Machine Learning for Robotics SS 2017
Instructor: Georg Martius <georg.martius@tuebingen.mpg.de>
Due-date: 18.05.2017 (hand in at beginning of recitation session)
Exercise Sheet 4
May 11, 2017

1 Multiclass Logistic Regression

(a) Logistic regression for binary classification with $y \in \mathcal{Y} = \{-1, +1\}$ we use the following model for the class probabilities:

$$\hat{p}(y|x; w) = \frac{1}{1 + \exp(-y\langle w, x\rangle)}, \tag{1}$$

Proof that $\hat{p}(y|x; w)$ is a well defined probability density w.r.t. y for any $w \in \mathbb{R}^d$.

(b) For $y \in \{0, 1\}$ we typically model only $p(y = 1|x; w)$ and defining $p(y = 0|x, w) = 1 - p(y = 1|x, w)$. The cost function becomes

$$\mathcal{L}(w) = \sum_{i=1}^{n} (-y \log(\hat{p}(y|x_i; w)) - (1 - y) \log(1 - p(y|x_i; w))) \,. \tag{2}$$

Show how this comes about.

For many classes we use a for each class $k = 1 \ldots K$ one parameter w_k output presenting $p(y = k|x, w)$ and model this by the *softmax*:

$$\hat{p}(y = k|x, W) = s_k(x, w) = \frac{\exp(\langle w_y, x\rangle)}{\sum_{j=1}^{M} \exp(\langle w_j, x\rangle)} \quad \text{for } y = 1, \ldots, M, \tag{3}$$

. We use a 1-of-K coding (also called one-hot representation) where the target class for sample i is encoded by a vector t_i which is a vector of K zeros except for the kth element (if sample i has class label k). The cost function is

$$\mathcal{L}(W) = -\sum_{i=1}^{n} \sum_{k=1}^{K} t_{ik} \log s_k(x^i, W) \tag{4}$$

which is called the *cross entropy* cost function.

(a) Show that for the of the softmax $s_k(z) = \frac{\exp(z_k)}{\sum_{j=1} \exp(z_j)}$ we have the following derivative: $\frac{\partial s_k(z)}{\partial z_j} = s_k(z)(I_{kj} - s_j(z))$

(b) Calculate the derivative of (4) w.r.t. W.
Solution: $\Delta_{w_j} L(w_j) = \sum_i (s_j(x^i) - t_{ij}) x^i$

2 Train an SVM and Logistic Regression Classifier

Here we want to train different SVMs and multiclass Logistic Regression Classifier on a digit recognition task. The data for this is part of the *sklearn* python package. You find a skeleton file on the Dropbox:

```
import numpy as np
import matplotlib.pyplot as plt
%matplotlib inline

# Import datasets, classifiers and performance metrics
from sklearn import datasets, svm, metrics

# The digits dataset
digits = datasets.load_digits()

# The data that we are interested in is made of 8x8 images of digits, let's
# have a look at the first 8 images, stored in the `images` attribute of the
# dataset.
images_and_labels = list(zip(digits.images, digits.target))
for index, (image, label) in enumerate(images_and_labels[:10]):
    plt.subplot(2, 5, index + 1)
    plt.axis('off')
    plt.imshow(image, cmap=plt.cm.gray_r, interpolation='nearest')
    plt.title('Training: %i' % label)
```

The dataset has 1797 points. Use half for training and a quarter for validation and testing respectively.

(a) Train a linear SVM and a SVM with rbf kernel and visualize the output, see module `svm.SVC`. Visualize some of the mistakes.

(b) For the rbf case there is the kernel size: gamma. Try 0.001 but also perform model selection.

(c) Implement a Logistic regression classifier (use the derivatives you calculated above) and apply it to the data. Here a litte hint:

```
def softmax(x):
  if x.ndim == 1: # for 1d input
      e = np.exp(x - np.max(x))  # prevent overflow
      return e / (np.sum(e, axis=0))
  else:  # for x having the shape: (samples,classes)
      e = np.exp(x - np.max(x,axis=1,keepdims=True))
      return e / (np.array([np.sum(e, axis=1)]).T)  # ndim = 2
```

For numeric reasons it is useful to shift the argument to exp, such that they are all smaller than 0.

Use gradient decent with a decreasing learning rate every step by lr=lr*0.999. Hint: the x values of the images are pixel values in [0,32] or so. Either scale them down to [0-1] or start with a small learning rate to avoid divergence.

(d) compare the performance

(e) Try to with some feature map of your choice, can you improve?