CSCI 3320: Fundamental of Machine Learning **2018-2019 Term 2**

Programming Assignment 2

Instructor: Prof. John C.S. Lui Due: 23:59 on Sunday, Mar. 24, 2019

1 Introduction

This programming assignment consists of two parts.

- The first part will guide you through a complete version of Linear Regression. In the first part, you will implement LR of one/multiple features using optimization techniques including gradient descent and normal equation. You will also have the idea of data cleaning, data standardization and regularization.
- In the second part, you will learn how to do classification using logistic regression.

1.1 Requirements

1. ✍ **Report**. When you meet a ✍ symbol, it means that you need to copy down some output/figures or write down your answers to the questions to `Assignment2.pdf`. (You can create a `Assignment2.doc` file and then save it as `Assignment2.pdf`.)
2. ⌨ **Python script**. When you meet an ⌨ symbol, it means that you need to write or add some Python scripts in a corresponding `.py` script file.

1.2 File Descriptions

To start, you need to download the `asgn2.zip` file from the course website. In `asgn2.zip`, we provide the following files for you:

- `imports-85.data`: contains 26 columns of featured data.
- `imports-85.names`: contains the descriptions of `imports-85.data`.
- `lr_rawdata.py`: an almost empty file. You should write scripts to this file to processing the raw data.
- `lr_mfeature.py`: an almost empty file. You should write scripts to this file to do linear regression with multiple features.
- `poly_regular.py`: contains some python scripts for you to study linear regression variants without/with regularization.
- `logistic_clf.py`: contains scripts for simple classification using logistic regression.

Note: Please do **not** change the file names (`*.py`) of the files described above.

2 Linear Regression

In the lecture, we told you that it's easy to write down a basic version of Linear regression with scikit-learn. The following sections are designed to let you practice what you have learned.

2.1 Explore Raw Dataset

In this part, you will explore the real world data using scikit-learn. Usually, the data from real world is messy. You have to pre-process the data and store it using the standard data representation in scikit-learn.

2.1.1 Read the raw data with `pandas.read_csv()`

For data pre-processing, `pandas` is a great Python package. To learn how to explore the raw data with `pandas`, we will walk you through the procedures of processing the raw data `imports-85.data`[1].

- ⌨ Add scripts to `lr_rawdata.py` to read the `imports-85.data`. You can start with the following scripts (Please replace commented the code `"'..."'` with your own.):

```
df = pd.read_csv('imports-85.data',
  header=None,
  names=['''Names of 26 features according to imports-85.names'''],
  na_values=('''Some values are missing, treat the '?' with NaN'''))
```

(*Hint*: read the document of `read_csv()` in pandas official website.)

2.1.2 Data cleaning: remove the data sample with missing values

The missing values in the raw data are denoted with `'?'`. If you have set the parameter `na_values` properly in the `read_csv()` function, then each missing value will be saved as an `NaN`. (`NaN` means Not a Number.) Now we want to remove the data samples with the value `NaN`.

- ⌨ Add scripts to `lr_rawdata.py` to remove all samples/rows that have `NaN` values. (*Hint*: the function `dropna()` is helpful.)

[1] We provided this data in the `asgn2.zip`. Data is downloaded from the Data web directory: `https://archive.ics.uci.edu/ml/machine-learning-databases/autos/`. Note that the file `imports-85.names` contains the data descriptions in this directory.

2.1.3 Data standardization

Many estimators perform better when they are trained on standardized data sets. Standardized data has **zero mean** and **unit variance**. The intuition is that if a feature's variance is orders of magnitude greater than the variances of the other features, that feature may dominate the learning algorithm and prevent it from learning from the other variables.

