# CS 229, Fall 2020
# Problem Set #1

**Due Wednesday, October 7 at 11:59 pm on Gradescope.**

**Notes:** (1) These questions require thought, but do not require long answers. Please be as concise as possible. (2) If you have a question about this homework, we encourage you to post your question on our Piazza forum, at `https://piazza.com/stanford/fall2020/cs229`. (3) If you missed the first lecture or are unfamiliar with the collaboration or honor code policy, please read the policy on the course website before starting work. (4) For the coding problems, you may not use any libraries except those defined in the provided `environment.yml` file. In particular, ML-specific libraries such as scikit-learn are not permitted. (5) To account for late days, the due date is Wednesday, October 7 at 11:59 pm. If you submit after Wednesday, October 7 at 11:59 pm, you will begin consuming your late days. If you wish to submit on time, submit before Wednesday, October 7 at 11:59 pm. Please make sure to reserve sufficient time for potentially compiling, scanning, and uploading your homework questions.

All students must submit an electronic PDF version of the written questions. We highly recommend typesetting your solutions via LaTeX, and we will award one bonus point for submissions typeset in LaTeX. Please make sure to tag your solutions properly on Gradescope. The graders reserve the right to penalize incorrectly tagged solutions by 0.5 points per question. All students must also submit a zip file of their source code to Gradescope, which should be created using the `make_zip.py` script. You should make sure to (1) restrict yourself to only using libraries included in the `environment.yml` file, and (2) make sure your code runs without errors. Your submission may be evaluated by the auto-grader using a private test set, or used for verifying the outputs reported in the writeup.

1. [**40 points] Linear Classifiers (logistic regression and GDA)**

In this problem, we cover two probabilistic linear classifiers we have covered in class so far. First, a discriminative linear classifier: logistic regression. Second, a generative linear classifier: Gaussian discriminant analysis (GDA). Both the algorithms find a linear decision boundary that separates the data into two classes, but make different assumptions. Our goal in this problem is to get a deeper understanding of the similarities and differences (and, strengths and weaknesses) of these two algorithms.

For this problem, we will consider two datasets, along with starter codes provided in the following files:

- `src/linearclass/ds1_{train,valid}.csv`
- `src/linearclass/ds2_{train,valid}.csv`
- `src/linearclass/logreg.py`
- `src/linearclass/gda.py`

Each file contains $n$ examples, one example $(x^{(i)}, y^{(i)})$ per row. In particular, the $i$-th row contains columns $x_1^{(i)} \in \mathbb{R}$, $x_2^{(i)} \in \mathbb{R}$, and $y^{(i)} \in \{0, 1\}$. In the subproblems that follow, we will investigate using logistic regression and Gaussian discriminant analysis (GDA) to perform binary classification on these two datasets.

(a) [10 points]

In lecture we saw the average empirical loss for logistic regression:

$$J(\theta) = -\frac{1}{n}\sum_{i=1}^{n}\left(y^{(i)}\log(h_\theta(x^{(i)})) + (1-y^{(i)})\log(1-h_\theta(x^{(i)}))\right),$$

where $y^{(i)} \in \{0,1\}$, $h_\theta(x) = g(\theta^T x)$ and $g(z) = 1/(1+e^{-z})$.

Find the Hessian $H$ of this function, and show that for any vector $z$, it holds true that

$$z^T H z \geq 0.$$

**Hint:** You may want to start by showing that $\sum_i \sum_j z_i x_i x_j z_j = (x^T z)^2 \geq 0$. Recall also that $g'(z) = g(z)(1-g(z))$.

**Remark:** This is one of the standard ways of showing that the matrix $H$ is positive semi-definite, written "$H \succeq 0$." This implies that $J$ is convex, and has no local minima other than the global one. If you have some other way of showing $H \succeq 0$, you're also welcome to use your method instead of the one above.

(b) [5 points] **Coding problem.** Follow the instructions in `src/linearclass/logreg.py` to train a logistic regression classifier using Newton's Method with the given step size $\alpha$ (i.e. $\theta := \theta - \alpha H^{-1}\nabla_\theta J(\theta)$). Starting with $\theta = \vec{0}$, run Newton's Method until the updates to $\theta$ are small: Specifically, train until the first iteration $k$ such that $\|\theta_k - \theta_{k-1}\|_1 < \epsilon$, where $\epsilon = 1 \times 10^{-5}$. Make sure to write your model's predicted probabilities on the validation set to the file specified in the code. You may NOT use numpy.gradient.

Include a plot of the **validation data** with $x_1$ on the horizontal axis and $x_2$ on the vertical axis. To visualize the two classes, use a different symbol for examples $x^{(i)}$ with $y^{(i)} = 0$ than for those with $y^{(i)} = 1$. On the same figure, plot the decision boundary found by logistic regression (i.e, line corresponding to $p(y|x) = 0.5$).

(c) [5 points] Recall that in GDA we model the joint distribution of $(x, y)$ by the following equations:

$$p(y) = \begin{cases} \phi & \text{if } y = 1 \\ 1-\phi & \text{if } y = 0 \end{cases}$$
$$p(x|y=0) = \frac{1}{(2\pi)^{d/2}|\Sigma|^{1/2}} \exp\left(-\frac{1}{2}(x-\mu_0)^T\Sigma^{-1}(x-\mu_0)\right)$$
$$p(x|y=1) = \frac{1}{(2\pi)^{d/2}|\Sigma|^{1/2}} \exp\left(-\frac{1}{2}(x-\mu_1)^T\Sigma^{-1}(x-\mu_1)\right),$$

where $\phi$, $\mu_0$, $\mu_1$, and $\Sigma$ are the parameters of our model.

