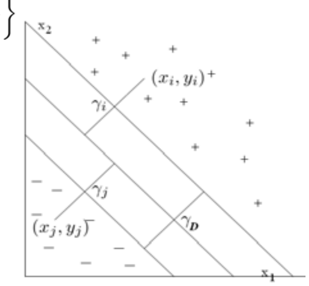
Definition of margins and linearly separable

Definition 2. The "geometric margin" of data *D* w.r.t hyper-plane *H* is,

$$\gamma_{(H,D)} = \min_i \left\{ \gamma_i = y_i(\langle \frac{w}{||W||}, x\rangle + \frac{b}{||W||}) \right\}$$

Definition 3. The margin of dataset *D* is the maximum geometric margin over all hyper-planes *H* ,

$$\gamma_{(D)} = \max_H \left\{ \min_i \{\gamma_i = y_i(\langle \frac{w}{||W||}, x\rangle + \frac{b}{||W||})\} \right\}$$



Definition 4. D is "linearly separable" if and only if the margin of D is positive, i.e. $\gamma_{(D)} > 0$

Definition 5. The empirical error of a hyper-plane is

$$Err = \frac{1}{m}\sum_{i=1}^{m} 1(\gamma_i < 0)$$

Learning the hyper-plane by gradient descent

Suppose *D* is linearly separable, we need to learn a hyper-plane that can separate it. We define an objective function, summing over all misclassified points,

$$J(H,D) = \sum_{i:\gamma_i>0} -\gamma_i = \sum_{\gamma_i<0} -y_i(\langle w, x_i\rangle + b)$$

Define the set of error point as: $\epsilon(t) = \{(x_i, y_i) : \gamma_i < 0\}$

J = 0 if and only if all the margins are larger than 0.

Our goal is to derive an algorithm that minimizes *J* by gradient descent.

$$\frac{dw}{dt} = -\eta\frac{\partial J}{\partial w} = \eta \sum_{\gamma_i<0} y_i x_i$$

$$\frac{db}{dt} = -\eta\frac{\partial J}{\partial b}R^2 = \eta \sum_{\gamma_i<0} y_i R^2$$

Remark: The normalization of ***b*** has two aspects:

1) $||W||$
2) it also depends on the number of data points, n. May define: $R^2 = \max_i ||x_i||^2$

Perceptron algorithm (primal form)

This is an "on-line" iterative algorithm.

Input: a set of training samples: $D = \{(x_i, y_i) : i = 1, 2, ..., m\}$

Initialize: $w \leftarrow 0,\ b \leftarrow 0,\ t \leftarrow 0,\ R^2 \leftarrow \max_{1\le i\le m} ||x_i||^2$

Repeat
    for i = 1 to m
        if $\gamma_i = y_i\,[\langle w, x_i\rangle + b] \le 0$ (i.e. misclassified)
        then $w \leftarrow w + \eta y_i x_i$
            $b \leftarrow b + \eta y_i R^2$
    $t \leftarrow t + 1$
until $\epsilon(t) = \{(x_i, y_i) : \gamma_i \le 0\} = \emptyset$

return $H^* = (w, b),\quad T \leftarrow t$

Perceptron algorithm continued

After convergence, we have:

$$w = \eta \sum_{i=1}^{m} \alpha_i y_i x_i$$

$$b = \eta \sum_{i=1}^{m} \alpha_i y_i R^2$$

where $\alpha_i = \sum_{t=1}^{T} 1(\gamma_i \le 0)$ is the number of times that a point is misclassified in the learning process. A larger number means more contribution to the final hyper-plane.

η, R^2 are constants which can be absorbed.

Then we obtain the final perceptron

$$\begin{aligned} g(x) &= \langle w, x \rangle + b && \text{Primal form} \\ &= \sum_{i=1}^{m} \alpha_i y_i \langle x_i, x \rangle + \sum_{i=1}^{m} \alpha_i y_i R^2 && \text{Dual form} \end{aligned}$$

Problem: it remembers all the points in D which is too much.

Perceptron Algorithm – the dual form

Theorem: If the data is linearly separable, with geometric margin γ then the number of mistakes made by the algorithm is at most:

$$||\vec{\alpha}||_1 = \sum_{i=1}^{m} \alpha_i \le \left(\frac{2R}{\gamma}\right)^2$$

This bound is perhaps the first machine learning theoretical result. Now we can rewrite the algorithm by updating α in g(x) because g(x) can be viewed as a function with either parameters *(w, b)* or *(α , b)*:

Initialize $\alpha_i \leftarrow 0,\ b \leftarrow 0,\ R^2 \leftarrow \max_i ||x_i||^2$

Repeat

for i = 1 to m

if $\gamma_i = y_i(\sum_{j=1}^{m} \alpha_j y_j \langle x_j, x_i \rangle + b) \le 0$ (misclassified)

then $\alpha_i \leftarrow \alpha_i + 1$

until $\epsilon = \emptyset$ then $b = \sum_{i=1}^{m} \alpha_i y_i R^2$

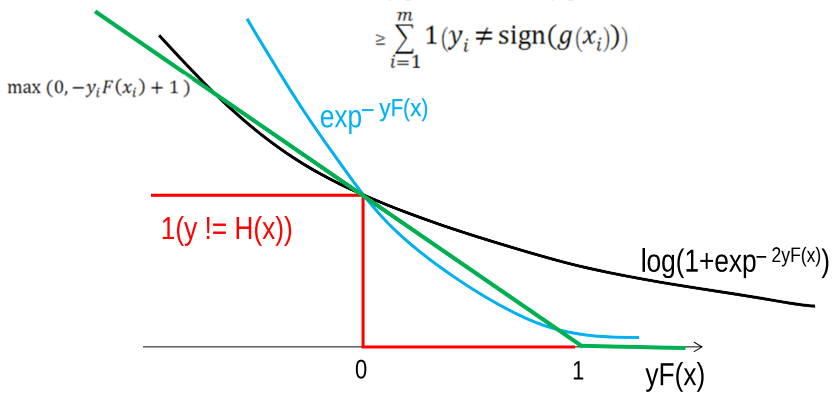
return $(\vec{\alpha}, b)$

Surrogate functions revisited

$$J(H, D) = \sum_{i:\gamma_i > 0} -\gamma_i = \sum_{\gamma_i < 0} -y_i(\langle w, x_i \rangle + b)$$

$$= \sum_{i=1}^{m} \max(0, -\gamma_i) = \sum_{i=1}^{m} \max\big(0, -y_i g(x_i)\big)$$

$$\geq \sum_{i=1}^{m} 1\big(y_i \neq \mathrm{sign}(g(x_i))\big)$$



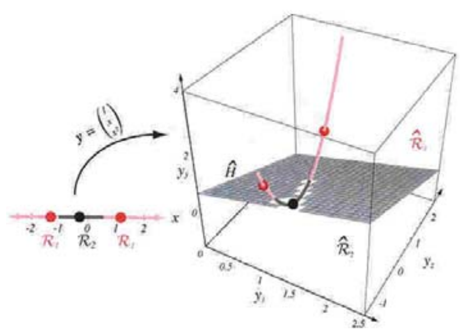
3, Kernel-induced feature space

The hyper-plane H has limited power for classification in $\boldsymbol{R}^n$. The obvious example is the xor problem.

The idea now is to map the data to a different "feature space" which is often higher dimensional. We have already called $\boldsymbol{X}$ the "feature", now we are mapping it to a new artificial feature space so that the data $\boldsymbol{D}$ will become linearly separable.

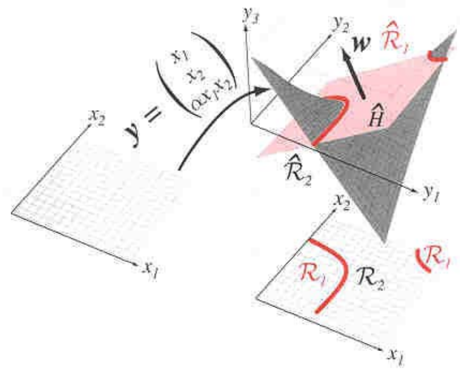
Example:

The three points in the 1D space are not separable by a line. If we map it to a 3D space, where a plane will easily separate them.



Classification in high-dimensional space

Linear classification in high-dimensional space gives non-linear boundary in the original feature space, and thus a hyper-plane is capable of generating complex decision boundaries.



