SVM Cost function derived from Constrained Optimization

Here is an alternate (and slightly more mathematical) derivation of the SVM Cost function.

Recall our dual objectives

- Maximize margin (width of buffer zone)
- Minimize classification loss (incorrectly classified or in-buffer training examples)

The natural way to express these objectives is as a Constrained Optimization problem

- Maximize margin
- subject to not violating a boundary constraint (all examples on correct side of margin boundary)

We already showed how maximizing the margin was equivalent to minimizing the margin penalty

$$\frac{1}{2}\Theta_{-0}^{T} \cdot \Theta_{-0}$$

Each per-example Classification Loss

$$\max\left(0, 1 - \dot{\mathbf{y}}^{(\mathbf{i})} * s(\hat{\mathbf{x}}^{(\mathbf{i})})\right)$$

is equivalent to an inequality constraint

$$1 - \dot{\mathbf{y}}^{(\mathbf{i})} * s(\hat{\mathbf{x}}^{(\mathbf{i})}) \leq 0$$

$$\dot{\mathbf{y}}^{(\mathbf{i})} * s(\hat{\mathbf{x}}^{(\mathbf{i})}) \geq 1 \qquad \text{re-arranging the terms}$$

So we can express the Loss Function minimization as a constrained optimization problem

$$\begin{aligned}&\text{minimize } \tfrac{1}{2}\Theta_{-0}^{T}\Theta_{-0}\\&\text{subject to } \dot{\mathbf{y}}^{(\mathbf{i})} * s(\hat{\mathbf{x}}^{(\mathbf{i})}) \geq 1 \text{for } i = 1, \ldots, m\end{aligned}$$

The objective

$$\frac{1}{2}\Theta_{-0}^{T}\Theta_{-0}$$

is quadratic so this is a Quadratic Optimization problem.

Solving Constrained Optimization: LaGrangian multipliers

It is beyond the scope of this course, but one way to solve constrained minimization is to create an objective function $\mathcal{L}$ that is the sum of

- the function to minimize
- λ times each constraint (when it is rewritten in a form where the inequality is with respect to 0)

The λ terms are called *Lagrangian multipliers*.

They serve as penalties when a constraint is violated (i.e., is greater than 0).

The (per example) objective function $\mathcal{L}$ becomes

$$\frac{1}{2}\Theta_{-0}^{T}\Theta_{-0} + \lambda * (\max(0, -1 * \dot{\mathbf{y}}^{(\mathbf{i})} * s(\mathbf{x}^{(\mathbf{i})})))$$

which is equal to the SVM Cost function that we constructed in the ad hoc fashion if we let $\lambda = C$.

Recall C expressed the relative weight between the Margin Penalty and Classification Loss.

Aside

- You will sometimes see the max replaced by a "slack" variable $\xi^{(\mathbf{i})}$

$$\xi^{(\mathbf{i})} = \max(0, -1 * \dot{\mathbf{y}}^{(\mathbf{i})} * s(\mathbf{x}^{(\mathbf{i})}))$$

  This is a way of turning an inequality

$$\dot{\mathbf{y}}^{(\mathbf{i})} * s(\hat{\mathbf{x}}^{(\mathbf{i})}) \geq 1$$

  into an equality

    - i.e., the "slack" is the amount of the difference of the expression from being equal
- Technically, there should be a separate Lagrangian for each constraint
    - should write $\lambda^{(\mathbf{i})}$ for $i = 1 \leq i \leq m$

Asides

- Write penalty with the $\frac{1}{2}$ in front so that the derivative with respect to $\mathbf{w}$ is $\mathbf{w}$.
- The Regularization Penalty is the same as the Margin Penalty of our prior derivation
- The relative weight between Classification Loss and the Regularization Penalty is λ
    - The Regularization Penalty is λ times more important than Classification Loss
    - Prior derivation: Classification Loss was C times as important as Regularization Penalty
    - so $C = \frac{1}{\lambda}$

Support Vector Machines: derivation via landmark similarity

We now show another derivation of the Support Vector Machine.

