1 Kernel Logistic Regression

We have seen in lecture how to kernelize ridge regression, and by going to the dual formulation, how to kernelize soft-margin SVMs as well. Here, we will consider how to kernelize logistic regression and compare its performance to kernelized SVMs using a real-world dataset.

(a) Imagine we have \(n\) different \(d\)-dimensional data points in an \(n \times d\) matrix \(X\), with associated labels in an \(n\)-dimensional vector \(y\). Let this be a binary classification problem, so each label \(y_i \in \{0, 1\}\). Remember, this means that each training data point is associated with a row in the matrix \(X\) and the vector \(y\).

Recall that logistic regression associates a point \(x\) with a real number from 0 to 1 by computing:

\[
f(x) = \frac{1}{1 + \exp(-w^T x)},
\]

This number can be interpreted as the estimated probability for the point \(x\) having a true label of +1. Since this number is \(\frac{1}{2}\) when \(w^T x = 0\), the sign of \(w^T x\) is what predicts the label of the test point \(x\).

As you’ve seen in earlier homeworks, the loss function is defined to be

\[
L = \sum_i -y_i \ln(f(x_i)) - (1 - y_i) \ln(1 - f(x_i)),
\]

where the label of the \(i\)th point \(x_i\) is \(y_i\).

Write down the gradient-descent update step for logistic regression, with step size \(\gamma\).

Assume that we are working with the raw features \(X\) for now, with no kernelization.

For convenience, define the logistic function \(s(\cdot)\) to be

\[
s(x) = \frac{1}{1 + e^{-x}}.
\]

**Solution:** Notice that

\[
\frac{\partial s}{\partial x} = \frac{e^{-x}}{(1 + e^{-x})^2} = s(x)(1 - s(x)).
\]

We can now rewrite our loss function as

\[
L = \sum_i -y_i \ln(s(w^T x_i)) - (1 - y_i) \ln(1 - s(w^T x_i)),
\]
and differentiate with respect to any particular \( w_j \) to obtain

\[
\frac{\partial L}{\partial w_j} = \sum_i -y_i \frac{s(w^T x_i)(1 - s(w^T x_i))}{s(w^T x_i)} x_i[j] + (1 - y_i) \frac{s(w^T x_i)(1 - s(w^T x_i))}{1 - s(w^T x_i)} x_i[j].
\]

Combining terms and stacking to compute a derivative with respect to \( w \) as a whole, we see that

\[
\left( \frac{\partial L}{\partial w} \right)^T = \sum_i (-y_i(1 - s(w^T x_i)) + (1 - y_i)s(w^T x_i)) x_i = \sum_i (s(w^T x_i) - y_i) x_i.
\]

Thus, the gradient descent step becomes

\[
w^{(t+1)} = w^{(t)} + \gamma \sum_i (y_i - s((w^{(t)})^T x_i)) x_i.
\]

(b) You should have found that the update \( w^{(t+1)} - w^{(t)} \) is a linear combination of the observations \([x_i]\). This suggests that gradient descent for logistic regression might be compatible with the kernel trick. After all, this is the same thing that we saw when we were doing least-squares iteratively by gradient descent, and that was certainly something that we could kernelize.

When we argued for the kernelization of least-squares, we did so by means of the augmented-features view of ridge regression. That had the pedagogic advantage of helping you internalize the importance of norm-minimization and made for an argument by which you could naturally discover the kernelization for yourself. Here, we will pursue a more direct path that has an inductive feeling to it.

Imagine that we start with some weight vector \( w^{(t)} = X^T a^{(t)} \) that is a linear combination of the \([x_i]\). (Notice that if we start with the zero vector as our base case, this is true for the base case.) Show that after one gradient step, \( w^{(t+1)} \) will remain a linear combination of the \([x_i]\), and so is expressible as \( X^T a^{(t+1)} \) for some “dual weight vector” \( a^{(t+1)} \). Then write down the gradient-descent update step for the dual weights \( a^{(t)} \) directly without referring to \( w^{(t)} \) at all. In other words, tell us how \( a^{(t+1)} \) is obtained from the data, the step size, and \( a^{(t)} \).

**Solution:** Substituting in the natural manner, we see that

\[
w^{(t+1)} = X^T a^{(t)} + \gamma \sum_i (y_i - s((w^{(t)})^T x_i)) x_i = X^T (a^{(t)} + \gamma(y - s(Xw^{(t)}))),
\]

applying \( s \) elementwise when a vector is passed in as an argument. We have thus shown that \( w^{(t+1)} \) remains a linear combination of the \( x_i \), and can be written as \( X^T a^{(t+1)} \) where

\[
a^{(t+1)} = a^{(t)} + \gamma(y - s(Xw^{(t)})) = a^{(t)} + \gamma(y - s(XX^T a^{(t)})),
\]

so we have found the gradient-descent update step for \( a^{(t)} \).

(c) You should see from the previous part that the gradient-descent update step for \( a^{(t)} \) can be written to depend solely on \( XX^T \), not on the individual \([x_i]\) in any other way. Since \( XX^T \) is just the Gram matrix of pairwise inner-products of training point inputs, this suggests that we can
use the kernel trick to quickly compute gradient steps for $a^{(t)}$ so long as we can compute the inner products of any pair of featurized observations.

Now suppose that you just have access to a similarity kernel function $k(x, z)$ that can be understood in terms of an implicit featurization $\phi(\cdot)$ so that $k(x, z) = \phi(x)^T \phi(z)$. Describe how you would compute gradient-descent updates for the dual weights $a^{(t)}$ as well as how you would use the final weights together with the training data to classify a test point $x$.

Note: You do not have access to the implicit featurization $\phi(\cdot)$. You have to use the similarity kernel $k(\cdot, \cdot)$ in your final answer.

**Solution:** We simply need to replace $X$ with $\Phi$ from our solution to the previous part, since the rest of the derivation is unchanged. Thus,

$$a^{(t+1)} = a^{(t)} + \gamma(y - s(\Phi^T a^{(t)})).$$

(d) You have now derived kernel logistic regression! Next, we will study how it relates to the kernel SVM, which we will do numerically. **Complete all the parts in the associated Jupyter notebook.**
Contributors:

- Anant Sahai
- Peter Wang
- Rahul Arya