Machine Learning 1, SS23
Homework 3
k-NN, SVM, Decision Trees

Horst Petschenig

Tutor:	Moritz Erlacher, moritz.erlacher@student.tugraz.at
Points to achieve:	25 pts
Info hour:	Will be announced via TeachCenter
Deadline:	07.06.2023 23:55
Hand-in procedure:	Use the **cover sheet** available in TeachCenter. Submit your **python files and pdf report** in TeachCenter. Do not zip the files. Do not upload the data folder
Submissions after the deadline:	Each missed day results in a (-5) points penalty.
Plagiarism:	If detected, all parties involved will receive 0 points. If this happens twice, the course will be graded as failed.
Course info:	TeachCenter

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, ...?) All plots have to be included in your report!
- **Every** result that should be graded must be included in your report.
- Your submission should run with Python 3.5+.

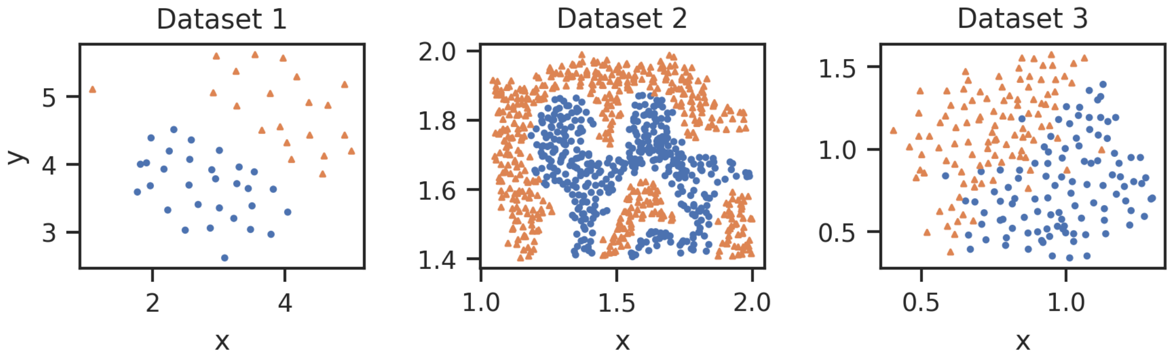

Figure 1: Dataset 1 would be linearly separable if it wasn't for this pesky little outlier at the bottom right. Dataset 2 is highly nonlinear but noise-free. Dataset 3 exhibits a large overlap of the two classes.

In this assignment we will evaluate a variety of common classification algorithms: k-nearest neighbors, support vector machines, random forests and bagging ensembles. To do so, we are going to test these classifiers on three simple toy datasets for binary classification; see Figure 1 for an overview.

> **Implementation note:** The code skeleton contains additional instructions, hints and references to documentation for all tasks described below. Carefully read the instructions and the comments in the code. When asked to perform grid search: You are advised to use `GridSearchCV` with parameters exponentially spaced apart, i.e. `'C': 10.**np.arange(-3, 4)`.

1 k-Nearest Neighbors [7 points]

There are many different ways to classify machine learning algorithms. One of the most important distinctions is the following: Does your model have a fixed number of trainable parameters or does the number of parameters change with the number of available training examples? If the number of parameters is fixed, it is called a parametric model (e.g. neural networks, . . .). One example of a non-parametric classifier is the k-Nearest Neighbor classifier. It just looks at the k points in the training set that are nearest to the test input $\boldsymbol{x}$ and counts how many members of each class are in this set. Based on a majority vote, the class membership is determined.

Tasks:

1. Implement the k-nearest neighbors algorithm. Compute the Euclidean distance between a test point and all points in the training dataset. Based on the class labels of the k closest training points, determine the class membership for the test input.

2. Apply the k-NN algorithm to the three datasets. Automatically determine the best value for k using 5-fold cross-validation using a grid search for different values of $k \in [1, \dots, 100]$. Plot the training and validation accuracy for varying values of k and report the performance of the classifier with your chosen value of k on the test set as well as the decision boundaries.