The SVM will apply a particular kind of transformation ϕ to $\mathbf{x}$ before fitting a linear model.

Landmarks and the Similarity transformation

Let us pick a set of distinguished points $\{\boldsymbol{\ell}^{(1)}, \boldsymbol{\ell}^{(2)}, \ldots\}$ in the input domain.

We will refer to these distinguished points as "landmarks" because we will use them as reference points from which we will measure the similarity (inverse of distance) of each example $\mathbf{x}^{(\mathbf{i})}$ in the training set.

In particular, let us choose n' landmarks and let

$$K(\mathbf{x}, \boldsymbol{\ell}^{(\mathbf{i})})$$

be a measure of similarity between vector $\mathbf{x}$ and the i^{th} landmark.

K will be referred to as a "similarity function" or "kernel".

Then the transformation of $\mathbf{x^{(i)}}$ into

$$\hat{\mathbf{x}}^{\mathbf{(i)}} = [K(\mathbf{x^{(i)}}, \boldsymbol{\ell}^{(1)}), K(\mathbf{x^{(i)}}, \boldsymbol{\ell}^{(2)}), \ldots, K(\mathbf{x^{(i)}}, \boldsymbol{\ell}^{(n')}),]$$

is a representation of the original $\mathbf{x^{(i)}}$ into a "similarity" vector.

The transformed features $\hat{x}^{\mathbf{(i)}}$ is of length n', each element representing the distance of $\mathbf{x^{(i)}}$ to one landmark.

We will do linear classification on these transformed features $\hat{\mathbf{x}}$ rather than the original $\mathbf{x}$.

The linear classifier creates a hyperplane to separate Positive and Negative examples by using the dot product to create a score

$$s(\hat{\mathbf{x}}^{(\mathbf{i})}) = \Theta^T \cdot \hat{\mathbf{x}}^{(\mathbf{i})}$$

based on transformed $\hat{x}^{(\mathbf{i})}$

This score will determine the prediction.

Score:

$$s(\hat{\mathbf{x}}^{(\mathbf{i})}) = \Theta^T \cdot \hat{\mathbf{x}}^{(\mathbf{i})} + b$$

Score to prediction:

$$\hat{\mathbf{y}}^{(\mathbf{i})} = \begin{cases} 0 & \text{if } s(\hat{\mathbf{x}}^{(\mathbf{i})}) < 0 \\ 1 & \text{if } s(\hat{\mathbf{x}}^{(\mathbf{i})}) \geq 0 \end{cases}$$

Note

- Θ and $\hat{\mathbf{x}}$ must have the same length
- The Θ in this derivation is thus **very different** than the Θ in the original derivation
    - in this derivation the length of Θ is m (number of examples) **not** n (number of original features)
    - the elements of this Θ correspond to *landmarks* **not** features in the original dimensions

So once again, the dot product is doing a form of "pattern matching" of features, but now

- we use transformed features: similarity to landmarks
- the pattern is identifying the relative importance of being similar to each landmark

So far, very similar to Logistic Regression: the score determines the prediction.

One difference is that Logistic Regression converts score to probability via a sigmoid function:

$$\hat{p}^{(\mathbf{i})} = \sigma\big(s(\mathbf{x}^{(\mathbf{i})})\big)$$

This probability is needed mainly for the Cost function for Logistic Regression (Binary Cross Entropy) but may be useful as an informative output as well.

We shall soon seen the main difference from Logistic Regression: Hinge Loss rather than Cross Entropy

Choosing the landmarks

TO DO Picture

How do we choose the landmarks ? How many do we choose ?

Let's choose each of the m examples in the training set as a landmark so that $\boldsymbol{\ell}^{(\mathbf{i})} = \mathbf{x}^{(\mathbf{i})}$.

This might initially seem to be a large number of landmarks.

Is it possible to choose fewer ?

Yes, and we will let Machine Learning decide which ones matter!

In [4]:
```
print("Done")
```

```
Done
```