Homework 3, Machine Learning, Fall 2022

***IMPORTANT* Homework Submission Instructions**

1. All homeworks must be submitted in one PDF file to Gradescope.
2. Please make sure to select the corresponding HW pages on Gradescope for each question.
3. For all theory problems, please type the answer and proof using Latex or Markdown, and export the tex or markdown into a PDF file.
4. For all coding components, complete the solutions with a Jupyter notebook/Google Colab, and export the notebook (including both code and outputs) into a PDF file. Concatenate the theory solutions PDF file with the coding solutions PDF file into one PDF file which you will submit.
5. Failure to adhere to the above submission format may result in penalties.

All homework assignments must be your independent work product, no collaboration is allowed. You may check your assignment with others or the internet after you have done it, but you must actually do it by yourself. **Please copy the following statements at the top of your assignment file**:

Agreement 1) This assignment represents my own work. I did not work on this assignment with others. All coding was done by myself.

Agreement 2) I understand that if I struggle with this assignment that I will reevaluate whether this is the correct class for me to take. I understand that the homework only gets harder.

1 Hoeffding and Beyond: Concentration Inequalities in Statistical Learning (Ed Tam)

In your probability courses, you learned about basic inequalities, such as Markov's Inequality. Below, we will review Markov's inequality, and build up from there a variety of other inequalities that are used extremely often in statistical learning theory. These inequalities are examples of what is generally known as "concentration inequalities." You may have seen these in probability class, but it is important to know them well, because they create the foundation of statistical learning theory.

Note: We understand that this concentration inequalities material is not easy. We chose this material because it is important and we want students to get exposure and practice. For this question, you are allowed to consult materials (e.g. textbooks, external lecture notes, online resources, etc.) to learn the related proof ideas/techniques. However, you must write up your own work and cite/reference any sources that you consulted. Make sure you prove exactly the expression we ask you for!

1.1 Markov's Inequality

Markov's inequality states that for any non-negative random variable X, we have

$$P(X \geq \epsilon) \leq \frac{E(X)}{\epsilon} \quad \text{(Markov's Inequality)}$$

for any positive real number $\epsilon > 0$.

Prove Markov's inequality.

1.2 Chebyschev's Inequality

Chebyschev's inequality states that for any random variable X with mean μ and variance σ^2, we have

$$P(|X - \mu| \geq \epsilon) \leq \frac{\sigma^2}{\epsilon^2} \quad \text{(Chebyschev's Inequality)}$$

for any positive real number $\epsilon > 0$.

Prove Chebyschev's Inequality.

1.3 Polynomial Markov's Inequality

The polynomial version of Markov's inequality states that for any random variable X with mean μ and kth central moment $E[|X - \mu|^k]$, we have

$$P(|X - \mu| \geq \epsilon) \leq \frac{E[|X - \mu|^k]}{\epsilon^k} \quad \text{(Polynomial Version of Markov's Inequality)}$$

for any positive real number $\epsilon > 0$.

Prove the polynomial version of Markov's inequality.

1.4 Chernoff Bound

You might have noticed a pattern by now. All the inequalities that we have seen so far give upper bounds on the probability of a random variable's deviations from the mean. Markov's inequality uses $E(X)$ (the first moment) to obtain an upper bound that is of order $O(\frac{1}{\epsilon})$, Chebyschev's inequality uses the variance (second central moment) to obtain an upper bound of order $O(\frac{1}{\epsilon^2})$, whereas in the polynomial Markov case the kth central moment is used to obtain an upper bound of order $O(\frac{1}{\epsilon^k})$. The higher the order of the moments used, the faster the rate of decay in the upper bound.

It turns out that we can obtain even nicer rates of decay, i.e., an upper bound that decays exponentially fast, by using the moment generating function (MGF) of a random variable (when it exists). We use the notation $M_X(\lambda) := E[\exp(\lambda X)]$ to denote the moment generating function of the random variable X.

Intuitively, the MGF is a function that captures "all the moment information" in a random variable (one can obtain all the moments of a random variable by differentiating the MGF).

