Support Vector Machines

CS434
Linear Separators

• Many linear separators exist that perfectly classify all training examples
• Which of the linear separators is the best?
Recall: Concept of Margin

- In Perceptron, we learned that the convergence rate of the Perceptron algorithm depends on a concept called *margin*
Intuition of Margin

• Consider points A, B, and C
• We are quite confident in our prediction for A because it is far from the decision boundary.
• In contrast, we are not so confident in our prediction for C because a slight change in the decision boundary may flip the decision.

Given a training set, we would like to make all of our predictions correct and confident! This can be captured by the concept of margin
Functional Margin

• One possible way to define margin:

\[ \hat{\gamma}^i = y^i (w \cdot x^i + b) \]

Note that \( \hat{\gamma}^i > 0 \) if classified correctly

• We define this as the **functional margin** of the linear classifier \( w.r.t \) training example \((x^i, y^i)\)

• The larger the value, the better – really?
Issue with Functional Margin

• What if we rescale \((w, b)\) by a factor \(\alpha\)?

\[ w \cdot x + b = 0 \quad \Rightarrow \quad \alpha w \cdot x + \alpha b = 0 \]

• Decision boundary remain the same

• Yet, functional margin gets multiplied by \(\alpha\)

• We can change the functional margin of a linear classifier without changing anything meaningful
What we really want

We want the distances between the examples and the decision boundary to be large – this quantity is what we call geometric margin.

But how do we compute the geometric margin of a data point w.r.t a particular line (parameterized by w and b)?
Basic facts about lines

\[ w \cdot x + b = 0 \]

To see this:
Let \( x^0 \) be the projection of \( x^1 \) on this line. We have:

\[ x^1 = x^0 + \text{the red vector} \]

the red vector = \( d \frac{w}{||w||} \)

\[ x^1 = x^0 + d \frac{w}{||w||} \]

Multiply both sides by \( w \)

\[ w \cdot x^1 = w \cdot x^0 + d \frac{w \cdot w}{||w||} = -b + d ||w|| \]

\[ d = \frac{w \cdot x^1 + b}{||w||} \]

Note: \( d \) can be positive or negative. Positive means the point is on the same side that \( w \) is pointing to, and negative means the opposite.
Geometric Margin

- The geometric margin of \((\mathbf{w}, b)\) \(w.r.t. \mathbf{x}^{(i)}\) is the distance from \(\mathbf{x}^{(i)}\) to the decision boundary.

- This distance can be computed as:
  \[
  \gamma^i = \frac{y^i (\mathbf{w} \cdot \mathbf{x}^i + b)}{\|\mathbf{w}\|}
  \]
  \(\gamma^i > 0\) if correctly classified.

- Given training set \(S=\{(\mathbf{x}^i, y^i): i=1, \ldots, N\}\), the geometric margin of the classifier \(w.r.t. S\) is:
  \[
  \gamma = \min_{i=1\ldots N} \gamma^{(i)}
  \]

Note that the points closest to the boundary are called the **support vectors** – in fact these are the only points that really matters, other examples are ignorable.
What we have done so far

• We have established that we want to find a linear decision boundary *whose geometric margin is the largest*

• We have a new learning objective
  – Given a *linearly separable* (will be relaxed later) training set $S=\{(x^i, y^i): i=1, \ldots, N\}$, we would like to find a linear classifier $(w, b)$ with maximum geometric margin.

• How can we achieve this?
  – Mathematically formulate this optimization problem and then solve it
  – In this class, we just need to know some basics about this process
Maximum Margin Classifier

• This can be represented as a constrained optimization problem.

\[
\begin{align*}
\max_{w,b} & \quad \gamma \\
\text{subject to :} & \quad y^{(i)} \frac{(w \cdot x^{(i)} + b)}{||w||} \geq \gamma, \quad i = 1, \ldots, N
\end{align*}
\]

• This optimization problem is in a nasty form, so we need to do some rewriting

• Let \( \gamma' \) be the functional margin (we have \(\gamma = \gamma' / ||w||\)), we can rewrite this as

\[
\begin{align*}
\max_{w,b} & \quad \frac{\gamma'}{||w||} \\
\text{subject to :} & \quad y^i (w \cdot x^i + b) \geq \gamma', \quad i = 1, \ldots, N
\end{align*}
\]
Maximum Margin Classifier

• Note that we can arbitrarily rescale \( w \) and \( b \) to make the functional margin \( \gamma' \) large or small.