Learning the perceptron in the mapped feature space

Define the mapping function $\phi(x)$:

$$\phi(x) = R^n \rightarrow R^N$$
$$x = (x_1, x_2, ..., x_n) \rightarrow \phi(x) = (\phi_1(x), \phi_2(x), ..., \phi_N(x))$$

So, the hyper-plane in the mapped feature space is

$$g(x) = \sum_{j=1}^{N} w_j \phi_j(x) + b = \langle w, \phi(x) \rangle + b$$

Following the perceptron algorithm, we have the dual form:

$$g(x) = \sum_{i=1}^{m} \alpha_i y_i \langle \phi(x_i), \phi(x) \rangle + b$$

Kernels

A kernel is a function mapping R^n x R^n to R:

$$K(x, x') = \langle \phi(x), \phi(x') \rangle \qquad \forall\, x, x' \in R^n$$

Therefore, we can rewrite the dual form of the hyper-plane H:

$$g(x) = \sum_{i=1}^{m} \alpha_i y_i K(x_i, x) + b$$

Reason for introducing the kernel:

1) We don't need to know the features $\phi_1(x), ..., \phi_N(x)$ explicitly.

2) We don't need to know the dimension N.

All we care about is the final inner-product. Thus, the computational complexity is independent of N.

Remark: In AdaBoost, we mentioned that the algorithm leaves the design of weak classifiers to the user and avoids the difficulty of learning the massive weight matrix $\boldsymbol{w_{ij}}$ in a 2-layer perceptron network. In SVM, we leave the problem to the user again to design the kernel K, which hopefully accounts for some intrinsic structure of the data D.

Mercer's Theorem

Suppose we don't design $\phi(x)$ explicitly, then what properties ensure $K(x,x')$ be a "kernel"?

Firstly, $K(x,x')$ must be symmetric:

$$K(x, x') = \langle \phi(x), \phi(x') \rangle = \langle \phi(x'), \phi(x) \rangle = K(x', x)$$

Secondly, we construct a Gram matrix for a finite set $D = \{(x_i, y_i) : i = 1...m\}$

$$G = (K_{ij})_{m \times m} \text{ with } K_{ij} = K(x_i, x_j)$$

Mercer's Theorem

$K(x,x')$ is a kernel if and only if the matrix $\boldsymbol{G}$ is positive semi-definite.

Mercer's Theorem Continued

Since G is a real symmetric matrix, we can decompose:

$G = \vee\Lambda\vee' = \sum_{i=1}^{m} \lambda_i v_i v_i'$ where $\vee = (v_1, v_2, ..., v_m)$ is an orthonormal matrix with each column an eigenvector and

$$\Lambda = \begin{pmatrix} \lambda_1 & & 0 \\ & \ddots & \\ 0 & & \lambda_m \end{pmatrix}$$

Now we can define:

$$\phi : x_i \in D \to \vec{\phi}(x_i) = (\sqrt{\lambda_1} v_{1i}, ..., \sqrt{\lambda_m} v_{mi}) = \sqrt{\vee}\vec{v}_i$$

then,

$$\langle \vec{\phi}(x_i), \vec{\phi}(x_j) \rangle = \sum_{t=1}^{m} \lambda_t v_{ti}' v_{tj} = K_{ij} = G_{ij}$$

Hilbert Space

Following the observations of the finite space, we extend to the Hilbert space of continuous functions:

K(x,x') is symmetric. It's eigenvalues are defined by: $\int K(x,z)\phi(z)\,\mathrm{d}z = \lambda\phi(x)$

K(x,x') is positive semi-definite if: $\int K(x,z) f(x) f(z)\,\mathrm{d}x\mathrm{d}z \geq 0 \quad \forall\ f(\cdot)$

The decomposition is: $$K(x,x') = \sum_{j=1}^{\infty} \lambda_j \phi_j(x)\phi_j(x')$$

We define $\phi(x) = (\phi_1(x), ..., \phi_n(x))$ therefore:

$$K(x,x') = \langle \phi \cdot \phi \rangle$$

is the inner product in Hilbert space. Thus, we have,

$$g(x) = \sum_{j=1}^{m} \alpha_j y_j K(x, x_j) + b$$

General Hyper-plane

Now we derive the general hyper-plane in the kernel induced space:

Dual form:

$$g(x) = \sum_{j=1}^{m} \alpha_j y_j K(x_j, x) + b = \sum_{j=1}^{m} \alpha_j y_j \left(\sum_{i=1}^{\infty} \lambda_i \phi_i(x_j) \phi_i(x) \right) + b$$

$H = (\alpha, b)$ $\quad \phi(x) = (\phi_1(x), ..., \phi_n(x), ...)$ are the eigen-functions which are "implicitly" defined

Primal form:

$$g(x) = \sum_{i=1}^{\infty} \lambda_i \psi_i(x) \phi_i(x) + b = \langle \psi, \phi(x) \rangle + b$$

$\psi = (\psi_1, \psi_2, ..., \psi_n, ...)$ take the role of w

$H = (\phi, b)$ $\quad \psi_i = \sum_{j=1}^{m} \alpha_j y_j \phi_j(x_j)$ is a constant.

Selection of Kernels

Some examples of kernels:

$$K(X, X') = e^{-\|X-X'\|^2 / 2\sigma^2}$$
$$K(X, X') = (X \cdot X' + 1)^p$$
$$K(X, X') = (X \cdot X')^2$$

In the 3rd example, the kernel implicitly induces a n^2-vector.

$$K(X, X') = (X \cdot X')^2 = \sum_{i,j} \sum_{m,n} x_i x_j \cdot x'_m x'_n$$

Kernels can be generated through compositions,

$$K(X, X') = aK_1(X, X') + bK_2(X, X')$$
$$K(X, X') = K_1(X, X') \cdot K_2(X, X')$$
$$K(X, X') = \exp^{K_1(X, X')}$$

Comments on using Kernels

In the practice of object recognition in vision, people rarely use kernel functions for a number of reasons.

1, Given the input image x=**I**, one can extract a large number of intuitive features, while the kernel function is less intuitive (black box);

$$x = (x_1, x_2, ..., x_n) \to \phi(x) = (\phi_1(x), \phi_2(x), ..., \phi_N(x))$$

2, The kernel function makes the computation hard: K() is computed for every data point (or each support vectors),

$$g(x) = \sum_{i=1}^{m} \alpha_i y_i K(x_i, x) + b$$

while in the prima case (linear SVM), one can sum over all the data points once off-line, and the computation is a simple product online.

$$g(x) = \sum_{i=1}^{m} \alpha_i y_i \langle \phi(x_i), \phi(x) \rangle + b$$

$$= \sum_{j=1}^{N} \beta_j \phi_j(x) + b =< \beta, \phi(x) > +b \qquad \text{where } \beta_j = \sum_{i=1}^{m} \alpha_{i,j} y_i \phi_j(x_i)$$

4, Support Vector Machines

So far, we have discussed the perceptron algorithm for learning the hyper-plane. The perceptron minimizes the following criterion (summing over the margins of all mis-classified points)

$$J(H, D) = \sum_{i:\gamma_i > 0} -\gamma_i = \sum_{\gamma_i < 0} -y_i(\langle w, x_i \rangle + b)$$

SVMs are a family of algorithms that seek for hyper-planes using a different criterion:

finding *H* so that the data has maximal margin

They use Quadratic program method to minimize the criterion. Advantages:

1, Maximizing the margin will lead to better hyper-planes;

2, We could relax the constraint that the data is linearly separable.

Margins

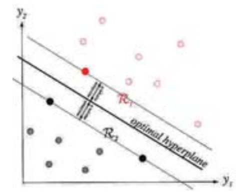
Recall: A hyper-plane can always be rescaled

$$H(w, b) = H(\lambda w, \lambda b)$$

For a linearly separable data set *D*,

the "functional margin" of *H* is the minimum:

$$\gamma_0(H = (w, b), D) = \min_i \{\gamma_i = y_i(\langle w, x\rangle + b)\}$$



The geometric margin is normalized.

$$\gamma_1(H, D) = \min_i \left\{\gamma_i = y_i(\left\langle \frac{w}{\|w\|}, x \right\rangle + \frac{b}{\|w\|})\right\}$$

$$\gamma_1(H, D) = \frac{\gamma_0}{\|w\|}$$

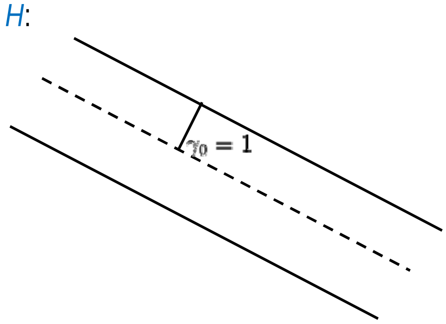
Maximal margin classifier

Suppose we scale the hyper-plane *H* so that its functional margin is always 1, then its geometric margin is always $\frac{1}{\|w\|}$.

