1 Honor Code

- Collaborator: None
- *I certify that all solutions are entirely in my own words and that I have not looked at another student's solutions. I have given credit to all external sources I consulted.*

Signature: _____

2 Logistic Regression with Newton's Method

1. Derive $\frac{ds(\gamma)}{d\gamma}$

$$\begin{aligned}\frac{ds(\gamma)}{d\gamma} &= \frac{d}{dx}(\frac{1}{x})\frac{d}{d\gamma}(1+e^{-\gamma}) \quad \text{where} \quad x = 1+e^{-\gamma} \\ &= -\frac{1}{x^2}(-e^{-\gamma}) \\ &= \frac{e^{-\gamma}}{(1+e^{-\gamma})^2} \\ &= s(\gamma)(1-s(\gamma))\end{aligned}$$

Derive $\nabla_w J(w)$

$$\begin{aligned}\nabla_w J(w) &= -\sum_{i=1}^{n}(\frac{y_i}{s_i}\frac{ds_i}{dw} - \frac{1-y_i}{1-s_i}\frac{ds_i}{dw}) \\ &= -\sum_{i=1}^{n}(\frac{y_i}{s_i} - \frac{1-y_i}{1-s_i})\frac{ds_i}{dw} \\ &= -\sum_{i=1}^{n}\frac{y_i(1-s_i)-s_i(1-y_i)}{s_i(1-s_i))}\frac{ds_i}{d\gamma_i}\frac{d\gamma_i}{dw} \\ &= -\sum_{i=1}^{n}\frac{y_i(1-s_i)-s_i(1-y_i)}{s_i(1-s_i)}s_i(1-s_i)\boldsymbol{x_i} \\ &= -\sum_{i=1}^{n}(y_i-s_i)\boldsymbol{x_i} \\ &= -X^T(\boldsymbol{y}-\boldsymbol{s}) \quad \text{where} \quad s = \begin{bmatrix} s_1 & s_2 & \dots & s_n \end{bmatrix}^T\end{aligned}$$

2. Derive $\nabla_w^2 J(w)$

$$\begin{aligned}\nabla_w^2 J(w) &= \nabla_w[\nabla_w J(w)]^T \\ &= \nabla_w[-X^T(\boldsymbol{y}-\boldsymbol{s})]^T \\ &= \nabla_w[-(\boldsymbol{y}^T-\boldsymbol{s}^T)X] \\ &= (\boldsymbol{s}^T(\boldsymbol{1}-\boldsymbol{s})X)^T X \\ &= X^T(1-\boldsymbol{s})^T\boldsymbol{s}X \\ &= X^T\Omega X \quad \text{where} \quad \Omega = \begin{bmatrix} s_1(1-s_1) & 0 & \dots & 0 \\ 0 & s_2(1-s_2) & \dots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \dots & s_n(1-s_n) \end{bmatrix}\end{aligned}$$

3. By the update rule of Newton's method derived in the lecture,

$$\begin{aligned}\boldsymbol{w}^{(n+1)} &= \boldsymbol{w}^{(n)} - (\nabla^2_{\boldsymbol{w}^{(n)}} J(\boldsymbol{w}^{(n)}))^{-1}\nabla_{\boldsymbol{w}^{(n)}} J(\boldsymbol{w}^{(n)}) \\ &= \boldsymbol{w}^{(n)} + (X^T\Omega^{(n)}X)^{-1}(X^T(\boldsymbol{y}-\boldsymbol{s}^{(n)}))\end{aligned}$$

4.

$$\boldsymbol{s}^{(0)} = \begin{bmatrix} 0.9478 \\ 0.8808 \\ 0.8022 \\ 0.5250 \end{bmatrix}$$

$$\boldsymbol{w}^{(1)} = \begin{bmatrix} 1.3247 \\ 3.0499 \\ -6.8291 \end{bmatrix}$$

$$\boldsymbol{s}^{(1)} = \begin{bmatrix} 0.9474 \\ 0.9746 \\ 0.0315 \\ 0.1044 \end{bmatrix}$$

$$\boldsymbol{w}^{(2)} = \begin{bmatrix} 1.3660 \\ 4.1585 \\ -9.1996 \end{bmatrix}$$

