# Homework

- Project: CoNLL 2004 shared task

- Homework for next week from today:
  - ⋆ Write a program which calculates the probability of each verb sense from the training data
  - ⋆ Don’t rely on the verb sense tags, though you can use them for debugging
  - ⋆ Turn in program, plus sense probabilities for *exchange* and *feel*

- Item read chapter 1 of *Learning with kernels* at http://www.learning-with-kernels.org/

# Perceptron

- Rosenblatt's perceptron algorithm constructs a linear decision boundary (*separating hyperplane*)
- For each misclassified training example, update weights (primal form):

$$w \leftarrow w + \eta(y_i - \hat{y}_i))x_i$$

- This is a form of gradient descent to minimize the negative margin of misclassified points:

$$\hat{w} = \operatorname*{argmin}_{w} - \sum_{i \in \mathcal{M}} y_i \left(x_i \cdot w\right)$$

# Perceptron

- The functional margin $\gamma$:

$$\gamma = \min_i y_i(x_i \cdot w)$$

  is non-negative if the data is separated by the hyperplane $w$, and the larger $\gamma$ is, the greater the separation

- Novikoff (1962): Suppose some weight vector $w_0$ (where $||w|| = 1$) correctly classifies all examples in the training set with margin $\gamma$, and $R = \max_i ||x_i||$. Then the number of corrections made by the perceptron algorithm is at most:

$$\left(\frac{2R}{\gamma}\right)^2$$

- The difficulty of learning a concept depends on the pattern length divided by the margin

# Perceptron

- The perceptron algorithm suffers from a few serious problems
- If the training data is not separable (i.e., $\gamma < 0$), it will not converge to a solution
- If the margin $\gamma$ is very small, convergence will be very slow
- When the margin is large, the solution is not unique and will depend on the starting conditions
- The first two problems can be addressed by increasing the dimensionality of the feature space to improve separation

