Computational Intelligence SS20
Homework 4
Support Vector Machine, Kernels & Multiclass classification

Ceca Kraisnikovic
Horst Petschenig

Tutor:	Valerian Drack, drack@student.tugraz.at
Points to achieve:	20 pts
Extra points:	0* pts
Info hour:	03.06.2020 Cisco WebEx Meeting, see TeachCenter
Deadline:	09.06.2020 23:55
Hand-in procedure:	Use the **cover sheet** available in TeachCenter. Submit your **python files and pdf report** in TeachCenter.
Course info:	TeachCenter

Contents

General remarks

Your submission will be graded based on...

- The correctness of your results (Is your code doing what it should be doing? Are your plots consistent with what algorithm XY should produce for the given task? Is your derivation of formula XY correct?)
- The depth and correctness of your interpretations (Keep your interpretations as short as possible, but as long as necessary to convey your ideas.)
- The quality of your plots (Is everything important clearly visible in the report, are axes labeled, ...?)
- **Every** result that should be graded must be included in your report.
- INOTE is an implementation-related note.

We use scikit-learn's implementation of SVM throughout with the `SVC` class. (online documentation available here and here)

1 Linear SVM [5 points]

The file `data.json` contains the training set (X, Y) of a binary classification problem with two input dimensions (X is the set of 2-dimensional points, Y are labels). Your task is to use a linear soft-margin SVM to solve this classification problem and to inspect how its parameter C, the positive cost factor that penalizes misclassified training examples (slack variables), influences the decision boundary found by SVM.

In function `ex_1` in file `svm_main.py` the data points are first loaded from the data file and plotted so that we can observe the structure of the data. Do the following:

a) Fill in function `ex_1_a` in file `svm.py` to train an SVM with a linear kernel with the provided training data 'x' and 'y'. Plot the training points, decision boundary and support vectors using function `plot_svm_decision_boundary` in file `svm_plot.py` (passing in only 'x' and 'y' as parameters 'x_train' and 'y_train').

INOTE Set the `kernel` parameter to 'linear' to train an SVM with a linear kernel.

b) In function `ex_1_b` in file `svm.py` add an additional data point (4,0) with label 1 to the data. Train a linear SVM again on this extended data. Plot results using `plot_svm_decision_boundary`.

INOTE Since 'x' and 'y' are NumPy arrays, use NumPy's vstack function to add a point to 'x' and hstack to add the label to 'y'.

c) In function `ex_1_c` in file `svm.py`, again extend the dataset as above, and train linear SVMs with different values of the parameter C: 10^6, 10^0, 10^{-1}, and 10^{-3}. Plot the decision boundaries and support vectors for each of these values of C.

INOTE Use the parameter `C` in the SVC class to set the value of C. The default value of this parameter is 1.0.

In your report:

- Include plots of all the results.
- For task b), discuss how and why the decision boundary changed when the new point was added.
- For task c):
  - Report how the parameter C influences the decision boundary found by the SVM.
  - How does the number of support vectors found by the SVM change with the value of C? Why?

2 Nonlinear (kernel) SVM [5 points]

The file `data_nl.json` contains the training (X, Y) and test (XT, YT) sets of a binary classification problem with two input dimensions. The goal of this task is to use different kernels (linear, polynomial and RBF) in combination with SVM in order to solve this nonlinear classification problem and to inspect the influence of kernel parameters on the classification error.

In function `ex_2` in file `svm_main.py` the data points are first loaded from the data file and plotted so that we can observe the non-linear structure of the data. The training points are plotted in dark colors, and the testing points in lighter colors.