Suppose we have already fit $\phi$, $\mu_0$, $\mu_1$, and $\Sigma$, and now want to predict $y$ given a new point $x$. To show that GDA results in a classifier that has a linear decision boundary, show the posterior distribution can be written as

$$p(y = 1 \mid x; \phi, \mu_0, \mu_1, \Sigma) = \frac{1}{1 + \exp(-(\theta^T x + \theta_0))},$$

where $\theta \in \mathbb{R}^d$ and $\theta_0 \in \mathbb{R}$ are appropriate functions of $\phi$, $\Sigma$, $\mu_0$, and $\mu_1$.

(d) [7 points] Given the dataset, we claim that the maximum likelihood estimates of the parameters are given by

$$\phi = \frac{1}{n}\sum_{i=1}^{n} 1\{y^{(i)} = 1\}$$
$$\mu_0 = \frac{\sum_{i=1}^n 1\{y^{(i)} = 0\}x^{(i)}}{\sum_{i=1}^n 1\{y^{(i)} = 0\}}$$
$$\mu_1 = \frac{\sum_{i=1}^n 1\{y^{(i)} = 1\}x^{(i)}}{\sum_{i=1}^n 1\{y^{(i)} = 1\}}$$
$$\Sigma = \frac{1}{n}\sum_{i=1}^{n}(x^{(i)} - \mu_{y^{(i)}})(x^{(i)} - \mu_{y^{(i)}})^T$$

The log-likelihood of the data is

$$\begin{aligned} \ell(\phi, \mu_0, \mu_1, \Sigma) &= \log \prod_{i=1}^{n} p(x^{(i)}, y^{(i)}; \phi, \mu_0, \mu_1, \Sigma) \\ &= \log \prod_{i=1}^{n} p(x^{(i)}|y^{(i)}; \mu_0, \mu_1, \Sigma)p(y^{(i)}; \phi). \end{aligned}$$

By maximizing $\ell$ with respect to the four parameters, prove that the maximum likelihood estimates of $\phi$, $\mu_0, \mu_1$, and $\Sigma$ are indeed as given in the formulas above. (You may assume that there is at least one positive and one negative example, so that the denominators in the definitions of $\mu_0$ and $\mu_1$ above are non-zero.)

(e) [5 points] **Coding problem.** In `src/linearclass/gda.py`, fill in the code to calculate $\phi$, $\mu_0$, $\mu_1$, and $\Sigma$, use these parameters to derive $\theta$, and use the resulting GDA model to make predictions on the validation set. Make sure to write your model's predictions on the validation set to the file specified in the code.

Include a plot of the **validation data** with $x_1$ on the horizontal axis and $x_2$ on the vertical axis. To visualize the two classes, use a different symbol for examples $x^{(i)}$ with $y^{(i)} = 0$ than for those with $y^{(i)} = 1$. On the same figure, plot the decision boundary found by GDA (i.e, line corresponding to $p(y|x) = 0.5$).

(f) [2 points] For Dataset 1, compare the validation set plots obtained in part (b) and part (e) from logistic regression and GDA respectively, and briefly comment on your observation in a couple of lines.

(g) [5 points] Repeat the steps in part (b) and part (e) for Dataset 2. Create similar plots on the **validation set** of Dataset 2 and include those plots in your writeup.

On which dataset does GDA seem to perform worse than logistic regression? Why might this be the case?

(h) [**1 points**] Based on your plots, do the GDA modeling assumptions look like they are satisfied on Datasets 1 and 2? If not, can you find a transformation of the $x^{(i)}$'s such that the GDA modeling assumptions look like they are satisfied for that dataset? What might this transformation be?

2. **[25 points] Poisson Regression**

In this question we will construct another kind of a commonly used GLM, which is called Poisson Regression. In a GLM, the choice of the exponential family distribution is based on the kind of problem at hand. If we are solving a classification problem, then we use an exponential family distribution with support over discrete classes (such as Bernoulli, or Categorical). Simiarly, if the output is real valued, we can use Gaussian or Laplace (both are in the exponential family). Sometimes the desired output is to predict counts, for e.g., predicting the number of emails expected in a day, or the number of customers expected to enter a store in the next hour, etc. based on input features (also called covariates). You may recall that a probability distribution with support over integers (i.e. counts) is the Poisson distribution, and it also happens to be in the exponential family.

In the following sub-problems, we will start by showing that the Poisson distribution is in the exponential family, derive the functional form of the hypothesis, derive the update rules for training models, and finally using the provided dataset train a real model and make predictions on the test set.

(a) [5 points] Consider the Poisson distribution parameterized by $\lambda$:

$$p(y; \lambda) = \frac{e^{-\lambda}\lambda^y}{y!}.$$

(Here $y$ has positive integer values and $y!$ is the factorial of $y$. ) Show that the Poisson distribution is in the exponential family, and clearly state the values for $b(y)$, $\eta$, $T(y)$, and $a(\eta)$.

