Linear Classifiers
(With slides from Najim Dehak)
Recap

• Classification and KNN..
MLSP

• Application of Machine Learning techniques to the analysis of signals

• Modeling
  – Classification: Model-Based vs instances-Based
Machine Learning

- **Supervised**: We are given input samples \((X)\) and output samples \((y)\) of a function \(y = f(X)\). We would like to “learn” \(f\), and evaluate it on new data. Types:
  - **Classification**: \(y\) is discrete (class labels).
  - **Regression**: \(y\) is continuous, e.g. linear regression.

- **Unsupervised**: Given only samples \(X\) of the data, we compute a function \(f\) such that \(y = f(X)\) is “simpler”.
  - **Clustering**: \(y\) is discrete
  - \(Y\) is continuous: Matrix factorization, Kalman filtering, unsupervised neural networks.
Machine Learning

• **Supervised:**
  – Is this image a cat, dog, car, house?
  – How would this user score that restaurant?
  – Is this email spam?
  – Is this blob a supernova?

• **Unsupervised**
  – Cluster some hand-written digit data into 10 classes.
  – What are the top 20 topics in Twitter right now?
  – Find and cluster distinct accents of people at Berkeley. (?)
Multi-class Image Classification
k-Nearest Neighbor classification

Given a query item:
Find k closest matches in a labeled dataset ↓
k-Nearest Neighbor classification

Given a query item: Find k closest matches
Return the most Frequent label
k-Nearest Neighbor classification

k = 3 votes for “cat”
k-Nearest Neighbors

2 votes for cat,
1 each for Buffalo, Deer, Lion

Cat wins...
Nearest neighbor method

- Majority vote within the $k$ nearest neighbors

\[
\hat{Y}(x) = \frac{1}{k} \sum_{x_i \in N_k(x)} y_i
\]

K= 1: blue
K= 3: green
Nearest neighbor method

• Weighted majority vote

\[ \hat{Y}(x) = \frac{1}{k} \sum_{i \in N_k(x)} w(x, x_i) y_i \]
Nearest neighbor method

- Weighted majority vote within the k nearest neighbors
- Not all Ys are equally important
  - Outliers and training instances far away from the "confusing" regions don’t really inform
  - Redundant training instances (very close to others) don’t really add anything new

\[ \hat{Y}(x) = \frac{1}{\sum_{i \in N_k(x)} \alpha_i} \sum_{i \in N_k(x)} w(x, x_i) \alpha_i y_i \]

- \( \alpha_i \)’s may be binary (useful vs. useless)
Nearest-Neighbor Classifiers

- Requires three things
  - The set of stored records
  - Distance Metric to compute distance between records
  - The value of $k$, number of nearest neighbors to retrieve

To classify new record:
- Compute distance to other training records
- Identify $k$ nearest neighbors
- Vote among nearest neighbors
Definition of Nearest Neighbor

K-nearest neighbors of a record $x$ are data points that have the $k$ smallest distance to $x$
1 nearest-neighbor

Voronoi Diagram
k-NN issues

The Data is the Model

• No training needed.
• Accuracy generally improves with more data.
• Matching is simple and fast (and single pass).
• Usually need data in memory, but can be run off disk.

Minimal Configuration:

• Only parameter is k (number of neighbors)
• Two other choices are important:
  – Weighting of neighbors (e.g. inverse distance)
  – Similarity metric
K-NN metrics

• **Euclidean Distance**: Simplest, fast to compute
  \[ d(x, y) = \|x - y\| \]

• **Cosine Distance**: Good for documents, images, etc.
  \[ d(x, y) = 1 - \frac{x \cdot y}{\|x\| \|y\|} \]

• **Jaccard Distance**: For set data:
  \[ d(X, Y) = 1 - \frac{|X \cap Y|}{|X \cup Y|} \]

• **Hamming Distance**: For string data:
  \[ d(x, y) = \sum_{i=1}^{n} (x_i \neq y_i) \]
K-NN metrics

• **Manhattan Distance**: Coordinate-wise distance

\[ d(x, y) = \sum_{i=1}^{n} |x_i - y_i| \]

• **Edit Distance**: for strings, especially genetic data.