3 Wine Classification with Logistic Regression

1.

$$\boldsymbol{w}^{(n+1)} = \boldsymbol{w}^{(n)} + \epsilon[2\lambda\boldsymbol{w}^{(n)} - X^T(y - s^{(n)})]$$

2. The "train_bgd_decay_False" (blue line) in the Figure 1 shows cost function versus iterations of Batch Gradient Descent(BGD) with the following parameters.

$$\epsilon = 10^{-7}$$
$$\lambda = 1$$

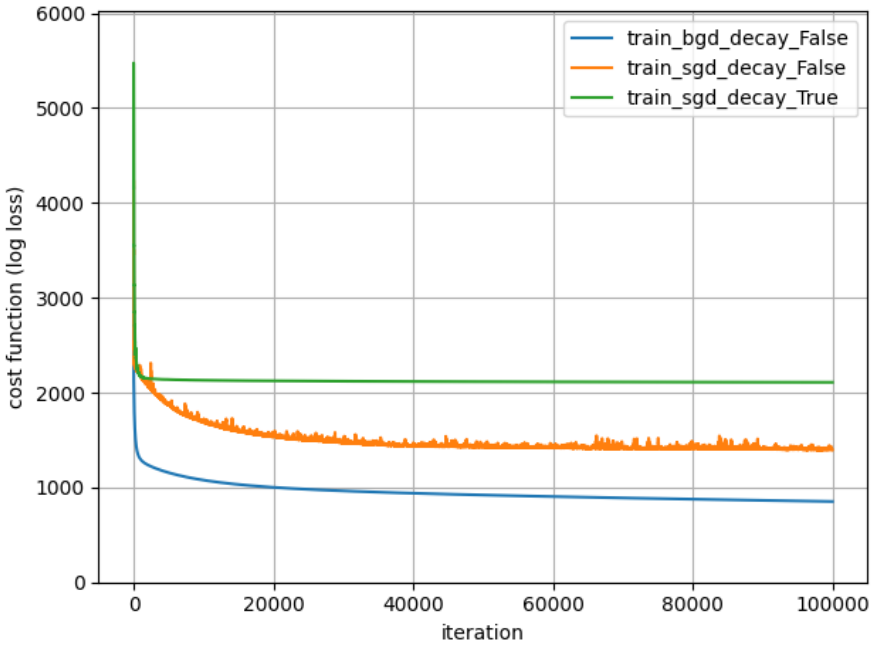

Figure 1: Update iterations and cost function

(Code is in the appendix. Corresponding pages are selected at Grade Scope submission)

3.

$$\boldsymbol{w}^{(n+1)} = \boldsymbol{w}^{(n)} + \epsilon[2\lambda\boldsymbol{w}^{(n)} - \boldsymbol{x_i}^T(\boldsymbol{y_i} - \boldsymbol{s_i}^{(n)})]$$

4. The "train_sgd_decay_False" (orange line) in the Figure 1 shows cost function versus iterations of Stochastic Gradient Descent(SGD) with thefollowingg parameters.

$$\epsilon = 10^{-5}$$
$$\lambda = 1$$

BGD converges more quickly than SGD because BGD uses all input for one update while SGD uses only one randomly selected input. If the input size is much larger, SGD may perform better because SGD takes shorter computational time for one iteration than BGD.
(Code is in the appendix. Corresponding pages are selected at Grade Scope submission)

5. The "train_sgd_decay_True" (green line) in the Figure 1 shows cost function versus iterations of SGD with the following parameters and shrinking learning rate.

$$\epsilon = 10^{-3}$$
$$\lambda = 1$$

SGD with shrinking learning rate needs more iterations to converge than SGD with static learning rate. Shrinking learning rate has possibility to reach the better optimum point by searching optimum from roughly to finely. However, in this problem, the shrinking policy ($\epsilon_t = \frac{\delta}{t}$) updates the learning rate too drastically.
(Code is in the appendix. Corresponding pages are selected at Grade Scope submission)