(b) [3 points] Consider performing regression using a GLM model with a Poisson response variable. What is the canonical response function for the family? (You may use the fact that a Poisson random variable with parameter $\lambda$ has mean $\lambda$.)

(c) [7 points] For a training set $\{(x^{(i)}, y^{(i)}); i = 1, \ldots, n\}$, let the log-likelihood of an example be $\log p(y^{(i)}|x^{(i)}; \theta)$. By taking the derivative of the log-likelihood with respect to $\theta_j$, derive the stochastic gradient ascent update rule for learning using a GLM model with Poisson responses $y$ and the canonical response function.

(d) [10 points] **Coding problem**

Consider a website that wants to predict its daily traffic. The website owners have collected a dataset of past traffic to their website, along with some features which they think are useful in predicting the number of visitors per day. The dataset is split into train/valid sets and the starter code is provided in the following files:

- `src/poisson/{train,valid}.csv`
- `src/poisson/poisson.py`

We will apply Poisson regression to model the number of visitors per day. Note that applying Poisson regression in particular assumes that the data follows a Poisson distribution whose natural parameter is a linear combination of the input features (*i.e.,* $\eta = \theta^T x$). In `src/poisson/poisson.py`, implement Poisson regression for this dataset and use *full batch gradient ascent* to maximize the log-likelihood of $\theta$. For the stopping criterion, check if the change in parameters has a norm smaller than a small value such as $10^{-5}$.

Using the trained model, predict the expected counts for the **validation set**, and create a scatter plot between the true counts vs predicted counts (on the validation set). In the scatter plot, let x-axis be the true count and y-axis be the corresponding predicted expected count. Note that the true counts are integers while the expected counts are generally real values.

3. **[15 points] Convexity of Generalized Linear Models**

In this question we will explore and show some nice properties of Generalized Linear Models, specifically those related to its use of Exponential Family distributions to model the output.

Most commonly, GLMs are trained by using the negative log-likelihood (NLL) as the loss function. This is mathematically equivalent to Maximum Likelihood Estimation (*i.e.,* maximizing the log-likelihood is equivalent to minimizing the negative log-likelihood). In this problem, our goal is to show that the NLL loss of a GLM is a *convex* function w.r.t the model parameters. As a reminder, this is convenient because a convex function is one for which any local minimum is also a global minimum, and there is extensive research on how to optimize various types of convex functions efficiently with various algorithms such as gradient descent or stochastic gradient descent.

To recap, an exponential family distribution is one whose probability density can be represented

$$p(y;\eta) = b(y)\exp(\eta^T T(y) - a(\eta)),$$

where $\eta$ is the *natural parameter* of the distribution. Moreover, in a Generalized Linear Model, $\eta$ is modeled as $\theta^T x$, where $x \in \mathbb{R}^d$ are the input features of the example, and $\theta \in \mathbb{R}^d$ are learnable parameters. In order to show that the NLL loss is convex for GLMs, we break down the process into sub-parts, and approach them one at a time. Our approach is to show that the second derivative (*i.e.,* Hessian) of the loss w.r.t the model parameters is Positive Semi-Definite (PSD) at all values of the model parameters. We will also show some nice properties of Exponential Family distributions as intermediate steps.

For the sake of convenience we restrict ourselves to the case where $\eta$ is a scalar. Assume $p(Y|X;\theta) \sim \text{ExponentialFamily}(\eta)$, where $\eta \in \mathbb{R}$ is a scalar, and $T(y) = y$. Note that $Y$ stands for the random variable and $y$ is a particular value. This makes the exponential family representation take the form

$$p(y;\eta) = b(y)\exp(\eta y - a(\eta)).$$

(a) [5 points] Derive an expression for the mean of the distribution. Show that $\mathbb{E}[Y;\eta] = \frac{\partial}{\partial\eta}a(\eta)$ (note that $\mathbb{E}[Y;\eta] = \mathbb{E}[Y|X;\theta]$ since $\eta = \theta^T x$). In other words, show that the mean of an exponential family distribution is the first derivative of the log-partition function with respect to the natural parameter.

**Hint:** Start with observing that $\frac{\partial}{\partial\eta}\int p(y;\eta)dy = \int \frac{\partial}{\partial\eta}p(y;\eta)dy$.

(b) [5 points] Next, derive an expression for the variance of the distribution. In particular, show that $\text{Var}(Y;\eta) = \frac{\partial^2}{\partial\eta^2}a(\eta)$ (again, note that $\text{Var}(Y;\eta) = \text{Var}(Y|X;\theta)$). In other words, show that the variance of an exponential family distribution is the second derivative of the log-partition function w.r.t. the natural parameter.

**Hint:** Building upon the result in the previous sub-problem can simplify the derivation.

(c) [5 points] Finally, write out the loss function $\ell(\theta)$, the NLL of the distribution, for a single example as a function of $\theta$. Then, calculate the Hessian of the loss w.r.t $\theta$, and show that it is always PSD. This concludes the proof that NLL loss of GLM is convex.

**Hint 1:** Use the chain rule of calculus along with the results of the previous parts to simplify your derivations.

**Hint 2:** Recall that variance of any probability distribution is non-negative.

**Remark:** The main takeaways from this problem are:

- Any GLM model is convex in its model parameters.