• So we can rescale them such that \( \gamma' = 1 \)

\[
\max_{w,b} \frac{\gamma'}{\|w\|}
\]

subject to: \( y^i (w \cdot x^i + b) \geq \gamma', \ i = 1, \cdots, N \)

\[
\max_{w,b} \frac{1}{\|w\|} \quad \text{(or equivalently min}_{w,b} \|w\|^2)
\]

subject to: \( y^i (w \cdot x^i + b) \geq 1, \ i = 1, \cdots, N \)

Maximizing the geometric margin is equivalent to minimizing the magnitude of \( w \) subject to maintaining a functional margin of at least 1.
Solving the Optimization Problem

\[
\min_{w,b} \frac{1}{2} \|w\|^2
\]
subject to: \( y^i(w \cdot x^i + b) \geq 1, \quad i = 1, \ldots, N \)

- This results in a quadratic optimization problem with linear inequality constraints.

- This is a well-known class of mathematical programming problems for which several (non-trivial) algorithms exist.
  - In practice, we can just regard the QP solver as a “black-box” without bothering how it works

- You will be spared of the excruciating details and jump to...
The solution

• We can not give you a close form solution that you can directly plug in the numbers and compute for an arbitrary data sets

• But, the solution can always be written in the following form

\[ w = \sum_{i=1}^{N} \alpha_i y^i x^i, \text{ s.t. } \sum_{i=1}^{N} \alpha_i y^i = 0 \]

• This is the form of \( w \), the value for \( b \) can be calculated accordingly using some additional steps

• A few important things to note:
  – The weight vector is a *linear combination of all the training examples*
  – Importantly, many of the \( \alpha_i \)'s are zeros
  – These points that have non-zero \( \alpha_i \)'s are the **support vectors**
A Geometrical Interpretation

$\alpha_5 = 0$
$\alpha_4 = 0$
$\alpha_9 = 0$
$\alpha_3 = 0$
$\alpha_6 = 1.4$
$\alpha_8 = 0.6$
$\alpha_{10} = 0$
$\alpha_7 = 0$
$\alpha_2 = 0$
$\alpha_1 = 0.8$

$w^T x + b = 1$
$w^T x + b = 0$
$w^T x + b = -1$
A few important notes regarding the geometric interpretation

- \( w^T x + b = 0 \) gives the decision boundary
- Positive support vectors lie on the line of
  \[ w^T x + b = 1 \]
- Negative support vectors lie on the line of
  \[ w^T x + b = -1 \]
- We can think of the above two lines as defining a fat decision boundary
  - The support vectors exactly touches the two sides of the fat boundary
  - Learning involves adjusting the location and orientation of the decision boundary so that it can be as fat as possible without eating up the training examples
Summarization So Far

- We defined margin (functional, geometric)
- We demonstrated that we prefer to have linear classifiers with large geometric margin.
- We formulated the problem of finding the maximum margin linear classifier as a quadratic optimization problem
- This problem can be solved using efficient QP algorithms that are available.
  - The solutions for $\mathbf{w}$ and $b$ are very nicely formed
- Do we have our perfect classifier yet?
Non-separable Data and Noise

What if the data is not linearly separable?
- The solution does not exist
- i.e., the set of linear constraints are not satisfiable
- But we should still be able to find a good decision boundary

What if we have noise in data?
- The maximum margin classifier is not robust to noise!
Soft Margin

