4 Classification

To obtain extra points for the written exam, prepare the solution for **Programming Task 4.3** using the Jupyter notebook provided and upload your work to StudOn. Students may discuss with each other while preparing the solution, but each student must submit their work individually. Refer the Supplements page in StudOn for details regarding the submission deadline.

4.1 Programming Task: Gaussian Naive-Bayes Classifier

The Iris dataset, containing measurements of the flower parts obtained from 3 different species of the Iris plant, is provided in the file `iris.csv`. The first four columns of the dataset contain the measurement values representing input features for the model and the last column contains class labels of the plant species: Iris-setosa, Iris-versicolor, and Iris-virginica. The goal of this task is to implement a Gaussian Naive-Bayes classifier for the Iris dataset.

i. What are the assumptions on the dataset required for the Gaussian Naive-Bayes model?

ii. Split the dataset into train and test by the 80:20 ratio.

iii. Estimate the parameters of the Gaussian Naive-Bayes classifier using the train set.

iv. Using the learned parameters, predict the classes for the samples in the test set. What is the accuracy of the model on the test set?

4.2 Programming Task: K-Nearest Neighbor

The datasets in files `train-knn.csv` and `test-knn.csv` contain samples from a synthetic dataset for training a K-Nearest Neighbor classifier. The dataset consists of 7 columns: the first six columns, denoted as $x1$, $x2$, ..., $x6$ represent the input features for each data sample, and the last column represents the class label given by 0 or 1. There are 200 samples in the `train-knn.csv` and 100 samples in the `test-knn.csv`.

i. Implement the K-Nearest Neighbor classification algorithm using NumPy and SciPy.

ii. Perform cross-validation (with 5 folds) on the train dataset `train-knn.csv` to determine a suitable value of K.

iii. Using the optimal value of k from the cross-validation, obtain the accuracy of your model on the test dataset `test-knn.csv`.

iv. Compare your result with the KNeighborsClassifier model from the scikit-learn library.

v. How do the bias and variance of each model vary as K increases?

4.3 Programming Task: Song popularity prediction using Logistic regression

The datasets `train-songs.csv` and `test-songs.csv` contain audio properties of various tracks collected from the music streaming site Spotify.

The goal of the task is to train a logistic regression classifier that predicts if a given track is popular or not. The dataset consists of 9 columns. The first 8 columns contain various audio properties [1] which are provided as input features to the model. The last column contains the class labels given by 0(not popular) and 1(popular)[2].

i. Implement the loss function and its gradient for the logistic regression model.

ii. Using the gradient descent algorithm, train the logistic regression model. You may reuse/modify the gradient descent algorithm from the previous supplement.

iii. Using model predictions on `test-songs.csv`, build the confusion matrix and subsequently calculate the precision, recall, and F1-score for a threshold of 0.4.

iv. Plot the ROC curve for the model and calculate the AUC metric of your model.

v. Consider the simpler models given below. The input $\mathbf{x}$ in these models takes only 4 input features from the given dataset.

- Model A $\mathbf{x} = [\text{ danceability, key, valence, tempo }]^\top$
- Model B $\mathbf{x} = [\text{ danceability, loudness, acousticness, instrumentalness }]^\top$
- Model C $\mathbf{x} = [\text{ key, liveness, valence, tempo }]^\top$

Train these models using your logistic regression implementation and determine the best performing model using the AUC metric.

[1] For the description of the audio features, refer `https://developer.spotify.com/documentation/web-api/reference/#/operations/get-audio-features`.

[2] Spotify uses a complex algorithm to calculate the popularity index of a track. The popularity index varies from 0(least popular) to 100(most popular). The values in last column of the datasets provided is obtained by thresholding the popularity index at 50. This results in a binary classification problem with classes: 'not popular' and 'popular'.