- The exponential family of probability distributions are mathematically nice. Whereas calculating mean and variance of distributions in general involves integrals (hard), surprisingly we can calculate them using derivatives (easy) for exponential family.

4. **[25 points] Linear regression: linear in what?**

In the first two lectures, you have seen how to fit a linear function of the data for the regression problem. In this question, we will see how linear regression can be used to fit non-linear functions of the data using feature maps. We will also explore some of its limitations, for which future lectures will discuss fixes.

(a) [5 points] **Learning degree-3 polynomials of the input**

Suppose we have a dataset $\{(x^{(i)}, y^{(i)})\}_{i=1}^n$ where $x^{(i)}, y^{(i)} \in \mathbb{R}$. We would like to fit a third degree polynomial $h_\theta(x) = \theta_3 x^3 + \theta_2 x^2 + \theta_1 x^1 + \theta_0$ to the dataset. The key observation here is that the function $h_\theta(x)$ is still linear in the unknown parameter $\theta$, even though it's not linear in the input $x$. This allows us to convert the problem into a linear regression problem as follows.

Let $\phi : \mathbb{R} \to \mathbb{R}^4$ be a function that transforms the original input $x$ to a 4-dimensional vector defined as

$$\phi(x) = \begin{bmatrix} 1 \\ x \\ x^2 \\ x^3 \end{bmatrix} \in \mathbb{R}^4 \qquad (1)$$

Let $\hat{x} \in \mathbb{R}^4$ be a shorthand for $\phi(x)$, and let $\hat{x}^{(i)} \triangleq \phi(x^{(i)})$ be the transformed input in the training dataset. We construct a new dataset $\{(\phi(x^{(i)}), y^{(i)})\}_{i=1}^n = \{(\hat{x}^{(i)}, y^{(i)})\}_{i=1}^n$ by replacing the original inputs $x^{(i)}$'s by $\hat{x}^{(i)}$'s. We see that fitting $h_\theta(x) = \theta_3 x^3 + \theta_2 x^2 + \theta_1 x^1 + \theta_0$ to the old dataset is equivalent to fitting a linear function $h_\theta(\hat{x}) = \theta_3 \hat{x}_3 + \theta_2 \hat{x}_2 + \theta_1 \hat{x}_1 + \theta_0$ to the new dataset because

$$h_\theta(x) = \theta_3 x^3 + \theta_2 x^2 + \theta_1 x^1 + \theta_0 = \theta_3 \phi(x)_3 + \theta_2 \phi(x)_2 + \theta_1 \phi(x)_1 + \theta_0 = \theta^T \hat{x} \qquad (2)$$

In other words, we can use linear regression on the new dataset to find parameters $\theta_0, \dots, \theta_3$.

Please write down 1) the objective function $J(\theta)$ of the linear regression problem on the new dataset $\{(\hat{x}^{(i)}, y^{(i)})\}_{i=1}^n$ and 2) the update rule of the batch gradient descent algorithm for linear regression on the dataset $\{(\hat{x}^{(i)}, y^{(i)})\}_{i=1}^n$.

*Terminology:* In machine learning, $\phi$ is often called the feature map which maps the original input $x$ to a new set of variables. To distinguish between these two sets of variables, we will call $x$ the input **attributes**, and call $\phi(x)$ the **features**. (Unfortunately, different authors use different terms to describe these two things. In this course, we will do our best to follow the above convention consistently.)

(b) [5 points] **Coding question: degree-3 polynomial regression**

For this sub-question question, we will use the dataset provided in the following files:

`src/featuremaps/{train,valid,test}.csv`

Each file contains two columns: $x$ and $y$. In the terminology described in the introduction, $x$ is the attribute (in this case one dimensional) and $y$ is the output label.

Using the formulation of the previous sub-question, implement linear regression with **normal equations** using the feature map of degree-3 polynomials. Use the starter code provided in `src/featuremaps/featuremap.py` to implement the algorithm.

Create a scatter plot of the training data, and plot the learnt hypothesis as a smooth curve over it. Submit the plot in the writeup as the solution for this problem.

*Remark:* Suppose $\widehat{X}$ is the design matrix of the transformed dataset. You may sometimes encounter a non-invertible matrix $\widehat{X}^T\widehat{X}$. For a numerically stable code implementation, always use `np.linalg.solve` to obtain the parameters directly, rather than explicitly calculating the inverse and then multiplying it with $\widehat{X}^T y$.

(c) [5 points] **Coding question: degree-$k$ polynomial regression**

Now we extend the idea above to degree-$k$ polynomials by considering $\phi : \mathbb{R} \to \mathbb{R}^{k+1}$ to be

$$\phi(x) = \begin{bmatrix} 1 \\ x \\ x^2 \\ \vdots \\ x^k \end{bmatrix} \in \mathbb{R}^{k+1} \tag{3}$$

Follow the same procedure as the previous sub-question, and implement the algorithm with $k = 3, 5, 10, 20$. Create a similar plot as in the previous sub-question, and include the hypothesis curves for each value of $k$ with a different color. Include a legend in the plot to indicate which color is for which value of $k$.

Submit the plot in the writeup as the solution for this sub-problem. Observe how the fitting of the training dataset changes as $k$ increases. Briefly comment on your observations in the plot.