6.
- Kaggle username: Takuma Kinoshita
- Best score: 0.99396
- Preprocess: Standardization ($x' = \frac{x-\mu}{\sigma}$) to all features to remove the scale difference
- Model: Logistic Regression with l_2 regularization ($C = 1$), fixed learning rate (10^{-5}) and 10^6 times update iterations

(Code is in the appendix. Corresponding pages are selected at Grade Scope submission)

4 Convergence of Batch Gradient Descent in Logistic Regression

1. Derive $\nabla_w J(w)$.

$$\begin{aligned} J(\boldsymbol{w}) &= \frac{\lambda||\boldsymbol{w}||^2}{2} - \boldsymbol{y}^T \ln(s(X\boldsymbol{w})) - (1-\boldsymbol{y})^T \ln(\boldsymbol{1} - s(X\boldsymbol{w})) \\ &= \frac{\lambda||w||^2}{2} - \sum_{i=1}^{n} y_i \ln(s(\boldsymbol{x_i}^T\boldsymbol{w})) - (1-y_i)\ln(1 - s((\boldsymbol{x_i}^T\boldsymbol{w}))) \\ \nabla_{\boldsymbol{w}} J(\boldsymbol{w}) &= \lambda\boldsymbol{w} - \sum_{i=1}^{n}(y_i - s(\boldsymbol{x_i}^T\boldsymbol{w}))\boldsymbol{x_i} \quad \text{(from Question 2.1)} \\ &= \lambda\boldsymbol{w} - X^T(\boldsymbol{y} - s(X\boldsymbol{w})) \end{aligned}$$

The update rule of batch gradient descent is:

$$\begin{aligned} \boldsymbol{w}^{(t+1)} &= \boldsymbol{w}^{(t)} - \epsilon\nabla_{\boldsymbol{w}} J(\boldsymbol{w}) \\ &= \boldsymbol{w}^{(t)} - \epsilon[\lambda\boldsymbol{w}^{(t)} - X^T(\boldsymbol{y} - s(X\boldsymbol{w}^{(t)}))] \\ &= g(\boldsymbol{w}^{(t)}) \end{aligned}$$

2. Derive $\nabla_w^2 J(w)$.

$$\begin{aligned} \nabla_{\boldsymbol{w}}^2 J(\boldsymbol{w}) &= \nabla_{\boldsymbol{w}}[\nabla_{\boldsymbol{w}} J(\boldsymbol{w})]^T \\ &= \nabla_{\boldsymbol{w}}[\lambda\boldsymbol{w}^T - (\boldsymbol{y}^T - s(X\boldsymbol{w})^T)X] \\ &= \lambda I + X^T\Omega X \quad \text{(from Question 2.2)} \end{aligned}$$

$$\text{where} \quad \Omega = \begin{bmatrix} s_1(1-s_1) & 0 & \dots & 0 \\ 0 & s_2(1-s_2) & \dots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \dots & s_n(1-s_n) \end{bmatrix}$$

First, λI is positive definite because $\lambda > 0$. Second, Ω is positive definite because all diagonal entities of diagonal matrix Ω is positive. When Ω is positive definite, $X^T\Omega X$ is positive semidefinite. Therefore, $\lambda I + X^T\Omega X$ is positive definite.

3. By Mean Value Theorem, there exist w_ρ satisfying:

$$\begin{aligned} \frac{g(\boldsymbol{w}_1)_i - g(\boldsymbol{w}_2)_i}{\boldsymbol{w}_{1i} - \boldsymbol{w}_{2i}} &= \nabla g(\boldsymbol{w}_\rho)_i \quad (i = 1, \cdots, d) \\ g(\boldsymbol{w}_1)_i - g(w_2)_i &= \nabla g(\boldsymbol{w}_\rho)_i(\boldsymbol{w}_{1i} - \boldsymbol{w}_{2i}) \\ ||g(\boldsymbol{w}_1) - g(\boldsymbol{w}_2)||^2 &= ||\nabla g(\boldsymbol{w}_\rho)^T(\boldsymbol{w}_1 - \boldsymbol{w}_2)||^2 \end{aligned}$$

