Machine Learning 1, SS24
Homework 3
k-NN, SVM, Decision Trees.

Thomas Wedenig, thomas.wedenig@tugraz.at

Tutor:	Marharyta Papakina, marharyta.papakina@student.tugraz.at
Points to achieve:	25 pts
Bonus points:	5* pts
Deadline:	21.06.2024 23:59
Hand-in procedure:	Use the **cover sheet** that you can find in the TeachCenter. Submit `main.py`, `knn.py`, `svm.py` and **a report (PDF)** to TeachCenter. Do not change the file name of the Python files. Do not upload a zip file.
Submissions after the deadline:	Each missed day brings a (-5) points penalty.
Plagiarism:	If detected, 0 points for all parties involved. If this happens twice, we will grade the group with "Ungültig aufgrund von Täuschung"
Course info:	TeachCenter, https://tc.tugraz.at/main/course/view.php?id=1648

Contents

General remarks

- **Do not add any additional `import` statements** anywhere in the code.
- **Do not modify the function signatures** of the skeleton functions (i.e., the function names and inputs).
- **Do not change the file name** of any of the Python files.
- Include **all plots and results** in your PDF report.

Failing to conform to these rules will lead to point deductions. Further, your submission will be graded based on:

- Correctness (Is your code doing what it should be doing? Is your derivation correct?)
- The depth of your interpretations (Usually, only a couple of lines are needed.)
- The quality of your plots (Is everything clearly readable/interpretable? Are axes labeled? …)
- Your submission should run with Python 3.9+.

In this assignment we will evaluate a variety of common classification algorithms: k-nearest neighbors, support vector machines, and random forests. 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. Include all plots you create in your report. To this end, call the `plt.savefig` function before `plt.show()`.

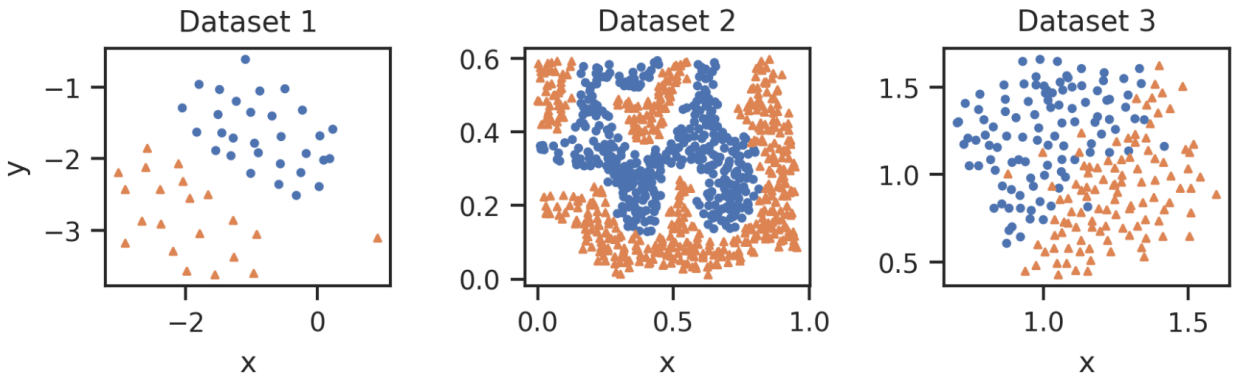

Figure 1: Dataset 1 would be linearly separable if it wasn't for this pesky little outlier at the bottom right. Dataset 2 is highly nonlinear but noise-free. Dataset 3 exhibits a large overlap of the two classes.

1 k-Nearest Neighbors [8 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 nonparametric 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 $\mathbf{x}$ 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. At inference time, predict the class membership of the new input based on the class labels of the k closest training points.

2. Apply the k-NN algorithm to the three datasets. For each dataset, determine the best value for k using 5-fold cross-validation using a grid search for different values of $k \in \{1, \ldots, 100\}$. Plot the mean training and mean validation accuracy for these values of k. Plot the decision boundary for each dataset. Finally, report the performance of the classifier with your chosen value of k on the test set.

3. Briefly discuss the effect of low or high values of k. Explain what happens to the *training* accuracy of a k-NN classifier when you set $k = 1$ and why this is the case.

4. Plot the decision boundaries for $k \in \{1, 30, 100\}$ for the *noisy* variant of dataset 2. Using the *noisy* version of dataset 2, automatically determine the best value for k using 5-fold cross-validation using a grid search for different values of $k \in \{1, 2, \ldots, 100\}$. Plot the mean training and validation accuracy for varying values of k. Finally, report the performance of the classifier with your chosen value of k on the test set.

2 Support Vector Machines [11 points]

In this task, we will train a linear Support Vector Machine (SVM) by hand, and will later use scikit-learn's built-in SVM classifier with different kernels. Also, we will investigate the influence of certain hyperparameters.