(d) [5 points] **Coding question: other feature maps**

You may have observed that it requires a relatively high degree $k$ to fit the given training data, and this is because the dataset cannot be explained (i.e., approximated) very well by low-degree polynomials. By visualizing the data, you may have realized that $y$ can be approximated well by a sine wave. In fact, we generated the data by sampling from $y = \sin(x) + \xi$, where $\xi$ is noise with Gaussian distribution. Please update the feature map $\phi$ to include a sine transformation as follows:

$$\phi(x) = \begin{bmatrix} 1 \\ x \\ x^2 \\ \vdots \\ x^k \\ \sin(x) \end{bmatrix} \in \mathbb{R}^{k+2} \tag{4}$$

With the updated feature map, train different models for values of $k = 0, 1, 2, 3, 5, 10, 20$, and plot the resulting hypothesis curves over the data as before.

Submit the plot as a solution to this sub-problem. Compare the fitted models with the previous sub-question, and briefly comment about noticable differences in the fit with this feature map.

(e) [5 points] **Overfitting with expressive models and small data**

For the rest of the problem, we will consider a small dataset (a random subset of the dataset you have been using so far) with much fewer examples, provided in the following file:

`src/featuremaps/small.csv`

We will be exploring what happens when the number of features start becoming bigger than the number of examples in the training set. Run your algorithm on this small dataset using the following feature map

$$\phi(x) = \begin{bmatrix} 1 \\ x \\ x^2 \\ \vdots \\ x^k \end{bmatrix} \in \mathbb{R}^{k+1} \tag{5}$$

with $k = 1, 2, 5, 10, 20$.

Create a plot of the various hypothesis curves (just like previous sub-questions). Observe how the fitting of the training dataset changes as $k$ increases. Submit the plot in the writeup and comment on what you observe.

**Remark:** The phenomenon you observe where the models start to fit the training dataset very well, but suddenly "goes wild" is due to what is called *overfitting.* The intuition to have for now is that, when the amount of data you have is small relative to the expressive capacity of the family of possible models (that is, the hypothesis class, which, in this case, is the family of all degree $k$ polynomials), it results in overfitting.

Loosely speaking, the set of hypothesis function is "very flexible" and can be easily forced to pass through all your data points especially in unnatural ways. In other words, the model explains the noises in the training dataset, which shouldn't be explained in the first place. This hurts the predictive power of the model on test examples. We will describe overfitting in more detail in future lectures when we cover learning theory and bias-variance tradeoffs.

5. **[25 points] Learning Imbalanced dataset**

In this problem, we study how to learn a classifier from an imbalanced dataset, where the marginal distribution of the classes/labels are imbalanced. Imbalanced datasets are ubiquitous in real-world applications. For example, in the spam detection problem, the training dataset usually has only a small fraction of spam emails (positive examples) but a large fraction of ordinary emails (negative examples). For simplicity, we consider binary classification problem where the labels are in $\{0, 1\}$ and the number of positive examples is much smaller than the number of negative examples.

(a) [10 points] **Evaluation metrics**

In this sub-question, we discuss the common evaluation metrics for imbalanced dataset. Suppose we have a validation dataset and for some $\rho \in (0, 1)$, we assume that $\rho$ fraction of the validation examples are positive examples (with label 1), and $1 - \rho$ fraction of them are negative examples (with label 0).

Define the accuracy as

$$A \triangleq \frac{\#\text{examples that are predicted correctly by the classifier}}{\#\text{examples}}$$

(i) (3 point) Show that for any dataset with $\rho$ fraction of positive examples and $1 - \rho$ fraction of negative examples, there exists a (trivial) classifier with accuracy at least $1 - \rho$.

The statement above suggests that the accuracy is not an ideal evaluation metric when $\rho$ is close to 0. E.g., imagine that for spam detection $\rho$ can be smaller than 1%. The statement suggests there is a trivial classifier that gets more than 99% accuracy. This could be misleading — 99% seems to be almost perfect, but actually you don't need to learn anything from the dataset to achieve it.

Therefore, for imbalanced dataset, we need more informative evaluation metrics. We define the number of true positive, true negative, false positive, false negative examples as

$$\begin{aligned}
TP &\triangleq \#\text{ positive examples with a correct (positive) prediction} \\
TN &\triangleq \#\text{ negative examples with a correct (negative) prediction} \\
FP &\triangleq \#\text{ negative examples with a incorrect (positive) prediction} \\
FN &\triangleq \#\text{ positive examples with a incorrect (negative) prediction}
\end{aligned}$$

Define the accuracy of positive examples as

$$A_1 \triangleq \frac{TP}{TP + FN} = \frac{\#\text{ positive examples with a correct (positive) prediction}}{\#\text{ positive examples}}$$

Define the accuracy of negative examples as

$$A_0 \triangleq \frac{TN}{TN + FP} = \frac{\#\text{ negative examples with a correct (negative) prediction}}{\#\text{ negative examples}}$$

We define the balanced accuracy as

$$\overline{A} \triangleq \frac{1}{2}(A_0 + A_1) \tag{6}$$

With these notations, we can verify that the accuracy is equal to $A = \frac{TP+TN}{TP+TN+FP+FN}$.