By Cauchy-Schwarz ineqeuality,

$$||\nabla g(\boldsymbol{w}_\rho)(\boldsymbol{w}_1 - \boldsymbol{w}_2)||^2 \le ||\nabla g(\boldsymbol{w}_\rho)||^2||(\boldsymbol{w}_1 - \boldsymbol{w}_2)||^2$$

By adjusting ϵ, there exist $||\nabla g(\boldsymbol{w}_\rho)||^2 \in (0, 1)$. Therefore, $g(\cdot)$ is a contraction.

$$||g(\boldsymbol{w}_1) - g(\boldsymbol{w}_2)||^2 \le \rho||(\boldsymbol{w}_1 - \boldsymbol{w}_2)||^2$$

4.

$$\begin{aligned}||\boldsymbol{w}^{(t)} - \boldsymbol{w}^{(t+1)}|| &\leq \rho||\boldsymbol{w}^{(t-1)} - \boldsymbol{w}^{(t)}|| \\ &\leq \rho^2||\boldsymbol{w}^{(t-2)} - \boldsymbol{w}^{(t-1)}|| \\ &\vdots \\ &\leq \rho^{t-1}||\boldsymbol{w}^{(0)} - \boldsymbol{w}^{(1)}||\end{aligned}$$

Because $\rho \in (0, 1)$,

$$\lim_{t\to\infty} \rho^{t-1}||\boldsymbol{w}^{(0)} - \boldsymbol{w}^{(1)}|| = 0$$

Thus,

$$\lim_{t\to\infty} ||\boldsymbol{w}^{(t)} - \boldsymbol{w}^{(t+1)}|| = 0$$

Therefore, $\boldsymbol{w}^{(t)}$ converges to the unique minimizer.

5. Find ϵ which minimizes $\rho = ||\nabla g(\boldsymbol{w}_\rho)||^2$

6. From Question 4-2, when $\lambda = 0$,

$$\nabla_{\boldsymbol{w}^2} J(\boldsymbol{w}) = X^T \Omega X$$

Because $X^T \Omega X$ is positive semidefinite, there are more than one optimal points. Therefore, the logistic regression problem does not converge.

5 A Bayesian Interpretation of Lasso

1. By Bayes' Theorem,

$$\begin{aligned} f(w|(\boldsymbol{x_i}, y_i)_{i\in[n]}) &= \frac{f(\boldsymbol{x_{i\in[n]}}, y_{i\in[n]}|\boldsymbol{w})f(\boldsymbol{w})}{f(\boldsymbol{x_{i\in[n]}}, y_{i\in[n]})} \\ &= \frac{f(y_{i\in[n]}|\boldsymbol{x_{i\in[n]}}, \boldsymbol{w})f(\boldsymbol{x_{i\in[n]}})f(\boldsymbol{w})}{f(\boldsymbol{x_{ii\in[n]}}, y_{i\in[n]})} \\ &\propto f(y_{i\in[n]}|\boldsymbol{x_{i\in[n]}}, \boldsymbol{w})f(\boldsymbol{w}) \end{aligned}$$

2. From Question 5.1

$$\begin{aligned} l(w) &= \ln f(\boldsymbol{w}|\boldsymbol{x_{i\in[n]}}, y_{i\in[n]}) \\ &\propto f(y_{i\in[n]}|\boldsymbol{x_{i\in[n]}}, \boldsymbol{w})f(\boldsymbol{w}) \\ &= \ln \prod_{i=1}^{n} \frac{1}{\sigma\sqrt{2\pi}} e^{\frac{-(y_i - \boldsymbol{w}\cdot\boldsymbol{x_i})}{2\sigma^2}} \prod_{j=1}^{d} \frac{e^{\frac{|w_j|}{b}}}{2b} \\ &\propto -\sum_{i=1}^{n} \frac{(y_i - \boldsymbol{w}\cdot\boldsymbol{x_i})^2}{2\sigma^2} + \sum_{j=1}^{d} \frac{|w_j|}{b} \\ &\propto -\sum_{i=1}^{n} (y_i - \boldsymbol{w}\cdot\boldsymbol{x_i})^2 + \frac{2\sigma^2}{b} \sum_{j=1}^{d} |w_j| \end{aligned}$$