Therefore, we derive a criterion for *H*:

$H^* = \arg\max \gamma_1$

Subject to $\gamma_0 = 1$



This is a constrained optimization problem.

Maximal margin classifier

Given $D = \{(x_i, y_i)\ i = 1 \dots m\}$

The learning problem

max margin γ = min the normal $\|w\|$

$$w^* = argmin(f(w) = \langle \vec{w} \cdot \vec{w} \rangle)$$

subject to

$$\gamma_i = y_i(\langle w \cdot x_i \rangle + b) \geq 1; i = 1 \dots m$$

We re-formulate it as,

$$\Rightarrow w^* = argmin\ \langle w, w \rangle \qquad \text{(quadratic)}$$

Subject to

$$-y_i(\langle w, x_i \rangle + b) + 1 \leq 0;\ i = 1 \dots m \qquad \text{(affine)}$$

Background: Constrained Optimization

Suppose we are minimizing (or equivalently maximizing) an objective function, subject to some equality and inequality constraints.

$$\min \quad f(w)$$
$$h_i(w) = 0, \quad i = 1, \dots, m;$$
$$g_j(w) \leq 0, \quad j = 1, \dots, k$$

Definition: a "feasible region" for *w* is a zone of Ω_w

$$\Omega_w = \{w : h_i(w) = 0\ i = 1 \dots m,\ g_j(w) \leq 0\ j = 1 \dots k\}$$

where the constraints are satisfied.

Thus the problem becomes

$$w^* = \operatorname*{argmin}_{w \in \Omega_w} f(w)$$

Background: Constrained Optimization

Solving such constrained optimization problem needs the well-known Lagrange (1797) method (for equality constraints) and Kuhn-Tucker (1951) (adding inequality constraints.)

We transform the problem to

$$w^* = \min_w L(w, \alpha, \beta) = f(w) + \sum_{i=1}^{k} \alpha_i g_i(w) + \sum_{i=1}^{m} \beta_i h_i(w)$$

$\alpha = (\alpha_1 \dots \alpha_k), \beta = (\beta_1 \dots \beta_m)$ are the "Lagrange multipliers"

Linear programming: if f(w), h_i(w), and g_i(w) are linear

Quadratic programming: if f(w) is quadratic and h_i(w), and g_i(w) i= 1…k are linear

Background: Constrained Optimization

Theorem (Kuhn-Tucker) , for L(w, α, β), if f(w) is convex wrt. w, g_i(w) and h_i(w) are affine, i.e. in the form of g(w) or h(w) = Aw+b Then the sufficient and necessary conditions for w* are the existence of α^*, β^* such that:

$$\begin{cases} \frac{\partial L(w^*, \alpha^*, \beta^*)}{\partial w} = 0 & \\ \frac{\partial L(w^*, \alpha^*, \beta^*)}{\partial \beta} = 0 & \\ \alpha_i^* g_i(w^*) = 0 & i = 1 \dots k \\ g_i(w^*) \leq 0 & i = 1 \dots k \\ \alpha_i \geq 0 & i = 1 \dots k \end{cases}$$

------ supplementary condition

$$\alpha_i = 0 \text{ if } g_i(w) < 0 \text{ "Inactive"}$$
$$\alpha_i > 0 \text{ if } g_i(w) = 0 \text{ "Active"}$$

Maximal margin classifier

Following the maximal margin formulation, we transform the constrained optimization problem

$$\Rightarrow w^* = argmin \langle w, w \rangle \qquad \text{(quadratic)}$$

Subject to

$$-y_i(\langle w, x_i \rangle + b) + 1 \leq 0; \; i = 1 \ldots m \qquad \text{(affine)}$$

into the problem below.

the primal Lagrangian is

$$L(\underbrace{w, b}_{\text{Prima variables}}, \underbrace{\alpha}_{\text{Dual variables}}) = \frac{1}{2} \langle w, w \rangle - \sum_{i=1}^{m} \alpha_i \left[y_i(\langle w, x_i \rangle + b) - 1 \right]$$

Prima variables

Dual variables

Maximal margin classifier

According to Kuhn-Tucker theorem, we solve for the prima variables by

$$\left. \begin{array}{l} \dfrac{\partial L}{\partial w} = 0 \\ \dfrac{\partial L}{\partial b} = 0 \end{array} \right\} \Rightarrow \begin{array}{l} w = \sum_{i=1}^{m} \alpha_i y_i \cdot \vec{x}_i \\ \sum_{i=1}^{m} y_i \alpha_i = 0 \end{array}$$

Plug in w in the primal Lagrangian,

$$L(w, b, \alpha) = \frac{1}{2} \langle \vec{w}, \vec{w} \rangle - \sum_{i=1}^{m} \alpha_i \left[y_i(\langle \vec{w}, \vec{x}_i \rangle + b) - 1 \right]$$

$$= \frac{1}{2} \sum_{i=1}^{m} \sum_{j=1}^{m} \alpha_i \alpha_j y_i y_j \langle \vec{x}_i, \vec{x}_j \rangle - \sum_{i=1}^{m} \alpha_i y_i \left(\sum_{j=1}^{m} \alpha_j y_j \langle \vec{x}_j, \vec{x}_i \rangle \right) - \sum_{i=1}^{m} \alpha_i y_i b + \sum_{i=1}^{m} \alpha_i$$

Then we get the dual form

$$Q(\alpha) \stackrel{\text{def}}{=} \sum_{i=1}^{m} \alpha_i - \frac{1}{2} \sum_{i=1}^{m} \sum_{j=1}^{m} \alpha_i \alpha_j y_i y_j \underbrace{\langle \vec{x}_i, \vec{x}_j \rangle}_{\text{Known constants}}$$

Known constants

Maximal margin classifier

Now as a dual problem, we solve $\alpha^* = argmin\ Q(\alpha)$

above still quadratic w.r.t. α

subject to $\sum_{i=1}^{m} y_i \alpha_i = 0$

$\alpha \geq 0 \ \ for\ i = 1 \ldots m$

$\alpha = (\alpha_1 \ldots \alpha_m)$ is the dual variable.

Solving this problem then we have

$$w^* = \sum_{i=1}^{m} \alpha_i y_i \vec{x}_i$$

$$b^* = -\frac{\max_{y_i=-1}(\langle w^*, x_i\rangle) + \min_{y_i=+1}(\langle w^*, x_i\rangle)}{2}$$ (by definition)

Maximal margin classifier

By the Kuhn-Tucker theorem, we have the conclusion that at the optimum $\alpha^*\ w^*\ b^*$

we have the supplementary condition equation $\alpha_i^* g_i^*(w, b) = 0$

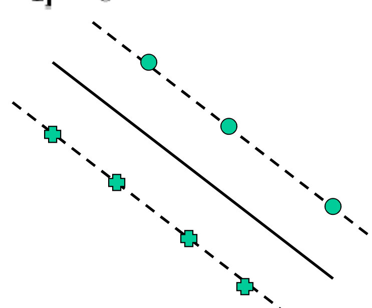
i.e. $$\alpha_i^* \left[(y_i(w^* x_i) + b^*) - 1 \right] = 0$$

Thus for data points not exactly on the marginal planes, we have

$$y_i(w^* x_i) + b^* - 1 \neq 0$$

Therefore $\alpha_i = 0$

$$w^* = \sum_{\alpha \neq 0} \alpha_i y_i \vec{x}_i$$



Note: in an n-dimensional space, we need $\geq n$ points to define (support) a hyper-plane.

Support vectors

Definition: the support vectors are data points on the two hyper-planes

$$SV = \{\langle x_i, y_i\rangle : y_i(\langle w^*, x_i\rangle + b) = 1\}$$

$$w^* = \sum_{i \in SV} \alpha_i y_i \vec{x}_i$$

Only the support vectors contribute to w^* instead of the massive m.

Recall that the perceptron solution $w^* = \sum_{i=1}^{m} \alpha_i y_i \vec{x}_i$ is the same, but it was an iterative solution for α .