(ii) (4 point) Show that

$$\rho = \frac{TP + FN}{TP + TN + FP + FN}$$

and

$$A = \rho \cdot A_1 + (1 - \rho)A_0 \tag{7}$$

Comparing equation (6) and (7), we can see that the accuracy and balanced accuracy are both linear combination of $A_0$ and $A_1$ but with different weighting.

(iii) (3 point) Show that the trivial classifier you constructed for part (i) has balanced accuracy 50%.

Partly because of (iii), the balanced accuracy $\overline{A}$ is often a preferable evaluation metric than the accuracy $A$. Sometimes people also report the accuracies for the two classes ($A_0$ and $A_1$) to demonstrate the performance for each class.

(b) [5 points] **Coding problem: vanilla logistic regression**

First, we use the vanilla logistic regression to learn an imbalanced dataset. For the rest of the question, we will use the dataset and starter code provided in the following files:

- `src/imbalanced/{train,validation}.csv`
- `src/imbalanced/imbalanced.py`

Each file contains $n$ examples, one example $(x^{(i)}, y^{(i)})$ per row. $x$ is two-dimensional, i.e., the $i$-th row contains columns $x_1^{(i)} \in \mathbb{R}$, $x_2^{(i)} \in \mathbb{R}$, and $y^{(i)} \in \{0, 1\}$. Let $\mathcal{D} = \{(x^{(i)}, y^{(i)})\}_{i=1}^n$ be our training dataset. $\mathcal{D}$ has $\rho n$ examples with label 1 and $(1 - \rho)n$ with label 0. In the dataset we constructed, $\rho = 1/11$.

You will train a linear classifier $h_\theta(x)$ with average empirical loss for logistical regression, where $h_\theta(x) = g(\theta^T x), g(z) = 1/(1 + e^{-z})$, similar to Problem 1 Part a:

$$J(\theta) = -\frac{1}{n}\sum_{i=1}^{n}\left(y^{(i)}\log(h_\theta(x^{(i)})) + (1 - y^{(i)})\log(1 - h_\theta(x^{(i)}))\right),$$

You are encouraged to use your code for problem 1, but you are also allowed to use any standard logistic regression library or package to optimize the objective above. After obtaining the classifier, compute the classifier's accuracy ($A$), balanced accuracy ($\overline{A}$), accuracies for the two classes ($A_0, A_1$) on the validation dataset, and report them in the writeup. You are expected to observe that the minority class (positive class) has significantly lower accuracy than the majority class.

Create a plot to visualize the validation set with $x_1$ on the horizontal axis and $x_2$ on the vertical axis. Use different symbols for examples $x^{(i)}$ with true label $y^{(i)} = 1$ than those with $y^{(i)} = 0$. On the same figure, plot the decision boundary obtained by your model (i.e, line corresponding to model's predicted probability = 0.5) in red color. Include this plot in your writeup. You do not need to report your model's predicted probabilities.

*Remark:* The logistic regression loss function is the average empirical loss across all of the training examples averaged over the entire dataset, without regard to which class a data point belongs to. Hence, logistic regression is good for overall accuracy. The focus on overall accuracy also means that logistic regression may make more mistakes on rare classes if there are fewer examples from that class that contribute to the overall loss.

(c) [5 points] **Re-sampling/Re-weighting Logistic Regression**

The relatively low accuracy for the minority class and the resulting low balanced accuracy are undesirable for many applications. Various methods have been proposed to improve the accuracy for the minority class, and learning imbalanced datasets is an active open research direction. Here we introduce a simple and classical re-sampling/re-weighting technique that helps improve the balanced accuracy in most of the cases.

We observe that the logistic regression algorithm works well for the accuracy but not for the balanced accuracy. Let $\mathcal{D} = \{(x^{(i)}, y^{(i)})\}_{i=1}^n$ denote the existing training dataset. We will create a new dataset $\mathcal{D}'$ such that the accuracy on $\mathcal{D}'$ is the same as the balanced accuracy on $\mathcal{D}$.

Assume $\rho < 1/2$ without loss of generality, and let $\kappa = \frac{\rho}{1-\rho}$. This means the number of positive examples is $\kappa$ times the number of negative examples in $\mathcal{D}$. Assume for convenience $1/\kappa$ is an integer.[1] Let $\mathcal{D}'$ be the dataset that contain each negative example once and $1/\kappa$ repetitions of each positive example in $\mathcal{D}$. Then we will apply the logistic regression on the new dataset $\mathcal{D}'$.

**Prove that** for any classifier, the balanced accuracy on $\mathcal{D}$ is equal to the accuracy on $\mathcal{D}'$. Moreover, **show that** the average empirical loss for logistical regression on the dataset $\mathcal{D}'$ is equal to

$$J(\theta) = -\frac{1+\kappa}{2n}\sum_{i=1}^{n} w^{(i)} \left( y^{(i)} \log(h_\theta(x^{(i)})) + (1 - y^{(i)}) \log(1 - h_\theta(x^{(i)})) \right),$$

where $w^{(i)} = 1$ if $y^{(i)} = 0$ and $w^{(i)} = 1/\kappa$ if $y^{(i)} = 1$.

Observe effectively we are re-weighting the loss function for each example by some constant that depends on the frequency of the class it belongs to.

