TTIC 31020: Introduction to Machine Learning **Winter 2025**

Homework 5

Due: 3 p.m., February 6th, 2025

Note *You may discuss these problems in groups. However, you must write up your own solutions and mention the names of the people in your group. Also, please do mention any books, papers or other sources you refer to. We prefer and encourage that you typeset solutions in LaTeX for written assignments. Handwritten solutions that are not neat, organized and with clear and easy-to-read handwriting will not be graded. Please submit your solutions as a PDF document on Canvas.*

Challenge and Optional Questions *Challenge questions are marked with* challenge *and optional questions are marked with* optional *. You can get extra credit for solving* challenge *questions. You are not required to turn in* optional *questions, but we encourage you to solve them for your understanding. Course staff will help with* optional *questions but will prioritize queries on non-*optional *questions.*

Question 1 was originally an optional problem (Question 2(c)) in HW4. However, it is now a required problem, so you must answer it even if you have previously answered it in HW4.

1. $0/1$ **Loss vs Squared Loss vs Hinge Loss**

   In machine learning, while we ultimately care about minimizing the $0/1$ loss, it is often difficult to optimize directly because the $0/1$ loss is discontinuous and non-convex. Instead, we minimize a continuous and often convex surrogate loss, such as the squared loss and the hinge loss, which are easier to optimize. However, this does not guarantee that the solution will also achieve a small $0/1$ loss.

   Recall that the squared loss is defined as $\ell^{\text{sq}}(h(x); y) = (y - h(x))^2$ and $L_S^{\text{sq}}(h) = \frac{1}{m}\sum_{i=1}^m \ell^{\text{sq}}(h(x_i); y_i)$. The hinge loss is defined as $\ell^{\text{hinge}}(h(x); y) = [1 - yh(x)]_+$ and $L_S^{\text{hinge}}(h) = \frac{1}{m}\sum_{i=1}^m \ell^{\text{hinge}}(h(x_i); y_i)$. We further define $\hat{h}_{\text{sq}} = \arg\min_{h\in\mathcal{H}} L_S^{\text{sq}}(h)$ being the predictor minimizing the squared loss and $\hat{h}_{\text{hinge}} = \arg\min_{h\in\mathcal{H}} L_S^{\text{hinge}}(h)$ being the predictor minimizing the hinge loss

   In the questions below, you can just consider points in $\mathcal{X} = \mathbb{R}^2$ with binary labels and the class of linear predictors $\mathcal{H} = \{h_w(x) = \langle w, x\rangle \mid w \in \mathbb{R}^2\}$.

   (a) We only consider realizable case in the above. Now we will see the optimal solution of the hinge loss won't guarantee a small $0/1$ loss in the unrealizable setting.
   Provide an explicit set of points $S = \{(x_i, y_i) \mid i \in [m]\}$ such that the following two conditions hold simultaneously:

   i. $\inf_{h\in\mathcal{H}} L_S^{01}(h) \le 0.1$.
   ii. $L_S^{01}(\hat{h}_{\text{hinge}}) \ge 0.5$.

   Write down:

   i. The set of points S such that the above conditions hold. You may pick any number of points to achieve this.
   ii. A predictor $h \in \mathcal{H}$ minimizing $0/1$ loss. Evaluate $L_S^{01}(h)$ and $L_S^{\text{hinge}}(h)$.
   iii. The predictor $\hat{h}_{\text{hinge}}$. Evaluate $L_S^{01}(\hat{h}_{\text{hinge}})$ and $L_S^{\text{hinge}}(\hat{h}_{\text{hinge}})$.

2. **Kernel Perceptron**

   Recall the Perceptron algorithm from class. Given a data set $S = ((x_1, y_1), \ldots, (x_m, y_m)) \in (\mathcal{X} \pm \{\pm 1\})^m$, it outputs $\hat{w} \in \mathbb{R}^d$ such that $\forall_i y_i \langle \hat{w}, \phi(x_i)\rangle > 0$, if such a vector exists. The implementation discussed in class is based on explicit access to the feature map $\phi(\cdot)$, and requires storing vectors in

$\mathbb{R}^d$. Here, we will see how to implement the Perceptron algorithm, and predict based on the learned predictor, using only access to a kernel $K(x, x')$ such that $K(x, x') = \langle \phi(x), \phi(x') \rangle$.