(Optional Note: For more informal intuition, given a random variable, one can define a formal Taylor series with the derivative terms in the Taylor series equal to the moments of the given random variable. That Taylor series, under some conditions, will lead to a well-defined function. That function is the MGF.)

Not every random variable has a MGF (sometimes the MGF doesn't exist), but when a random variable does have an MGF, we can use it to obtain an inequality known as the Chernoff bound. You can assume in the rest of this question that all random variables under consideration has a well-defined, finite MGF.

A simple version of the Chernoff bound says the following:

$$P(X - \mu \geq \epsilon) \leq \inf_{\lambda \geq 0} \frac{M_{X-\mu}(\lambda)}{\exp(\lambda\epsilon)} \quad \text{(Chernoff Bound)}$$

where μ is the mean of X, $\lambda > 0$ is any positive real number and ϵ is any real number.

Prove the above version of the Chernoff Bound.

1.5 Hoeffding's inequality

In order to apply Chernoff Bound in practice, we need some way to evaluate/control the expression $E[\exp(\lambda(X-\mu))]$.

It turns out that for any bounded random variable X, Hoeffding's lemma provides a simple way for us to control $E[\exp(\lambda(X-\mu))]$. More precisely, Hoeffding's lemma says that for any random variable X taking values in the interval $[a, b]$ (i.e., $P(a \leq X \leq b) = 1$), the following bound holds true for any real number λ:

$$E[\exp(\lambda(X-\mu))] \leq \exp\left(\frac{\lambda^2(b-a)^2}{8}\right). \quad \text{(Hoeffding's Lemma)}$$

(For this assignment, you can treat Hoeffding's lemma as a given fact and use it freely without proving/deriving it.)

Given $X_1, X_2, \cdots, X_n$ as I.I.D. bounded random variables in the range $[a, b]$ with a common mean μ, for any positive integer $n > 0$ and any real number ϵ, prove the following inequality, which is a one-sided version of Hoeffding's Inequality:

$$P\left(\left(\frac{1}{n}\sum_{i=1}^{n} X_i\right) - \mu \geq \epsilon\right) \leq \exp\left(\frac{-2n\epsilon^2}{(b-a)^2}\right). \quad \text{(Hoeffding's Inequality (one-sided))}$$

(Hint: to find the best λ to use, think about taking an appropriate derivative and setting it to 0)

1.6 Applying concentration inequalities to classifier evaluation

There is a two-sided version of Hoeffding's inequality that is used very often in the literature:

$$P\left(\left|\frac{1}{n}\sum_{i=1}^{n} X_i - \mu\right| \geq \epsilon\right) \leq 2\exp\left(\frac{-2n\epsilon^2}{(b-a)^2}\right). \quad \text{(Hoeffding's Inequality (two-sided))}$$

In the subsequent subquestions, you can use the two sided Hoeffding's inequality directly without needing to prove it (the technique used to prove it is basically the same as the one-sided proof except a bit more tedious).

Suppose we are given a dataset $S := \{(\boldsymbol{x}_i, y_i)\}_{i=1}^n$ drawn i.i.d. from some unknown distribution D, where x_i's represent features in $\mathbb{R}^p$ and the y_i's are binary labels in $\{0, 1\}$.

Suppose we already have a single trained classifier g that maps from $\mathbb{R}^p$ to $\{0, 1\}$, and we would like to evaluate g's performance under the $0-1$ classification loss, denoted l_{01}.

In an ideal world, we would evaluate g based on the "true" expected loss of the classifier g (also known as the risk of g), which is defined and denoted as

$$R(g) := E_D[l_{01}(y, g(\boldsymbol{x}))]$$

where the expectation is taken over the unknown distribution D.

However, in practice, nobody knows what the distribution D is, hence the expectation $E_D[l_{01}(y, g(\boldsymbol{x}))]$ can never be explicitly calculated. In ML, we often use the following expression instead as an approximation:

$$R_S(g) := \frac{1}{n}\sum_{i=1}^{n} l_{01}(y_i, g(\boldsymbol{x}_i))$$

This is known as the empirical risk. We are interested in finding out how many samples n we need in order to for the empirical risk $R_S(g)$ to be a good approximation of the true risk $R(g)$.