• **Mahalanobis Distance**: Normalized by the sample covariance matrix – unaffected by coordinate transformations.
Scalp EEG Acquisition
10-second EEGs: Seizure Evolution

Normal

Pre-Seizure

Seizure Onset

Post-Seizure

K-Nearest Neighbor for seizure detection

Time series distances: (1) Euclidean, (2) Dynamic Time Warping
Example: Digit Recognition

- Yann LeCunn – MNIST Digit Recognition
  - Handwritten digits
  - 28x28 pixel images: $d = 784$
  - 60,000 training samples
  - 10,000 test samples

<table>
<thead>
<tr>
<th>Method</th>
<th>Test Error Rate (%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Linear classifier (1-layer NN)</td>
<td>12.0</td>
</tr>
<tr>
<td>K-nearest-neighbors, Euclidean</td>
<td>5.0</td>
</tr>
<tr>
<td>K-nearest-neighbors, Euclidean, deskewed</td>
<td>2.4</td>
</tr>
<tr>
<td>K-NN, Tangent Distance, 16x16</td>
<td>1.1</td>
</tr>
<tr>
<td>K-NN, shape context matching</td>
<td>0.67</td>
</tr>
</tbody>
</table>
More generally: Supervised classification

• A minor shift of gears..

• Given a set of labelled training instances, learn to classify a new test instance..
  – (K)NN was only one method
Binary classification
Gender identification

**Supervised learning:** A class (category) label for each pattern in the training set is provided.
Multidimensional Time Series Classification in Medical Data

- Positive versus Negative
- Responsive versus Unresponsive
- Multidimensional Time Series Classification
- Multisensor medical signals (e.g., EEG, ECG, EMG)
Classification and *discriminant* functions

- Define a “discriminant function” $g_i(x)$ for each class $\omega_i$ such that:

- the classifier assigns a feature vector $x$ to class $\omega_i$ if
  $$g_i(x) > g_j(x) \quad \text{for all } j \neq i$$
  
  - For two-category case, $g(x) = g_1(x) - g_2(x)$
    
    Decide $\omega_1$ if $g(x) > 0$; otherwise decide $\omega_2$

- An example
  - Minimum-Error-Rate Classifier
    $$g(x) \equiv p(\omega_1 \mid x) - p(\omega_2 \mid x)$$
Discriminant Function

- It can be arbitrary functions of $x$, such as:

\[ g(x) = \mathbf{w}^T \mathbf{x} + b \]

Nearest Neighbor

Linear Functions

Nonlinear Functions
The equation for a hyperplane

\[ \mathbf{W}^T \mathbf{X} = 0 \]

- \( \mathbf{W}^T \mathbf{X} = 0 \) is the equation representing the set of all vectors that are orthogonal to \( \mathbf{W} \)
The equation for a hyperplane

- The set of all vectors that are a distance $\frac{b}{\|W\|}$ from the blue plane.

- $W^TX - b = 0$ is the equation representing plane that is orthogonal to $W$ and a distance $\frac{b}{\|W\|}$ from origin.
The equation for a hyperplane

Trivial proof:

- On the red plane any $X = X_p + \left( \frac{b}{\|W\|} \right) \frac{W}{\|W\|}$

- $W^T X = W^T X_p + b \frac{W^T W}{\|W\|^2} = b$
Distance from a hyperplane

- The distance of any $X_{test}$ from the plane $W^T X - b = 0$ is $d = \frac{W^T X_{test} - b}{||W||}$
- This can be positive (in the direction of $W$) or negative (opposite to $W$)
Sign of distance from hyperplane

- The sign of $W^T X - b$ signifies which side of the plane the point $X$ is on.

$$\text{sign}(W^T X_{\text{test}} - b) > 0$$

$$\text{sign}(W^T X_{\text{test2}} - b) < 0$$
• The plane $W^T X - b$ is a linear classifier
  – The class is given by $\text{sign}(W^T X_{\text{test}} - b)$
Linearly separable data

- Data where the two classes are separated by a hyperplane
  - And classification can be performed by $\text{sign}(W^T X_{\text{test}} - b)$ for any separating hyperplane
• Classes are linearly separable
• Dots represent “training” instances
• **Training problem**: Given these training instances find a separating hyperplane
The separating hyperplane

