SS 2025
© Prof. Georgia Koppe

Course Coordinator: Andre Herz
andre.herz@iwr.uni-heidelberg.de

Machine Learning Essentials

Exercise Sheet 1

Due: 05.05.2025 11:15

This sheet covers **binary linear classification**, starting with the **geometric properties of decision boundaries**. You'll then explore the **Perceptron** algorithm, one of the earliest learning rules for linear classifiers, and also explore linear **Support Vector Machines (SVMs)**. Finally, you'll get a glimpse of how **feature transformations** and the **kernel trick** can extend linear classifiers to non-linearly separable data.

Regulations

Please submit your solutions via Moodle in teams of **3** students. The coding tasks must be completed using the template `MLE25_sheet01.ipynb`. Each submission must include **exactly** one file:

- Upload a `.pdf` file containing both your Jupyter notebook and solutions to analytical exercises.

The Jupyter notebook can be exported to pdf by selecting **File** → **Download as** → **pdf** in JupyterLab. If this method does not work, you may print the notebook as a pdf instead. Your analytical solutions can be either scanned handwritten solutions or created using LaTeX.

Exercise 1: Linear Decision Boundaries

In this exercise, you'll analyze a binary linear classifier in a 3D feature space. The classifier separates two classes in a 3D feature space (x_1, x_2, x_3) using the function:

$$y = \text{sign}(\boldsymbol{w}^\top \boldsymbol{x} + b)$$

where the parameters are given by:

$$\boldsymbol{w} = \begin{bmatrix} 1 & -2 & 3 \end{bmatrix}^\top, \quad b = -1.$$

Tasks

1. Write down the equation of the **decision boundary** explicitly. Explain why this decision boundary is a **hyperplane** in 3D.

   (2 pts.)

2. State the **normal vector** $\boldsymbol{n}$ to the decision boundary and explain what it represents.

   (1 pts.)

3. The **signed (orthogonal) distance** $d(\boldsymbol{x})$ from a point $\boldsymbol{x}$ to the decision boundary can be computed using:

$$d(\boldsymbol{x}) = \frac{\boldsymbol{w}^\top \boldsymbol{x} + b}{\|\boldsymbol{w}\|}.$$

   (a) Compute the signed distance of the point $\boldsymbol{x} = \begin{bmatrix} 1 & 1 & 1 \end{bmatrix}^\top$ from the decision boundary.

   (1 pts.)

   (b) What does the sign of d tell you?

   (0.5 pts.)

   (c) How does the distance relate to the model's confidence in its prediction?

   (0.5 pts.)

   (d) Compute the orthogonal projection of $\boldsymbol{x}$ onto the decision boundary.

   (0.5 pts.)

   (e) How does computing this orthogonal projection relate to finding **support vectors** for the **maximal margin classifier** that's provided by SVMs?

   (0.5 pts.)

Exercise 2: The Perceptron Algorithm

The single-layer Perceptron is one of the simplest linear classification models. However, it's also the ancestor of modern neural networks. In this exercise, you'll implement the **online**[1] **Perceptron algorithm** and evaluate its predictions. The data that you are going to apply the Perceptron classifier to will be the **Wisconsin Breast Cancer** dataset. This dataset consists of features that describe cell nuclei characteristics of breast tumors, labeled as malignant or benign (i.e. cancerous or non-cancerous).

[1] A learning method is called an "online" algorithm, if it is able to update the model incrementally, i.e. process each data point one by one rather than "waiting" for the entire dataset.

Tasks

1. Load the data using `sklearn.datasets.load_breast_cancer`, and print the names of the features. Select the features ``mean radius'' and ``mean texture''. Then, **standardize** both features to zero mean and unit variance. Create a **scatterplot** of the resulting dataset to visualize the data distribution. Why might the standardization be useful?

   (2 pts.)