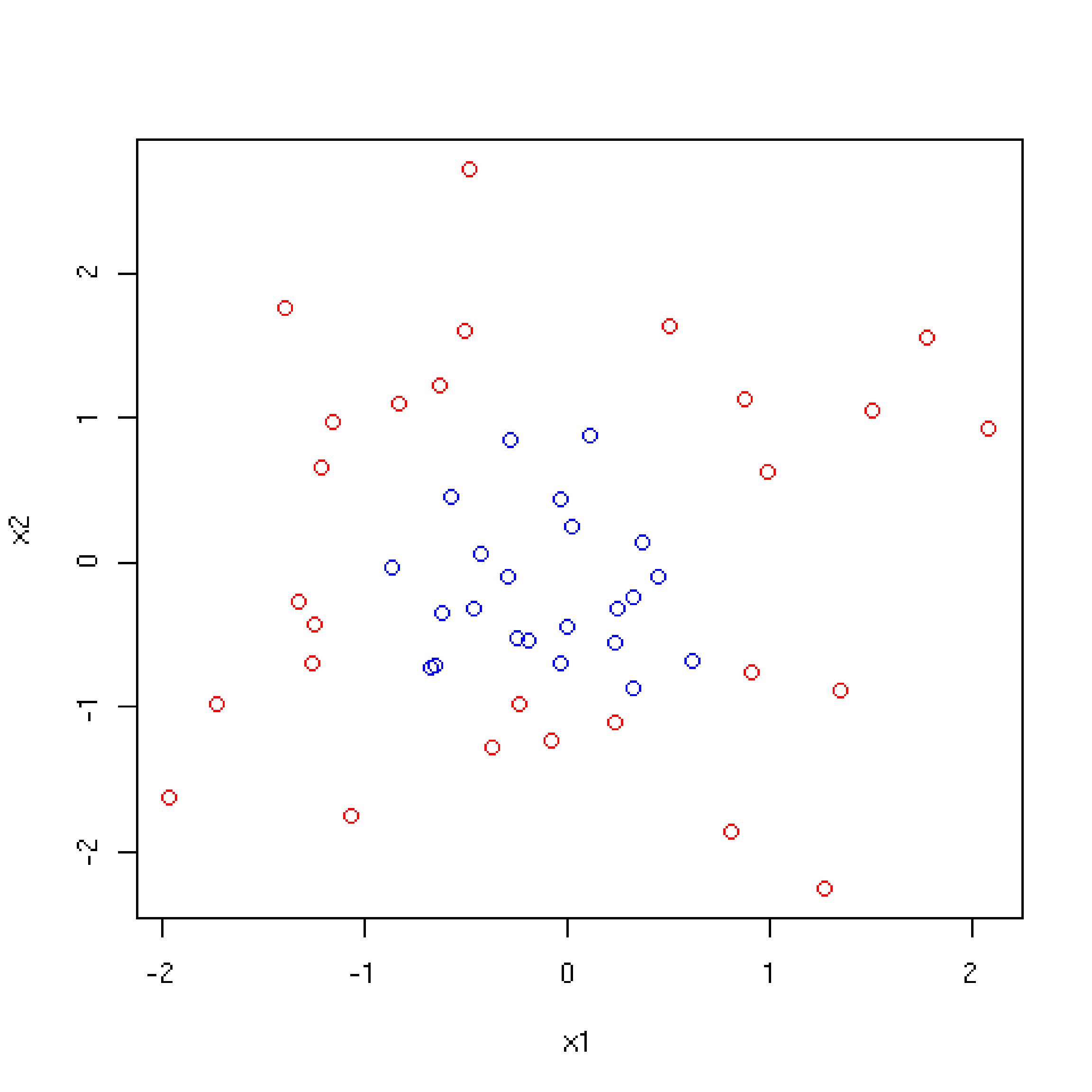
# Feature spaces

- For example, we could expand a two dimensional input space to a six-dimensional polynomial feature space:

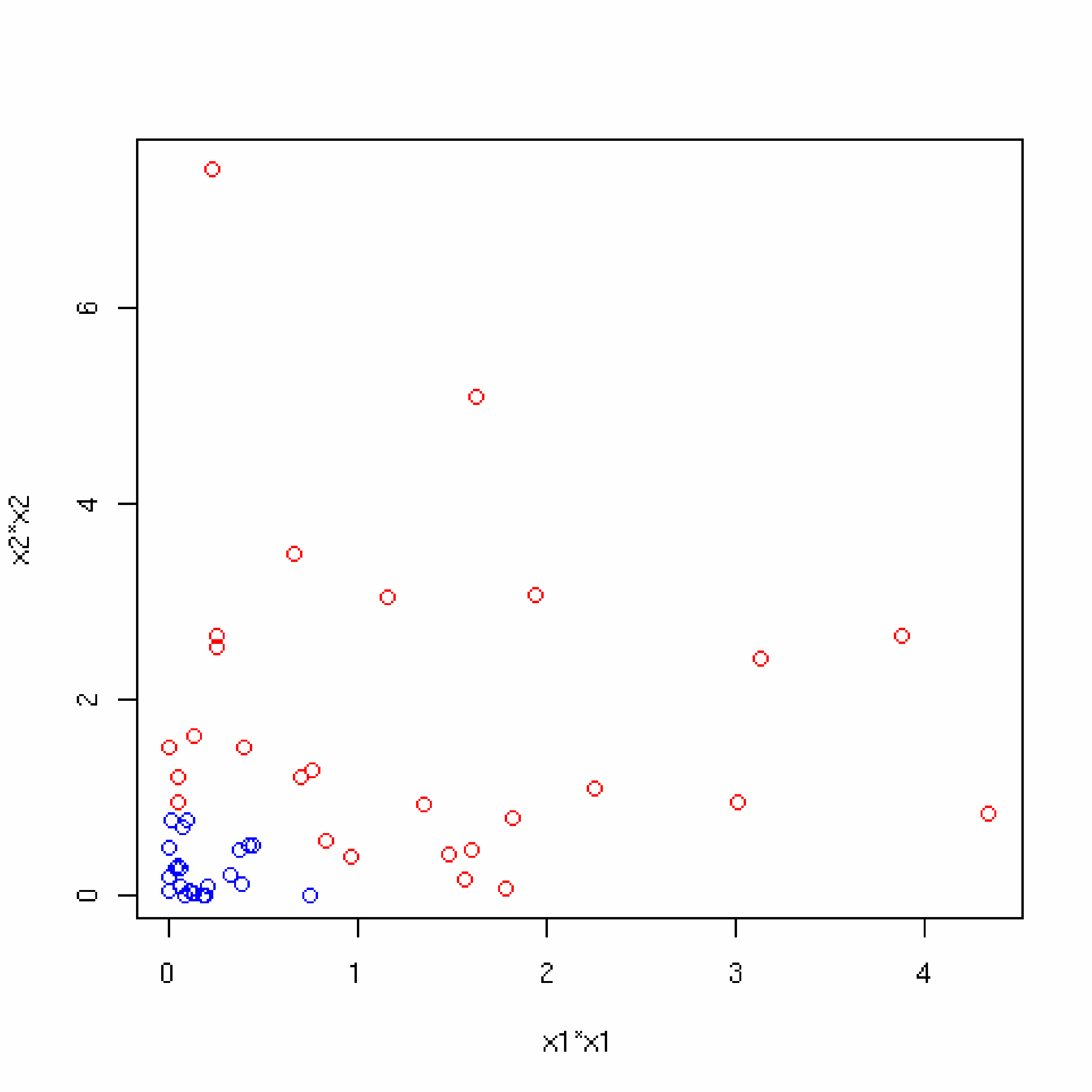
$$\Phi : (x_1, x_2) \longrightarrow (x_1, x_2, x_1\,x_2, x_2\,x_1, x_1^2, x_2^2)$$