(d) [5 points] **Coding problem: re-weighting minority class**

In `src/imbalanced/imbalanced.py`, implement the logistic regression algorithm on $\mathcal{D}'$. Compute and report the classifier's accuracy ($A$), balanced accuracy ($\overline{A}$), accuracies for the two classes ($A_0, A_1$) on the validation dataset. You are expected to see that the accuracy of minority class (class 1) improved significantly whereas that of the majority class dropped compared to vanilla logistic regression. However, the balanced accuracy is significantly greater than that of the vanilla logistic regression.

Create a plot to visualize the validation set with $x_1$ on the horizontal axis and $x_2$ on the vertical axis. Use different symbols for examples $x^{(i)}$ with true label $y^{(i)} = 1$ than those with $y^{(i)} = 0$. On the same figure, plot the decision boundary obtained by your model (i.e, line corresponding to model's predicted probability = 0.5) in red color. Include this plot in your writeup. You do not need to report your model's predicted probabilities.

[1] otherwise we can round up to the nearest integer with introducing a slight approximation.

6. **[30 points] Convergence and Learning Rate for Gradient Descent**

When running an optimization algorithm, finding a good learning rate may require some tuning. If the learning rate is too high, the gradient descent (GD) algorithm might not converge. If the learning rate is too low, GD may take too long to converge. In this question, we will investigate some theoretical guarantees for the convergence of GD on convex objective functions, and their implications on choosing the learning rate.

**Recap and notation.** Suppose we have a convex objective function $J(\theta)$. At each iteration, GD updates the iterate $\theta^{[t]}$ as follows:

$$\theta^{[t]} = \theta^{[t-1]} - \alpha \nabla J(\theta^{[t-1]})$$

where $\alpha$ is the learning rate and $\nabla J(\theta^{[t-1]})$ is the gradient of $J(\theta)$ evaluated at $\theta^{[t-1]}$.

(a) **[8 points] Quadratic Objective, Scalar Variable**

In this part, consider the simple objective function with one-dimensional variable $\theta$:

$$J(\theta) = \frac{1}{2}\beta\theta^2$$

where $\theta \in \mathbb{R}$ is our parameter and $\beta > 0$ is a positive, constant scalar. We initialize GD at some initialization $\theta^{[0]} \neq 0$ and run GD with learning rate $\alpha$.

i. **Derive** the range of $\alpha$ such that the iterate of GD converges in the sense that there exists a scalar $\theta^\dagger$ such that $\lim_{t\to\infty} |\theta^{[t]} - \theta^\dagger| = 0$. Provide the value of $\theta^\dagger$ when GD converges and show that it's equal to the global minimum $\theta^* = \arg\min_\theta J(\theta)$. The range of $\alpha$ can potentially depend on $\beta$.

ii. Given a desired accuracy $\epsilon > 0$, for all the $\alpha$ in the range that you derived, **derive** the minimum number of iterations $T$ required to reach a point $\theta^{[T]}$ such that $|\theta^{[T]} - \theta^*| \leq \epsilon$. $T$ can potentially depend on $\epsilon$, $\theta^{[0]}$, $\alpha$, and $\beta$. Investigate the behavior of $T$ and whether $T$ is increasing or decreasing for different values of $\alpha$. Briefly discuss why choosing an inappropriate $\alpha$ can cause the algorithm to take longer to converge.

*Hint:* Express $\theta^{[t]}$ in terms of $\theta^{[0]}$, $\alpha$, $t$, and $\beta$.

*Hint:* $\lim_{t\to\infty} c^t = 0, \quad \forall c \in \mathbb{R}\ s.t.\ |c| < 1$

(b) **[4 points] Quadratic Objective, d-dimensional Variable**

In this part of the question, consider the objective function with $d$-dimensional variable $\theta$:

$$J(\theta) = \frac{1}{2}\sum_{i=1}^{d} \beta_i \theta_i^2$$

where $\theta_i$'s are our parameters and $\beta_i \in \mathbb{R}\ s.t.\ \beta_i > 0$ are positive, constant scalars. Assume that we start from some $\theta^{[0]}$ where $\theta_i^{[0]} \neq 0$ for all $i$ and run GD with learning rate $\alpha$. Derive the range of learning rate $\alpha$ such that GD converges in the sense that there exists a vector $\theta^\dagger \in \mathbb{R}^d$ such that $\lim_{t\to\infty} \|\theta^{[t]} - \theta^\dagger\|_2 = 0$. Provide the value of $\theta^\dagger$ when GD converges. The range of $\alpha$ can potentially depend on $\beta_i$ where $i \in \{1, \dots, d\}$.

(c) **[4 points] Coding Question: Quadratic Multivariate Objective**

Let the objective function be

$$J(\theta) = \theta^\top A\theta$$

where $A \in \mathbb{R}^{2\times 2}$ is a $2 \times 2$ real, positive semi-definite matrix. Do the following exercises:

i. Implement the `update_theta` and `gradient_descend` function in `src/gd_convergence/experiment.py`. You can stop the GD algorithm when either of the following condition is satisfied:

   A. $|J(\theta^{[t]}) - J(\theta^{[t-1]})| < \epsilon$ where $\epsilon = 10^{-50}$ is given as the function parameter. This is when we assume the algorithm converged.

   B. $J(\theta^{[t]}) > 10^{20}$. This is to prevent an infinite loop when the algorithm does not converge.

   To test your implementation, run `python src/gd_convergence/experiment.py`, which checks that your $\theta$ (approximately) converges to the optimal value.