Thus, when $\frac{2\sigma^2}{b} = \lambda$, maximizing $l(\boldsymbol{w})$ is equivalent of minimizing $\sum_{i=1}^{n}(y_i - \boldsymbol{w}\cdot\boldsymbol{x_i})^2 + \lambda||\boldsymbol{w}||_1$

6 l_1-regularization, l_2-regularization, and Sparsity

1.

$$\begin{aligned}
J_1(\boldsymbol{w}) &= ||X\boldsymbol{w} - \boldsymbol{y}||^2 + \lambda||\boldsymbol{w}||_1 \\
&= \boldsymbol{y}^T\boldsymbol{y} - (X\boldsymbol{w})^T\boldsymbol{y} - \boldsymbol{y}^T(X\boldsymbol{w}) + (X\boldsymbol{w})^T X X\boldsymbol{w} + \lambda||\boldsymbol{w}||_1 \\
&= \boldsymbol{y}^T\boldsymbol{y} - 2\boldsymbol{y}^T X\boldsymbol{w} + \boldsymbol{w}^T X^T X\boldsymbol{w} + \lambda||\boldsymbol{w}||_1 \\
&= \boldsymbol{y}^T\boldsymbol{y} - 2\boldsymbol{y}^T X\boldsymbol{w} + n\boldsymbol{w}^T\boldsymbol{w} + \lambda||\boldsymbol{w}||_1 \\
&= ||\boldsymbol{y}||^2 + \sum_{i=1}^{d} -2\boldsymbol{y}^T\boldsymbol{x_{*i}}w_i + nw_i^2 + \lambda|w_i| \\
&= ||\boldsymbol{y}||^2 + \sum_{i=1}^{d} f(\boldsymbol{x_{*i}}, w_i)
\end{aligned}$$

2. When $w_i^* > 0$,

$$\begin{aligned}
\nabla_w J_1(\boldsymbol{w})_i &= -2\boldsymbol{y}^T\boldsymbol{x_{*i}} + 2nw_i + \lambda = 0 \\
\Rightarrow w_{*i} &= \frac{2\boldsymbol{y}^T\boldsymbol{x_{*i}} + \lambda}{2n}
\end{aligned}$$

When $w_i^* < 0$,

$$\begin{aligned}
\nabla_w J_1(\boldsymbol{w})_i &= -2\boldsymbol{y}^T\boldsymbol{x_{*i}} + 2nw_i - \lambda = 0 \\
\Rightarrow w_{*i} &= \frac{2\boldsymbol{y}^T\boldsymbol{x_{*i}} - \lambda}{2n}
\end{aligned}$$

When $w_i^* = 0$, by subgradient,

$$\begin{aligned}
\nabla_{\boldsymbol{w}} J_1(\boldsymbol{w})_i &= [-2\boldsymbol{y}^T\boldsymbol{x_{*i}} + 2nw_i - \lambda, -2\boldsymbol{y}^T\boldsymbol{x_{*i}} + 2nw_i + \lambda] = 0 \\
&\Rightarrow \begin{cases} -2\boldsymbol{y}^T\boldsymbol{x_{*i}} + 2nw_i - \lambda \le 0 \\ -2\boldsymbol{y}^T\boldsymbol{x_{*i}} + 2nw_i + \lambda \ge 0 \end{cases} \\
&\Rightarrow -\lambda \le 2\boldsymbol{y}^T\boldsymbol{x_{*i}} + 2nw_i \le \lambda \\
&\Rightarrow -\lambda \le 2\boldsymbol{y}^T\boldsymbol{x_{*i}} \le \lambda \quad (w_i^* = 0)
\end{aligned}$$