Support vector machine

In summary, the maximal margin classifier leads to a hyper-plane.

primal $g(x : w, b) = \langle w, x\rangle + b$

or

dual $g(x : a, b) = \sum_{i \in SV} \alpha_i y_i \langle x_i, x\rangle + b$

This is very similar to the perceptron,

again we have the kernel for $\langle x_i, x\rangle$

Support vector machine

Furthermore, we want to know the maximum margin at w^*

$$\gamma^*_{max} = \frac{1}{\|w^*\|_2}$$

Plug in $w^* = \sum_{i \in SV} \alpha_i y_i \vec{x}_i$ and we have

$$\langle w^*, w \rangle = \sum_{i \in SV} \alpha_i y_i \sum_{j \in SV} \alpha_j y_j \langle x_i x_j \rangle$$

Using two conclusions from Kuhn-Tucker

i. for $i \in SV$, we have $\gamma_i = y_i (\langle w^*, x_i \rangle + b^*)$

$$\therefore y_i \left[\sum_{j \in SV} \alpha^*_j y_j \langle x_i, x_j \rangle + b \right] = 1$$

ii. $\sum_{i \in SV} \alpha^*_i y_i = 0$

The maximum margin

Then $\langle w^* \cdot w^* \rangle = \sum_{i \in SV} \alpha^*_i y_i \sum_{j \in SV} \alpha_j y_j \langle x_i y_i \rangle$

$$= \sum_{i \in SV} \alpha^*_i (1 - y_i b^*) \quad = \sum_{i \in SV} \alpha^*_i > 0$$

Proposition: for Dataset $D = \{(x_i, y_i)\, i = 1 \dots m\}$

Let α^*, b^* be the solution of the dual problem, then

$w^* = \sum_{i \in SV} \alpha_i y_i \vec{x}_i$ realizes the maximum margin hyper-plane

with geometric margin $\gamma^* = \frac{1}{\|w\|_2} = \frac{1}{\sqrt{\sum_{i \in SV} \alpha_i}}$

Drawback: we still assume the data D is linearly seperable.

Support vector machine in kernel induced space

Now we can easily extend the SVM on X-space to the ϕ-feature space, or the feature space induced by Kernel K

$$K(x, x') \to (\phi_1(x), \ldots \phi_n(x), \ldots)$$

Recall the SVM (max. margin classifier) – dual problem

$$(\alpha^*) = argmin \underbrace{\sum_{i=1}^{m} \alpha_i - \frac{1}{2} \sum_{i=1}^{m} \sum_{j=1}^{m} \alpha_i \alpha_j y_i y_j \langle x_i, x_j \rangle}_{Q(\alpha)}$$

The following proposition summarizes the result.

Support vector machine in kernel induced space

Proposition:

Given $D = \{(x_i, y_i)\ i = 1 \ldots m\}$ which is linearly separable in the features space implicitly defined by a Kernel $K(x, x')$

Suppose α^*, b^* solves the following quadratic maximization problem

$$\alpha^* = argmax \sum_{i=1}^{m} \alpha_i - \frac{1}{2} \sum_{i=1}^{m} \sum_{j=1}^{m} \alpha_i \alpha_j y_i y_j K(x_i, x_j)$$

subject to $\sum_{i=1}^{m} y_i \alpha_i = 0$ // form $\frac{\partial L}{\partial b} = 0$

$\alpha_i \geq 0$ // from Kuhn-Tucker

Support vector machine in kernel induced space

Then the design rule

$$H(x) = sign(\sum_{i=1}^{m} y_i \alpha_i K(x, x_i) + b^*)$$

is equivalent to the max margin hyper-plane in the feature space implicitly defined by Kernel $K(x, x')$, that hyper-plane (in the feature space) has geometric margin

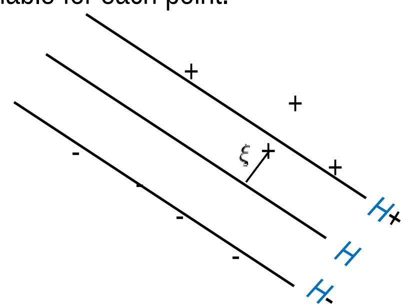
$$\gamma^* = \frac{1}{\|w\|_2} = \frac{1}{\sqrt{\sum_{i \in SV} \alpha_i}}$$

Soft margin maximization

Max margin classifier is a simple SVM, a main drawback is the assumption that the data is linearly separable.

This leads to over-fitting (when the data contains noise and outliers and are not linearly separable).

We introduce a slack variable for each point.



Soft margin maximization

The criterion becomes $min \langle w, w \rangle + c \sum_{i=1}^{m} \xi_i^2$

subject to $y_i [\langle w, x_i \rangle + b] \geq 1 - \xi_i$

1. Here $\xi_i \geq 0$, the case of $\xi_i < 0$ is penalized by minimizing ξ_i^2
2. Parameter c is selected in a large range through cross validation for reaching smaller testing errors.

Soft margin maximization

The primal Lagrangian is

$$L(w, b, \xi, \alpha) = \frac{1}{2} \langle w, w \rangle + \frac{c}{2} \sum_{i=1}^{m} \xi_i^2 - \sum_{i=1}^{m} \alpha_i \left[y_i (\langle w, x_i \rangle + b) - 1 + \xi_i \right]$$

$$\frac{\partial L}{\partial w} = 0 \Rightarrow \vec{w} = \sum_{i=1}^{m} y_i \alpha_i \vec{x}_i$$

$$\frac{\partial L}{\partial \xi} = 0 \Rightarrow c \cdot \xi = \alpha$$

$$\frac{\partial L}{\partial b} = 0 \Rightarrow \sum_{i=1}^{m} y_i \alpha_i = 0$$

Soft margin maximization

Then we obtain the dual form by plugging in w etc

$$L(w,b,\xi,\alpha) = \sum_{i=1}^{m} \alpha_i - \frac{1}{2}\sum_{i=1}^{m}\sum_{j=1}^{m} y_i y_j \alpha_i \alpha_j \langle \vec{x_i}, \vec{x_j} \rangle - \frac{1}{2c}\langle \vec{\alpha}, \vec{\alpha} \rangle$$

$$= \sum_{i=1}^{m} \alpha_i - \frac{1}{2}\sum_{i=1}^{m}\sum_{j=1}^{m} [y_i y_j \alpha_i \alpha_j (\langle \vec{x_i}, \vec{x_j} \rangle) + \frac{1}{c}\delta_{ij}]$$

$$\delta_{ij} = 1 \; if \; i = j$$

By Kuhn-Tucker theorem:

$$\alpha_i [y_i (\langle w, x_i \rangle + b) - 1 + \xi_i] = 0 \;, \; \forall i$$

Soft margin maximization

Proposition: Give training data $D = \{(x_i, y_i)\, i = 1 \dots m\}$ with feature space implicitly defined by a Kernel $K(x, x')$, solving the dual problem w.r.t. α.

$$min L(\alpha) = \sum_{i=1}^{m} \alpha_i - \frac{1}{2}\sum_{i=1}^{m}\sum_{j=1}^{m} y_i y_j \alpha_i \alpha_j \left[K(x_i, x_j) + \frac{1}{c}\delta_{ij}\right]$$

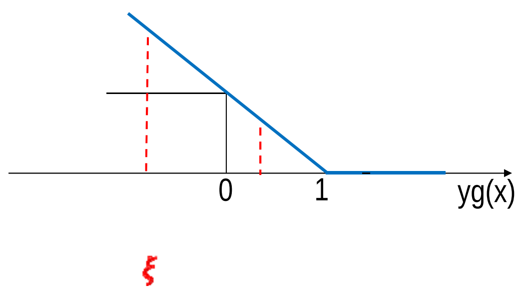
Subject to $\sum_{i=1}^{m} y_i \alpha_i = 0 \qquad \alpha_i \geq 0 \,, i = 1 \dots l$

Then at $\alpha^0 = \alpha^*$

the margin $\gamma = \frac{1}{\|w^*\|_2} = \frac{1}{\sqrt{\sum_{i \in SV} \alpha_i^* - \frac{1}{c}\langle \alpha^*, \alpha^* \rangle}}$

Statistical perspective on the SVM

The SVM: $min \langle w, w \rangle + c \sum_{i=1}^{m} \xi_i^2$ subject to $y_i [\langle w, x_i \rangle + b] \geq 1 - \xi_i$



Statistical perspective on the SVM

$< \omega, \omega > = |\omega|_2$ is just a regularizor penalizing model complexity.