We consider the following kernels:

- Linear kernel: $K(\boldsymbol{x}, \boldsymbol{y}) = \boldsymbol{x}^T \boldsymbol{y}$
- Polynomial (non-homogeneous) kernel: $K(\boldsymbol{x}, \boldsymbol{y}) = (\boldsymbol{x}^T \boldsymbol{y} + r)^d$
- RBF kernel: $K(\boldsymbol{x}, \boldsymbol{y}) = exp(-\gamma ||\boldsymbol{x} - \boldsymbol{y}||^2)$ where $\gamma = \frac{1}{2\sigma^2}$

Do the following:

a) Fill in function `ex_2_a` in file `svm.py` to train an SVM with a linear kernel with the provided training data 'x_train' and 'y_train'. Plot the training points, decision boundary and support vectors using function `plot_svm_decision_boundary` in file `svm_plot.py`. Calculate the test score (using the provided test data 'x_test' and 'y_test').

INOTE Pass in all the parameters 'x_train', 'y_train', 'x_test', 'y_test' to `plot_svm_decision_boundary`.

INOTE Use the `SVC`'s `score` method to calculate the test score. It returns the mean accuracy of classification on the given dataset.

b) In function `ex_2_b` in file `svm.py` train and test SVMs with a polynomial kernel for different degrees of the polynomial $d = 1, 2, \ldots, 19, 20$ (Setting the value of r to 1). Plot the variation of training and testing score with polynomial degree using function `plot_score_vs_degree` in file `svm_plot.py`. Plot the decision boundary and support vectors for the kernel with the highest test score using `plot_svm_decision_boundary`.

INOTE In the SVC class, set the `kernel` parameter to 'poly' to use a polynomial kernel, set the degree of the polynomial using the `degree` parameter, and set the value of r using the `coef0` parameter.

c) Fill in function `ex_2_c` in file `svm.py` to train and test SVMs with an RBF kernel with values of $\gamma = 0.01, 0.03, \ldots, 1.97, 1.99$. Plot the variation of training and testing score with the value of γ using the function `plot_score_vs_gamma` in file `svm_plot.py`. Plot the decision boundary and support vectors for the kernel with the highest test score using `plot_svm_decision_boundary`.

INOTE In the SVC class, set the `kernel` parameter to 'rbf' to use an RBF kernel; set the value of γ using the `gamma` parameter.

In your report:

- Include plots of all the results.
- For task a), report the test score achieved.
- For task b), which degree of the polynomial produces the highest test score (accuracy)? Report this test score.
- For task c), which value of gamma produces the highest test score (accuracy)? Report this test score.
- Compare results obtained by each of these three kernels:
  - Which of the considered kernels performs best and why?
  - Compare the complexity of decision boundaries and the number of Support Vectors found.
  - Which kernel generalizes best for the given dataset?

3 Multiclass classification [5 points]

For this task you will use a linear SVM as a binary classifier to solve a multiclass problem. For this purpose use the data file `data_mnist.json` which contains training (X, Y) and test (XT, YT) sets of handwritten digits. To simplify the problem we consider only the first 5 digits: $1, \ldots, 5$. Each data sample in X (a handwritten digit) is an image of 28×28 pixels. Note that here the labels are not binary, but rather each data sample belongs to one of the 5 classes. Load the data file and plot several data samples to see handwritten digits (use `plot_mnist`). The goal of this task is to use the one-vs-all approach with SVM for multiclass classification in order to recognize digits. In all simulations of this task use the parameter $C = 10$.

In function `ex_3` in file `svm_main.py` the data points are first loaded from the data file and plotted so that we can observe the structure of the MNIST dataset. Do the following:

a) In function `ex_3_a` in the file `svm.py`, use the scikit learn implementation of one-versus-rest multi-class classification. This is done by specifying the argument `decision_function_shape='ovr'` when creating the classifier object. Train the classifier on the MNIST dataset for a LINEAR kernel and an RBF kernel with gamma ranging from 10^{-5} to 10^5 (more precisely, 10^i, $i = -5, -4, ..., 4, 5$). Plot the results with `plot_score_vs_gamma`, and specify the score obtained with a linear kernel using the optional argument `lin_score_train`. Note that the chance level has changed for this example.

b) In function `ex_3_b` in the file `svm.py`, train a multi-class SVC with a linear kernel. Use the function `confusion_matrix` from scikit-learn to get the confusion matrix. Plot the confusion matrix with `plot_confusion_matrix`. Plot the first 10 images from the test set that misclassified (classified as the most misclassified digit) using `plot_mnist`. For example, if digit 1 is the most misclassified digit, plot the first 10 images misclassified **as** 1 in the test set.