- Problem: Given these training instances find a separating hyperplane
- Many ways of finding this hyperplane
  - Any number of solution algorithms are possible
• **Simplifying assumption:** The separating hyperplane always goes through origin
  
  – Easily enforced by appending a constant 1 to every vector
A Simple Method: The Perceptron Algorithm

- **Initialize:** Randomly initialize the hyperplane
  - I.e. randomly initialize the normal vector $W$
  - Classification rule $\text{sign}(W^T X)$
  - The random initial plane will make mistakes
Perceptron Algorithm

• Given $N$ training instances $(X_1, Y_1), (X_2, Y_2), \ldots, (X_N, Y_N)$
  - $Y_i = +1$ or $-1$

• Initialize $W$

• Cycle through the training instances:

• While more classification errors
  - For $i = 1 \ldots N_{\text{train}}$
    $O(X_i) = \text{sign}(W^T X_i)$

• If $O(X_i) \neq Y_i$
  $W = W + Y_i X_i$
Perceptron Algorithm: Summary

• Cycle through the training instances
• Only update $W$ on misclassified instances
• If instance misclassified:
  – If instance is positive class
    \[ W = W + X_i \]
  – If instance is negative class
    \[ W = W - X_i \]
Perceptron Algorithm

Initialization

+1 (green)  -1 (Red)
Perceptron Algorithm

Misclassified positive instance
Perceptron Algorithm

+1 (green)
-1 (Red)
Perceptron Algorithm

Updated weight vector
Perceptron Algorithm

+1 (green)  -1 (Red)

Updated hyperplane
Convergence of Perceptron Algorithm

• Guaranteed to converge
  – After no more than $\frac{R^2}{\gamma^2}$ misclassifications
  – $R$ is length of longest training point
  – $\gamma$ is the best case closest distance of a training point from the classifier
    • I.e the largest distance to the closest training instance to any appropriate classifier
Problems with perceptron algorithm

- Final solution depends on order of processing of inputs
  - Can get different solutions for the same initial vector by changing the order in which instances are considered
Problems with perceptron algorithm

- Final solution depends on order of processing of inputs
  - Can get different solutions for the same initial vector by changing the order in which instances are considered
Problems with perceptron algorithm

• Final solution depends on order of processing of inputs
  – Can get different solutions for the same initial vector
  – No assurance about whether this solution will work for new test data
Problems with perceptron algorithm

- Final solution depends on order of processing of inputs
  - Can get different solutions for the same initial vector
  - No assurance about whether this solution will work for new test data

Figure relaxes the simplifying assumption that the classifier runs through origin
Convergence of Perceptron Algorithm

• Guaranteed to converge
  – After no more than \( \frac{R^2}{\gamma^2} \) misclassifications
  – \( R \) is length of longest training point
  – \( \gamma \) is the best case closest distance of a training point from the classifier
    • I.e the largest distance to the closest training instance to any appropriate classifier

• Although the number of iterations is bounded by the distance of a “best-case” classifier, no guarantee that we will actually find this best-case classifier
  – Algorithm stops updating after perfect training classification
Modification of perceptron to find margin

• Instead of updating only on misclassified instances, update on any vector within $0.5\gamma$ of boundary
• Guaranteed to converge
• Problem – you specify $\gamma$.
  – Overall optimality not guaranteed
  – But still, a pretty good algorithm
Enter: Support Vector Machines

• Find a classifier that is maximally distant from the closest instances from either class
Any linear classifier has some *closest* instances

These instances will be at some distance from the boundary

Changing the classifier will change both, the closest instance, and their distance from the boundary
Returning to the *Perceptron* algorithm

- Guaranteed to converge
  - After no more than \( \frac{R^2}{\gamma^2} \) misclassifications
  - \( R \) is length of longest training point
  - \( \gamma \) is the *best case* closest distance of a training point from the classifier
    - I.e the largest distance to the closest training instance to *any* appropriate classifier
- No guarantee that we will actually find this best-case classifier
  - Algorithm stops updating after perfect training classification
- **Can we actually *make it find this best case classifier***
A Better Approach

- Search through all classifiers such that the distance to the closest points is maximized
  - Very conservative
  - Focuses on *worst-case* scenario
  - Maximizes the chance that the classifier will work well on new unseen data
A Better Approach