Fix any $1 > \delta > 0$ and $\epsilon > 0$. Using Hoeffding's inequality, give a lower bound on the number of samples n needed to ensure that with probability at least $1-\delta$, $|R_S(g) - R(g)|$ is less than ϵ.

Similarly, assuming that each $l_{01}(y_i, g(\boldsymbol{x}_i))$ has variance equal to 1, using Chebyschev's inequality, give a lower bound on the number of samples n needed to ensure that with probability at least $1-\delta$, $|R_S(g)-R(g)|$ is less than ϵ.

Based on the above, in no more than two sentences, give a reason why in practice machine learning researchers often prefer to use the sample complexity lower bound/guarantee given by Hoeffding's inequality over the one given by Chebyschev's inequality.

1.7 Application to Algorithmic Stability

Recall that a learning algorithm (for classification) is a function that takes training data points as input and outputs a classifier.

In statistical learning theory, an important notion is the "stability" of a learning algorithm. Intuitively, a learning algorithm is stable if the risk of the resulting classifier does not change much if we change the training data by a little bit. It turns out that algorithms/models that are stable enjoy good generalization properties. We will explore some elementary aspects of this in this subquestion.

We first define an important mathematical notion:

A function $f : \mathbb{R}^n \to \mathbb{R}$ is said to satisfy the bounded difference property if

$$\sup_{x_1,\cdots,x_n,x'_k} |f(x_1,\cdots,x_k,\cdots,x_n) - f(x_1,\cdots,x'_k,\cdots,x_n)| \le c_k$$

holds for any $1 \le k \le n$. In other words, if you substitute the value x_k at the kth argument of the function by any other arbitrary value x'_k, then the difference in the function is at most c_k.

There is a very commonly used concentration inequality in learning theory called McDiarmid's inequality, one version of which says the following: Let $X_1, \cdots, X_n$ be IID random variables taking values in $\mathbb{R}$, and $f : \mathbb{R}^n \to \mathbb{R}$ be a function that satisfies the bounded differences property. Then

$$P\left(|f(X_1, \cdots, X_n) - E[f(X_1, \cdots, X_n)]| \geq \epsilon\right) \leq 2\exp\left(\frac{-2\epsilon^2}{\sum_{i=1}^n c_i^2}\right). \quad \text{(McDiarmid's Inequality)}$$

In the subquestions below, you can directly use McDiarmid's inequality as a tool without having to prove it.

Suppose we have a training dataset $S := \{\boldsymbol{z}_i = (\boldsymbol{x}_i, y_i)\}_{i=1}^n$ sampled IID from some unknown distribution D. Let S^k be the dataset obtained by replacing the kth training sample $(\boldsymbol{x}_k, y_k)$ in S by another training sample $(\boldsymbol{x}'_k, y'_k)$ that is independently drawn also from D.

A learning *algorithm* that learns a binary classifier g is called β-stable if for any dataset S of size n drawn IID from D and for any $1 \leq k \leq n$,

$$\sup_{x', y'} |l_{01}(g_S(x'), y') - l_{01}(g_{S^k}(x'), y')| \leq \beta$$

where g_S and g_{S^k} denote the classifiers trained on S and S^k respectively.

We are interested the risk of a learning algorithm. This is very related to, albeit slightly different from, the risk of a classifier. The (empirical) risk of a learning algorithm A on a dataset $S := \{(\boldsymbol{x}_i, y_i)\}_{i=1}^n$, is a function of the data points $(\boldsymbol{x}_1, y_1), \cdots, (\boldsymbol{x}_n, y_n)$. Perhaps the most suggestive notation for this risk would be:

$$R_A(S) := R_A((\boldsymbol{x}_1, y_1), \cdots, (\boldsymbol{x}_n, y_n)) := \frac{1}{n}\sum_{i=1}^n l_{01}(g_S(\boldsymbol{x}_i), y_i)$$

where g_S is the binary classifier learned by the algorithm A on the dataset S.

