Intro to Machine Learning (CS771A, Autumn 2018)

Practice Problem Set 3

September 14, 2018

Problem 1

Consider a *hard-margin* version of the support vector data description (SVDD) problem

$$\arg\min_{\boldsymbol{c},R} R^2, \text{s.t.} \quad ||\boldsymbol{x}_n - \boldsymbol{c}||^2 \leq R^2, n = 1, \ldots, N$$

So, basically, we want to find the smallest radius ball with center $\boldsymbol{c} \in \mathbb{R}^D$ that encloses *all* the points. Note that we are calling it the hard-margin version because doesn't allow for slacks, i.e., no points are allowed by violate the within-the-ball constraint (unlike the version with slacks we saw in the class when talking about SVDD).

Construct the Langangian for this problem, obtain the expression for the optimal center $\boldsymbol{c}$ and the optimal radius R in terms of the Lagrange multipliers (just like we did for SVMs where the weight vector $\boldsymbol{w}$ was expression in terms of the α_n's), and finally construct the dual problem. You do not need to optimize the dual problem to solve for the Lagrange multipliers. What will be the support vectors for this problem?

Problem 2

Consider a kernel function $k(\boldsymbol{x}, \boldsymbol{z}) = \boldsymbol{x}^\top \boldsymbol{z} + 2(\boldsymbol{x}^\top \boldsymbol{z})^2$. As we know, each kernel has an associated feature mapping $\boldsymbol{\phi}$ such that $k(\boldsymbol{x}, \boldsymbol{z}) = \boldsymbol{\phi}(\boldsymbol{x})^\top \boldsymbol{\phi}(\boldsymbol{z})$. Assume that each input originally has two features (so $\boldsymbol{x} = \{x_1, x_2\}, \boldsymbol{z} = \{z_1, z_2\}$). Write down the feature mapping $\boldsymbol{\phi}$ associated with the kernel defined above.

Problem 3

In the class, we have seen gradient based (i.e., first order) method for learning the weight vector $\boldsymbol{w} \in \mathbb{R}^D$ of the logistic regression model $p(y_n = 1|\boldsymbol{x}_n, \boldsymbol{w}) = \frac{1}{1+\exp(-\boldsymbol{w}^\top \boldsymbol{x}_n)}$ (assuming $y_n \in \{0, 1\}$). Use Newton's (i.e., second order) method to derive the updates of $\boldsymbol{w}$ in each step. The Newton updates are of the form $\boldsymbol{w}^{(t+1)} = \boldsymbol{w}^{(t)} - {\mathbf{H}^{(t)}}^{-1} \boldsymbol{g}^{(t)}$ where $\boldsymbol{g}^{(t)}$ and $\mathbf{H}^{(t)}$ denote the gradient and Hessian, respectively, of the NLL.

Denote the $N \times D$ feature matrix as $\mathbf{X}$ and the $N \times 1$ label vector as $\boldsymbol{y}$. Show that each Newton update can be reduced to the form $\boldsymbol{w}^{(t+1)} = (\mathbf{X}^\top \mathbf{S}^{(t)} \mathbf{X})^{-1} \mathbf{S}^{(t)} \mathbf{X}^\top \hat{\boldsymbol{y}}^{(t)}$ where $\mathbf{S}^{(t)}$ is an $N \times N$ diagonal matrix and $\hat{\boldsymbol{y}}^{(t)}$ is a modified output vector (that changes in each iteration). Note that this expression of $\boldsymbol{w}^{(t+1)}$ is similar to the solution of a weighted least squares regression problem of the form $\arg\min_{\boldsymbol{w}} \sum_{n=1}^N s_n (y_n - \boldsymbol{w}^\top \boldsymbol{x}_n)^2$, where each input $\boldsymbol{x}_n$ has a different importance s_n, and the importances are changing in each iteration. Note: This is popularly known as the iteratively reweighted least squares algorithm for learning logistic regression.

Problem 4

Consider an $N \times M$ matrix $\mathbf{X}$ and let us assume that we are approximating it by a product of two matrices as $\mathbf{X} \approx \mathbf{U}\mathbf{V}^\top$, where $\mathbf{U} \in \mathbb{R}^{N \times K}$ and $\mathbf{V} \in \mathbf{R}^{M \times K}$. Assume K to be some small number. This is an example of matrix factorization. Note that one of the benefits of this factorization is that it enables us to approximate the $N \times M$ matrix $\mathbf{X}$ using only $(N + M) \times K$ parameters (which will give substantial compression when K is small enough).

Suppose we want our approximation to be such that $||\mathbf{X} - \mathbf{U}\mathbf{V}^\top||_F^2 = \sum_{n=1}^N \sum_{m=1}^N (X_{nm} - \boldsymbol{u}_n^\top \boldsymbol{v}_m)^2$ is minimized, so we want to solve the

$$\{\hat{\mathbf{U}}, \hat{\mathbf{V}}\} = \arg\min_{\mathbf{U},\mathbf{V}} ||\mathbf{X} - \mathbf{U}\mathbf{V}^\top||_F^2$$

Jointly optimizing this objective is not possible. Give an alternating optimization scheme for $\mathbf{U}$ and $\mathbf{V}$ that optimizes one row $\boldsymbol{u}_n \in \mathbb{R}^K$ of $\mathbf{U}$ at a time, keeping all other rows and $\mathbf{V}$ as fixed to the current estimates, and likewise one row $\boldsymbol{v}_m \in \mathbb{R}^K$ of $\mathbf{V}$ at a time, keeping all other rows and $\mathbf{U}$ as fixed to the current estimates. Show that each of these subproblems is a *regression* problem with a certain "feature matrix" and a response vector, and derive the expression for the optimal $\boldsymbol{u}_n$ and $\boldsymbol{v}_m$ (you don't need to derive it; just identifying that each of the subproblems is a regression problem, you can write down the solution since you know what the expression for the optimal regression weight vector is when the loss function is squared loss).

Problem 5

Consider a clustering algorithm somewhat similar in spirit to K-means but with a minor difference. Instead of modeling each cluster by a mean μ_k, we will model it by a Gaussian distribution with mean μ_k and covariance matrix Σ_k. The algorithm will mostly be identical to K-means except how we would assign points to clusters (and the fact that we also need to update the covariances in each iteration of the algorithm).

Point-to-cluster assignment: Unlike in K-means where we compare the distances of a point from the each of the K means and assign the point to the cluster with the closest mean, in this algorithm we will compute the *probability density* of the point under each of the K Gaussians and assign the point to the cluster under which it has the largest probability density. Derive the expression that performs this assignment. How it is different (and in good/bad way) from the cluster assignment expression in K-means? In what condition will this cluster assignment rule be exactly the same as the K-means cluster assignment rule?

Also give the full sketch of this algorithm similar to K-means, where the first step does point to cluster assignment (based on the above criteria) and the second step computes (MLE for) the Gaussian's parameters (*given* the cluster assignments from the first step). For the second step, you don't need to derive it. We have already seen the MLE equations for Gaussians.

Does the above algorithm fix the issue of K-means where a point's assignment to a cluster does not depend on how many points are (currently) assigned to that cluster?