Exercise 1

Deadline: 04.05.2016, 2:00 pm
Procedure for the exercises:

- You will work on the exercises in groups of 2-3 people (due to your large group size I will unfortunately not be able to correct single submission)
- Please send your solutions to `mlhd1516@gmail.com` in a single zip file containing (*well documented*) code, results and plots in the following format:
  Subject line: *"[Ex01] Ghahramani, Hinton, Koller"*
  Attachement: `ex01_ghahramani_hinton_koller.zip`
  (If your group were to consist of Zoubin Ghahramani, Geoffrey Hinton and Daphne Koller)

Please note, that there won't be exercise groups this Friday (29.04) and next week (06.05), but we will have an exercise group instead of the lecture next Wednesday (04.05).

This exercise focuses on training a binary classifier on handwritten numbers from the scikit-learn digits dataset (similar to the MNIST datatset). You will implement optimization methods for minimizing the logistic regression objective function and compare their convergence rates.

1 Reminder: Logistic Regression

In the lecture you have used logistic regression as a binary classifier to assign a label $y_i \in \{-1, 1\}$ for a sample $X_i \in \mathbb{R}^D$ by

$$\hat{y} = \text{argmax}_{k \in \{-1,1\}} P(y = k | X_i, \beta) = \begin{cases} 1 & \sigma(X_i \beta) > 0.5 \\ -1 & \sigma(X_i \beta) \leq 0.5 \end{cases}$$

with the parameter $\beta \in \mathbb{R}^D$ and σ the sigmoid function (see Section 2.1). In the following we will apply different training methods to find the optimal β.

2 Loading the Dataset (2 points)

The following lines show how to load the digits dataset from sklearn and how to extract the necessary data:

```
from sklearn.datasets import load_digits

digits = load_digits()

print digits.keys()

data         = digits["data"]
images       = digits["images"]
target       = digits["target"]
target_names = digits["target_names"]

import numpy as np
print np.dtype(data)
```

Note that data is a vectorized version of the images.

We want restrict the dataset to two similar looking numbers to form a binary classification problem. Therefore, create the feature Matrix X by slicing dataset such that it only contains the feature

vectors X_i where the target[i] is 3 or 8. Create a matching groundtruth vector with

$$y_i = \begin{cases} 1 & \text{target}[i] = 3 \\ -1 & \text{target}[i] = 8 \end{cases}$$

The slicing reduces the dataset to N samples and you should have the matching numpy arrays $X \in \mathbb{R}^{N \times D}$ and $y \in \mathbb{R}^N$ with $D = 64$ and $N = 357$.

2.1 Basic Functions(8 Points)

For the training of our classifier we require some basic functions. Please implement:

- sigmoid(Z) that applies the sigmoid function to every element of the array Z with

$$\sigma(z) = \frac{1}{1 + e^{-z}}.$$

- gradient(β, X, y) that returns the normalized gradient of the loss function.

$$\nabla\text{loss} = \frac{1}{M} \sum_{i=0}^{M} \nabla\text{loss}_i = \frac{1}{M} \sum_{i=0}^{M} \left(1 - \sigma(y_i X_i \beta)\right) \left(-y_i X_i^T\right)$$

  where $M = \text{len}(y)$
- predict(β, X) that returns a vector with one class label y_i for each row X_i in X.
- zero_one_loss($\text{y}_{\text{prediction}}$, $\text{y}_{\text{groundtruth}}$) that counts the number of wrongly classified samples

Hint:

Numpy functions such as `numpy.exp(X)` are applied elementwise. To increase the speed of your code you should avoid unnecessary for loops. Visit `http://ufldl.stanford.edu/wiki/index.php/Logistic_Regression_Vectorization_Example` for a suggestion on how to use Matrix multiplication in order to calculate the gradient.

2.2 Optimization Method(20 Points)

Use the functions from section 2.1 to implement the following optimization methods(without regularization):