- A separating hyperplane in this feature space corresponds to a 2nd degree polynomial decision boundary in the input space
- By using the perceptron algorithm in this derived feature space, we can find a non-linear boundary in the input space
- While increasing the dimensionality generally improves separation, it can also introduce it’s own problems

# Feature spaces



# Feature space



# Feature spaces

- One drawback to this is computational: our our feature vectors and weight will get very long in a hurry

- A solution is to use the *dual form* of the perceptron algorithm, which represents the decision boundary by the embedding strength of the training examples:

$$w = \sum_i \alpha_i \, y_i \, x_i$$

and the decision boundary is:

$$\hat{y} = \mathrm{sign}(\sum \alpha_i \, y_i \, (x_i \cdot x))$$

# Kernel functions

- Now the training data only enters into our calculations via dot products $(x_i \cdot x_j)$

- For certain feature spaces, the dot product can be computed efficiently without actually constructing the complete feature vectors

- And, it turns out that a broad class of *kernel functions* $K(x_i, x_j)$ are dot products in some (possibly $\infty$-dimensional) feature space

- Representing kernel Hilbert spaces (RKHS)

- The dual form of the perceptron algorithm can use any of these kernel functions in place of the dot product:

$$\hat{y} = \mathrm{sign}(\sum \alpha_i \, y_i \, K(x_i, x))$$

# Kernel functions

- Linear kernel

$$K(x_i, x_j) = (x_i \cdot x_j)$$

- Polynomial kernel

$$K(x_i, x_j) = (x_i \cdot x_j)^d$$

- Radial basis function (RBF) kernel

$$K(x_i, x_j) = \exp\left(\frac{-|x_i - x_j|^2}{\sigma^2}\right)$$

- Kernels are a kind of similarity measure

# Kernel functions

- Other kernel functions are used occasionally, but these are the most important
- Linear kernels are very useful when feature vectors are sparse (e.g., texts represented as a bag of words)
- Polynomial kernels fit curves
- RBFs are Gaussian blobs centered on training examples ($k$ nearest neighbors?)
- Using a kernel function, the inputs need not even be in a vector space (string kernels, tree kernels)
- ‘Kernelizing’ the perceptron can improve performance on inseparable or poorly separated training data, with the risk of overtraining

# Perceptron

- We still don’t have a unique solution from the perceptron algorithm for eparable problems

- We can fix this by imposing a stronger constraint

- Rather than minimizing the negative margin of misclassified points:

$$\hat{w} = \operatorname*{argmin}_w - \sum_{i \in \mathcal{M}} y_i \left( x_i \cdot w \right)$$

we can instead maximize the functional margin:

$$\gamma = \min_i y_i (x_i \cdot w)$$

- Recall how increasing the margin improved things with boosting

# Canonical hyperplane

- A problem: many weight vectors represent the same hyperplane, so we can rescale the weights (and increase the functional margin) without changing the classification decisions

- We define the *geometric margin*:

$$\rho = \frac{\gamma}{||w||}$$

  and we say that a separating hyperplane $w$ is *canonical* if the functional margin $\gamma = 1$

- For a canonical hyperplane, the distance to the closest data point is $1/||w||$

# Optimal margin classifier

- We've already seen plenty of evidence that large margins are good
- Here's a theoretical bound on the generalization error of a separating hyperplane
- Suppose $||w|| \leq \Lambda$ and $||x|| \leq R$ for $\Lambda, R > 0$, and the *margin error* $\nu$ is the fraction of the $m$ training examples with margin smaller than $\frac{\rho}{||w||}$ for $\rho > 0$. Then, with probability at least $1 - \delta$, the probability of misclassifying a test example is bounded by:

$$\nu + \sqrt{\frac{c}{m}\left(\frac{R^2\Lambda^2}{\rho^2}(\log m)^2 + \log\frac{1}{\delta}\right)}$$

# Optimal margin classifier

- Some observations about this:

$$\nu + \sqrt{\frac{c}{m}\left(\frac{R^2\Lambda^2}{\rho^2}(\log m)^2 + \log\frac{1}{\delta}\right)}$$

- The error is the sum of the margin error $\nu$ (i.e., training error) and a capacity term ($\sqrt{\cdots}$)

- As $m$ gets large, the capacity term gets small

- The capacity term gets larger as $\Lambda$ and $R$ (which are more or less fixed by the training data) increase

- Increasing the margin $\rho$ will decrease the capacity term, but increase the margin error