3. Similary,

$$\begin{aligned}
J_2(w) &= ||X\boldsymbol{w} - \boldsymbol{y}||^2 + \lambda||\boldsymbol{w}||_2 \\
&= ||\boldsymbol{y}||^2 + \sum_{i=1}^{d} -2\boldsymbol{y}^T\boldsymbol{x_{*i}}w_i + nw_i^2 + \lambda w_i^2 \\
\nabla_{\boldsymbol{w}} J_2(\boldsymbol{w})_i &= -2\boldsymbol{y}^T\boldsymbol{x_{*i}} + 2nw_i - 2\lambda w_i = 0 \\
&\Rightarrow w_i^* = \frac{\boldsymbol{y}^T\boldsymbol{x_{*i}}}{n + \lambda}
\end{aligned}$$

When $w_i^{\#} = 0$,

$$w_i^* = \frac{\boldsymbol{y}^T \boldsymbol{x_{*i}}}{n + \lambda} = 0$$
$$\Rightarrow \boldsymbol{y}^T \boldsymbol{x_{*i}} = 0$$

4. w^* (l_1 regularization) is more likely to be sparse than $w^{\#}$ (l_2 regularization). In l_1 regularization, to be $w_i = 0$, $\boldsymbol{y}^T \boldsymbol{x_{*i}}$ need to in the range between $-\lambda$ and λ. However, in l_2 regularization, $\boldsymbol{y}^T \boldsymbol{x_{*i}}$ need to be exactly 0.

7 Appendix: Source Codes

1. utils.py (Question 3.2, 3.4, 3.5, 3.6)

```
from collections import defaultdict

import numpy as np
import pandas as pd
import scipy.io as sio
from scipy.special import expit
import matplotlib.pyplot as plt
from sklearn.metrics import accuracy_score

from tqdm import tqdm


def load_data(path):
    data = sio.loadmat(path)
    return data['X'], data['X_test'], data['y']


def my_log_loss(y, s):
    t1 = y.T @ np.log(s)
    t2 = (~y.astype(bool)).astype(int).T @ np.log(1 - s)
    return -float(t1 + t2)


def plot_costs(costs_df, fig_path):
    fig, ax = plt.subplots()
    costs_df.plot(ax=ax)
    ax.set_xlabel('iteration')
    ax.set_ylabel('cost function (log loss)')
    ax.set_ylim(0, float(costs_df.max().max()) * 1.1)
    ax.grid()
    fig.tight_layout()
    fig.savefig(fig_path)


class LogisticRegression():

    def __init__(self, C, lr, lr_decay=False, max_iter=1e2, solver='sgd'):
        self.C = C
        self.lr = lr
        self.lr_decay = lr_decay
        self.solver = solver
        self.max_iter = max_iter
        self.solver = solver
        if lr_decay:
            self.lr_init = lr

    def fit(self, x_tr, y_tr, verbose=False):
        # initialize
        ones = np.ones((x_tr.shape[0], 1))
        x_tr = np.hstack((x_tr, ones))
        self.w = np.zeros((x_tr.shape[1], 1))

        costs = defaultdict(list)
```

```
54
55          for i in tqdm(range(self.max_iter)):
56              # output prob
57              s = expit(x_tr @ self.w)
58
59              # cost function
60              if verbose:
61                  cost_train = my_log_loss(y_tr, s)
62                  costs[f'train_{self.solver}_decay_{self.lr_decay}'].append(
    cost_train)
63
64              # learning rate decay
65              if self.lr_decay:
66                  self.lr = self.lr_init / (i + 1)
67
68              # update weight
69              if self.solver == 'bgd':
70                  self.w = self.w - self.lr * (2 * self.C * self.w - x_tr.T @ (y_tr -
     s))
71              if self.solver == 'sgd':
72                  rand_i = np.random.randint(0, x_tr.shape[0], 1)
73                  self.w = self.w - self.lr * (2 * self.C * self.w - (y_tr[rand_i] -
    s[rand_i]) * x_tr[rand_i].T)
74
75          if verbose:
76              return pd.DataFrame(costs)
77
78      def predict_proba(self, x):
79          ones = np.ones((x.shape[0], 1))
80          x = np.hstack((x, ones))
81          return expit(x @ self.w).ravel()
82
83      def predict(self, x):
84          prob = self.predict_proba(x)
85          return np.round(prob).ravel()
86
87      def score(self, x, y):
88          pred = self.predict(x)
89          return accuracy_score(y, pred)
```

