01.112 MACHINE LEARNING (2017)
HOMEWORK 2

SHAOWEI LIN

The problems which require your solutions will be marked (a), (b), (c), ..., at the end of every section. Each problem is worth one point. This assignment has a total of 12 points.

1. Ridge Regression With Offset

In this problem, we will continue from where Homework 1 Question 3 left off, and explore the effects of ridge regression on generalization. We will also try different software solutions for gradient descent. As before, we will be using the following data set.

`https://www.dropbox.com/s/oqoyy9p849ewzt2/linear.csv?dl=1`.

In ridge regression with offset, our model consists of linear functions

$$f : \mathbb{R}^d \to \mathbb{R}, \quad f(x; w) = w^\top x = w_d x_d + \cdots + w_1 x_1 + w_0.$$

where $w = (w_d, \ldots, w_1, w_0)$, $x = (x_d, \ldots, x_1, x_0)$ and $x_0 = 1$. We want to find the parameter $\hat{w}$ that minimizes the training loss

$$\mathcal{L}_{n,\lambda}(w; \mathcal{S}_n) = \frac{1}{2n} \sum_{(x,y)\in\mathcal{S}_n} (f(x; w) - y)^2 + \frac{\lambda}{2}(w_1^2 + \cdots + w_d^2).$$

Note that the offset w_0 is not penalized in the regularizer. The training gradient is then

$$\nabla\mathcal{L}_{n,\lambda}(w; \mathcal{S}_n) = \lambda(w_d, \ldots, w_1, 0)^\top + \frac{1}{n}(X^\top X)w - \frac{1}{n}X^\top Y$$

where $X = (x^{(1)}, \ldots, x^{(n)})^\top \in \mathbb{R}^{n\times d}$ and $Y = (y^{(1)}, \ldots, y^{(n)})^\top \in \mathbb{R}^n$. Setting the gradient to zero and solving the resulting equations, we find that the optimal parameter is given by

$$\hat{w} = (n\lambda J + X^\top X)^{-1} X^\top Y, \quad J = \begin{pmatrix} 1 & 0 & \cdots & 0 & 0 \\ 0 & 1 & \cdots & 0 & 0 \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & \cdots & 1 & 0 \\ 0 & 0 & \cdots & 0 & 0 \end{pmatrix} \in \mathbb{R}^{d\times d}.$$

To find a suitable value for λ, we will set aside a small subset of the provided data set for estimating the test loss. This subset is called the *validation set*, which we use to compute the *validation loss*. The remainder of the data will be called the *training set*.

Date: October 2, 2017.

(a) Let the first 10 entries of the data set be the validation set, and the last 40 entries be the training set. Concatenate their features into matrices `vX` and `tX`, and their responses into vectors `vY` and `tY`. Print the shapes of `vX`, `tX`, `vY` and `tY`.

(b) Write a program in Theano that performs ridge regression by using a regularization penalty of $\frac{1}{2}\lambda\|w\|^2$ with $\lambda = 0.15$. You may use the source codes from Homework 1. Print the resulting value of w. Which feature may we assume to be irrelevant?

(c) Compute the optimal solution using BFGS optimizer from `scipy`. Below is a simple example where the cost (or loss) and the gradient is given to the optimizer.

```
import numpy as np
from scipy.optimize import fmin_l_bfgs_b as minimize
def costgrad(x,a):
    cost = np.sum((x-a)**2)
    grad = 2*(x-a)
    return cost,grad
a = np.array([1,2])        #data to pass into costgrad
x = np.random.randn(2)     #parameter to be optimized
optx,cost,messages = minimize(costgrad,x,args=[a])
```

Print the resulting value of w for $\lambda = 0.15$.

(d) Write a function

```
ridge_regression(tX, tY, l)
```

that takes the training features, training responses and regularizing parameter λ, and outputs the exact solution w for ridge regression with offset. Print the resulting value of w for $\lambda = 0.15$.