NOTE: Rows of the confusion matrix represent true classes, columns represent predicted classes. If we want to plot digits that are misclassified **as** digit, for example, 1, we are plotting examples that are counted in the corresponding **column** of digit 1 in this particular case (all elements of the column except the element that is on the main diagonal).

In your report:

- Recall the algorithms 'One-versus-Rest' (or versus-all) and 'One-versus-one' multi-class classification procedures. How many binary classifiers need to be trained in both cases?
- Include plots for `ex_3_a` with the scores of a linear and an RBF kernel.
- Discuss those results. In particular, why does a linear kernel perform well on images?
- Find the digit class for which you get the highest error rate (the most misclassified digit). Report the numbers from the corresponding row of the confusion matrix, and explain what they represent.
- Include plots for `ex_3_b` of the confusion matrix and the first 10 images from the test set that are classified as the most misclassified digit.
- With the help of these two plots, discuss why the classifier is doing these mistakes.

4 SVM with Gradient Descent [5 points]

Instead of relying on the pre-built `SVC` classifier from scikit-learn, you will implement a linear support vector machine classifier using GD yourself. When we optimize the parameters in the primal formulation using GD, we cannot make use of the kernel trick (directly). The objective of SVM can be written as:

$$\min_{\mathbf{w},b} J(\mathbf{w}, b) = \min_{\mathbf{w},b} \frac{1}{2} \|\mathbf{w}\|^2 + \frac{C}{m} \sum_{i=1}^{m} \max(0, 1 - y_i(\mathbf{w}^T \mathbf{x}_i + b)), \tag{1}$$

where $\mathbf{x}_i \in \mathbb{R}^p$ are training examples, $y_i \in -1, 1$ are binary class labels, $\mathbf{w} \in \mathbb{R}^p$ is the parameter vector, $b \in \mathbb{R}$ is the bias and C is the penalty term. In this simple example, we will not make use of slack variables. The gradient of this function is:

$$\nabla J_{\mathbf{w}} = \mathbf{w} - \frac{C}{m} \sum_{i=1}^{m} \mathbb{I}_i y_i \mathbf{x}_i \tag{2}$$

$$\frac{dJ}{db} = -\frac{C}{m} \sum_{i=1}^{m} \mathbb{I}_i y_i \tag{3}$$

$$\mathbb{I}_i = \begin{cases} 0 & \text{if } 1 - y_i(\mathbf{w}^T \mathbf{x}_i + b) \leq 0 \\ 1 & \text{otherwise.} \end{cases} \tag{4}$$

Fill the following blocks in the code you are provided:

- In `gd.py` implement the cost funtion (Equ. 1) as the python function `cost`.
- In `gd.py` implement the gradient of the cost with respect to $\mathbf{w}$ and b (Equ. 2, 3, 4) as the python function `grad`.
- In `gd.py` implement a generic gradient descent solver as the function `gradient_descent`.

The dataset used in this task is again `data.json`, as in Task 1. For training, use 80% of the dataset; for testing, use the remaining 20%. To calculate the predictions on the test set, use the following:

$$\hat{y}_i = \begin{cases} 1 & \text{if } \mathbf{w}^T \mathbf{x}_i + b \geq 0 \\ -1 & \text{otherwise.} \end{cases} \tag{5}$$

Choose an appropriate learning rate and the maximum number of iterations. Set the penalty term $C = 1$. Initialize $\mathbf{w}$ and b to zero. In your report:

- Report the optimal parameter $\theta^* = (\mathbf{w}, b)$ found and the final value of the cost function.
- Report the accuracy obtained on the test set.
- Use the function `plot_decision_function` in `svm_plot.py` to plot the decision function of your trained support vector machine. Compare it to the plot from Task 1.
- What is the main drawback of minimizing the support vector machine given in the form of Equ. 1?