# Optimal margin classifier

- This bound suggests we want to find a separating hyperplane that maximizes the margin $\rho$ without increasing the margin error $\nu$

- For a canonical hyperplane, $\gamma = 1$ and:

$$\rho = \frac{1}{||w||}$$

  Thus maximizing $\rho$ is the same as minimizing $||w||$

- In other words, we want to find a canonical hyperplane where $||w||$ is small and no training points have margin less than $1/||w||$

- In other other words, we want no points where $y\,(w \cdot x) < 1$

# Optimal margin classifier

- We can frame this a *quadratic programming* problem:

  find: $\min_w \frac{1}{2}||w||^2$

  such that: $y_i\,(x_i \cdot w) \geq 1$, for all $i = 1, \ldots, m$

- To solve this, we introduce the Lagrangian:

$$L(w, \alpha) = \frac{1}{2}||w||^2 - \sum_i \alpha_i\,(y_i\,(x_i \cdot w) - 1)$$

  where $\alpha_i \geq 0$

- We minimize $L$ with respect to $w$ by setting the derivatives to zero, which gives us:

$$w = \sum_i \alpha_i\, y_i\, x_i$$

# Optimal margin classifier

- With some substitutions and tinkering, we get the dual problem:

  find: $\max_\alpha \sum_i \alpha_i - \frac{1}{2} \sum_i \sum_j \alpha_i \, \alpha_j \, y_i \, y_j \, (x_i \cdot x_j)$

  such that: $\alpha_i \geq 0$

- This can be solved using standard convex optimization techniques

- Because this is stated in terms of the dual parameters $\alpha_i$, it is easily kernelized

# Optimal margin classifier

- The decision boundary is given by:

$$\hat{y} = \mathrm{sign}(\sum \alpha_i \, y_i \, (x_i \cdot x))$$

- At the solution, $\alpha_i \, (y_i \, (x_i \cdot w) - 1) = 0$ for all $i$
- So, if $\alpha_i > 0$, then $y_i \, (x_i \cdot w) = 1$ and $x_i$ is right on the margin
- If $y_i \, (x_i \cdot w) > 1$ and $x_i$ is not on the margin, then $\alpha_i = 0$
- Thus, the decision boundary is a linear combination of training points which lie precisely on the margin (*support vectors*). The other training points are irrelevant.

# Support vector machines

- By directly controlling capacity, optimal margin classifiers reduce the dangers of overtraining
- Because most training points don’t lie right on the margin, the dual solution will tend to be sparse
- Optimal margin classifiers improve on the perceptron, but still fail for non-separable problems
- Support Vector Machines (SVM) extend optimal margin classifiers to non-linear decision boundaries (using kernel functions) and non-separable problems (using slack variables)

# Support vector machines

- A *slack variable* $\xi_i$ reflects the extent to which a point fails to satisfy the margin

- Now the quadratic program is:

  find: $\min_w \frac{1}{2}||w||^2 + \frac{C}{m}\sum_i \xi_i$

  such that: $y_i\,(x_i \cdot w) \geq 1 - \xi_i$, for all $i = 1, \ldots, m$

- The dual problem is:

  find: $\max_\alpha \sum_i \alpha_i - \frac{1}{2}\sum_i \sum_j \alpha_i\,\alpha_j\,y_i\,y_j\,(x_i \cdot x_j)$

  such that: $0 \leq \alpha_i \leq \frac{C}{m}$ and $\sum_i \alpha_i\,y_i = 0$

- The constant $C$ reflects the trade-off between our conflicting goals: to maximize the margin and to minimize the margin error

# Support vector machines

- Soft margin SVMs will find a solution even for non-separable problems
- The support vectors (for which $\alpha_i \neq 0$) are either on the margin, with $\alpha_i < \frac{C}{m}$ and $\xi_i = 0$, or they are training errors, with $\alpha_i = \frac{C}{m}$ and $\xi_i > 0$
- There's no obvious way to set the constant $C$, other than cross-validation, etc. (but: $\nu$-SVMs)
- demo, demo, demo

# Support vector machines

- The hype about SVMs claimed that the combination of the kernel trick with capacity control makes them transcend the curse of dimensionality

- Not so: SVMs will overtrain given the right circumstances

- Choice of kernel function is crucial, and a bit of an art

| Method | Error (4+0 feats) | Error (4+6 feats) |
|---|---:|---:|
| SVM (linear) | 0.450 | 0.472 |
| SVM (poly 2) | 0.078 | 0.152 |
| SVM (poly 5) | 0.180 | 0.370 |
| SVM (poly 10) | 0.230 | 0.434 |
| BRUTO | 0.084 | 0.090 |
| MARS | 0.156 | 0.173 |
| Bayes optimal | 0.029 | 0.029 |