3. Briefly discuss the effect of low or high values of k. What happens when you set $k = 1$? Plot the decision boundaries for $k \in \{1, 20, 100\}$ for the noisy variant of dataset 2. Automatically determine the best value for k using 5-fold cross-validation using a grid search for different values of $k \in [1, \dots, 100]$. Plot the training and validation accuracy for varying values of k and report the performance of the classifier with your chosen value of k on the test set.

2 Support Vector Machines [13.5 points]

Linear models of the form $y(\boldsymbol{x}) = \boldsymbol{w}^T\phi(\boldsymbol{x}) + b$ where $\phi(\boldsymbol{x}) = [\kappa(\boldsymbol{x}, \boldsymbol{x}_1), \dots, \kappa(\boldsymbol{x}, \boldsymbol{x}_N)]$ denotes a fixed feature-space transformation have been discussed in the first assignment and can be readily solved using methods such as the least squares approach. For example, consider the polynomial kernel $\kappa(\boldsymbol{x}, \boldsymbol{x}') = (\gamma \boldsymbol{x}^T \boldsymbol{x}' + r)^M$ with $r > 0$. With $M = 2, \gamma = 1, r = 1$ and $\boldsymbol{x}, \boldsymbol{x}' \in \mathbb{R}^2$ we have

$$\kappa(\boldsymbol{x}, \boldsymbol{x}') = (1 + \boldsymbol{x}^T \boldsymbol{x}')^2 = (1 + x_1 x_1' + x_2 x_2')^2.$$

This can be written as $\psi(\boldsymbol{x})^T \psi(\boldsymbol{x}')$ where

$$\psi(\boldsymbol{x}) = \left[1, \sqrt{2}x_1, \sqrt{2}x_2, x_1^2, x_2^2, \sqrt{2}x_1 x_2\right]^T.$$

However, some kernels such as the popular Gaussian kernel

$$\kappa(\boldsymbol{x}, \boldsymbol{x}') = \exp\left(-\frac{1}{2}(\boldsymbol{x} - \boldsymbol{x}')^T \Sigma^{-1} (\boldsymbol{x} - \boldsymbol{x}')\right)$$

cannot be expressed explicitly in this way. The support vector machine works around this problem by using the kernel trick where we replace all inner products of the form $\langle \boldsymbol{x}^T \boldsymbol{x} \rangle$ with a call to our kernel $\kappa(\boldsymbol{x}, \boldsymbol{x}')$.

Tasks:

1. Consider a training data set with N input vectors $\boldsymbol{x}_1, \ldots, \boldsymbol{x}_N$ with corresponding target labels $y_1, \ldots, y_N$, where $y_i \in \{-1, +1\}$. The maximum margin solution for linear support vector classifiers without slack variables is found by solving

$$\min_{\boldsymbol{w}, b} \frac{1}{2} \boldsymbol{w}^T \boldsymbol{w} + C \sum_{i=1} \max\left(0, 1 - y_i(\boldsymbol{w}^T \phi(\boldsymbol{x}_i) + b)\right) \tag{1}$$

   where $\phi(\boldsymbol{x}) = \boldsymbol{x}$ is the identity function. This is equivalent to the formulation we have seen in the lecture

$$J(\boldsymbol{w}, b, \boldsymbol{\alpha}) = \frac{1}{2}\|\boldsymbol{w}\|^2 - \sum_{i=1}^{m} \alpha_i \left[y^{(i)}\left(\boldsymbol{w}^T \boldsymbol{x}^{(i)} + b\right) - 1\right],$$

   where $\alpha_i \geq 0$ are Lagrange multipliers. Although $f(x) = \max(0, x)$ is a convex function, it is not differentiable. However, it is possible to express the gradient in a piece-wise fashion:

$$\frac{\partial f}{\partial x} = \begin{cases} \frac{\partial x}{\partial x}, & \text{if } x \geq 0 \\ \frac{\partial 0}{\partial x} & \text{else} \end{cases} = \begin{cases} 1, & \text{if } x \geq 0 \\ 0 & \text{else} \end{cases}$$

   Derive the gradients for $\boldsymbol{w}$ and b and fill in the missing code of the gradient descent routine. *Hint:* Don't forget to apply the chain rule of calculus!