gradient descent	$\beta^{t+1} = \beta^t - \tau \nabla \text{loss}^t$
stochastic gradient descent	$\beta^{t+1} = \beta^t - \tau_t \nabla \text{loss}_i^t$
SG minibatch	$\beta^{t+1} = \beta^t - \frac{\tau_t}{\lvert B \rvert} \sum_{i \in B} \nabla \text{loss}_i^t$
SG momentum	$g^{t+1} = \mu g^t + (1-\mu) \nabla \text{loss}_i^t$
	$\beta^{t+1} = \beta^t - \tau_t g^t$
average stochastic gradient	$g^{t+1} = g^t - \tau_t \left(1 - \sigma(y_i X_i g^t)\right)(-y_i X_i^T)$
	$\beta^{t+1} = (1-\mu)\beta^t + \mu g^{t+1}$
stochastic average gradient	$d^0 = 0, \forall i' : d_{i'}^0 = 0$
	$\forall i' : d_{i'}^{t+1} = \begin{cases} \nabla \text{loss}_{i'}^t & i' = i \\ d_{i'}^t & \text{otherwise} \end{cases}$
	$d^{t+1} = \frac{1}{N} \sum_{i''} d_{i''}^{t+1} = d^t + \frac{1}{N}(d_i^{t+1} - d_i^t)$
	$\beta^{t+1} = \beta^t - \tau_t d^{t+1}$
Dual Coordinate Ascent	$\alpha^0 = \text{random}(0,1)$
	$\beta^0 = \sum_{i'} \alpha_{i'} y_{i'} X_{i'}^T$
	$\alpha_i^{t+1} = \underbrace{\alpha_i^t - \frac{y_i X_i \beta^t}{X_i X_i^T}}_{\text{clip to [0,1]}}$
	$\beta^{t+1} = \beta^t + (\alpha_i^{t+1} - \alpha_i^t) y_i X_i^T$
Weighted Least Squares	$z^t = X\beta^t$
	$V_{i'i'}^t = \sqrt{\frac{1}{N}\sigma(z_{i'}^t)(1-\sigma(z_{i'}^t))}$ (V^t is an $N \times N$ diagonal matrix)
	$\forall i' : \tilde{y}_{i'}^t = \frac{y_{i'}}{\sigma(y_{i'} z_{i'}^t)}$
	$\tilde{z}^t = (z^t + \tilde{y}^t) V^t$
	$\tilde{X}^t = X V^t$
	$\beta^{t+1} = \text{argmin}_\beta (\tilde{z}^t - \tilde{X}^t \beta)^2$

For each method define a function that makes m update steps and returns β^m. The sample i and the batch B should be drawn randomly from the whole dataset in every iteration. In contrast, i' runs over all instances. Please choose to either sample with or without replacement and give a (one sentence) explanation why. Note that stochastic gradient methods need a reducing learning rate in order to converge. Use the scaling

$$\tau_t = \frac{\tau_0}{1 + \gamma t}$$

for all stochastic gradient descent methods other than average stochastic gradient where

$$\tau_t = \frac{\tau_0}{(1 + \gamma t)^{\frac{3}{4}}}$$

is proven to give a faster convergence. Please make the X, y, β^0, τ, τ_0, γ and μ available as function parameters.

Hint:

You can compute the gradient $\sum_{i \in B} \nabla \text{loss}_i$ for a subset B of samples simply by slicing the input matrix X and y of gradient(β, X, y). Use `numpy.clip` for the dual coordinate ascent.

2.3 Comparison(15 Points)

For our analysis split the data into a training and test with

```
from sklearn import cross_validation
X, X_test, y, y_test = cross_validation.train_test_split(X, y, test_size=0.3,
    random_state=0)
```

For every run initialize $\beta^0 = 0$ and $g^0 = 0$. The initialization of the dual coordinate ascent needs to be different from zero. Here you can use random numbers: α^0 = np.random.uniform(size=N)

Learning Rate

Use the cross validation tool from sklearn to find a learning rate and free parameters of each optimization algorithm. You can generate multiple training- and testsets with

```
from sklearn import cross_validation
kf = cross_validation.KFold(y.shape[0], n_folds=10)
for train_index, validation_index in kf:
    X_train, X_validation = X[train_index], X[validation_index]
    y_train, y_validation = y[train_index], y[validation_index]
```

For the cross validation set the parameters sequentially to the following values:

τ or τ_0	0.001	0.01	0.1
μ	0.1	0.2	0.5
γ	0.0001	0.001	0.01

Use 150 iterations for all stochastic methods and 10 otherwise. Calculate the total error by summing over the zero_one_loss(y_validation, predict(β, X_validation)), where β was trained on (X_train,y_train). Finally, list the parameters with the lowest total error.

Speed

We would like to compare the convergence speeds for the algorithms. Unfortunately we can not simply use a timing tool since this is highly dependent on the optimization of each algorithms. Alternatively, we can compute estimates T of the ideal times by multiplying the number of iterations with the computational complexity of one update step. The complexities per iteration are $O(ND)$ for gradient descent, $O(|B|D)$ for SG minibatch and $O(D)$ for all other stochastic processes and $O(ND^2)$ for Weighted Least Squares. Make one run for each method (with the parameters found in 2.3) while saving T, training- and test error in every operation. Create a graph where you plot both errors over T. You may want to use logarithmic axes. Which of the algorithms is the fastest to converge? Are there qualitative differences between methods?

Regulations

Please hand in the python code, figures and explanations (describing clearly which belongs to which). Non-trivial sections of your code should be explained with short comments, and variables should have self-explanatory names. Plots should have informative axis labels, legends and captions. Please enclose all results into a single .pdf document and hand in the .py files that created the results. Please email the solutions to `mlhd1516@gmail.com` before the deadline. You may hand in the exercises in

teams of maximally three people, which must be clearly named on the solution sheet (one email is sufficient). Discussions between different teams about the exercises are encouraged, but the code must not be copied verbatim (the same holds for any implementations which may be available on the WWW). Please respect particularly this rule, otherwise we cannot give you a passing grade. If you have 50% or more points in the end of the semester you will be allowed to participate in the final project.