2. For each data point $\{\boldsymbol{x}_i, y_i\}$, where $\boldsymbol{x}_i \in \mathbb{R}^2$ are the input feature vectors and $y_i \in \{-1, 1\}$ is the corresponding true label, the Perceptron makes predictions

   $$\hat{y}_i = \phi(\boldsymbol{w}^\top \boldsymbol{x}_i),$$

   using the **activation function** $\phi(z) = \text{sign}(z)$. For convenience, the bias b here has been absorbed into w_0 by setting $\boldsymbol{x}_i = \begin{bmatrix} 1 & x_{i1} & x_{i2} \end{bmatrix}^\top$. This lets us simplify the notation:

   $$\sum_{i=1}^{2} w_i x_i + b = \boldsymbol{w}^\top \boldsymbol{x}_i.$$

   The update rule of the Perceptron updates the weights and bias only in the case of a mistake:

   $$\boldsymbol{w} \leftarrow \boldsymbol{w} + \gamma y_i \boldsymbol{x}_i, \quad \text{if } y_i \neq \hat{y}_i,$$

   where $0 < \gamma \leq 1$ is the **learning rate**. **Implement the Perceptron's training algorithm**. Initialize the weights and bias as zero.

   (2 pts.)

3. Split the data into a **training set** (80%) and a **test set** (20%). Train the Perceptron on the training set and compute the classification **accuracy** on both sets over 100 **epochs**.

   (2 pts.)

4. Plot the **decision boundaries** of the first 5 consecutive iterations to observe how they evolve.

   (1 pts.)

5. How many updates are needed until **convergence** (i.e., until no more model updates occur)? Explain why.

   (1 pts.)

6. Repeat the training process 100 times, each time randomly splitting the dataset into 80% training and 20% test sets. After each run, compute and store the **test accuracy**.

(a) Plot a **histogram** of the test accuracies. What does the shape of the histogram tell you?

(0.5 pts.)

(b) Compute the sample **mean** and **standard deviation** of the test accuracy.

(0.5 pts.)

(c) Given enough data points and many training runs, what type of **probability distribution** would the histogram approximate and why?

(0.5 pts.)

(d) To test the robustness of the Perceptron, add artificial **noise** by randomly flipping $p\%$ of the labels. Visualize the effect on test accuracy by comparing histograms for $p = 0, 10, 20, \ldots, 50$. Interpret the results.

(0.5 pts.)

Exercise 3: SVM

The SVM is a more powerful linear classifier than the Perceptron. While the Perceptron simply finds a separating hyperplane, SVMs find the **optimal** separating hyperplane by maximizing the margin between classes. In this exercise, you'll implement the **dual formulation** of the SVM and explore how it performs on linearly and non-linearly separable datasets in binary classification tasks. Lastly, you'll also see how **feature transformations** and the **kernel trick** can enable linear classifiers to learn nonlinear decision boundaries. The **dual formulation** of the SVM is given by:

$$\max_{\boldsymbol{\alpha}} \sum_{i=1}^{n} \alpha_i - \frac{1}{2} \sum_{i=1}^{n} \sum_{j=1}^{n} \alpha_i \alpha_j \, y_i y_j \, K(\boldsymbol{x}_i, \boldsymbol{x}_j),$$

subject to:

$$\sum_{i=1}^{n} \alpha_i y_i = 0 \quad \text{and} \quad 0 \leq \alpha_i \leq C.$$

Here, α_i are the Lagrange multipliers, $y_i \in \{-1, 1\}$ are the class labels, $K(\boldsymbol{x}_i, \boldsymbol{x}_j)$ is the kernel function that measures similarity between data points, and C is a regularization hyperparameter that balances the trade-off between margin maximization and misclassification error. For large values of C one also speaks of using a hard margin, whereas for small C it's called soft margin.

