Lab Class ML:VI

Until **Wednesday, Jan. 22nd, 2020, 23:59 pm CET**, the following exercises must be submitted:

Students receiving 4.5 ECTS: 2, and 3
Students receiving 6 ECTS: 1, 2, and 3

Lab Class General Instructions. Exercises should be completed in groups of three students; which and how many exercises you must submit depends on the number of ECTS credits awarded for the course, based on which degree programme you're studying:

	4.5 ECTS	6 ECTS
Degree programmes	CS4DM, CSM, MI, HCI*	DE, HCI*
	*(HCI students can choose number of credits in electives module)	
For admission to the final exam:		
Must reach at least	80 points	120 points
Out of maximum	120 points	180 points

For each exercise sheet, all required exercises must be submitted until the stated deadline. Each individual exercise will receive a score between zero and ten. Submissions received after the deadline will not be considered for grading. In order to be admitted to the final exams, you must reach the minimum total score shown.
Upload your solutions to Moodle as a single `.pdf` file, plus one `.ipynb` file for any programming exercises. Make it clear which solution corresponds to which exercise. Programming exercises are marked with [P] – their solutions must be documented extensively with docstrings and/or inline comments. ml.weimar.webis.de

Exercise 1 : Gradient Descent

(a) What are the differences between the perceptron training rule and the gradient descent method?

(b) What are the requirements for gradient descent being successful as a learning algorithm?

(c) What are the differences between batch gradient descent and incremental gradient descent?

Exercise 2 : Perceptron Learning

Solve the following problems on learning boolean functions with perceptrons. Use the values 0 for *false* and 1 for *true*, and the threshold function $\varphi(x) = \mathsf{max}(sign(x), 0)$.

(a) Design a single perceptron with two inputs x_A and x_B. This perceptron shall implement the boolean formula $A \wedge \neg B$ with a suitable function $y(x_A, x_B)$. Hint: to start, determine the training data and draw it, and a suitable decision boundary, in a coordinate system; then, determine a set of suitable weights $\mathbf{w} = (w_0, w_1, w_2)$.

(b) Instead of directly computing w, train the perceptron with two iterations of the batch gradient descent algorithm computed by hand, with a learning rate η of 0.1 and the weights initialized with $w_0 = -0.5$ and $w_1 = w_2 = 0.5$. Use the following examples in the given order:

x_1	x_2	$c(\mathbf{x})$
0	0	0
0	1	0
1	0	1
1	1	0

(c) Why can the boolean formula A *XOR* B not be learned by a single perceptron? Justify your answer with a drawing.

(d) Design a two-layer perceptron which implements the boolean formula A *XOR* B.

Hint: decompose XOR into simpler boolean functions.

Exercise 3 : P Perceptron Training

Your task is to implement perceptron learning for a $p = 2$-dimensional feature space. Our dataset D with $n = 300$ points will be defined as a tuple $(X, \mathbf{c})$ where X is a n-by-3 matrix with the first column equal to 1, and the second and third columns corresponding to the x- and y-coordinates of the points; $\mathbf{c}$ is an n-by-1 vector with $c_i \in \{-1, 1\}$.

The following Python function generates the dataset D:

```
def separable_dataset():
    from sklearn.datasets import make_classification
    X, C = make_classification(300, 2, 2, 0, 0, 2, 1,
                               class_sep=30, flip_y=0,
                               hypercube=False, random_state=2)
    X = np.hstack([np.ones((X.shape[0], 1)), X])
    C = 2*C - 1
    D = (X, C)
    return D
```

(a) Implement the helper function `random_select(D)` that takes as input a dataset $D = (X, \mathbf{c})$ such as the one returned by the function above, and returns a tuple $(x_i, c(x_i))$ by selecting the same random row i from both X and C. Consider also the following simple test case:

```
>>> random_select(([[1,0,0]], [-1]))
([1, 0, 0], -1)
```

(b) Implement the helper function `convergence(D, w)` that takes a dataset D, and a 3-element vector w, and returns `True` if and only if a perceptron with weight vector w classifies every point from D correctly. A basic test case:

```
>>> X = np.array([[1, 0, 0]])
>>> C = np.array([1])
>>> w = np.array([1, 1, 1])
>>> convergence(D=(X, C), w=w)
True
>>> convergence(D=(X, C), w=-w)
False
```

Hint: since in this exercise, $c \in \{-1, 1\}$, we must use the sign function (`numpy.sign()`) instead of the Heaviside function.

(c) Following the description of the PT algorithm from the lecture, implement the function `pt(D, eta, tmax=300)` that trains a perceptron on the dataset D with learning rate `eta` for a maximum of `tmax` iterations (with a default maximum iteration number of 300), and returns the 3-element weight vector w.

Hint: again, use the sign function in place of Heaviside.

(d) Implement a function `misclass_rate(D, w)` that computes the misclassification rate for a perceptron with weight vector w on dataset D. Verify that your implementation achieves a misclassification rate of zero on the seperable dataset defined above:

```
>>> D = separable_dataset()
>>> w = pt(D, eta=0.1)
>>> misclass_rate(D, w)
0.0
```

(e) Replace the dataset D with one generated by the following function:

```
def nonseparable_dataset():
    from sklearn.datasets import make_classification
    X, C =  make_classification(300, 2, 2, 0, 0, 2, 1,
                                class_sep=10, flip_y=0.1,
                                hypercube=False, random_state=2)
    X = np.hstack([np.ones((X.shape[0], 1)), X])
    C = 2*C - 1
    return X, C
```

Run your implementation of the PT algorithm on this new dataset 1000 times, with a maximum of 300 iterations each time. For each of the 1000 runs, record the misclassification rate, and examine the results (minimum, mean, and maximum misclassification rate).

(f) Implement the batch gradient descent algorithm from the lecture as a function `bgd(D, eta, tmax=300)`, and repeat the previous experiment with 1000 runs of BDG instead of PT. Compare the results.

Hint: use a smaller learning rate for BGD.