   Note that we have provided you a matrix $A = \begin{bmatrix} 1 & 0 \\ 0 & 2 \end{bmatrix}$ and $\theta^{[0]} = [-1, 0.5]$ at the beginning of the file for you to experiment with. Therefore, the objective function is a special case of the objective function in part (b) with dimension $d = 2$. Check if your theoretical derivation of the feasible range of the learning rate indeed matches empirical observations.

ii. Now, suppose we rotate the matrix $A$, what do we observe? Plot the trajectories of the GD algorithm using the following learning rates: 0.05, 0.1, 0.2, 0.3, 0.4, 0.45, 0.5, and 1. We have provided the rotation and plotting function for you. You need to simply run `python src/gd_convergence/plotting.py lr1 lr2 lr3 ...` with `lr*` replaced with desired learning rates. Include the output file `trajectories.png` and `trajectories_rotated.png`, as well as a brief discussion on the printed output of `plotting.py` in the write-up.

Remark: Setting the learning rate too high will cause the objective function to not converge. The convergence properties of these learning rates are also rotational invariant. We will show this formally in the next part of the problem.

(d) **[12 points] Convergence of Gradient Descent for a General Convex Objective**

Now let's consider any convex, twice continuously differentiable objective function $J(\theta)$ that is bounded from below. Suppose that the largest eigenvalue of the Hessian matrix $H = \nabla^2 J(\theta)$ is less than or equal to $\beta_{\max}$ for all points $\theta$.

Show that when running gradient descent, choosing a step size $0 < \alpha < \frac{1}{\beta_{\max}}$ guarantees that $J(\theta^{[t]})$ will converge to some finite value as $t$ approaches infinity.[2] Specifically, you should derive an inequality for $J(\theta^{[t]})$ in terms of $J(\theta^{[t-1]})$, $\nabla J(\theta^{[t-1]})$, $\alpha$, and $\beta_{\max}$, and show that the objective function is strictly decreasing during each iteration, i.e. $J(\theta^{[t]}) < J(\theta^{[t-1]})$.

*Hint:* You can assume that, by Taylor's theorem, the following statement is true:

$$J(y) = J(x) + \nabla J(x)^\top (y - x) + \frac{1}{2}(y - x)^\top \nabla^2 J(x + c(y - x))(y - x) \text{ for some } 0 \le c \le 1$$

*Hint:* The Hessian of a convex function is symmetric and positive semi-definite at any point $\theta$, which means all of its eigenvalues are real and non-negative.

*Hint:* The Hessian matrix is symmetric. Consider using the spectral theorem introduced in problem 3 in homework 0.

[2] This definition of convergence is different from the definition in previous questions where we explicitly prove that the parameter $\theta$ converges to an optimal parameter $\theta^*$. In this case, we do not know the optimal value $\theta^*$, and it would be difficult to prove convergence using the previous definition.

***Optional (no credit):*** Using the gradient descent inequality that you derived, show that the GD algorithm converges in the sense that $\lim_{t\to\infty} ||\nabla J(\theta^{[t]})||_2^2 = 0$.[3]

*Remark:* This question suggests that a smaller learning rate should be applied if we believe the curvature of the objective function is big.

(e) **[2 points] Learning Rate for Linear Regression**

Consider using GD on the LMS objective introduced in lecture:

$$J(\theta) = \frac{1}{2}||X\theta - y||_2^2$$

where $X \in \mathbb{R}^{n\times d}$ is the design matrix of our data, $\theta \in \mathbb{R}^d$ is our parameter, and $\vec{y} \in \mathbb{R}^n$ is the response variable.

Let $\beta_{\max}$ be the largest eigenvalue of $X^\top X$. Prove that for all $\alpha \in (0, \frac{1}{\beta_{\max}})$, GD with learning rate $\alpha$ satisfies that $J(\theta^{[t]})$ converges as $t \to \infty$.

*Hint:* You can invoke the statements in the part (d) (including the optional question in part (d)) to solve this part (even if you were not able to prove part (d).)

**Remark:** The conclusion here suggests that for linear regression, roughly speaking, you should use smaller learning rates if the scale of your data $X$ is big or if the data points are correlated with each other, both of which will enable a large top eigenvalue of $XX^\top$.

**Remark:** Even though in many cases the LMS objective can be solved exactly by solving the normal equation as shown in lecture, it is still useful to be able to guarantee convergence when using the Gradient Descent algorithm in situations where it is difficult to solve for the optimal $\theta^*$ directly (such as having large amount of data making inverting $X$ very expensive).

[3]Note that $\lim_{t\to\infty} ||\nabla J(\theta^{[t]})||_2^2 = 0$ does not necessarily mean that there exists a vector $\hat{\theta}$ such that $\theta^{[t]} \to \hat{\theta}$. (Even an 1-dimensional counterexample exists.) However, for convex functions, when $\theta^{[t]}$ stays in a bounded set, $\lim_{t\to\infty} ||\nabla J(\theta^{[t]})||_2^2 = 0$ does imply that $J(\theta^{[t]}) \to \inf_\theta J(\theta)$ as $t \to \infty$. Therefore, $\lim_{t\to\infty} ||\nabla J(\theta^{[t]})||_2^2 = 0$ is typically considered a reasonable definition for an optimization algorithm's convergence.