Practice Session Logistic Regression

G Gasso

- Implement logistic regression on synthetic and real datasets
- Investigate the importance of variables normalization
- Extend to multi-class classification problem
- *Provided codes*: you will require some useful functions provided in the file `utility.py` available on Moodle.

1 Synthetic data

Let consider a binary classification problem with class $y \in \{0, 1\}$. The conditional distribution of each class k, $(k = 0, 1)$, is $p(x/y = k) = \mathcal{N}(\mu_k, \mathbf{S}_k)$ i.e. a gaussian distribution with mean μ_k and covariance matrix $\mathbf{S}_k$.

1. Generate $n_1 = n_2 = 100$ training samples per class using $\mu_1 = \begin{pmatrix} 0 \\ 2 \end{pmatrix}$, $\mathbf{S}_1 = \frac{3}{2}\begin{pmatrix} 1 & 0.1 \\ 0.1 & 1 \end{pmatrix}$, $\mu_2 = \begin{pmatrix} -2 \\ 2 \end{pmatrix}$ and $\mathbf{S}_2 = \frac{7}{2}\begin{pmatrix} 1 & -0.25 \\ -0.25 & 1/2 \end{pmatrix}$

```
from utility import gen_data_twogaussians_2d
import numpy as np
# class 1
n1 = 100
mu1 = np.array([0, 2]); S1 = 1.5*np.array([[1, 0.1], [0.1, 1]])
# class 2
n2 = n1
mu2 = np.array([-2, -2]); S2 = 3.5*np.array([[1, -0.25], [-0.25, 1/2]])
X_train, Y_train = gen_data_twogaussians_2d(mu1, S1, mu2, S2, n1, n2)
```

2. Learn a logistic regression model with no penalty term (`penalty="none"`) and `solver="lbfgs"`. You may refer to the documentation of logistic regression for more details.

```
from sklearn import linear_model
# define the model
clf_reglog = linear_model.LogisticRegression(penalty="none", solver="lbfgs")
# fit the parameters
clf_reglog.fit(X_train, Y_train)
```

Plot the decision frontier (beware that the function `plot_decision_regions_2d` is only valid if the samples x_i yo be plotted are in 2D i.e. $\mathbf{x}_i \in \mathbb{R}^2$)

```
from utility import plot_decision_regions_2d
import matplotlib.pyplot as plt
plot_decision_regions_2d(X_train, Y_train, clf_reglog, resolution=0.02, title="Logistic
    regression")
```

2 Spam classification

We consider a spam classification problem with a dataset containing 4601 e-mails, from which 57 features have been extracted. These are:

- 48 features, in $[0, 100]$, giving the percentage of words in a message which match a given word on a fixed list. The list contains words such as "business", "free"...
- 6 features, in $[0, 100]$, giving the percentage of characters in the e-mail that match a given character among "(", "[", "!", "$", "#"
- Features 55-57 respectively represent the average length, the longest length and the sum of the length of uninterrupted sequence of capital letters

The dataset `spambase.data` and the features name `spambase_variables.csv` are in text format and available on Moodle. The goal is to classify an email as spam (class 1) or not (class 0).

1. Read the files and extract the inputs X and the output Y (last column in the dataset).

```
import numpy as np
dataspam = np.loadtxt("spambase.data", delimiter=",")
features_name = np.genfromtxt("spambase_variables.csv", delimiter=",", dtype="str")
features_name = features_name[:,0]
# get the inputs
X = dataspam[:,0:57]
# extract the output
Y = dataspam[:,-1]
```

2. How many samples do you have in each class (0 or 1)? Show the boxplot of the inputs and comment the graphics.

3. Split the data into training and test sets. The test set size should be 1/3 of the available data.

```
from sklearn.model_selection import train_test_split
...
```

4. The goal is to learn a logistic regression model able to classify properly the incoming e-mails.
   (a) Learn a linear logistic regression model. Let denote `clf_spam` the classifier.