• Allow functional margins to be less than 1

Originally functional margins need to satisfy:
\[ y_i(w \cdot x^i + b) \geq 1 \]

Now we allow it to be less than 1:
\[ y_i(w \cdot x^i + b) \geq 1 - \xi_i \quad \& \quad \xi_i \geq 0 \]

The objective changes to:
\[
\min_{w,b} \left\| w \right\|^2 + c \sum_{i=1}^{N} \xi_i
\]
Soft-Margin Maximization

\[
\begin{align*}
\min_{w,b} & \|w\|^2 \\
\text{subject to: } & y^i (w \cdot x^i + b) \geq 1, \quad i = 1, \ldots, N
\end{align*}
\]

\[
\begin{align*}
\min_{w,b} & \|w\|^2 + c \sum_{i=1}^{N} \xi_i \\
\text{subject to: } & y^i (w \cdot x^i + b) \geq 1 - \xi_i, \quad i = 1, \ldots, N \\
& \xi_i \geq 0, \quad i = 1, \ldots, N
\end{align*}
\]

- This allows some functional margins < 1 (could even be < 0)
- The \( \xi_i \)'s can be viewed as the “errors” of our fat decision boundary
- Adding \( \xi_i \)'s to the objective function to minimize errors
- We have a tradeoff between making the decision boundary fat and minimizing the error
- Parameter \( c \) controls the tradeoff:
  - Large \( c \): \( \xi_i \)'s incur large penalty, so the optimal solution will try to avoid them
  - Small \( c \): small cost for \( \xi_i \)'s, we can sacrifice some training examples to have a large classifier margin
Solutions to soft-margin SVM

\[ w = \sum_{i=1}^{N} \alpha_i y^i x^i, \quad \text{s.t.} \sum_{i=1}^{N} \alpha_i y^i = 0 \]

No soft margin

\[ w = \sum_{i=1}^{N} \alpha_i y^i x^i, \quad \text{s.t.} \sum_{i=1}^{N} \alpha_i y^i = 0 \text{ and } 0 \leq \alpha_i \leq c \]

With soft margin

- \( c \) effectively puts a **box constraint** on \( \alpha \), the weights of the support vectors
- It limits the influence of individual support vectors (maybe outliers)
- In practice, \( c \) is a parameter to be set, similar to \( k \) in k-nearest neighbor
- It can be set using cross-validation
How to make predictions?

For classifying with a new input \( z \)

Compute

\[
 w \cdot z + b = \left( \sum_{j=1}^{s} \alpha_{i_j} y^{i_j} x^{i_j} \right) \cdot z + b = \sum_{j=1}^{s} \alpha_{i_j} y^{i_j} (x^{i_j} \cdot z) + b
\]

classify \( z \) as + if positive, and - otherwise

Note: \( w \) need not be computed/stored explicitly, we can store the \( \alpha_i \)'s, and classify \( z \) by using the above calculation, which involves taking the dot product between training examples and the new example \( z \)

In fact, in SVM both learning and classification can be achieved using dot products between pair of input points:
--- this allows us to do the **Kernel trick**, that is to replace the dot product with something called a kernel function.
Non-linear SVMs

- Datasets that are linearly separable with some noise work out great:

- But what are we going to do if the dataset is just too hard?
Mapping the input to a higher dimensional space can solve the linearly inseparable cases.
Non-linear SVMs: Feature Spaces

- General idea: For any data set, the original input space can always be mapped to some higher-dimensional feature spaces such that the data is linearly separable:

\[ x \rightarrow \Phi(x) \]
Example: Quadratic Feature Space

- Assume $m$ input dimensions
  $$x = (x_1, x_2, \ldots, x_m)$$
- Number of quadratic terms:
  $$1 + m + m + m(m-1)/2 \approx m^2$$
- The number of dimensions increase rapidly!