- ⌨ Add scripts to `lr_rawdata.py` to split the dataframe `df` into training data and test data. For simplicity, please make the former 80% of the whole dataset as training data and the latter 20% of the whole dataset as test data.
- ⌨ Add scripts to `lr_rawdata.py` to standardize both the training data and test data of horsepower and price using `StandardScaler` as follows:

```
from sklearn.preprocessing import StandardScaler
X_scaler = StandardScaler()
X_train_scaled = X_scaler.fit_transform(X_train)
X_test_scaled = X_scaler.transform(X_test)
```

2.1.4 Linear regression on the preprocessed data

Now we have the cleaned and standardized data without any `NaN` values represented in the pandas dataframe (`df`). Linear regression on the dataframe is similar to what you have done on numpy array (matrix) in the previous programming assignment. For this part, you should read more about the documentation of `pandas`.

- ⌨ Add scripts to `lr_rawdata.py` to build a linear regression model of standardized `price` as the label on standardized `horsepower` (as the only feature) using the standardized training data.
- ⌨ Add scripts to `lr_rawdata.py` to plot the standardized `price` test data versus standardized `horsepower` test data, and the price predictions on the standardized `horsepower` test data versus standardized `horsepower` test data in the same plot using different markers and colors. The label of x axis should be `Standardized horsepower`, the the label of y axis should be `Standardized price` and the title of the figure should be `Linear regression on cleaned and standardized test data`.
- ✍ Copy down the figure to `Assignment2.pdf`.

2.2 Linear regression with multiple features

Now we study linear regression with multiple features, which is also referred to as multiple linear regression. In this part, we only use four features `city-mpg`, `horsepower`, `engine-size` and `peak-rpm`.

- ⌨ Add scripts to `lr_mfeature.py` to standardize the five columns (`city-mpg`, `horsepower`, `engine-size`, `peak-rpm` and `price`) on the cleaned data (You can copy the code from `lr_rawdata.py` to do data preprocessing. For simplicity, we use the whole dataset as the training set and don't split the data this time.). Now the feature matrix $\mathbf{X}$ has four standardized features (`city-mpg`), (`horsepower`), (`engine-size`) and (`peak-rpm`). The standardized label vector $\mathbf{y}$ is the feature `price`.

- **Solve multiple linear regression with normal equation**. The linear regression model is $\mathbf{y} = \mathbf{X}'\boldsymbol{\theta}$. You can get the coefficient $\boldsymbol{\theta}$ by solving the normal equation as follows:

$$\boldsymbol{\theta} = (\mathbf{X}'^{\mathrm{T}}\mathbf{X}')^{-1}\mathbf{X}'^{\mathrm{T}}\mathbf{y} \tag{1}$$

  - ⌨ Add scripts to `lr_mfeature.py` to solve the parameter $\boldsymbol{\theta}$. Here you should add the **unit feature** to $\mathbf{X}$ to get $\mathbf{X}'$, i.e., $\mathbf{X}' = [\mathbf{1}, \mathbf{X}]$. (*Hint*: `dot()`, `inv()` and `transpose()` in `numpy` are useful functions.)
  - ⌨ Add scripts to `lr_mfeature.py` to print out the calculated theta with the following format:

```
Parameter theta calculated by normal equation: #your answer
```

  - ✍ Copy down the above output to `Assignment2.pdf`.

- **Solve multiple linear regression with gradient descent.** When the dataset is large and cannot fit into the memory, you should solve linear regression with gradient descent method.

  - ⌨ Add scripts to `lr_mfeature.py` to use the `SGDRegressor()` in scikit-learn to solve the linear regression of $\mathbf{y}$ versus $\mathbf{X}$. Here you can try using different parameters in `SGDRegressor()`. (*Hint*: you can reuse sample code in Tutorial 3.)
  - ⌨ Add scripts to `lr_mfeature.py` to print out the calculated theta with the following format: (*Hint*: In scikit-learn, $\boldsymbol{\theta}$ is composed of `model.intercept_` and `model.coef_`.)

```
Parameter theta calculated by SGD: #your answer, e.g. (1,2,3,4,5)
```

  - ✍ Copy down this above output to `Assignment2.pdf`.