   ```
   clf_spam = ...
   ```

   (b) Compute the accuracy of your model both on training and test sets. Comment the obtained performances.

   ```
   from sklearn.metrics import accuracy_score

   # training set
   acc_train = ...
   print("Accuracy on training data = {}".format(100*acc_train))

   # test set
   acc_test = ...
   print("Accuracy on test data = {}".format(100*acc_test))
   ```

   (c) To see which features are most important, let sort the parameters of the model in increasing order according to their absolute value...

```
# sort in ascending order
index = np.argsort(np.abs(np.squeeze(clf_spam.coef_)))
# reverse the order
index = index[::-1]
```

. . . and print them in that order

```
print("{:<38s}{:<18s}".format("Variables", "Coefficient"))
for var, coef in zip(features_name[index], clf_spam.coef_[0,index]):
  print("{:<28s}{:>16.2f}".format(var, coef))
```

What are the 10 most important features ?

5. We have performed the classification without normalizing the features as commonly prescribed. This may impact the learned model. Normalize the variables and repeat question 4. How do your previous results change? Comment.

```
# centering and scaling
from sklearn.preprocessing import StandardScaler
# define the scaler
sc = StandardScaler(with_mean=True, with_std=True)
...
```

6. Finally we will attempt a third logistic regression model. For this, transform the features using $\log(x_{ij} + \epsilon)$ with $\epsilon = 0.01$.

```
# log transform the features
epsilon = 0.01
X_train_n = np.log(X_train + epsilon)
X_test_n = np.log(X_test+epsilon)
```

Train a logistic regression model using these transformed features and evaluate its performance. Normally you should achieve an accuracy around 95.3%. Can you explain why do we improve the performances?
Hint: you can plot the histograms of the features without and with the log transform.

```
import matplotlib.pyplot as plt
plt.figure(figsize=(18, 18))
vartype = "log" # apply the transform or not
for i in range(X.shape[1]):
  plt.subplot(8,8,i+1)
  if vartype=="log":
    plt.hist(np.log(X[:,i]+epsilon))
  else:
    plt.hist(X[:,i])
  plt.xticks(fontsize=5); plt.yticks(fontsize=6);
```

3 Extension to multi-class classification: digits classification

On Moodle you will find the MNIST training and test data-sets containing respectively 60,000 and 10,000 gray images of dimension 28×28 vectorized into vectors of size 784. The images represent the digits from 0 to 9.
Caution: the datasets are heavy and may result in time consuming load or processing.

1. Load the datasets

```python
import numpy as np
# training set
mnist_train = np.loadtxt("./mnist/mnist-app.csv", delimiter=",")
Y_train = mnist_train[:,-1]
X_train = mnist_train[:, 0:784]
# test set
mnist_test = np.loadtxt("./mnist/mnist-test.csv", delimiter=",")
Y_test = mnist_test[:,-1]
X_test = mnist_test[:, 0:784]
```

2. Visualize some of the digits

```python
import matplotlib.pyplot as plt
fig, ax_array = plt.subplots(6, 6)
axes = ax_array.flatten()
for i, ax in enumerate(axes):
  ax.imshow(X_train[i].reshape(28, 28), cmap="gray_r")
plt.setp(axes, xticks=[], yticks=[], frame_on=False)
plt.tight_layout(h_pad=0.5, w_pad=0.01)
```

3. Perform a classification by the logistic regression method trained with a ℓ_2 penalty and $C = 1$

```python
from sklearn import linear_model
clf = linear_model.LogisticRegression(tol=1e-2, multi_class="multinomial", solver="lbfgs",C=1,
    max_iter=2500)
```

Remind the fundamentals of machine learning: data normalization; we calculate the model on the training set; we select possible hyper-parameters on the validation data; we test the final model on the test set.

Compute the test error rate of your model. How do you compare to the leader-board on Mnist?