(e) Use the following code to plot graphs of the validation loss and training loss as λ varies on a logarithmic scale from $\lambda = 10^{-5}$ to $\lambda = 10^{0}$.

```
tn = tX.shape[0]
vn = vX.shape[0]
tloss = []
vloss = []
index = -np.arange(0,5,0.1)
for i in index:
    w = ridge_regression(tX,tY,10**i)
    tloss = tloss+[np.sum((np.dot(tX,w)-tY)**2)/tn/2]
    vloss = vloss+[np.sum((np.dot(vX,w)-vY)**2)/vn/2]
import matplotlib.pyplot as plt
plt.plot(index,np.log(tloss),'r')
plt.plot(index,np.log(vloss),'b')
```

Write down the value of λ that minimizes the validation loss.

2. Clustering

In this problem, we will compress an image by using k-means clustering to create smaller color palettes for the pixels of the image. We first load the 686×1030 image of SUTD that uses a palette with 16.8 million colors. We will reduce this palette down to 32 colors.

```
%matplotlib inline
import numpy as np
import numpy.random as rng
import matplotlib.pyplot as plt
import matplotlib.image as mpimg
from sklearn.cluster import KMeans
from sklearn.metrics import pairwise_distances_argmin

n_colors = 32
pic = 'https://www.dropbox.com/s/bmwwfct2qxjfje4/sutd.png?dl=1'
img = mpimg.imread(pic)
img = img[:,:,:3]
```

We then reshape the $686 \times 1030 \times 3$ array `img` into a 706580×3 array.

```
w, h, d = tuple(img.shape)
image_array = np.reshape(img, (w * h, d))
```

The following helper function will also come in useful. It displays a picture given its palette, its width w, its height h and the labels for the colors of each pixel. The `palette` is a $k \times 3$ array which gives the RGB values of k colors, where each R, G and B value is between 0 and 1. The `labels` array is one-dimensional with wh entries, and it lists the colors of the pixels of the picture in order starting from the top left corner and ending with the bottom right corner. Each entry of `labels` is an integer between 0 and $k - 1$ inclusive.

```
def recreate_image(palette, labels, w, h):
    d = palette.shape[1]
    image = np.zeros((w, h, d))
    label_idx = 0
    for i in range(w):
        for j in range(h):
            image[i][j] = palette[labels[label_idx]]
            label_idx += 1
    return image
```

Your objective is to derive `kmeans_palette` and `kmeans_labels` using k-means clustering. We compare the quality of the compressed image with another that uses a random palette and its corresponding labels, `random_palette` and `random_labels`.

```
plt.figure(1)
plt.clf()
ax = plt.axes([0, 0, 1, 1])
plt.axis('off')
plt.title('Original image (16.8 million colors)')
plt.imshow(img)

plt.figure(2)
plt.clf()
ax = plt.axes([0, 0, 1, 1])
plt.axis('off')
plt.title('Compressed image (K-Means)')
plt.imshow(recreate_image(kmeans_palette, kmeans_labels, w, h))

plt.figure(3)
plt.clf()
ax = plt.axes([0, 0, 1, 1])
plt.axis('off')
plt.title('Compressed image (Random)')
plt.imshow(recreate_image(random_palette, random_labels, w, h))
plt.show()
```

(a) First, sample 1000 pixels at random from the original image. Use

```
sklearn.cluster.KMeans
```

to partition the colors of these pixels into 32 clusters. Extract the cluster centers as `kmeans_palette`, and use the trained `KMeans` object to predict the pixel labels.

(b) Now, sample 32 pixels at random from the original image. Use their colors to form a `random_palette`. Compute the label for each pixel of the original image by finding the color in the palette that is closest to the color of the pixel. You may apply

```
sklearn.metrics.pairwise_distances_argmin
```

directly to find the closest representative.