Other regularizor: $|\omega|_1$ — Lasso regression (Least Absolute Shrinkage and Selection Operator).

Other regressors in the statistical literature: ridge regression, group lasso etc.

Struct-SVM

The SVM methods are also used in two other ways.

1, Tuning (learning) parameters in an inference problem that maximizing a score, i.e. computing the optimal solution from input x:

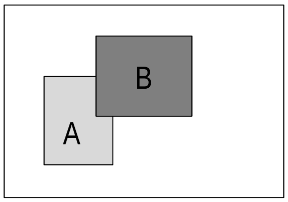
$$\widehat{pg} = \text{argmax} < \omega, \phi(pg|\, \text{x}) >$$

For example, x is an input image, and pg is a parse graph
--- the structured output as we discussed in syntactic pattern recognition.
Now, suppose the output of the algorithm is compared to a ground truth annotation, pg^*. This means the current parameter ω needs to be adjusted such that

$$< \omega, \phi(pg^*|\, \text{x}) > \; - < \omega, \phi(pg|\, \text{x}) > \;\geq\; 0, \qquad \forall pg$$

Rank SVM

2, In some applications, the input to us is ranked pairs, e.g. B is better than A.



We need to learn the parameters to satisfy those ranked pairs:

$$< \omega, \phi(B) > \; - < \omega, \phi(A) > \;\geq\; 0, \qquad \forall (A,B) \text{ pairs}$$

$\phi(\quad)$ is a vector of feature extracted from A or B, or *pg*. We will see the example in project 3.

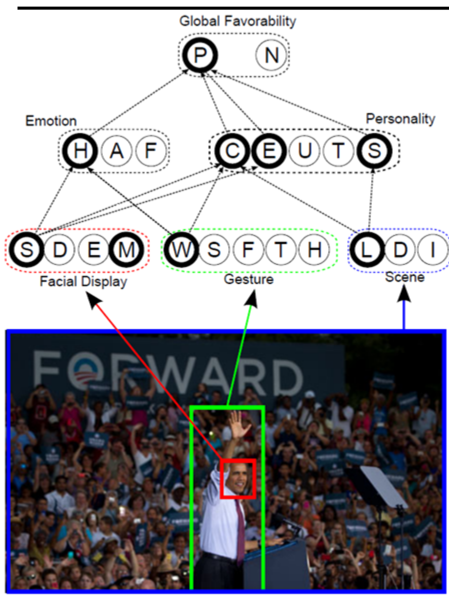
Project 3:

Face social attributes and Political Election Analysis by SVM

This project is created from:

Jungseock Joo et al, “Automated Facial Trait Judgment and Election Outcome Prediction: Social Dimensions of Face,” *Int’l Conf. on Computer Vision*, 2015.

Overview



Communication intent and Favorability

Rank SVMs

Social dimensions and Attributes

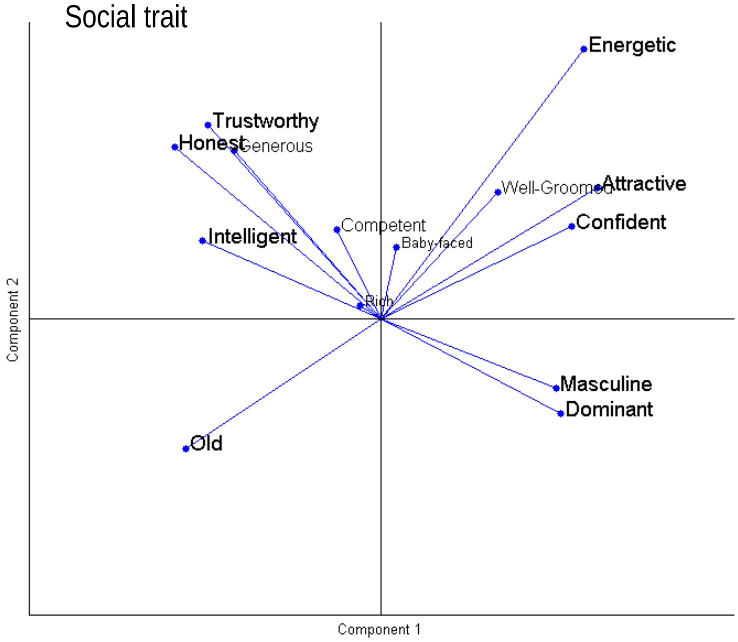
Binary SVMs

Input Image
Low-level features

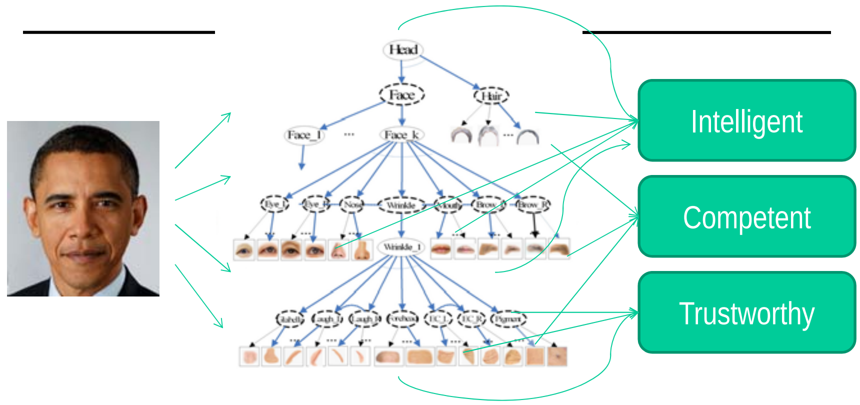
Judgment by Impression from Face

Which person is the more competent?

Social trait



Automated Trait Inference



Finer-grained feature analysis

- Facial feature, shape, attribute decomposition.
- Reverse-engineering of social perception.
- "What makes for a competent face?"

X_i : Image feature

For each intent dimension, we learn :

$\vec{w}$: Intent model params (Ranking SVM)

$$\text{Minimize}: \ \frac{1}{2}\|\vec{w}\|_2^2 + C\sum \xi_{i,j}$$

$$\text{subject to}: \ \langle \vec{w}, F_i \rangle \geq \langle \vec{w}, F_j \rangle + 1 - \xi_{i,j},$$

$$\xi_{i,j} \geq 0, \quad \forall (i,j) \in D$$

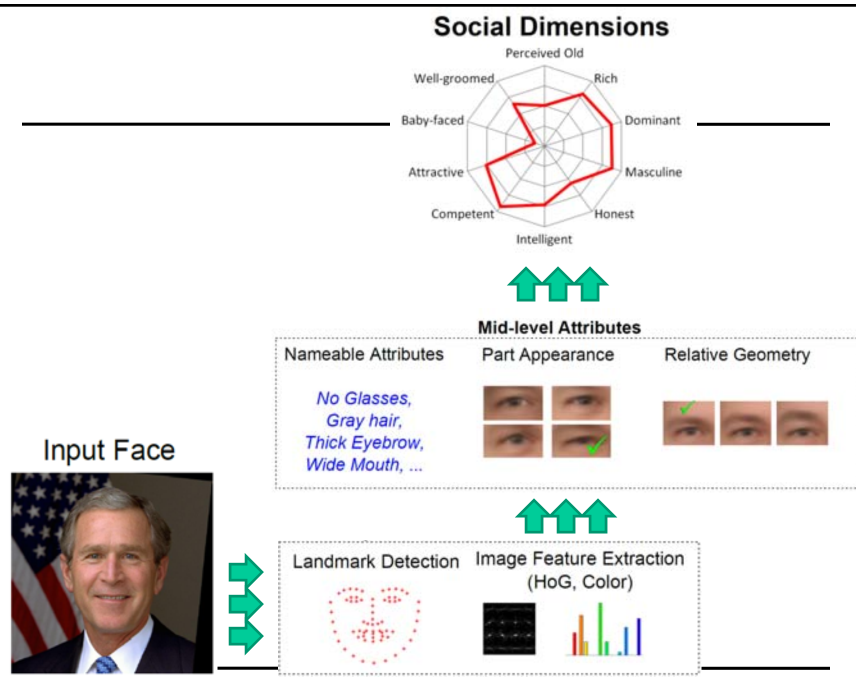
Dataset

550 facial photographs of US politicians.

