Homework #6, EECS 553, F26. Due **Wed. Nov. 4**, by 11:00PM

Problems

1. [6] **Optimal soft-margin hyperplane and errors**

This problem is an accidental duplicate of HW5.4. Ignore.

Let $(\boldsymbol{w}^*, b^*, \boldsymbol{\xi}^*)$ denote the solution to the **soft-margin hyperplane quadratic program** defined as

$$\begin{aligned} \underset{\boldsymbol{w},\, b,\, \boldsymbol{\xi}}{\arg\min}\ & \frac{1}{2}\|\boldsymbol{w}\|_2^2 + \frac{C}{n}\sum_{i=1}^n \xi_i \\ \text{s.t. } & y_i\left(\boldsymbol{w}^\top \boldsymbol{x}_i + b\right) \geq 1 - \xi_i,\ \forall i \\ & \xi_i \geq 0,\ \forall i. \end{aligned}$$

(a) [3] Show that if $\boldsymbol{x}_i$ is misclassified by the optimal soft-margin hyperplane classifier, then $\xi_i^* \geq 1$. Conclude that $\frac{1}{n}\sum_i \xi_i^*$ upper bounds the training error. Hence, the OSM objective balances margin maximization with minimizing a bound on the training error.
(Note: the training error is defined as the number of $\boldsymbol{x}_i$ misclassified$/n$)

(b) [3] Show that if $\xi_i^* > 0$, then ξ_i^* is proportional to the distance from $\boldsymbol{x}_i$ to the margin hyperplane associated with class y_i (that is, the set $\{\boldsymbol{x} : \langle \boldsymbol{w}^*, \boldsymbol{x}\rangle + b^* = y_i\}$), and give the constant of proportionality.

(c) [0] (Not graded – do submit) Consider the optimization problem

$$\begin{aligned} \underset{\boldsymbol{w},b,\boldsymbol{\xi}}{\arg\min}\ & \frac{1}{2}\|\boldsymbol{w}\|_2^2 + \frac{D}{n}\sum_{i=1}^n \xi_i \\ \text{s.t. } & y_i(\boldsymbol{w}^\top \boldsymbol{x}_i + b) \geq \kappa - \xi_i && \forall i = 1, \ldots, n \\ & \xi_i \geq 0 && \forall i = 1, \ldots, n \end{aligned}$$

where $D > 0$, $\kappa > 0$. Show that the above quadratic program yields the same classifier as the optimal soft-margin classifier for a certain parameter C, and express this C in terms of D and κ.

2. [9] **Bayes risk**

Consider multiclass classification with K classes, with class prior probabilities $\{\pi_k\}$. Consider the 0-1 loss here.

(a) [3] For a given set $\{\pi_k\}$, what is the largest possible value of the risk R over all possible classifiers? Explain.

(b) [3] Over all possible sets $\{\pi_k\}$ and all possible classifiers, what is the largest possible value of the risk R? Explain.

(c) [3] What is the largest possible value of the Bayes Risk R^*? Explain.

3. [7] **Regression application: house prices**

In some HW problems, you will implement regression methods and apply them to the real-world application of predicting house prices from $d = 58$ named features like the year built, the lot area, and the number of bedrooms. You will use the Ames, Iowa Housing dataset that has 2000 train instances and 925 test instances. The target/response is house price in \$1000, *i.e.*, K\$. Being able to predict house prices from known features is clearly useful to sellers, buyers, and real estate apps.

This JULIA demo illustrates linear regression with **Tikhonov regularization** applied to that dataset. (It reads the data from this repo, so that no manual downloading is needed.) You should start by studying that demo or the Python version.

(a) [3] The demo starts by "sphering" or "normalizing" the raw features. This type of normalization is typically essential, because different features have different units (like "area" in square feet) or no units (like number of bathrooms).
Suppose we did no data normalization. Feature x_1 (Lot frontage) has units "feet" and feature x_2 (Lot area) has units "square feet." The response y_i has units K\$.
If we do linear regression using the model $f(\boldsymbol{x}) = \boldsymbol{w}^\top \boldsymbol{x} + b$, what are the units w_1, w_2 and b?
In this situation, does it make sense to compute the norms $\|\boldsymbol{X}^\top \boldsymbol{w} - \boldsymbol{y}\|_2$ and $\|\boldsymbol{w}\|_2$?