- Search through all classifiers such that the distance to the closest points is maximized
  - Very conservative
  - Focuses on *worst-case* scenario
  - Maximizes the chance that the classifier will work well on new unseen data
A Better Approach

• Search through all classifiers such that the distance to the closest points is maximized
  – Very conservative
  – Focuses on *worst-case* scenario
  – Maximizes the chance that the classifier will work well on new unseen data
• Search through all classifiers such that the distance to the closest points is maximized
  – Very conservative
  – Focuses on *worst-case* scenario
  – Maximizes the chance that the classifier will work well on new unseen data
Find the classifier such that the distance to the *closest* points is maximized

I.e. solve *two* problems: find the closest points, and the classifier, such that the distance is maximum

- Position the classifier in the *middle* so that the distance to the closest green = distance to the closest red

Is this a combinatorial optimization problem??
Solution Approach

- For any hyperplane (linear classifier) $W^T X - b = 0$
- Choose two hyperplanes $W^T X - b = 1$ and $W^T X - b = -1$
  - The distance of these hyperplanes from the classifier is $1/\|W\|$
  - The total distance between the hyperplanes is $2/\|W\|
Solution Approach

• Constraint: Perfect classification with a margin
• Choose the hyperplanes such that
  – All positive points are on the positive side of the positive hyperplane
  – All negative points are on the negative side of the negative hyperplane

\[ W^T X - b = 0 \]
Solution Approach

- The distance between the hyperplanes is $\frac{2}{\|W\|}$

- Find the $W$ (and $b$) such that this is maximized, while maintaining the constraint that all training points are on the “outside” of the appropriate hyperplane
The distance between the hyperplanes is \[ \frac{2}{\|W\|} \]

Find the \( W \) (and \( b \)) such that this is maximized, while maintaining the constraint that all training points are on the “outside” of the appropriate hyperplane.
Solution Approach

- The distance between the hyperplanes is $\frac{2}{\|W\|}$
- Find the $W$ (and $b$) such that this is maximized, while maintaining the constraint that all training points are on the “outside” of the appropriate hyperplane

Expanding the gap by changing the direction of $W$
Solution Approach

- The distance between the hyperplanes is \( \frac{2}{\|W\|} \)
- Find the \( W \) (and \( b \)) such that this is maximized, while maintaining the constraint that all training points are on the “outside” of the appropriate hyperplane
Solution Approach

• The distance between the hyperplanes is \( \frac{2}{\|W\|} \)
• Maximize this distance. I.e. ..
• Minimize \( \|W\| \) such that
  – all training points are on the “outside” of the appropriate hyperplane
Solution Approach

The distance between the hyperplanes is \( \frac{2}{\|W\|} \)

Maximize this distance. I.e. ...

Minimize \( \|W\|^2 \) such that

- all training points are on the “outside” of the appropriate hyperplane
Let's formalize this

• Constraint: Ensuring that all training instances are on the proper side of their respective hyperplanes

• For positive training instances $X_i$:
  \[ W^T X_i - b \geq 1 \]

• For negative instances
  \[ W^T X_i - b \leq -1 \]

• Generically stated, for all instances we want
  \[ Y_i(W^T X_i - b) \geq 1 \]
Solution Formalism

• Minimize $\|W\|$ such that

• For all training instances
  \[ Y_i(W^T X_i - b) \geq 1 \]

• Formally

\[ \hat{W} = \arg \min_{W,b} \|W\|^2 \]
\[ \text{s.t. } \forall i \quad Y_i(W^T X_i - b) \geq 1 \]
Solving the optimization

- This is a quadratic programming problem!
  \[
  \hat{W} = \arg\min_{W,b} \|W\|^2
  \]
  \[
  s.t. \ \forall i \quad Y_i(W^T X_i - b) \geq 1
  \]

- A variety of techniques can be applied
  - Interior point methods, active set methods, gradient descent, conjugate gradient
  - The objective function is convex, QP will find the (near) optimal solution

- Most useful solution is based on Lagrangian duals
  - Later..
The solution

- Maximizes the margin
- This is a max-margin classifier
- The boundary samples are called support vectors
  - All the information about the classifier is in these support vectors
Challenges

• What if the classes are not linearly separable

• What if the classes are not \textit{linearly} separable?

• What if the classes are not \textit{linearly separable}?
What if they are not separable?

