Machine Learning Project Report

Valentin Herrmann, Noah Wedlich
Applied Machine Learning in Python – LMU Munich

July 16, 2025

1 Task Overview

In this project we tested if common models like kernel svms, perceptrons as well as decision tree ensembles suffer from the bias-variance tradeoff. For this purpose we trained and tested the models on simple datasets like concentric bands, interleaving half-moons, spirals and separated Gaussian clusters to produce signs of underfitting and especially of overfitting.

2 Methods

2.1 Implementation

The implementation of this project can essentially be divided into two parts: the tunable models and the samplers, both of which we will shortly describe here. We chose an object-oriented approach to allow for easy extension and reuse of the code and to integrate with the Courselib.

Tunable Model: This class provides a convenient way to create and train models on all (or a subset of) the hyperparameter combinations specified. When initialized, it receives a subclass of the Courselib `TrainableModel` class and a dictionary mapping parameter names to their possible values. When training the model, it will instantiate a model for each combination of hyperparameters and train it on the given data. Optionally, one can also provide a validator function which can limit the training to a subset of the combinations. Since the different models are independent of each other, the training is parallelized using Python's `multiprocessing` module. We also utilize queues to communicate with the main process, which allows us to display the training progress in real-time. If an optimizer is used for training, we will further wrap it to provide granular progress updates.

Samplers: The samplers are used to generate the datasets on which the models are trained. The subclasses provided here will sample from a distribution on $S \times L \subseteq [-1, 1]^2 \times \mathbb{N}$ where S is some 2D shape and L is a set of labels. The Samplers also provide methods to apply pre- and postprocessing to the data, which can be used to transform the data into a suitable format for training and to apply transformations to the data after sampling. This allows to easily test the robustness of models using domain shifts or label noise.

3 Experiments and Results

In this project, we focused on Kernel SVMs with the RBF kernel, Perceptrons and Random Forests. We chose one or two complexity-parameters for each model and repeatedly trained the models on different combinations of these parameters. We then used several metrics and visualizations to analyze the results for signs of over- and underfitting.

3.1 Kernel SVMs using the RBF kernel

We first considered Kernel Support Vector Machines (SVMs) with the RBF kernel. We chose the kernel width σ as the complexity parameter, as it controls the sensitivity of the model to the data, i.e. how strongly the model will adapt to the training data. A small σ should lead to a model that is very sensitive to the training data, while a large σ should lead to a model that is more robust to noise and outliers. We varied σ between ?? and ?? and trained the model on spirals and separated Gaussian clusters. Some results of this are shown in Figure **??**.
As can be seen, the training accuracy steadily increases with decreasing σ, reaching ??% for $\sigma =$??. However, the test accuracy only reaches a maximum of ??% for $\sigma =$?? and then starts to decrease again, only being at ??% for $\sigma =$??.

3.2 Perceptrons

We then considered (multilayer) Perceptrons, i.e. fully connected feedforward neural networks, with the ReLU activation function. We chose the number of hidden layers and the number of neurons per layer as the complexity parameters, as they control the capacity of the model to learn complex functions. We varied these parameters one-by-one to isolate their effects on the model's performance. First we fixed the number of neurons per layer to ?? and varied the number of hidden layers between ?? and ??, the results of which can be seen in Figure **??**.
Then we fixed the number of hidden layers to ?? and varied the number of neurons per layer between ?? and ??, as shown in Figure **??**.
As can be seen in the figures, the accuracies of the models follow similar trends as the Kernel SVMs, albeit in a less pronounced way, with the difference being stronger when varying the layers.

3.3 Random Forests

Finally, we considered Random Forests using average voting. We chose both the amount of trees in the ensemble and the maximum tree depth as the complexity parameters. The number of trees controls the amount of noise in the model, as more trees will lead to a more stable model, while the maximum tree depth controls how complex the individual trees can be. First used ?? observers and varied the maximum tree depth between ?? and ??, as shown in in Figure **??**.
Then we fixed the maximum tree depth to ?? and varied the number of trees between ?? and ??. The results of this are presented in Figure **??**.
In contrast to the previous models, the training and test accuracies had a similiar evolution during both experiments, with the training accuracy being slightly higher than the test accuracy.

3.4 Final Comparison

Finally we trained all three types of models on the same dataset, varying the above mentioned complexity parameters. The results of this are shown in Figure **??**.

4 Discussion