(b) [3] Specifically, the demo uses the "sphered" arrays `Xbar` and `Xbar_test` for training and testing, respectively, where $\bar{\boldsymbol{X}} = \begin{bmatrix} 1 & \cdots & 1 \\ \boldsymbol{x}_1 & \cdots & \boldsymbol{x}_n \end{bmatrix}$ and $\boldsymbol{x}_i$ denotes adjusted data where we had subtracted the sample mean of each feature and divided by the empirical standard deviation of each feature.

Let $\boldsymbol{C} \triangleq \frac{1}{n}\bar{\boldsymbol{X}}\bar{\boldsymbol{X}}^\top$ denote the sample covariance of the normalized data (augmented with the constant 1).
Study the demo code and determine analytically the first column of that sample covariance: $\boldsymbol{C}_{:,1}$. Explain.
Hint. If you are unsure, look at it numerically first.

(c) [1] You will apply each regression method to the train and test housing price data `Xbar` and `Xbar_test` in the demo linked above, producing scatter plots similar to the one in the demo that shows the predicted house prices versus the actual house price for both the training data and the test data. The legend reports the RMSE errors (in K$).
In some cases, we will also sweep a hyperparameter like λ and plot RMSE vs λ like in this plot.
What is the lowest RMSE for the test data over all values of λ considered in the demo?
If you were buying a house, would you be comfortable with the RMS error provided by the ridge regression estimator?

---

4. [27] **Regression: generative method and kernel method**

In this problem, you will implement two regression methods and apply them to the house price prediction application described in a previous problem.

(a) [10] Implement the (plug-in) generative approach to regression that assumes that the features $\mathbf{X}$ and responses Y have joint gaussian distributions. Use maximum likelihood estimates of the gaussian model parameters. The MMSE (aka conditional mean) estimator depends on the inverse of the feature covariance matrix. That matrix might be singular or poorly conditioned, so add $\lambda \boldsymbol{I}$ to it *before* inverting it to ensure numerical stability.
Your submitted file should be named `mmse_regress.jl` or `mmse_regress.py` and should contain the following function:

```
"""
    yhat = mmse_regress(X, y, Xtest; lambda::Real = 0)

Apply (plug-in) MMSE regression to test data `Xtest`
using the training data `X` and responses/labels `y`.

# In
- `X` : `d` features × `n` samples training data
- `y` : `n` training responses (labels)
- `Xtest` : `d` features × `ntest` samples test data
- `lambda` : nonnegative regularization parameter
  to stabilize the inverse of the feature covariance matrix.
# Out
- `yhat` : vector of `ntest` predictions

The MMSE estimator involves the inverse of a feature covariance matrix.
That matrix might not be invertible,
or might be very poorly conditioned,
so add `lambda * I` to it before inverting.
"""
function mmse_regress(
    X::AbstractMatrix{<:Number}, y::AbstractVector{<:Number},
    Xtest::AbstractMatrix{<:Number}, lambda::Real = 0,
)
```

Submit your solution to autograder.io
Python version:

```
def mmse_regress_sol( # PEP 8 style formatting
    X: np.ndarray,
    y: np.ndarray,
    Xtest: np.ndarray,
    lambda_val: float = 0.0
) -> np.ndarray:
```

(b) [3] Apply your MMSE regression method to the housing price data described in a previous problem. The MMSE approach does *not* require sphered data, so use the arrays `Xtrain` and `Xtest` for training and testing, respectively. Make a scatter plot similar to the one in the demo that shows the MMSE predicted house prices versus the actual house price for both the

training data and the test data. Show the RMSE errors (in K$) in the legend. Use $\lambda = 0.01$.
Submit to gradescope a screenshot of the code that you added to the demo code, and a screenshot of your scatter plot.

(c) [10] Implement the kernel ridge regression (KRR) method without offset as described in lecture, for a gaussian kernel.
Your submitted file should be named `krr.jl` or `krr.py` and should contain the following function:

```
"""
     yhat = krr(X, y, Xtest, lambda, sigma)

Apply kernel ridge regression (without offset)
to test data `Xtest` using the training data `X` and responses/labels `y`.
Use the gaussian kernel: `k(u,v) = exp(-‖u - v‖₂² / (2 σ²))`
and Tikhonov regularization parameter `lambda`.
The non-kernelized, no-offset, ridge regression cost function is
``J(w) = (1/n) ‖ X w - y ‖₂² + λ ‖ w ‖₂²``.

# In
- `X` : `d` features × `n` samples training data
- `y` : `n` training responses (labels)
- `Xtest` : `d` features × `ntest` samples test data
- `lambda` : nonnegative regularization parameter.
- `sigma` : kernel bandwidth parameter.
# Out
- `yhat` : vector of `ntest` predictions

`X` and `Xtest` could include a leading `1` in the first row.
"""
function krr(
    X::AbstractMatrix{<:Number},
    y::AbstractVector{<:Number},
    Xtest::AbstractMatrix{<:Number},
    lambda::Real = 0,
    sigma::Real = 1,
)
```