2. Try different values of C and η for dataset 1 with the outlier removed and plot the decision boundary after you trained the classifier until convergence (monitor the loss). How do you know that the classifier converged on a solution, i.e. the loss will not improve anymore? Try to describe your findings in terms of equation 1. Does the decision boundary change when you vary C? Do you have to adapt the learning rate η? Try to describe your findings in terms of equation 1. *Hint:* Would you expect to see this result for the other datasets as well?

3. For this task we will use scikit-learn's built-in support vector machine classifier. It supports slack variables, various kernels such as linear, polynomial or Gaussian kernels and more. Perform a grid search over C (and γ) for all three datasets for linear and Gaussian kernels and report your results. You should include the hyperparameters you have found to work best as well as the mean cross-validated accuracy and accuracy on the test set and the decision boundaries for each dataset.

3 Decision Trees & Ensemble Methods [4.5 points]

Decision trees are defined by recursively partitioning the input space, and defining a local model in each resulting region of input space. This can be represented by a tree, with one leaf per region. Whilst being easy to interpret and relatively robust to outliers, they don't predict very accurately compared to other models. One way to reduce the variance of an estimate is to average together many estimates. For example, we can train many different trees on different subsets of the training data and input variables, chosen randomly with replacement. This technique is known as random forests.

Tasks:

1. Fit `RandomForestClassifier`s on the three datasets. Discuss the effects of varying the number of trees and the maximum tree depths on the decision boundary and the performance. Start with `n_estimators = 1` and vary the `max_depth` of the tree. Report the mean cross-validated accuracy and accuracy on the test set for the best parameters. Plot the decision boundaries for each dataset.

   Next, set the number of estimators to 100 and vary the `max_depth` of the tree. Report the mean cross-validated accuracy and accuracy on the test set for the best parameters. Plot the decision boundaries for each dataset. Will the random forest be more or less affected by outliers or noise in the data as you increase the number of trees in the forest?

2. Ensemble methods have the useful property that they can be used to determine the importance of individual features for classification or regression. Many real-world datasets contain a huge number of features, many of which are not useful for a particular task. Pre-selecting the right features is an important step to get a good performance. In this task, we will explore a dataset we know nothing about. It has 25 features and a total of 800 training examples and 200 test examples with 8 different classes.

   Fit a `RandomForestClassifier` on the dataset and find the most important features. After fitting the model to the data, the depth of the nodes in the decision trees can be used to estimate the relative importance of a feature. The features of the nodes closest to the root of the tree are the "most important" ones because they partition the largest amounts of data. A simple estimate of feature importance is thus the expected number of samples affected by this partition. To make this estimate more robust, the importance of a feature is averaged over the decision trees in the random forest. Conveniently, the `RandomForestClassifier` provides the `feature_importances_` attribute once we fit it to the dataset. Plot the relative feature importance and report the performance of a `SVC` classifier trained on the full dataset.

   The goal of recursive feature elimination (RFE) is to select features by recursively considering smaller and smaller sets of features. First, an estimator is trained on the initial set of features and the importance of each feature is obtained either through any specific attribute (such as `coef_`, `feature_importances_`) or callable. Then, the least important features are pruned from current set of features. That procedure is recursively repeated on the pruned set until the desired number of features to select is eventually reached. `RFECV` performs RFE in a cross-validation loop to find the optimal number of features. We will use this approach to filter the dataset and train another `SVC` on the new dataset. Report the mean cross-validated accuracy and accuracy on the test set for the best parameters.