- What if the data are not separable?
Original Problem

• This is a quadratic programming problem!

\[ \hat{W} = \underset{W}{\arg\min} \|W\|^2 \]

\[ s.t. \ \forall \ i \quad Y_i(W^T X_i - b) \geq 1 \]

• Maximize the distance between the planes
• Subject to the constraint that all training data instances are on the “correct” side of the plane
• When data are not linearly separable, this constraint can never be satisfied
Introducing the *slack* variable

- What if the data are not separable?
Introducing the *slack* variable

- For every training instance, introduce a *slack* variable $\xi$
- The slack variable is the maximum distance you have to *shift* the boundary plane to move the point to the “correct” side
Introducing the *slack* variable

- For every training instance, introduce a *slack* variable $\xi$
- The slack variable is the *reverse* distance from the *margin* plane of the training instance
  - This will be non-zero only for some instances
  - Ideally this should be minimum
Introducing the *slack* variable

- The total length of slack variables varies with the boundary
- If you push the boundaries too far you will have a greater length of slack variable
  - Which contradicts our desire that they should be minimum
Introducing the *slack* variable

- If they are very close, only the *inseparable points* will have non-zero slack variable
  - The minimum slack value is when the margin planes coincide with the linear classifier
Introducing the *slack* variable

• If they are very close, only the *inseparable points* will have non-zero slack variable
  – The minimum slack value is when the margin planes coincide with the linear classifier
• For linearly separable classes, if the boundary planes are close enough, the total slack length will be 0
Introducing the *slack* variable

- Problem: If they are too close, the planes violate our desire to *maximize* the margin
Introducing the *slack* variable

- Contradicting requirements..

Need: Push the margin planes close together to minimize total slack

But this contradicts our objective that the distance between the planes must be maximized
New Objective

• Simultaneously
  – Maximize distance between planes
  – Minimize total slack length
Quantifying Slack Length

- We need a formula for the total slack length first..
Quantifying Slack Length

- The positive margin plane is given by
- \( W^T X - b - 1 = 0 \)

- This plane is at a distance \( \frac{1}{\|W\|} \) from the decision boundary on the positive side of the decision plane (in the direction of \( W \))
  - Ideally all positive training points would be to the right of it
Quantifying Slack Length

- The (unnormalized) distance of any $X$ from this plane

$$W^T X - b - 1 = 0$$

- This will be negative for instances on the “wrong” side (in the direction away from $W$), but positive for those on the “right” side
Quantifying Slack Length

• The negated (unnormalized) distance of any $X$ from this plane
  
  \[ 1 - (W^T X - b) \geq 0 \]

• This will be positive for instances on the wrong side of the margin plane, but negative for instances on the right side of it
Quantifying Slack Length

We do not care about the actual distance of instances to the right of the plane.

So the slack value of any point is

$$\max(0, 1 - (W^T X - b))$$
Quantifying Slack Length

\[ W^T X - b + 1 = 0 \]

- The *negative* margin plane is given by
  \[ W^T X - b + 1 = 0 \]
  - Ideally all negative training points would be to the left of it
Quantifying Slack Length

The (unnormalized) distance of any $X$ from this plane

$$W^T X - b + 1 = 0$$

• This will be positive for vectors on the “wrong” side, but negative for vectors on the right side.
Quantifying Slack Length

• We do not care about the actual distance of instances to the left of the plane
• So the slack value of any point is

\[ \max(0, 1 + W^T X - b) \]
Quantifying Slack Length

$W^T X - b + 1 = 0$

• Combining the following for negative instances

$$\max(0, 1 + (W^T X - b))$$
Quantifying Slack Length

- And the following for positive instances
  \[ \max(0, 1 - (W^T X - b)) \]
Quantifying Slack Length

\[ W^T X - b + 1 = 0 \]

\[ W^T X - b - 1 = 0 \]

• Generic Slack length for any point
  \[ \max(0, 1 - y(W^T X - b)) \]

• This is also called a *hinge loss*
Total Slack Length

• Total slack length for *all* training instances

\[
\sum_{i} \max(0, 1 - y(W^T X - b))
\]