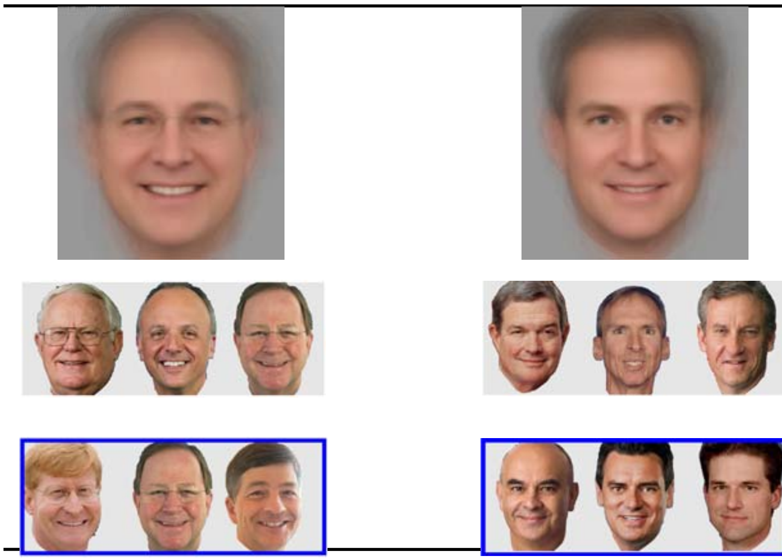
- No background, No clothing, smile, white (Caucasian)

14 trait annotations by pair-wise comparisons

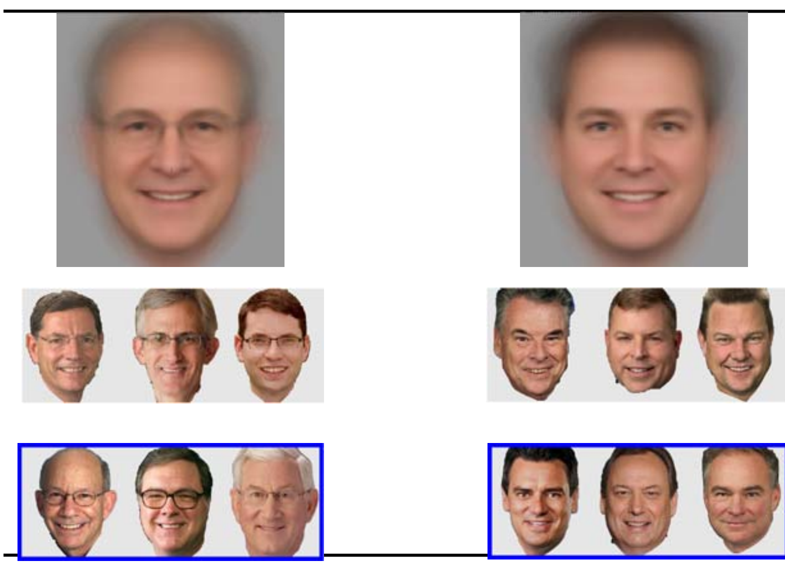
- Amazon Mechanical Turk
- Age independent