For convenience, we will use the following matrix notation:

$$\Phi = \begin{pmatrix} - & \phi(x_1)^T & - \\ & \vdots & \\ - & \phi(x_m)^T & - \end{pmatrix} \in \mathbb{R}^{m \times d} \quad y = \begin{pmatrix} y_1 \\ \vdots \\ y_m \end{pmatrix} \in \mathbb{R}^m .$$

We also define the Gram matrix $G \in \mathbb{R}^{m \times m}$, where $G(i, j) = \langle \phi(x_i), \phi(x_j) \rangle$, i.e., $G = \Phi\Phi^\top$.

(a) Show that, in any iteration t, the iterate w_t maintained by the Perceptron algorithm is in the span of $\{\phi(x_1), \ldots, \phi(x_m)\}$, i.e., we can represent w_t as $\sum_{i=1}^m \alpha_t[i]\phi(x_i) = \Phi^\top \alpha_t$ for some $\alpha_t \in \mathbb{R}^m$ (you may want to use induction over t).

(b) Show how to implement each step of the Perceptron algorithm with access only to K but not ϕ, and represent w_t in terms of α_t. In other words, your algorithm should compute a sequence of iterates $\alpha_t \in \mathbb{R}^m$ such that $\Phi^\top \alpha_t$ are exactly the iterates w_t of the Perceptron algorithm, but with access only to K and not ϕ.

(c) If we use TIME_K to denote the time for computing one $K(x, x')$, what is the runtime for each Perceptron iteration? Note that this runtime is independent of d. challenge Describe an implementation where the runtime per iteration is $O(m \cdot \text{TIME}_K)$.

(d) Assume that there exists $w^* \in \mathbb{R}^d$ such that $\forall_i y_i \langle w^*, \phi(x_i) \rangle \geq \gamma$. Use the Perceptron analysis from HW4 to show that the number of iterations for which the Perceptron algorithm will run is at most $T_{max} := \frac{\|w^*\|_2^2 \cdot \max_{i \in [m]} K(x_i, x_i)}{\gamma^2}$. Use this to bound the overall runtime and overall memory requirements of the Kernelized Perceptron, in terms of T_{max}, m, and TIME_K to compute $K(x, x')$.

(e) After running the Kernelized Perceptron algorithm, what must be stored in order to predict the label of test points? In other words, what must the .fit() method store in the predictor instance to be used in .predict(). Using only K and not ϕ, how can the prediction $\text{sign}(\langle w_T, \phi(x) \rangle)$ on any input test point x be computed? Here w_T is the last iterate and hence predictor obtained by Perceptron. What is the memory requirement and the prediction runtime per test point?

challenge If we want the learned predictor to generalize well, how would you expect m and $T_{\max}$ to compare? Use this when considering the runtime and suggest an implementation of training and prediction with the best runtime considering this relationship.

3. **Kernel Ridge Regression**

We discussed in class learning with the squared loss, and in particular by minimizing the empirical squared loss

$$L_S(w) = \frac{1}{m} \sum_{i=1}^m (\langle w, \phi(x_i) \rangle - y_i)^2$$

where $S = ((x_1, y_1), \ldots, (x_m, y_m))$ is the training set, $\phi(x_i) \in \mathbb{R}^d$, $y_i \in \mathbb{R}$ and $w \in \mathbb{R}^d$. Consider the same notation of Φ and y as in the previous question, then $L_S(w) = \frac{1}{m}\|\Phi w - y\|^2$.

As with binary classification, we might want to add a regularization term encouraging low norm solutions. This is known as *Ridge Regression*, as is defined by minimizing the following objective:

$$L_{S,\lambda}(w) = L_S(w) + \lambda \|w\|^2/2 .$$

A more general version of the Representer theorem holds in this case. Any minimizer $\hat{w}_\lambda$ of $L_{S,\lambda}$ is a linear combination of $\phi(x_1), \ldots, \phi(x_m)$. In other words, there exist $\alpha = (\alpha_1, \ldots, \alpha_m) \in \mathbb{R}^m$ such that

$$\hat{w}_\lambda = \sum_{i=1}^m \alpha_i \phi(x_i) .$$

General form of the Representer theorem. Consider any objective function $J(w)$ that depends on w only through $\langle w, \phi(x_i)\rangle$, i.e., it can be written as $J(w) = J_S(\langle w, \phi(x_1)\rangle, \ldots, \langle w, \phi(x_m)\rangle)$. In particular, this includes the empirical loss $L_S(h_w)$ for linear predictors $h_w(x) = \langle w, \phi(x)\rangle$. Then for any $\hat{w} = \arg\min J(w) + \lambda\|w\|_2^2$ for any $0 < \lambda < \infty$, or alternatively, any Pareto optimal solution of $J(w)$ and $\|w\|_2^2$, we have that $\hat{w}$ is in the span of $\{\phi(x_1), \ldots \phi(x_m)\}$.