You may be wondering about the $\sqrt{2}$’s
At least they won’t hurt anything!
You will find out why they are there soon!
Dot product in quadratic feature space

\[ \Phi(a) \cdot \Phi(b) = \]

\[
1 + 2 \sum_{i=1}^{m} a_i b_i + \sum_{i=1}^{m} a_i^2 b_i^2 + \sum_{i=1}^{m} \sum_{j=i+1}^{m} 2a_i a_j b_i b_j
\]

Now let’s just look at another interesting function of \((a \cdot b)\):

\[(a \cdot b + 1)^2\]

\[= (a \cdot b)^2 + 2(a \cdot b) + 1\]

\[= (\sum_{i=1}^{m} a_i b_i)^2 + 2 \sum_{i=1}^{m} a_i b_i + 1\]

\[= \sum_{i=1}^{m} \sum_{j=1}^{m} a_i a_j b_i b_j + 2 \sum_{i=1}^{m} a_i b_i + 1\]

\[= \sum_{i=1}^{m} a_i^2 b_i^2 + 2 \sum_{i=1}^{m} \sum_{j=i+1}^{m} a_i a_j b_i b_j + 2 \sum_{i=1}^{m} a_i b_i + 1\]

They are the same! And the later only takes \(O(m)\) to compute!
Kernel Functions

• If every data point is mapped into high-dimensional space via some transformation \( x \rightarrow \phi(x) \), the dot product that we need to compute for classifying a point \( x \) becomes:

\[
<\phi(x^i) \cdot \phi(x)> \text{ for all support vectors } x^i
\]

• A kernel function is a function that is equivalent to an dot product in some feature space.

\[
k(a,b) = <\phi(a) \cdot \phi(b)>
\]

• We have seen the example:

\[
k(a,b) = (a \cdot b + 1)^2
\]

This is equivalent to mapping to the quadratic space!

• A kernel function can often be viewed measuring similarity
More kernel functions

• Linear kernel: \( k(a, b) = (a \cdot b) \)

• Polynomial kernel: \( k(a, b) = (a \cdot b + 1)^d \)

• Radial-Basis-Function kernel:

\[
K(a, b) = \exp\left( -\frac{(a - b)^2}{2\sigma^2} \right)
\]

In this case, the corresponding mapping \( \phi(x) \) is *infinite-dimensional!* Lucky that we don’t have to compute the mapping explicitly!

\[
w \cdot \Phi(z) + b = \sum_{j=1}^{s} \alpha_{i,j} y^{i,j} (\Phi(x^{i,j}) \cdot \Phi(z)) + b = \sum_{j=1}^{s} \alpha_{i,j} y^{i,j} K(x^{i,j} \cdot z) + b
\]

Note: We will not get into the details but the learning of \( w \) can be achieved by using kernel functions as well!
Nonlinear SVM summary

- Using kernel functions in place of dot product, we, in effect, map the input space to a high dimensional feature space and learn a linear decision boundary in the feature space.
- The decision boundary will be nonlinear in the original input space.
- Many possible choices of kernel functions.
  - How to choose? Most frequently used method: cross-validation.
Strength vs weakness

- **Strengths**
  - The solution is globally optimal
  - It scales well with high dimensional data
  - It can handle non-traditional data like strings, trees, instead of the traditional fixed length feature vectors
    - Why? Because as long as we can define a kernel (similarity) function for such input, we can apply svm

- **Weakness**
  - Need to specify a good kernel
  - Training time can be long if you use the wrong software package

Table 1. Training time in CPU-seconds

<table>
<thead>
<tr>
<th></th>
<th>Pegasos</th>
<th>SVM-Perf</th>
<th>SVM-Light</th>
</tr>
</thead>
<tbody>
<tr>
<td>CCAT</td>
<td>2</td>
<td>77</td>
<td>20,075</td>
</tr>
<tr>
<td>Covertype</td>
<td>6</td>
<td>85</td>
<td>25,514</td>
</tr>
<tr>
<td>astro-ph</td>
<td>2</td>
<td>5</td>
<td>80</td>
</tr>
</tbody>
</table>

2007 2006 1998