(c) Given a cluster $\{x^{(1)}, x^{(2)}, \ldots, x^{(m)}\}$ of points, prove that the point z minimizing

$$\sum_{i=1}^{m} \|x^{(i)} - z\|^2$$

is the centroid

$$z = \frac{1}{m} \sum_{i=1}^{m} x^{(i)}.$$

3. Logistic Regression

In this problem, we will attempt the *Titanic: Machine Learning from Disaster* competition on Kaggle, by applying logistic regression and using the `pandas` and `sklearn` modules.

`https://www.kaggle.com/c/titanic`.

First, sign up for an account on Kaggle, and download the training and the test data from the competition. Import the modules, and load the data as DataFrames. Split `train.csv` into a training set and a validation set. The validation set will act as a proxy for the text set to show us how well our classifier is doing. Separate the features X from the labels Y.

```
import numpy as np
import pandas as pd
X_data  = pd.read_csv("train.csv")
X_test  = pd.read_csv("test.csv")
X_valid = X_data.sample(frac=0.2,random_state=200)
X_train = X_data.drop(X_valid.index)
Y_data  = X_data["Survived"]
Y_valid = X_valid["Survived"]
Y_train = X_train["Survived"]
ID_test = X_test["PassengerId"]
```

We take a peek at the head and the descriptions of some of the DataFrames.

```
from IPython.display import display
display(X_data.head())
display(X_data.describe())
display(X_test.head())
display(X_test.describe())
```

Let us prepare the data for logistic regression. Initially, we will only pre-process the training data, but later, we will apply these same changes to the test data. Our first task is to drop unnecessary information such as the `PassengerId`, `Survived`, `Name`, `Ticket` and `Cabin`.

```
df = X_train
df.drop(["Survived"],axis=1,inplace=True,errors="ignore")
df.drop(["PassengerId","Name","Ticket","Cabin"],axis=1,inplace=True)
```

From the DataFrame descriptions above, we note that there are missing values for `Embarked`, `Fare`, and `Age`. The first statement below fills in `NaN`'s in `Embarked` with the most common value. Complete the next two statements with the median and mean respectively.

```
df["Embarked"].fillna(df["Embarked"].mode()[0],inplace=True)
df["Fare"].fillna(                            ,inplace=True)
df["Age"].fillna(                             ,inplace=True)
```

Next, we replace categorical features with one-hot encodings (or dummies). The `Embarked` column has been done for you. Complete the statements below for `Sex` and `Pclass`.

```
df = df.join(pd.get_dummies(df["Embarked"]))
df.drop(["Embarked"],axis=1,inplace=True)
df = df.join(                                )
df.drop(                                     )
df = df.join(                                )
df.drop(                                     )
```

Finally, we create two new binary features which will help us in our classification problem. Set `Family` equal to 1 if the passenger has siblings, spouses, parents or children (i.e. `SibSp` or `Parch` is non-zero). Let `Child` be 1 if the passenger's age is less than 16.

```
df.loc[:,"Family"] =
df.loc[:,"Child"] =
```

We now write and apply a function to automate the pre-processing of the data.

```
def preprocess(df):
    :
    return df
```

(a) Run the following statements, and print out the displayed information.

```
X_train = preprocess(X_train)
X_valid = preprocess(X_valid)
X_data = preprocess(X_data)
X_test = preprocess(X_test)
display(X_train.head())
```

(b) Using `LogisticRegression` from the `sklearn.linear_model` module, fit a classifier to the training set `X_train` and `Y_train`. Evaluate the accuracy of the classifier via the validation set `X_valid` and `Y_valid`. What is the score?

(c) Fit a classifier to the data `X_data` and `Y_data`. What is the value of the parameter vector $\theta = (\theta_0, \theta_1, \ldots, \theta_d)$?

(d) Apply the classifier in (c) to the test data `X_test`. Let `Y_test` denote the predicted labels. You are welcome to tweak the code to improve your model. Use the following code to prepare a submission file for Kaggle.

```
ans = pd.DataFrame({"PassengerId":ID_test,"Survived":Y_test})
ans.to_csv("submit.csv", index=False)
```

Submit the file on Kaggle. Write down your Kaggle ID and competition score.