Now suppose we are given a β-stable algorithm A. Using McDiarmid's inequality, show that for any $\epsilon > 0$ and for any $n \geq 1$, the following inequality holds:

$$P(|R_A(S) - E[R_A(S)]| \geq \epsilon) \leq 2\exp\left(\frac{-2\epsilon^2}{n(\beta + \frac{1}{n})^2}\right).$$

2 Classic Exercises in VC Dimension (Ed Tam)

In class we learned about the VC dimension as an important mathematical tool for us to characterize the complexity/simplicity of a family of models, which in turn gives us ways to give guarantees on the generalization performance of the model.

In this question we will get some practice on figuring out the VC dimensions of various classes of classifiers.

In all subquestions of Q2, we are concerned with binary classification.

2.1 VC dimension of a step functions

Find the VC dimension of the following hypothesis class. $\mathcal{F} = \{f : [0, 1] \to \{0, 1\}, f(x) = \mathbf{1}_{x<t}, t \in [0, 1]\}$

2.2 VC dimension of intervals

Find the VC dimension of the following hypothesis class. $\mathcal{F} = \{f : [0,1] \to \{0,1\}, f(x) = \mathbf{1}_{t_1 < x < t_2}, t_1, t_2 \in [0,1]\}$

2.3 VC dimension of hyperplanes through origin

Find the VC dimension of the following hypothesis class. $\mathcal{F} = \{f : \mathbb{R}^d \to \{-1,1\}, f(\mathbf{x}) = \text{sign}(\mathbf{b}^T\mathbf{x}), \mathbf{b} \in \mathbb{R}^d\}$

2.4 VC dimension of binary decision trees

What is the VC dimension of the set of all binary decision trees with number of leaves at most l?

What is the VC dimension of the set of all binary decision trees with number of splits at most d?

In the corner case where we just have a root node in the tree being a leaf, then there is no splits at all so the number of splits is 0.

2.5 VC dimension upper bounds for a finite class

Let $\mathcal{F}$ be a finite hypothesis class for binary classification, i.e. $|\mathcal{F}| < \infty$. Show that the VC dimension of F is upper bounded by $\log_2 |\mathcal{F}|$.

3 Logistic Regression and Kernels (Ed Tam)

Let $\{(\mathbf{x}_i, y_i)\}_{i=1}^n$ be a set of training data, where $\mathbf{x}_i \in \mathbb{R}^d$ for all i, and $y_i \in \{-1, 1\}$.

Consider the l_2 regularized logistic regression model with parameter $\boldsymbol{\theta}$, where we want to find an $f_{\boldsymbol{\theta}}$ that minimizes the loss function

$$\sum_{i=1}^{n} \ln(1 + \exp(-y_i(f_{\boldsymbol{\theta}}(\mathbf{x}_i))) + \lambda \|f_{\boldsymbol{\theta}}\|_{\mathcal{H}}^2.$$

where $f_{\boldsymbol{\theta}}(\mathbf{x})$ is of the form $\boldsymbol{\theta}^\top \mathbf{x}$, and $\|f_{\boldsymbol{\theta}}\|_{\mathcal{H}}^2 = \boldsymbol{\theta}^\top \boldsymbol{\theta}$.

3.1 RKHS

Let $\mathcal{H}$ be the reproducing kernel Hilbert space that corresponds to the above l_2 regularized logistic regression. Using the representer theorem, what is the form of the optimal predictive function?

3.2 L2 Regularization

Define the function g as $g(\zeta) = \ln(1 + \exp(-\zeta))$.

It turns out that l_2 regularized logistic regression is closely related to the following primal optimization problem

$$\min_{\mathbf{w}, \boldsymbol{\zeta}} \frac{1}{2}\|\boldsymbol{w}\|^2 + C\sum_{i=1}^{n} g(\zeta_i)$$

subject to

$$y_i(\boldsymbol{w}^T\mathbf{x}_i) \geq \zeta_i, \forall i$$

where $C > 0$ is some positive constant.