• This must be minimized
Overall Optimization

• Minimize $\|W\|^2$ to maximize the distance between margin planes

• Minimize total slack length to minimize the distance of *misclassified* instances to margin planes

\[ \sum_i \max(0, 1 - y(W^T X - b)) \]

– This will make the margin planes *closer*

• The two objectives must be traded off..
Support Vector Machine for Inseparable data

• Minimize

$$\arg\min_{W,b} \frac{1}{N} \sum_{i} \max(0, 1 - y(W^T X - b)) + \lambda \|W\|^2$$

• $\lambda$ is a “regularization” parameter that decides the relative importance of the two terms

• This is just a regular optimization problem that can be solved through gradient descent
Support Vector Machine for Inseparable data

• $\lambda$ is typically set using *held-out* training data
  – Train the classifier for various values of $\lambda$
  – Test each of these classifiers on some held-out portion of the training data that was not included in training the SVM
  – Pick the $\lambda$ for which the classifier gave best performance
  – Retrain the SVM using the entire training data and this $\lambda$

• Frequently, instead of a single held-out set, $\lambda$ is set through K-fold cross validation
Equivalent Slack Formalism

\[
\text{argmin}_{W,b} \|W\|^2 + C \sum_i \xi_i
\]

• Subject to

\[
Y_i(W^T X_i - b) \geq 1 - \xi_i
\]

• This is a quadratic programming problem

• Slack parameter \(C\) is determined through held-out data as earlier (or through K-fold cross-validation)
How to deal with non-linear boundaries?

• First some math..
Recall: The Lagrange Method

• Optimize \( f(x, y) \) subject to \( g(x, y) = c \)

\[
L(x, y, \lambda) = f(x, y) - \lambda(g(x, y) - c)
\]

• to maximize \( f(x, y) \): \( \max_{x,y} \left( \min_{\lambda} L(x, y, \lambda) \right) \)

• to minimize \( f(x, y) \): \( \min_{x,y} \left( \max_{\lambda} L(x, y, \lambda) \right) \)
Optimization with inequality constraints

- Optimization problem with constraints
  \[
  \min_x f(x) \\
  s.t. \ g_i(x) \leq 0, \ i = \{1, \ldots, k\} \\
  \ h_j(x) = 0, \ j = \{1, \ldots, l\}
  \]

- Lagrange multipliers \( \lambda_i \geq 0, \nu \in \mathbb{R} \)
  \[
  L(x, \lambda, \nu) = f(x) + \sum_{i=1}^{k} \lambda_i g_i(x) + \sum_{j=1}^{l} \nu_j h_j(x)
  \]

- The optimization problem
  \[
  \arg\min_x \max_{\lambda, \nu} L(x, \lambda, \nu)
  \]
Revisiting the *linearly separable case*

- This is a quadratic programming problem!

\[
\hat{W} = \arg\min_W \|W\|^2 \\
\text{s.t. } \forall i \quad Y_i(W^T X_i - b) \geq 1
\]

- Can be stated using Lagrangians as

\[
\arg\min_{W,b} \max_{\alpha > 0} \|W\|^2 + \sum_i \alpha_i(Y_i(W^TX_i - b) - 1)
\]
Linearly separable case: Lagrangian formalism

• Can be stated using Lagrangians as

\[
\arg\min_{W,b} \max_{\alpha > 0} \|W\|^2 + \sum_i \alpha_i (Y_i(W^T X_i - b) - 1)
\]

• The optimum satisfies the *Karush Kuhn-Tucker* conditions, hence we can rewrite it as

\[
\arg\max_{\alpha > 0} \min_{W,b} \|W\|^2 + \sum_i \alpha_i (Y_i(W^T X_i - b) - 1)
\]
Linearly separable case: Lagrangian formalism

• Under the KKT conditions

$$\arg\max_{\alpha > 0} \min_{W,b} \|W\|^2 + \sum_i \alpha_i (Y_i (W^T X_i - b) - 1)$$

• Taking the derivative w.r.t $W$ and setting to 0, we get

$$2W = -\sum_i \alpha_i Y_i X_i$$
Linearly separable case: 
Lagrangian formalism

- Under the KKT conditions

\[
\arg\max_{\alpha > 0} \min_{W,b} \|W\|^2 + \sum_i \alpha_i (Y_i (W^T X_i - b) - 1)
\]