Submit your solution to autograder.io
Python version:

```
def krr(
    X: np.ndarray,
    y: np.ndarray,
    Xtest: np.ndarray,
    lam: float = 0.0,
    sigma: float = 1.0,
) -> np.ndarray:
```

(d) [3] Apply your KRR method to the housing price data. Here you should use the "sphered" arrays `Xbar` and `Xbar_test` for training and testing, respectively.
Make a scatter plot similar to the one in the demo that shows the predicted house prices versus the actual house price for both the training data and the test data. Show the RMSE errors (in K$) in the legend. Use $\lambda =$ `2.5e-6` and $\sigma = 32$.
Submit to gradescope a screenshot of the code that you added to the demo code, and a screenshot of your scatter plot.

(e) [1] Comment briefly on how the RMSE values for the train and test data changed for the KRR method compared to the original ridge regression method and the method based on the generative model in part (b).
The "M" in MSE stands for "minimum" so you might expect the MMSE method to provide better predictions (lower RMSE) than the KRR method, but here it does not. Explain.

5. [19] **Sparse linear regression (Lasso) using subgradient method**

In this problem, you will implement the **Lasso** method for linear regression and apply it to predicting house prices as described in a previous problem. The Lasso cost function is

$$J(\boldsymbol{\theta}) = J(b, \boldsymbol{w}) \triangleq \left(\frac{1}{2n} \sum_{i=1}^{n} (\boldsymbol{w}^\top \boldsymbol{x}_i + b - y_i)^2 \right) + \lambda \|\boldsymbol{w}\|_1 = \frac{1}{n} \sum_{i=1}^{n} J_n(\boldsymbol{\theta}), \quad J_i(\boldsymbol{\theta}) \triangleq \frac{1}{2} (\boldsymbol{w}^\top \boldsymbol{x}_i + b - y_i)^2 + \lambda \|\boldsymbol{w}\|_1,$$

where $\lambda \geq 0$, $\boldsymbol{w} = [w_1, \ldots, w_d]^\top$ is the vector of weights, not including the offset term b, and $\boldsymbol{x}_i$ is the ith feature vector of dimension d. (This cost function is not differentiable because of the 1-norm regularizer, so gradient descent is inapplicable.)

(a) [10] Implement the *incremental*[1] **subgradient method** (SGM) for solving the Lasso problem, where there is an *outer* loop over iteration t, and each outer iteration (from $\boldsymbol{\theta}_t$ to $\boldsymbol{\theta}_{t+1}$) involves an *inner* loop over all training samples:

$$\begin{aligned}
\tilde{\boldsymbol{\theta}}_t^{(0)} &\leftarrow \boldsymbol{\theta}_t \\
&\quad \text{for } i \in 1, \ldots, n \\
&\qquad \tilde{\boldsymbol{\theta}}_t^{(i)} \leftarrow \tilde{\boldsymbol{\theta}}_t^{(i-1)} - \alpha_t g_i(\tilde{\boldsymbol{\theta}}_t^{(i-1)}) \\
&\quad \text{end} \\
\boldsymbol{\theta}_{t+1} &\leftarrow \tilde{\boldsymbol{\theta}}_t^{(n)},
\end{aligned}$$

where g_i denotes any subgradient of J_i. For the autograder, use 0 as the subgradient of the absolute value function at 0. In other words, use `sign` as the subgradient of `abs`. A constant step size does *not* ensure SGM converence, so your function must accept a **vector** of step sizes (one for each outer iteration). SGM does not decrease J monotonically, so your function must track $J(\boldsymbol{\theta}_t)$ and finally return the $\boldsymbol{\theta}_t$ having the lowest value of J.

Incremental (or stochastic) gradient descent is used extensively for training deep networks, so here we explore it for a convex optimization problem.