Derive the dual formulation of this optimization problem (you are not required to solve the dual problem, you are just required to state it). You should simplify the dual formulation that you obtain so that it would only depend on $\boldsymbol{\alpha}$, the y_i's and $\boldsymbol{x}_i$'s, as well as C.

4 Kernel Power on SVM and Regularized Logistic Regression (Ray)

The kernel trick states that if we have an algorithm, like SVM in this particular problem, where the examples/observations appear only in inner products, you can freely replace the inner product with a different one. In other words, we replace $x_i^T x_l$ (i.e., $\langle x_i, x_l \rangle_{\mathbb{R}^p}$) with $k(x_i, x_l)$, where k (kernel) happens to be an inner product in some feature space, $k(x_i, x_l) = \langle \Phi(x_i), \Phi(x_l) \rangle_{\mathbb{H}_k}$.

nonlineardata.csv contains a set of 2D data (last column is label) that is not linear-separable. The following questions require you to train different classifiers and plot the boundary using the language of your choice. If you are using Python, we provide you a helper function **plotClassifier** to plot the boundary. It accepts any fitted model, as long as the model has a callable **predict** function. Feel free to draw the boundary in your own way if you like. Use 80% of the dataset for training and 20% for testing, if you use **train_test_split** from sklearn, please set the random state to 2022, so that we can reproduce your result.

Let's start with our regular (linear) soft-margin SVM, which minimizes

$$C \sum_{i=1}^{N} \max(0, 1 - y_i \boldsymbol{w}^T \boldsymbol{x}) + \frac{1}{2} \boldsymbol{w}^T \boldsymbol{w}.$$

For simplicity we omit the intercept term, feel free to add it back in if you like. You could use **SVC** from sklearn (https://scikit-learn.org/stable/modules/generated/sklearn.svm.SVC.html#sklearn.svm.SVC). Note that the sklearn SVM is not good, an alternative is to use LIBSVM (https://pypi.org/project/libsvm/), but you won't be able to use our provided plotting method in that case. Feel free to make your own choice. You can find more documentation on LIBSVM in the following resources:

- Python documentation: https://github.com/cjlin1/libsvm/tree/master/python
- Practical guide: https://www.csie.ntu.edu.tw/~cjlin/papers/guide/guide.pdf
- FAQs: https://www.csie.ntu.edu.tw/~cjlin/libsvm/faq.html

1. Train a soft-margin SVM with linear kernel and $C = 100$. Plot the decision boundary and report training and testing error rate.
2. Train a soft-margin SVM with polynomial kernel $k(x_i, x_l) = (1 + x_i^T x_l)^2$ and $C = 100$. Plot the decision boundary and report the training and testing error rates.
3. Train a soft-margin SVM with RBF kernel $k(x_i, x_l) = \exp(-\frac{\|x_i - x_l\|^2}{\sigma^2})$ and $C = 100, \sigma = 0.5$. Plot the decision boundary and report training and testing error rate.
4. If we write $\gamma = \frac{1}{\sigma^2}$, the RBF kernel can be rewritten as $\exp(-\gamma \|x_i - x_l\|^2)$. Try several values of γ (i.e, $\gamma \in [0.0001, 0.001, 0.01, 0.1, 1, 10, 100, 1000]$ and for the purpose of this problem fix $C = 100$ for all your SVM classifiers) and plot the training and test error rates vs γ. What do you notice about the error rates as γ increases? Why this is the case? Finally, in the bias variance tradeoff, does small γ correspond to high bias or high variance?