Because the objective is quadratic and the constraints are linear, we may use **quadratic programming (QP)** to optimize it. This efficiently provides a globally optimal solution to the problem. A **standard QP problem** can be formulated as:

$$\min_{\boldsymbol{\alpha}} \quad \frac{1}{2} \boldsymbol{\alpha}^T \boldsymbol{P} \boldsymbol{\alpha} + \boldsymbol{q}^T \boldsymbol{\alpha},$$

subject to:

$$\boldsymbol{G\alpha} \leq \boldsymbol{h} \quad \text{and} \quad \boldsymbol{A\alpha} = \boldsymbol{b}.$$

In case of the SVM dual problem, the objective thus corresponds to:

$$P_{ij} = y_i y_j \, K(\boldsymbol{x}_i, \boldsymbol{x}_j), \quad \boldsymbol{q} = -\mathbf{1} \quad \text{(i.e. a vector of } -1\text{'s)}.$$

For the constraints, the equality constraint

$$\sum_{i=1}^{n} \alpha_i y_i = 0 \quad \text{gets represented as:} \quad \boldsymbol{A\alpha} = 0, \text{ with } \boldsymbol{A} = \begin{pmatrix} y_1 & y_2 & \cdots & y_n \end{pmatrix}.$$

Finally, the inequality constraint $0 \leq \alpha_i \leq C$ corresponds to

$$\boldsymbol{G} = \begin{pmatrix} -\boldsymbol{I} \\ \boldsymbol{I} \end{pmatrix}, \quad \boldsymbol{h} = \begin{pmatrix} \mathbf{0} \\ C\mathbf{1} \end{pmatrix},$$

where $\boldsymbol{I}$ is the $n \times n$ identity matrix, $\mathbf{0}$ is the zero vector, and $C\mathbf{1}$ is a vector with all entries equal to C.

Tasks

1. Complete the implementation of the provided `DualSVM` class by correctly defining:

   - The **linear kernel** $K(\boldsymbol{x}_i, \boldsymbol{x}_j) = \boldsymbol{x}_i^\top \boldsymbol{x}_j$.
   - The **radial basis function (RBF) kernel** $K(\boldsymbol{x}_i, \boldsymbol{x}_j) = \exp\bigl(-\gamma \|\boldsymbol{x}_i - \boldsymbol{x}_j\|^2\bigr)$.
   - The **optimization objective and constraints**.
   - The **decision function** (i.e., the signed distance) $d(\boldsymbol{x}) = \sum_i \alpha_i y_i K(\boldsymbol{x}, \boldsymbol{x}_i) + \boldsymbol{b}$, where the index i goes over support vectors $\boldsymbol{x}_i$ and their associated class labels y_i and Lagrange multipliers α_i.

   (4 pts.)

2. Use `sklearn.datasets.make_blobs` to generate a dataset with 100 samples, 2 features, and 2 classes. Train a linear SVM on the dataset and plot the decision boundary.

   (2 pts.)

3. Use `sklearn.datasets.make_circles` to generate a dataset with 100 samples, 2 features, and 2 classes. Use two different radii for each class. Train a linear SVM on the dataset and plot the decision boundary.

   (2 pts.)

4. Using a **feature transformation**, we can map data into a higher-dimensional space where it becomes linearly separable. Transform the data from Task 3 with

$$f(\boldsymbol{x}) = \begin{pmatrix} x_1 \\ x_2 \\ x_1^2 + x_2^2 \end{pmatrix}.$$

   Using such a transformation mimics the effect of using the **kernel trick**, an important method in ML that allows models to operate in an implicitly high-dimensional feature space without explicitly computing its coordinates. Instead, they're able to use **kernel functions** to efficiently compute inner products of that space.
   Train a linear SVM on the transformed data and plot the decision boundary.

   (2 pts.)

5. Now, train an SVM using a **(nonlinear) RBF kernel** on the dataset from Task 3 and plot the decision boundary.

   (2 pts.)

6. Compare the decision boundaries from Tasks 3, 4, and 5. How does feature transformation differ from using an RBF kernel? When would one approach be preferable to the other?

   (1 pts.)

7. Besides the dual formulation, SVMs also have an equivalent primal formulation. The key factor in choosing which one to use as the optimization criterion is the dimensionality of the features. Explain why.

   (1 pts.)