2.3 Polynomial Regression with Regularization

To study the effect of regularization on linear regression, we first introduce the **polynomial regression**, a special case of linear regression with multiple features. An n-th order **polynomial regression** is given by the following formula:

$$y = \theta_0 + \theta_1 x + \theta_2 x^2 + ... + \theta_n x^n. \tag{2}$$

Given that training data is

```
X_train = [[5.3], [7.2], [10.5], [14.7], [18], [20]]
y_train = [[7.5], [9.1], [13.2], [17.5], [19.3], [19.5]]
```

and the test data is

```
X_test = [[6], [8], [11], [22]]
y_test = [[8.3], [12.5], [15.4], [19.6]]
```

We want to build a polynomial regression model between the above data `X_train` and `y_train`. The problem now is how to choose what order polynomial to fit the data. Let's try different orders of polynomial regression without or with regularization.

2.3.1 Polynomial regression on training data

Now we assume that order of the polynomial is 5, i.e., $y = \theta_0 + \sum_{i=1}^{5} \theta_i x^5$.

- ⌨ Add scripts to `poly_regular.py` get the transformed training and test data with the following scripts.

  ```
  poly = PolynomialFeatures(degree=5) #order 5 feature constructor
  X_train_poly = poly.fit_transform(X_train)
  X_test_poly = poly.transform(X_test)
  ```

- ⌨ Add scripts to `poly_regular.py` do build a linear regression model of `y_train` given `X_train_poly` using the transformed training data.

- ✍ Fill in the blank of the regression equation below

  `y1= __ + __ x + __ x*x + __ x*x*x + + __ x*x*x*x + __ x*x*x*x*x`

  write it down to `Assignment2.pdf`. (*Hint:* Please read scikit-learn document about the variables `intercept_` and `coef_`)

- ⌨ Add scripts to `poly_regular.py` to print the score of the linear regression model on the test data in the following format:

  ```
  Linear regression (order 5) score is: your answer
  ```

  (*Hint:* the "score" for the `model` can be got by `model.score(X_test_poly, y_test)`)

- ✍ Copy down the output above to `Assignment2.pdf`.

- ⌨ Add script to `poly_regular.py` to get predictions for transformed `xx` denoted as `xx_poly` where `xx = np.linspace(0, 26, 100)`. It is as follows:

```
xx = np.linspace(0, 26, 100)
xx_poly = poly.transform(xx.reshape(xx.shape[0], 1))
yy_poly = pr_model.predict(xx_poly)  #order 5 PR model.
```

- ⌨ Add scripts to `poly_regular.py` to plot the predicted output `yy_poly` versus `xx`, and also the test data (`y_test` versus `X_test`) in the same plot. The title of the figure should be `Linear regression (order 5) result`.
- ✍ Copy down the figure to `Assignment2.pdf`.

2.3.2 Ridge Regression (with regularization)

Ridge regression performs "L2 regularization" (L2 means 2-norm), i.e. it adds a factor of sum of squares of coefficients in the optimization objective. The ridge coefficients minimize a penalized residual sum of squares, specifically,

$$\min_{\boldsymbol{\theta}} \|\mathbf{X}\boldsymbol{\theta} - \mathbf{y}\|_2^2 + \alpha \|\boldsymbol{\theta}\|_2^2, \alpha \geq 0. \tag{3}$$

A Ridge regression model can be easily built to do regularized regression if you follow the steps below.