Now look at L2 regularized Logistic Regression, which minimizes

$$C \sum_{i=1}^{N} \log(1 + \exp(-y_i \boldsymbol{w}^T \boldsymbol{x})) + \frac{1}{2}\boldsymbol{w}^T \boldsymbol{w}. \tag{1}$$

If we multiply a constant $\lambda = \frac{1}{C}$, it is equivalent to minimizing

$$\sum_{i=1}^{N} \log(1 + \exp(-y_i \boldsymbol{w}^T \boldsymbol{x})) + \frac{\lambda}{2}\boldsymbol{w}^T \boldsymbol{w}, \tag{2}$$

where again we omit the intercept for simplicity (and feel free to decide whether to add an intercept). Also feel free to write your own code if you don't want to use our template classifier. We already wrote a kernelized logistic regression classifier with L2 regularization **kernelLogRegL2** for you, and we also provide a linear kernel function **kernel_linear** as an example to start. **Hint:** The classifier uses the linear kernel by default, take a look at what is constructed using the kernel function, which is helpful for next three questions.

5. Train a L2 regularized Logistic Regression classifier with linear kernel, $\lambda = 0.01$. Plot the decision boundary and report training and testing error rate.

6. Train a L2 regularized Logistic Regression classifier with polynomial kernel $k(x_i, x_l) = (1 + x_i^T x_l)^2, \lambda = 0.01$. Plot the decision boundary and report training and testing error rate. **Hint:** Write a kernel function that takes two matrices as input and returns a matrix K.

7. Train a L2 regularized Logistic Regression classifier with RBF kernel $k(x_i, x_l) = \exp(-\frac{\|x_i - x_l\|^2}{\sigma^2}), \lambda = 0.01, \sigma = 0.5$. Plot the decision boundary and report training and testing error rate.

8. If we write $\gamma = \frac{1}{\sigma^2}$, the RBF kernel can be rewritten as $\exp(-\gamma\|x_i - x_l\|^2)$.Try several values of γ (i.e, $\gamma \in [0.0001, 0.001, 0.01, 0.1, 1, 10, 100, 1000]$ and for the purpose of this problem fix $C = 100$ for all your L2 regularized logistic regression classifiers) and plot the training and test error rates vs γ.

5 Sparse Logistic Regression (Ray)

In this problem, we will attempt to solve sparse logistic regression in a very simple way – we will call a logistic regression solver for many subsets of features and pick the subset with the best training objective. Separate the **SPECTF** data into a test set and training set, where 3/4ths of the data is the training set.

1. On the training dataset, plot ROC curves for all the individual features on a single plot.

2. Create an algorithm for choosing 300 different subsets of these features. Generally it's useful to include some of the best features, and some features that are not highly correlated with them. You can start by using all feature subsets of size 1, 2, and maybe 3, perhaps being more strategic from there so that your subsets are generally likely to give good results. Describe your algorithm. Be creative so you can get good performance.

3. Train 300 logistic regression models by calling a logistic regression solver on each subset of data. Plot all 300 sparse classifiers on another ROC curve. Report the training AUC for the best model, call it f. We'll use this model f for testing.

4. Put the training ROC for f and the test ROC for f on the same figure and report the test AUC. Hopefully your model generalized! Did this technique work as well as ℓ_2-regularized logistic regression?

6 Ridge Regression and its friends (Ray)

1. Generate a dataset with 100 samples and 5,000,000 features randomly from a simulated distribution of your choice. Fit Ridge regression and Kernel ridge regression respectively with a similar strength regularization. Both use linear kernel. Compare their running time. Which one is faster? Why does that one find solution faster than the other? **Hint:** Take a look at their closed form solution.

2. Generate a dataset with 1000 samples and 500 features randomly this time. Fit Ridge regression and Lasso regression respectively with moderate strength of regularization. Try various λ to see what happened to the parameters learned by these two models. What difference did you observe? Based on what you observed, name one possible advantage of Lasso over Ridge.

3. On the dataset with 1000 samples and 500 features that you just generated, use different regularization strengths (e.g *alpha* from 0.01 to 0.1 in Sklearn Implementation). Plot the non-zero coefficients learned versus regularization strength (put each coefficient in a different color on the same plot). This is called a "regularization path." Typically on the left of the plot at value 0 (perhaps 0 is 1/regularization parameter), all the coefficients start at 0 (strong regularization), and as you weaken the regularization and move to the right of the plot, the coefficients each follow a continuous path. Create one plot like this for Lasso and another for Ridge.