(a) For $w(\alpha) = \sum_{i=1}^m \alpha_i \phi(x_i)$, show that $L_{S,\lambda}(\alpha) := L_{S,\lambda}(w(\alpha))$ can be expressed in terms of $\alpha = (\alpha_1, \ldots, \alpha_m)$ and (G, y, λ), without directly referring to w nor ϕ. Recall that G is the Gram matrix.

(b) Solve the minimization problem $\hat{\alpha}_\lambda = \arg\min L_{S,\lambda}(\alpha)$ and write down an explicit closed form expression for $\hat{\alpha}_\lambda$ in terms of G, y and λ.

(c) Show how to use $\hat{\alpha}_\lambda$ to calculate the prediction $\langle \hat{w}_\lambda, \phi(x)\rangle$ for any test point x, using only K but not ϕ.

4. **Experimentation** Look at the Jupyter Notebook. You will use the least-squares formulation derived in (b) to implement ridge regression on the California housing dataset in `scikit-learn`.

5. **Experimentation: SVM with non-standard kernels**

(a) **Prelude**: Look at the Jupyter Notebook for an exploratory exercise using kernel SVM on the spiral dataset.

(b) **Sentiment Analysis**: We have learned the definition of kernelized SVM, and we have mostly discussed kernels over feature vectors. In this part, we will use a kernel directly on the objects in question. We will implement SVM classifiers for a text classification task to classify the sentiment of tweets to airlines. See the Jupyter Notebook for more details.

6. **Review and Comprehension: SVM Objective** optional

In this optional question we will better understand the SVM objective.

Recall that in order to be used in SVM learning, a kernel function $K(x, x')$ must satisfy there exists some feature map $\phi(x)$ such that $K(x, x') = \langle \phi(x), \phi(x')\rangle$. A necessary and sufficient condition is that K is positive semi-definite. That is, for any m and any set of points $x_1, \ldots, x_m$, the Gram matrix $G \in \mathbb{R}^{m\times m}$ defined by $G_{ij} = K(x_i, x_j)$ is positive semi-definite (for every non-zero column vector $z \in \mathbb{R}^m$, $z^T G z \geq 0$, or equivalently, G is symmetric and all eigenvalues are non-negative).

(a) optional Show that if K_1 and K_2 are valid kernels then their sum $K(x, x') = K_1(x, x') + K_2(x, x')$ is also a valid kernel. Do so by writing down the feature map ϕ corresponding to K, in terms of the feature maps ϕ_1 and ϕ_2 corresponding to K_1 and K_2 respectively.

In the next question we'll understand what can go wrong if K is not positive semi-definite. Consider a training set consisting of two points $(x_1, y_1)(x_2, y_2)$ where $K(x_1, x_1) = K(x_2, x_2) = 1$, $K(x_1, x_2) = 2$, $y_1 = 1$ and $y_2 = -1$.

(b) optional Write down the Gram matrix, its two eigenvectors, and its two eigenvalues. Notice that one of them is negative. You can use `numpy.linalg.eig` to compute the eigenvalues and eigenvectors numerically.

(c) optional What is the infimum of the SVM training objective for this data with $\lambda = 1$? Construct an explicit sequence of solutions to the SVM training objective that converges to this infimum.

We will now see that this cannot happen if the Kernel is positive semi-definite. In particular, we will see that, at least if $\lambda > 0$, the infimum of the SVM objective is always attained.

(d) optional Prove that with a positive semi definite kernel and for any $\lambda \geq 0$, the infimum of the SVM objective is always between zero and one (i.e. bounded in $[0, 1]$).

(e) optional For $\lambda > 0$, provide a bound on the norm of the minimizer of the SVM objective. In other words, you need to give a bound $B(\lambda)$, such that the SVM optimization problem does not change if we impose the additional constraint $\|w\| \leq B(\lambda)$. You can do so by showing that for any w where $w > B(\lambda)$, there is always some other (trivial) w' such that the value of the SVM

objective on w' is better then on w, hence we never need to consider w with $\|w\| > B(\lambda)$. (This is an important observation, since it both allows us to relate λ to the norm of the solution of SVM training, and it allows us to limit our attention only to predictors with small norm, which is often convenient).