- Taking the derivative w.r.t \(b\) and setting to 0, we get

\[
0 = \sum_i \alpha_i Y_i
\]
Linearly separable case:

- Restating (and ignoring the factor of 2)
  \[
  \arg\max_{\alpha > 0} \sum_i \alpha_i - \sum_{i,j} \alpha_i \alpha_j Y_i Y_j X_i^T X_j - b \sum_i \alpha_i Y_i
  \]

- Since the last term is 0
  \[
  \arg\max_{\alpha} \sum_i \alpha_i - \sum_{i,j} \alpha_i \alpha_j Y_i Y_j X_i^T X_j
  \]
  \[s.t. \alpha_i \geq 0\]
  \[
  \sum_i \alpha_i Y_i = 0
  \]
Large Margin Linear Classifier with Slack

- Formulation: (Lagrangian Dual Problem)

\[
\text{maximize } \sum_{i=1}^{n} \alpha_i - \sum_{i=1}^{n} \sum_{j=1}^{n} \alpha_i \alpha_j y_i y_j x_i^T x_j
\]

such that

\[0 \leq \alpha_i \leq C\]

\[\sum_{i=1}^{n} \alpha_i y_i = 0\]
The usual simple SVM can also be solved through the ugly form

\[
\arg\max_{\alpha} \sum_i \alpha_i - \sum_{i,j} \alpha_i \alpha_j Y_i Y_j X_i^T X_j
\]

s.t. \( C \geq \alpha_i \geq 0 \)

\[
\sum_i \alpha_i Y_i = 0
\]

- This is for the linear case. Note that the optimization is in terms of \( X_i^T X_j \)
- Also \( W = -\sum_i \alpha_i Y_i X_i \)
- So the classifier on any test instance has the form:

\[\text{sign}\left(-\sum_i \alpha_i Y_i X_{test}^T X_i - b\right)\]
The SVM as KNN classification

\[
\text{argmax}_{\alpha} \sum_{i} \alpha_i - \sum_{i,j} \alpha_i \alpha_j Y_i Y_j X_i^T X_j \\
\text{s.t. } C \geq \alpha_i \geq 0 \\
\sum \alpha_i Y_i = 0
\]

Weighted-nearest neighbor classifier

- This is for the linear case. Note that the optimization is in terms of 
  \(X_i^T X_j\)
- Also \(W = - \sum \alpha_i Y_i X_i\)
- So the classifier on any test instance has the form:
  \[
  \text{sign}\left(- \sum \alpha_i Y_i X_{test}^T X_i - b\right)
  \]
The SVM as KNN classification

This is for the linear case. Note that the optimization is in terms of $X_i^T X_j$.

Also $W = - \sum_i \alpha_i Y_i X_i$

So the classifier on any test instance has the form:

$$\text{sign} \left( - \sum_i \alpha_i Y_i X_{test}^T X_i - b \right)$$
The Kernel Trick

\[
\operatorname{argmax}_\alpha \sum_i \alpha_i - \sum_{i,j} \alpha_i \alpha_j Y_i Y_j X_i^T X_j \\
\text{s.t. } C \geq \alpha_i \geq 0 \\
\sum_i \alpha_i Y_i = 0
\]

- This is for the linear case. Note that the optimization is in terms of \(X_i^T X_j\)
- Also \(W = -\sum_i \alpha_i Y_i X_i\)
- So the classifier on any test instance has the form:

\[
\operatorname{sign} \left( -\sum_i \alpha_i Y_i X_{test}^T X_i - b \right)
\]
The Kernel Trick

\[
\arg\max_{\alpha} \sum_{i} \alpha_i - \sum_{i,j} \alpha_i \alpha_j Y_i Y_j K(X_i, X_j)
\]

s.t. \( C \geq \alpha_i \geq 0 \)
\[
\sum_{i} \alpha_i Y_i = 0
\]

• For classification:

\[
\text{sign}\left(-\sum_{i} \alpha_i Y_i K(X_i, X_{test}) - b\right)
\]
The Kernel Trick

For classification:

\[
\arg\max_{\alpha} \sum_i \alpha_i - \sum_{i,j} \alpha_i \alpha_j Y_i Y_j K(X_i, X_j)
\]

\[
s.t. C \geq \alpha_i \geq 0
\]

\[
\sum_i \alpha_i Y_i = 0
\]