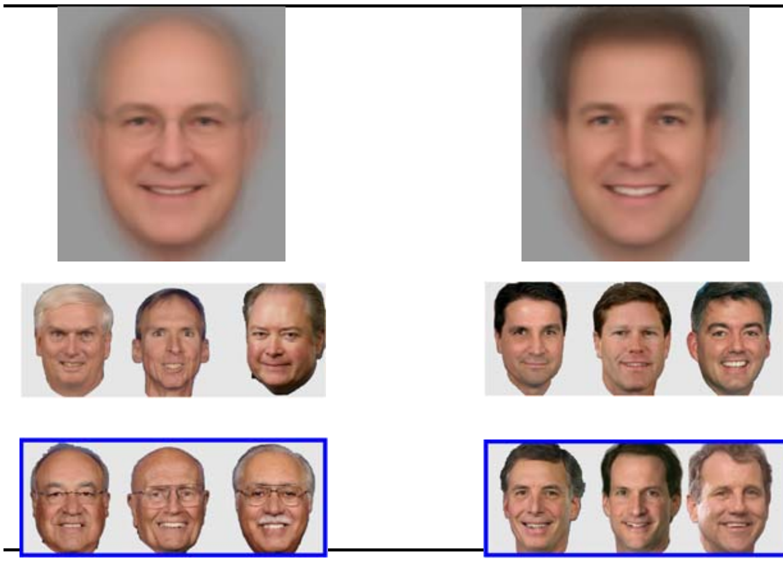
Generous



Intelligent

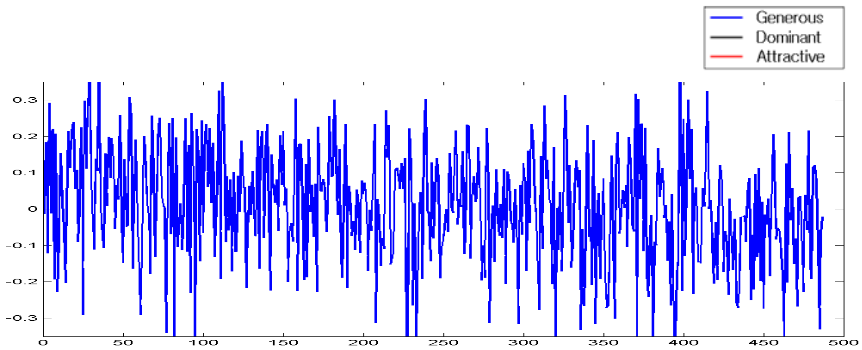


Not Attractive

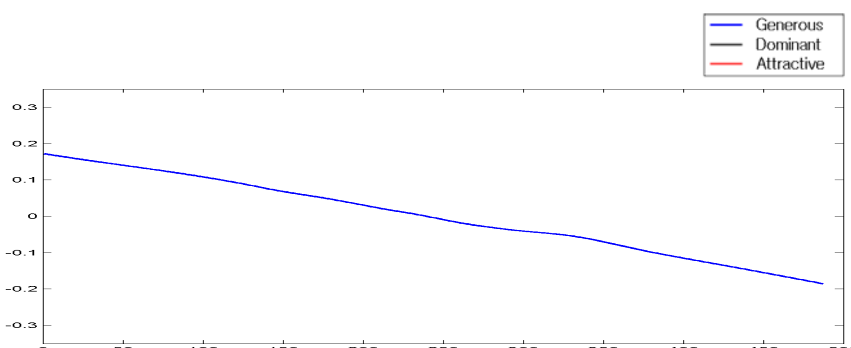


Distance between eyelid and eyebrow

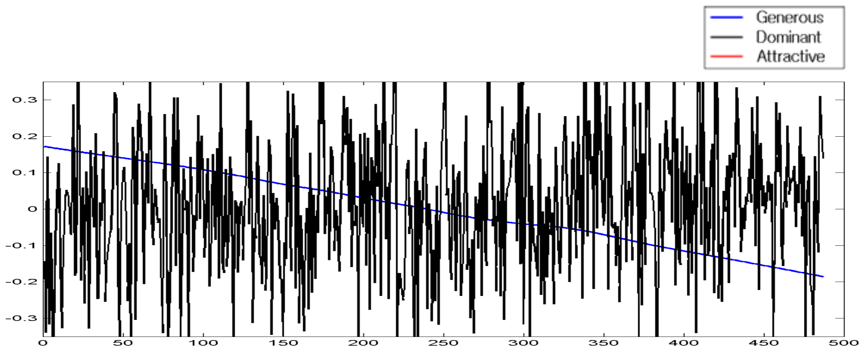
Distance between eyelid and eyebrow



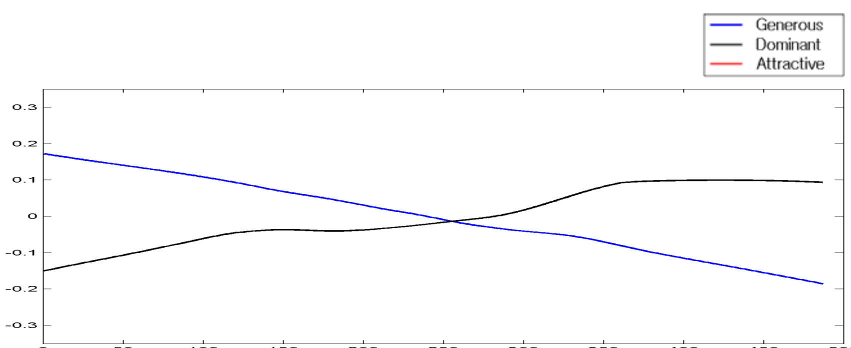
Distance between eyelid and eyebrow



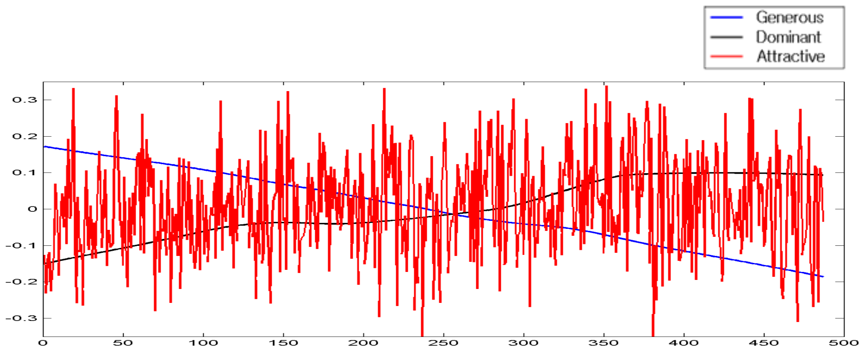
Distance between eyelid and eyebrow



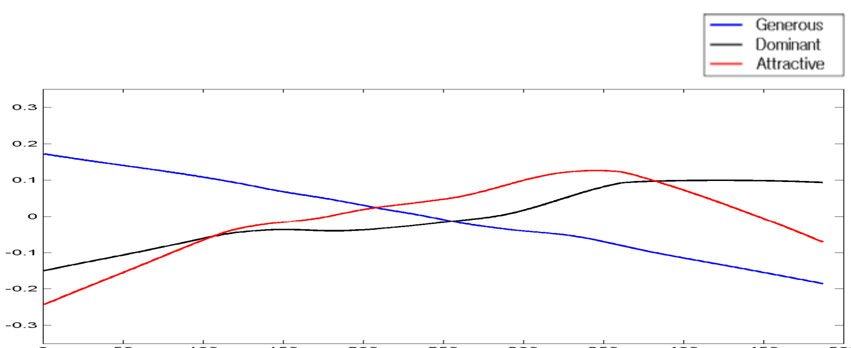
Distance between eyelid and eyebrow



Distance between eyelid and eyebrow



Distance between eyelid and eyebrow



Election Prediction from Traits

Using voting share differences

$F_i = [f_i^1, \ldots, f_i^K]$: A trait vector (from Ranking SVMs)

To learn :

W : Election prediction model (another Ranking SVM).

$$\text{Minimize}: \ \frac{1}{2}\|W\|_2^2 + C \cdot e_{i,j} \sum \xi_{i,j}$$

$$\text{subject to}: \ \langle W, F_i \rangle \geq \langle W, F_j \rangle + 1 - \xi_{i,j},$$

$$\xi_{i,j} \geq 0, \quad \forall (i, j) \in D,$$

$e_{i,j}$: Vote share difference in each race

Winning Traits

Traits	Governor ($n = 122$) r	p-value	Senator ($n = 110$) r	p-value
Confident	.434	$< .0001$		
Dominant	.396	$< .0001$		
Energetic	.354	$< .0001$	-.198	0.03
Attractive	.337	.0002		
Masculine	.325	.0003		
Well-groomed	.206	.01		
Competent			.289	.001
Rich			.338	.0002
Perceived Old	-.174	.05	.198	.04
Intelligent	-.214	.01	.228	.01
Trustworthy	-.231	.01		

* Elections from 2000 - 2014

Winning Features

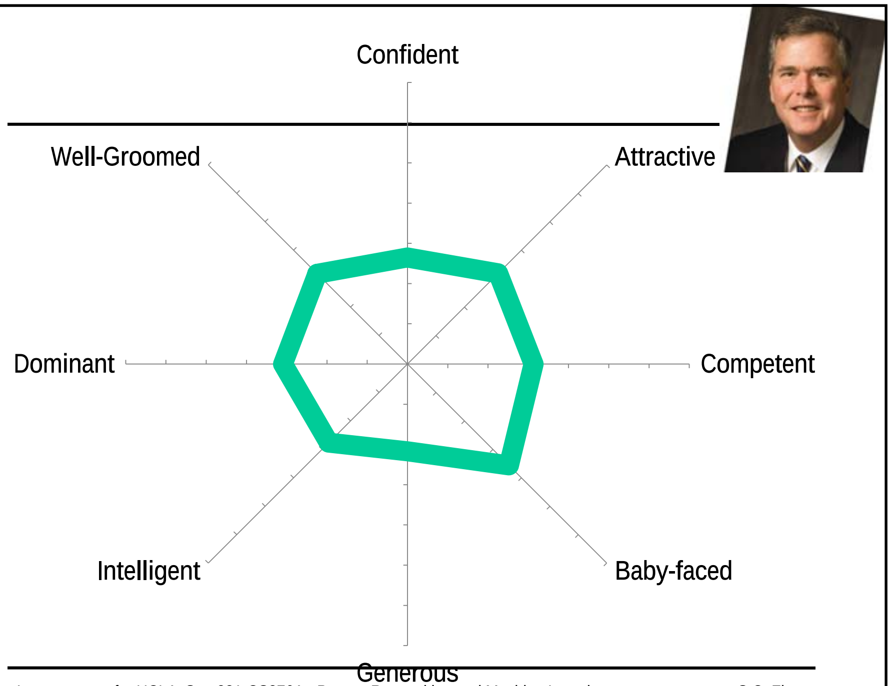
Traits	Governor ($n = 122$)		Senator ($n = 110$)	
	r	p-value	r	p-value
Eye size	.234	(.01)	-.165	(.07)
Eye width	.292	(.001)		
Distance between eyes	-.259	(.004)		
Eye slope	.220	(.01)	-.205	(.02)
Mouth size	.211	(.01)	-.339	(.0001)
Lip thickness			-.358	(.0001)
Tall face			-.234	(.01)