2. gd_logreg.py (Question 3.2, 3.4, 3.5)

```
1  import numpy as np
2  import pandas as pd
3  from pathlib import Path
4
5  from utils import load_data, LogisticRegression, plot_costs
6
7
8  if __name__ == "__main__":
9      # set seed
10     np.random.seed = 289
11
12     # laod data
13     data_path = Path('data/data.mat')
14     x_train, x_test, y_train = load_data(data_path)
15
```

```
    # parameters
    bgd_params = {
        'C': 1,
        'lr': 1e-7,
        'max_iter': int(1e5),
        'solver': 'bgd'
    }
    sgd_params = {
        'C': 1,
        'lr': 1e-5,
        'max_iter': int(1e5),
        'solver': 'sgd'
    }
    decay_params = {
        'C': 1,
        'lr': 1e-3,
        'lr_decay': True,
        'max_iter': int(1e5),
        'solver': 'sgd'
    }

    # models
    model = LogisticRegression(**bgd_params)
    bgd_costs_df = model.fit(x_train, y_train, verbose=True)

    model = LogisticRegression(**sgd_params)
    sgd_costs_df = model.fit(x_train, y_train, verbose=True)

    model = LogisticRegression(**decay_params)
    decay_costs_df = model.fit(x_train, y_train, verbose=True)

    # plot
    costs_df = pd.concat([bgd_costs_df, sgd_costs_df, decay_costs_df], axis=0)
    fig_path = Path('figures/iterations_vs_costs.png')
    plot_costs(costs_df, fig_path)
```

3. kaggle.py (Question 3.6)

```
from pathlib import Path
from collections import defaultdict

import numpy as np
import pandas as pd
from sklearn.model_selection import StratifiedKFold
from sklearn.metrics import accuracy_score
from sklearn.preprocessing import StandardScaler

from utils import load_data, plot_costs, LogisticRegression
from save_csv import results_to_csv


def cross_validation(x_train, y_train, params, n_splits, random_state):
    skf = StratifiedKFold(n_splits=n_splits, shuffle=True, random_state=seed)
    scores = defaultdict(list)

    for tr_idx, va_idx in skf.split(x_train, y_train):
        x_tr, x_va = x_train[tr_idx], x_train[va_idx]
```

```
        y_tr, y_va = y_train[tr_idx], y_train[va_idx]

        model = LogisticRegression(**params)
        model.fit(x_tr, y_tr)

        scores['acc_tr'].append(model.score(x_tr, y_tr))
        scores['acc_va'].append(model.score(x_va, y_va))

    return pd.DataFrame(scores)


if __name__ == "__main__":

    # set seed
    seed = 289
    np.random.seed = seed

    # load data
    data_path = Path('data/data.mat')
    x_train, x_test, y_train = load_data(data_path)

    # parameters
    params = {
        'C': 1,
        'lr': 1e-5,
        'lr_decay': False,
        'max_iter': int(1e6),
        'solver': 'bgd'
    }

    # preprocessing
    scaler = StandardScaler()
    scaler.fit(np.vstack((x_train, x_test)))
    x_train = scaler.transform(x_train)
    x_test = scaler.transform(x_test)

    # evaluate
    scores_df = cross_validation(x_train, y_train, params=params, n_splits=5,
    random_state=seed)
    print(scores_df)
    print(scores_df.mean(axis=0))

    # predict
    model = LogisticRegression(**params)
    model.fit(x_train, y_train)
    y_pred = model.predict(x_test)

    # save
    results_to_csv(y_pred)
```