- ⌨ Add the following scripts to `poly_regular.py` to built a ridge regression model of `y_train` on the transformed data `X_train_poly`:

```
ridge_model = Ridge(alpha=1, normalize=False) #build ridge model
ridge_model.fit(X_train_poly, y_train)
```

- ✍ Write down the regression equation in the form of

  `y2= __ + __ x + __ x*x + __ x*x*x + + __ x*x*x*x +__ x*x*x*x*x`

  to `Assignment2.pdf`.
- ⌨ Add scripts to `poly_regular.py` to print the score of the ridge regression model on the transformed test data in the following format:

  `Ridge regression (order 5) score is:`
- ✍ Copy down the output above to `Assignment2.pdf`.
- ⌨ Add scripts to `poly_regular.py` to plot the predicted output `yy_ridge` versus `xx`, and also the test data (`y_test` versus `X_test`) in the same plot. The title of the figure should be `Ridge regression (order 5) result`. Note that

```
yy_ridge = ridge_model.predict(xx_poly) #get predictions for xx_poly
```

- ✍ Copy down the figure to `Assignment2.pdf`.

2.3.3 Comparisons

In this part, we compare the three models: the linear regression (order 1), polynomial regression (order 5), and the ridge regression (order 5).
✍ Write down your answers to the following questions to `Assignment2.pdf`.

- **Q1**: Which model has the highest score?
- **Q2**: Does a larger α result in a larger coefficient for x^5 in the regression equation in Ridge model (order 5)?
- **Optional ungraded exercise.** You can change the parameter `degree` in the function `PolynomialFeatures()` and try to find the best `degree` parameter for polynomial regression and Ridge regression with (`alpha=1`).

3 Linear Discrimination/Classification

In this section, we will use the `LogisticRegression` estimator in scikit-learn to do classification. Logistic regression is a linear model for classification rather than regression.

In `logistic_clf.py`, we generate two clusters of data points with the following script:

```
n_samples = 10000
centers = [(-1, -1), (1, 1)]
X, y = make_blobs(n_samples=n_samples, n_features=2, cluster_std=1.8,
                  centers=centers, shuffle=False, random_state=19)
y[:n_samples // 2] = 0 # half of the points belong to class 0
y[n_samples // 2:] = 1 # the other half belongs to class 1
```

Then we split the data to training and test data, and create a logistic regression estimator with:

```
X_train, X_test, y_train, y_test = train_test_split(X, y, random_state=19)
log_reg = linear_model.LogisticRegression()
```

Note: please do **not** change the parameter `random_state` in the file `logistic_clf.py`.

3.1 Binary Classification

There are two different classes/clusters in the X and y. We want to classify the test data using the logistic regression model that is built on the training data.

- ⌨ Add scripts to `logistic_clf.py` to train/fit the `LogisticRegression` model using `X_train` and `y_train`, and get predictions for the test data `X_test`. *Hint*: remember the estimators have uniform interface with `fit` and `predict` method.

- **Q**: Does the predictions of `X_test` contain values other than `0` or `1`?

  ✍ Write down your answer to `Assignment2.pdf`

⌨ Add scripts to `logistic_clf.py` to plot the data points in `X_test` using the function `scatter()` with different colors for different *predicted* classes. The title of the figure should be `Classification with Logistic Regression`.

- ✍ Copy down the figure to `Assignment2.pdf`.

3.2 Classfication Statistics

- **Q**: How many wrong predictions does the LogisticRegression estimator make on the test data? Compare the predictions on the test data with the ground truth `y_test`.

  ⌨ Add scripts to `logistic_clf.py` to calculate and print out the number of wrong predictions. Your print message should be in the following format:

  ```
  Number of wrong predictions is: your answer here
  ```

  ✍ Copy down the output above to `Assignment2.pdf`.

4 Submission

Instructions for the submission are as follows. **Please follow them carefully.**

1. Make sure you have answered all questions in your report.
2. Test all your Python scripts before submission. Any script that has syntax error will not be marked.
3. Zip all Python script files, i.e., the `*.py` files in `asgn2.zip` (Please do not change the filenames of the scripts.) and your report (`Assignment2.pdf`) into a single zipped file named `<student-id>_asgn2.zip`, where `<student-id>` should be replaced with your own student ID. e.g., `1155012345_asgn2.zip`.
4. Submit the zipped file `<student-id>_asgn2.zip` via Blackboard System no later than 23:59 on Sunday, Mar. 24, 2019.