Your submitted file should be named `lasso_sgm.jl` or `lasso_sgm.py` and should contain the following function:

```
"""
    θ = lasso_sgm(X, y, λ, steps; θ0, costs!)

Apply incremental subgradient method (SGM)
to the Lasso regression cost function:
``J(θ) = 1/(2n) \\sum_{i=1}^n (⟨θ, [1;x_i]⟩ - y_i)² + λ ‖ w ‖_1``
where `θ = [b; w]`.  Note that only `w` is regularized.

Number of (outer) iterations is simply `length(steps)`.

In:
- `X` : `(d+1) x n` matrix of training features, aka ``\\bar{X}``
  (top row is all `1` and typically the remaining rows are sphered)
- `y` : `n` vector of training labels (responses)
- `lambda` : regularization parameter λ ≥ 0
- `steps` vector of (positive) step sizes, one for each iteration

Option:
- `costs!`: (mutated) vector to store cost `J(θ_t)` at each iteration,
   including `J(θ_0)`, so it is of length `1 + length(steps)`
- `θ0` : initial weight vector in ℝ^{d+1}; default `zeros(size(X,1))`

Out:
- `θ` : length-`d+1` vector of regression weights
  Returns the `θ_t` with lowest `J` value
"""
function lasso_sgm(
```

[1] In practice, one would use a *stochastic* subgradient method where the loop over sample index i is randomly permuted each iteration. Here we use the *incremental* version with the deterministic order $1, \ldots, n$ to simplify the autograder.

```
    X::AbstractMatrix,
    y::AbstractVector,
    lambda::Real,
    steps::AbstractVector;
    costs!::AbstractVector = zeros(1 + length(steps)),
    θ0::AbstractVector = zeros(size(X,1)),
)
```

Submit your solution to autograder.io
Python version

```
from typing import Optional
import numpy as np
def lasso_sgm(X:np.ndarray, y:np.ndarray, lambda_param:float, steps:np.ndarray,
    costs:Optional[np.ndarray]=None, theta0:Optional[np.ndarray]=None):
```

(b) [3] Write at least one simple test case for your code. At least one test case should use $\lambda = 0$ where $\bar{\boldsymbol{X}}$ has full column rank, so there is an analytical minimizer and you should verify that your code converges to that minimizer. Submit a screenshot of your test code.

(c) [3] Ask your favorite AI tool about how to choose the (diminishing) step sizes $\{\alpha_t\}$. Report what step sizes you chose to use for the next part and why.

(d) [3] Apply your SGM code to the sphered data `Xbar` `Xbar_test` in the JULIA demo or the Python version.
Use $\lambda = 7$ and run 1000 iterations. Make a plot of predicted versus true house price like the plot in the demo where you label the RMSE (in K$) for both the training and test data.
Submit a screenshot of your added code and the plot.
Look at the output $\hat{\boldsymbol{\theta}}$ from SGM. Is it sparse?

---

6. [5] **Make your own problem**

Make a clearly stated problem that could be used on an EECS 553 exam based on the topics covered by the due date of this HW set, and provide your own solution to the problem.

Follow all the instructions from HW2 except create your problem in `hw6.tex` using the Overleaf link on EECS 553 Canvas home page.

Non-graded problems

7. [6] **Maximum likelihood estimation for Bernoulli**

Let $X_1, \ldots, X_n \overset{\text{iid}}{\sim} \text{Bernoulli}(\theta)$, a Bernoulli trial taking values 0 and 1, and "success probability" θ, where 1 corresponds to "success."

(a) [3] Determine the maximum likelihood estimator of θ as a function of observations (realizations) $x_1, \ldots, x_n$ of $X_1, \ldots, X_n$.

(b) [3] Calculate the Hessian of the log-likelihood function and verify that the estimator you found is indeed the unique maximizer of the log-likelihood function for all possible values of $x_1, \ldots, x_n$.

8. [6] **Constant classifier**

Consider a multiclass classifier f that is constant as a function of the feature vector $\boldsymbol{x}$, that is, it always outputs the same label $k \in \{1, \ldots, K\}$, where K is the number of classes. Let π_k denote the prior class probability of class k.

(a) [3] In terms of $\pi_1, \ldots, \pi_K$, what class should the classifier output to achieve the smallest probability of error (among constant classifiers)? Explain.

(b) [3] And what is the probability of error of this best constant classifier?