Machine Learning 1 **Fall 24**

Homework 1

1. In the 2020 US Presidential election, there was a significant gap between the probability of a Biden victory assigned by poll aggregators (e.g., fivethirtyeight.com) and betting markets. According to Bayesian decision theory, a rational agent should choose actions (in this case, bet size) by writing down:

   - a probability distribution $P(X)$ over the state X of the world describing what they think will happen,
   - a utility function U describing the value of each action under certain state of the world,

   and then choose the action that maximizes their expected utility (equivalently, minimizes expected risk). That is, a rational agent will choose

   $$a^* = \arg\max_a \mathbb{E}_{X\sim P}\left[U(a, X)\right].$$

   Suppose that Victor is a rational agent with d dollars of total wealth, and that he thinks $P(\text{biden wins}) = 0.85$ (the fivethirtyeight prediction), and that Betting markets give even money bets at $P(\text{biden wins}) = 0.6$.

   (a) Prove that if Victor's utility function is just the amount of money he has then he'll bet d (his entire bankroll) on Biden winning. Does this kind of linear utility of money make sense?

   (b) Now, suppose that Victor's utility from y dollars is simply $U(y) = \log(y)$. How much (of his d dollars) should he bet on Biden winning?

   (c) Suppose you had a net worth of \$500,000. Would you personally make the bet from part (b)?

2. Suppose that Y is a categorical random variable, and X are covariates that we wish to use to predict Y. A loss function L on categorical data is called a *proper scoring rule* if the predictive function f^* that minimizes the implied risk is the conditional probability of Y given X. That is, if

   $$f^* := \arg\min_f \mathbb{E}_{X,Y}\left[L\left(Y, f(X)\right)\right] \text{ is given by } f^*(x) = P(Y|X = x).$$

   Prove that the Cross Entropy loss is a proper scoring rule in the case where Y is binary.

3. As we saw in class, a requirement for stochastic gradient descent to converge to the minimum of an objective function is that the step size ρ_t decays fast enough that $\sum_t \rho_t^2 < \infty$. In actual machine learning practice, it's a common practice to just use a constant step size, violating the convergence requirement. Why might this be reasonable in a finite sample setting?

4. One common justification for using square error loss in machine learning is that the residuals of the fitted model should be (approximately) normal. This justification can be approximately tested by fitting a model to the data and then testing the residuals for normality! To do that, you are asked to fit several ML models to the Average Localization Error (ALE) dataset and run a statistical test to see if the residuals are approximately normally distributed. You should predict **ale** using the first 4 variables (you can ignore **sd_ale**). You can find different regression models in the sklearn package. Try whatever you like (for example, linear model, svm, decision tree, or neural network). To test the normality, you are allowed to use any test method. One recommendation is to draw a qqplot and run normaltest in the scipy package.

5. This next problem will teach you how to use automatic differentiation software to fit a model with a custom loss function. You can use any automatic differentiation software you like (e.g., PyTorch, JAX, etc.).

Classification: For this part, you will use the Breast Cancer Wisconsin dataset, which you can load using
sklearn.datasets.load_breast_cancer. The dataset contains 30 features and a binary label. Begin by splitting the data (80/20) into a training and test set.

As a baseline, use sklearn's implementation of logistic regression to fit a logistic regression model to the data and evaluate its accuracy on the test set, where accuracy is defined as the proportion of correctly classified observations.

(a) Now you will use automatic differentiation to fit a logistic regression model.
Begin by defining your logistic regression model:

$$f(x_i) = \sigma(w^T x_i + b)$$

where x_i are the features for observation i, w are the weights, b is the bias, and σ is the sigmoid function. Next, define a function for the binary cross-entropy loss:

$$L(y_i, f(x_i)) = -\left[y_i \log(\sigma(w^T x_i + b)) + (1 - y_i) \log(1 - \sigma(w^T x_i + b))\right]$$

Now use your preferred automatic differentiation software to fit the model to the training data by minimizing the loss with SGD. Note that you may need to clip the output of the sigmoid function to avoid numerical instability.

Evaluate the performance of your model on the test set. How does it compare to the baseline?

(b) Logistic regression uses the function $\sigma(z_i)$ to map the real-valued $z_i := w^T x_i + b$ to a probability between 0 and 1. However, this is not the only way to map a real-valued output to a probability. Probit regression uses $\Phi(z_i)$, where Φ the cumulative distribution function of the standard normal distribution. Then the model is:

$$f(x_i) = \Phi(w^T x_i + b)$$

Consequently, the loss function is:

$$L(y_i, f(x_i)) = -\left[y_i \log(\Phi(w^T x_i + b)) + (1 - y_i) \log(1 - \Phi(w^T x_i + b))\right]$$

If you are using JAX, you can use jax.scipy.stats.norm.cdf to implement this loss function. If you are using PyTorch, you can use torch.distributions.Normal.

Implement probit regression by defining the model and loss function as above. Use your preferred automatic differentiation software to fit the model to the data by minimizing the loss with SGD. As before, you may need to clip the output of the cumulative distribution function to avoid numerical instability.

Evaluate the performance of your model on the test set. How does it compare to the baseline?

Regression: For this part, you will use the average localization error (ALE) dataset from the previous question. Begin by splitting the data (80/20) into a training and test set.

As a baseline, use sklearn's implementation of linear regression to fit a linear regression model to the data and evaluate its performance on the test set using the mean absolute error.

(a) Now you will use automatic differentiation to fit a linear regression model with squared error loss. Begin by defining your linear regression model:

$$f(x_i) = w^T x_i + b$$

where x_i are the features for observation i, w are the weights, and b is the bias. Next, define a function for the mean squared error loss:

$$L(y_i, f(x_i)) = (y_i - f(x_i))^2$$

Now use your preferred automatic differentiation software to fit the model to the training data by minimizing the loss with SGD.

Evaluate the performance of your model on the test set. How does it compare to the baseline?

(b) Earlier you saw that the squared error loss can be derived from the assumption that the residuals of the fitted model should be (approximately) normal. However, the normal distribution has weak tails, so it is sensitive to outliers. An alternative is the t-distribution, which has heavier tails.

The t-distribution is parameterized by the degrees of freedom ν. When $\nu \to \infty$, the t-distribution converges to the standard normal distribution. The probability density function of the t-distribution is:

$$p(y_i - f(x_i), \nu) = \frac{\Gamma\left(\frac{\nu+1}{2}\right)}{\sqrt{\nu\pi}\Gamma\left(\frac{\nu}{2}\right)} \left(1 + \frac{(y_i - f(x_i))^2}{\nu}\right)^{-\frac{\nu+1}{2}}$$

where Γ is the gamma function.

The corresponding loss function is the negative log-likelihood:

$$L(y_i, f(x_i)) = -\log p(y_i - f(x_i), \nu)$$

For present purposes, use the t-distribution with $\nu = 5$. If you are using JAX, you can use jax.scipy.stats.t.logpdf to implement this loss function. If you are using PyTorch, you can use torch.distributions.StudentT, which has a `log_prob` method.

Use your preferred automatic differentiation software to fit the model to the training data by minimizing the loss with SGD.

Evaluate the performance of your model on the test set. How does it compare to the baseline?