Tasks:

1. Consider a training data set with N input vectors $\mathbf{x}^{(1)}, \ldots, \mathbf{x}^{(N)}$ 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 can be found by minimizing the following loss:

$$\min_{\mathbf{w},b} \frac{1}{2}\|\mathbf{w}\|_2^2 + C\sum_{i=1}^{N} \max\left(0, 1 - y^{(i)}\left(\mathbf{w}^T\mathbf{x}^{(i)} + b\right)\right) \tag{1}$$

   Although $f(x) = \max(0, x)$ is a convex function, it is not differentiable everywhere:

$$\frac{df(x)}{dx} = \begin{cases} 1, & \text{if } x > 0 \\ 0, & \text{if } x < 0 \\ \text{undefined} & \text{else} \end{cases}$$

   Using pen and paper, derive the gradients of Equation 1 w.r.t. $\mathbf{w}$ and b. Clearly state all steps in your report. For your derivation, you can assume[1] that we have $y^{(i)}\left(\mathbf{w}^T\mathbf{x}^{(i)} + b\right) \neq 1$ for all $i, \mathbf{w}, b$. Explain what (theoretical) problem could occur if we drop this assumption.

   Use your result to fill in the missing code of the gradient descent routine. In the code, you can simply assume $\frac{df}{dx}(0) = 1$.

2. Try different values for C and for the learning rate η using dataset 1 (with the outlier removed). Use a grid where parameters are exponentially spaced apart, e.g., `'C' : 10.**np.arange(-3, 4)` (this serves just as an example – pick sensible parameter ranges yourself). Plot the decision boundary after you trained the classifier until convergence (monitor the loss).

   In general: Looking at Equation 1 above, what tradeoff does C control? Explain what happens as $C \to 0$ (from the positive side) and as $C \to \infty$.

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. We will try out linear and Gaussian kernels. For all three datasets, perform a grid search over C, `kernel`, and – if `kernel` is `'rbf'` – over γ. Pick sensible parameter ranges, state the best parameter configuration and mean cross-validation accuracy for each dataset. Include a plot of the decision boundaries for each dataset in your report. Finally, for each dataset, report the accuracy on the test set.

[1] While this is not necessarily true in practice, there are a number of reasons why we do not have to worry about these bad singular points.

3 Decision Trees & Ensemble Methods [6 points, 5* bonus 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 over 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. For each of the three datasets and for each `n_estimators` $\in \{1, 100\}$, we will search over `max_depth` $\in \{1, 2, \ldots, 25\}$ using `GridSearchCV` and fit a `RandomForestClassifier` with `random_state=0`.

   For each dataset and `n_estimators` $\in \{1, 100\}$, report the best value for `max_depth`, as well as the mean cross-validation accuracy (6 values in total). Also, include a plot of the decision boundary for each of the 6 configurations in your report.

   For each dataset, the code skeleton creates a plot that shows the relationship between `max_depth` $\in \{1, 2, \ldots, 25\}$ (x-axis) and both the mean test CV score and the mean train CV score (y-axis).

   For each dataset, instantiate a model using the best of the two configurations you have tried, train the model with the full training dataset, and report the test accuracy.

   Discuss the effects of varying the number of trees and the maximum tree depths on the decision boundary and the performance. 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. In general: Assume you're given a training dataset $\mathcal{D} = \{(\mathbf{x}^{(i)}, y^{(i)})\}_{i=1}^{N}$ where each $\mathbf{x}^{(i)}$ is a unique[2] feature vector and each target $y^{(i)}$ is one of k classes. Is it always possible to construct a single decision tree that achieves 100% train accuracy? Briefly explain why you think this is true/false.

Bonus task [5* bonus point]: Tree-based 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 large number of features, many of which are not useful for a particular task. Pre-selecting the right features is often an important step to achieve 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. Conveniently, the RandomForestClassifier provides the `feature_importances_` attribute once we fit it to the dataset. Plot the relative feature importance and report the CV validation accuracy of a `SVC` classifier. Briefly explain the intuition on how scikit-learn can compute feature importances in a tree-based model (no formulas needed, just a concise high-level explanation).

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, typically through a specific attribute (e.g., `coef_` or `feature_importances_`). 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 automatically. We will use this approach to filter the dataset and train another SVC on the new dataset. Report the mean cross-validated accuracy and compare it to the CV accuracy that you achieved without feature elimination. If this classifier performs better, use it to report the test accuracy. Otherwise, use the previous SVC classifier. Report which features (indices) are still in the pruned feature set.

[2] i.e., $\forall\, \mathbf{x}^{(i)}, \mathbf{x}^{(j)} \in \mathcal{D} : \left(\mathbf{x}^{(i)} = \mathbf{x}^{(j)}\right) \implies (i = j)$.