This is a quadratic programming problem

- For classification:

\[
\text{sign}\left(-\sum_i \alpha_i Y_i K(X_i, X_{\text{test}}) - b\right)
\]
Nonlinear SVMs: The Kernel Trick

- Examples of commonly-used kernel functions:
  - Linear kernel: \( K(x_i, x_j) = x_i^T x_j \)
  - Polynomial kernel: \( K(x_i, x_j) = (1 + x_i^T x_j)^p \)
  - Gaussian (Radial-Basis Function (RBF)) kernel:
    \[
    K(x_i, x_j) = \exp\left(-\frac{\|x_i - x_j\|^2}{2\sigma^2}\right)
    \]
  - Sigmoid:
    \[
    K(x_i, x_j) = \tanh(\beta_0 x_i^T x_j + \beta_1)
    \]

- In general, functions that satisfy Mercer’s condition can be kernel functions.
Nonlinear SVM: Optimization

- Formulation: (Lagrangean Dual Problem)

\[
\text{maximize } \sum_{i=1}^{n} \alpha_i - \frac{1}{2} \sum_{i=1}^{n} \sum_{j=1}^{n} \alpha_i \alpha_j y_i y_j K(x_i, x_j)
\]

such that

\[
0 \leq \alpha_i \leq C
\]

\[
\sum_{i=1}^{n} \alpha_i y_i = 0
\]

- The solution of the discriminant function is

\[
g(x) = \sum_{i \in SV} \alpha_i K(x_i, x) + b
\]

- The optimization technique is the same.
Support Vector Machine: Algorithm

• 1. Choose a kernel function

• 2. Choose a value for $C$

• 3. Solve the quadratic programming problem (many software packages available)

• 4. Construct the discriminant function from the support vectors
Some Issues

• Choice of kernel
  - Gaussian or polynomial kernel is default
  - if ineffective, more elaborate kernels are needed
  - domain experts can give assistance in formulating appropriate similarity measures

• Choice of kernel parameters
  - e.g. $\sigma$ in Gaussian kernel
  - $\sigma$ is the distance between closest points with different classifications
  - In the absence of reliable criteria, applications rely on the use of a validation set or cross-validation to set such parameters.

• Optimization criterion – Hard margin v.s. Soft margin
  - a lengthy series of experiments in which various parameters are tested
Summary: Support Vector Machine

• 1. Large Margin Classifier
  – Better generalization ability & less over-fitting

• 2. The Kernel Trick
  – Map data points to higher dimensional space in order to make them linearly separable.
  – Since only dot product is used, we do not need to represent the mapping explicitly.
Multi-class generalization Pairwise
Multi-class generalization One-vs-all
Support Vector Machine for seizure detection

Feature 1

Feature 2

Feature 3

Pre-Seizure

Normal

A data vector of EEG sample

1 2 3 4 \ldots n
Example: Digit Recognition

- Yann LeCunn – MNIST Digit Recognition
  - Handwritten digits
  - 28x28 pixel images: \( d = 784 \)
  - 60,000 training samples
  - 10,000 test samples
- Nearest neighbour is competitive

<table>
<thead>
<tr>
<th>Model</th>
<th>Test Error Rate (%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Linear classifier (1-layer NN)</td>
<td>12.0</td>
</tr>
<tr>
<td>K-nearest-neighbors, Euclidean</td>
<td>5.0</td>
</tr>
<tr>
<td>K-nearest-neighbors, Euclidean, deskewed</td>
<td>2.4</td>
</tr>
<tr>
<td>K-NN, Tangent Distance, 16x16</td>
<td>1.1</td>
</tr>
<tr>
<td>K-NN, shape context matching</td>
<td>0.67</td>
</tr>
<tr>
<td>1000 RBF + linear classifier</td>
<td>3.6</td>
</tr>
<tr>
<td>SVM deg 4 polynomial</td>
<td>1.1</td>
</tr>
</tbody>
</table>
Linear Classifiers: Conclusion

• Simple linear classifiers can be surprisingly effective
  – Particularly when trained to maximize a margin
    • Whereupon the “simple” arithmetic magically becomes complicated

• Kernel trick enables classification of even non-linear problems

• Most commonly used classifier, still