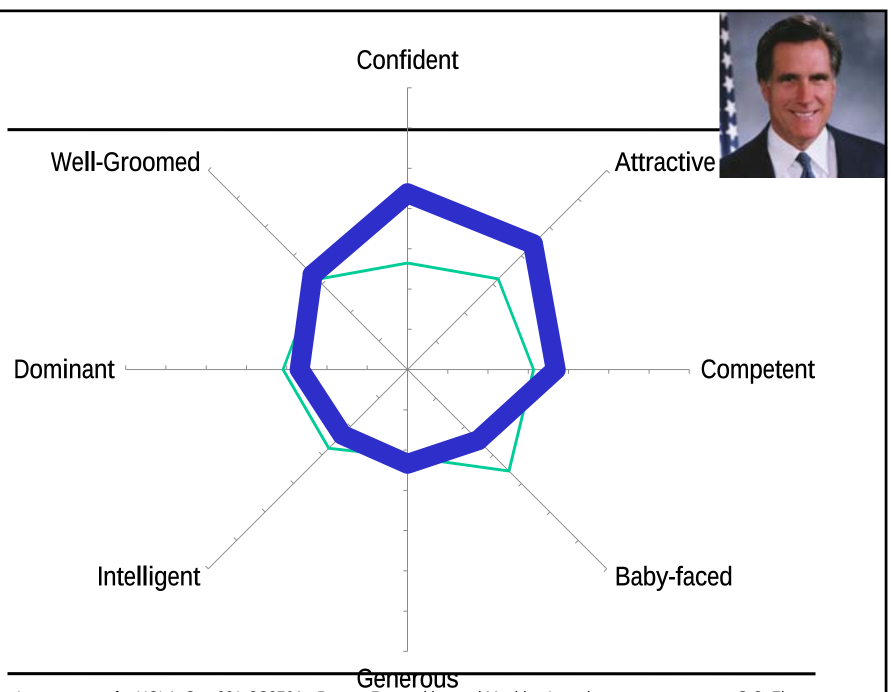
* Elections from 2000 - 2014

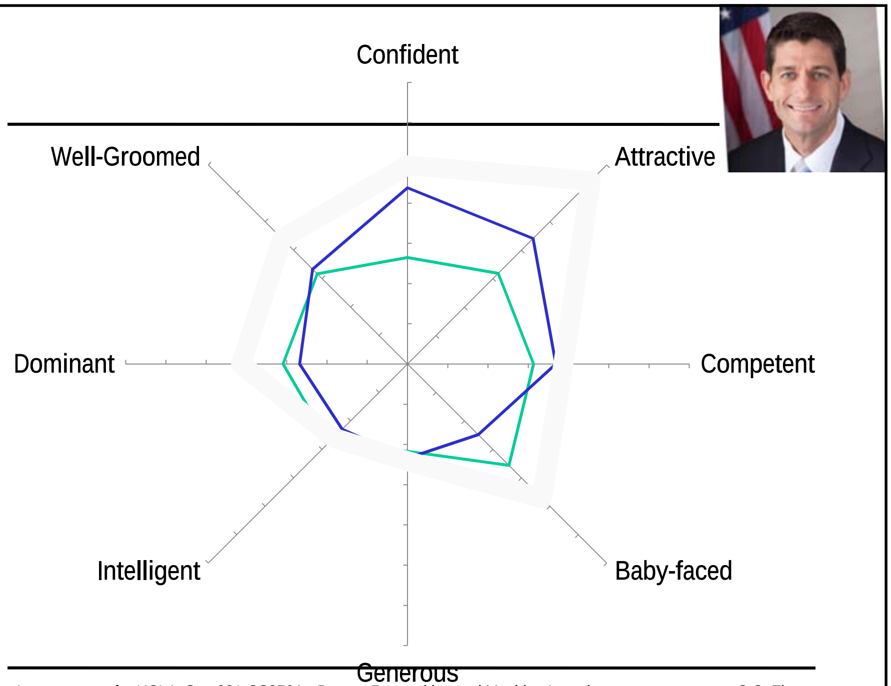
DEM vs GOP

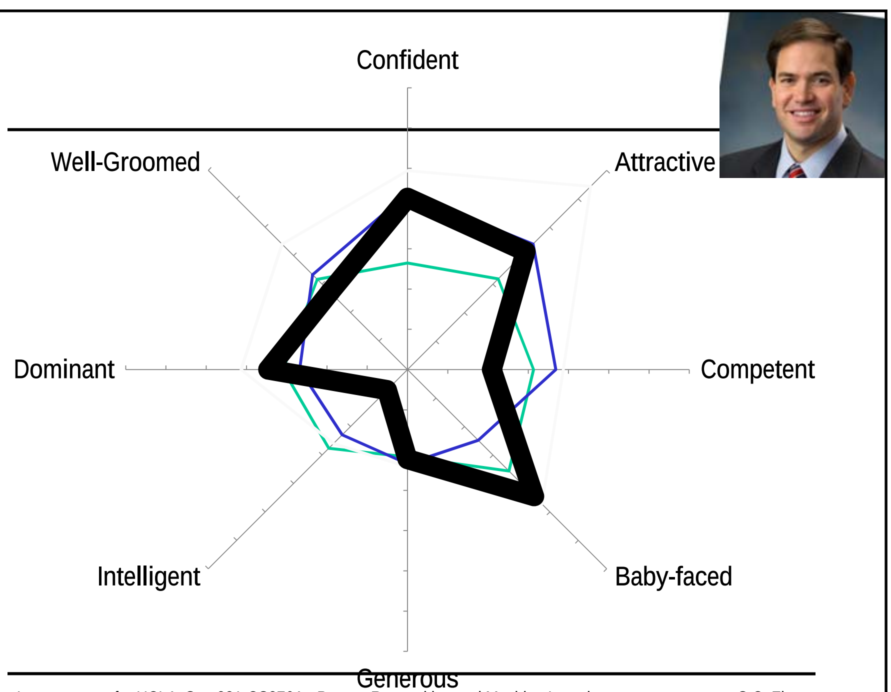
Traits	Whole Set ($n = 491$)		Winner Set ($n = 343$)	
	r	p-value	r	p-value
Intelligent	.155	(.0006)	.199	(.0002)
Perceived Old	.113	(.01)	.160	(.003)
Attractive	-.110	(.01)	-.105	(.05)
Babyfaced	-.106	(.01)	-.143	(.008)
Competent	.096	(.03)	.147	(.006)

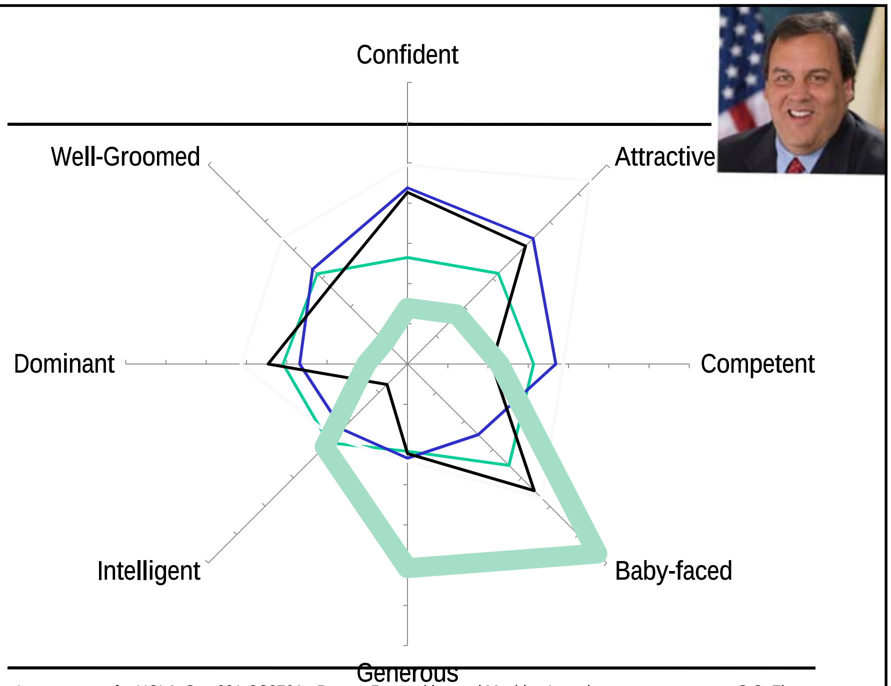
* Positive correlations: more Democratic.

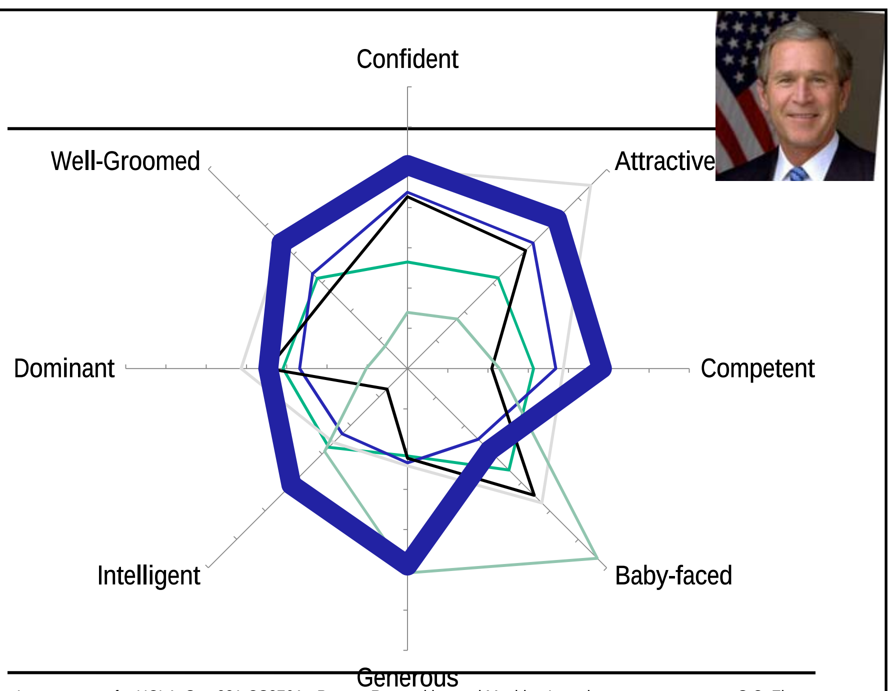

Confident
Well-Groomed
Attractive
Dominant
Competent
Intelligent
Baby-faced
Generous


Confident
Well-Groomed
Attractive
Dominant
Competent
Intelligent
Baby-faced
Generous


Confident
Attractive
Competent
Baby-faced
Generous
Intelligent
Dominant
Well-Groomed


Confident
Attractive
Competent
Baby-faced
Generous
Intelligent
Dominant
Well-Groomed


Confident
Well-Groomed
Attractive
Dominant
Competent
Intelligent
Baby-faced
Generous


Confident
Well-Groomed
Attractive
Dominant
Competent
Intelligent
Baby-faced
Generous