# Pattern Recognition and Machine Learning

**Team 5** - Tzanetis Savvas, Zoidis Vasileios

January 14, 2025

## Part A

In this first part of the assignment, we are tasked with quantifying stress levels of video game players based on play patterns and the intensity of button presses. Given the stress index $x$, which reflects the frequency and pressure of key presses, we are tasked with implementing a **Maximum Likelihood Estimator** that should correctly predict whether a player is experiencing stress, by distinguishing between two classes $\omega_1$ (no stress) and $\omega_2$ (stress).

## Part A

We are also given:

- The **PDF** function for the indicator $x$

$$p(x|\theta) = \frac{1}{\pi(1 + (x - \theta)^2)}$$

- The discriminant function

$$g(x) = \log P(x|\hat{\theta}_1) - \log P(x|\hat{\theta}_2) + \log P(\omega_1) - \log P(\omega_2)$$

## Part A1

The first requirement for this part of the assignment is to estimate the variables $\hat{\theta}_1$ and $\hat{\theta}_2$. In order to achieve this, we need to implement the **Log Likelihood** function:

$$\log L(\theta|D) = \sum_{x \in D} \log p(x|\theta)$$

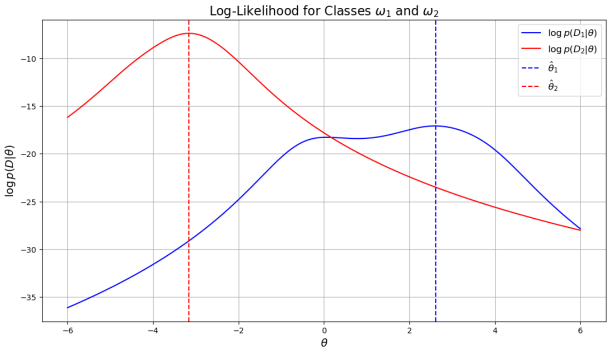
As well as define a range of candidate $\hat{\theta}$ values, which will likely contain the true $\hat{\theta}$.

## Part A1

This is done because using an approach like the gradient of $\log L(\theta|D)$ would be computationally expensive, as this function does not have a closed-form expression. The range of $\hat{\theta}$ candidates should be a slightly wider that the range of our data **[-4.5, 4.1]** to ensure that the optimal value will be included. This range will be **[-6.0, 6.0]**.

## Part A1

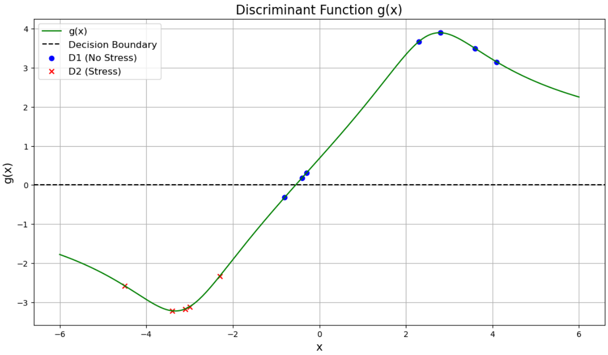
The estimated $\hat{\theta}_1$ and $\hat{\theta}_2$ values, as well as the $\log p(D_1|\theta)$ and $\log p(D_2|\theta)$, are shown in the graph below:



## Part A2

Next we need to classify the two datasets $D_1$ and $D_2$, using the discriminant function that was provided in the beginning:

$$g(x) = \log P(x|\hat{\theta}_1) - \log P(x|\hat{\theta}_2) + \log P(\omega_1) - \log P(\omega_2)$$



## Part A2

In conclusion, with the decision boundary at $g(x) = 0$, meaning that any point in $g(x) < 0$ is classified as **stress** and any point in $g(x) > 0$ is classified as **no stress**, we have the following results:

- **11** out of **12** values of the $D_1$ dataset are classified correctly.
- All of the values of the $D_2$ dataset are classified correctly.

## Part B

In this second part of the assignment, we are again tasked with quantifying the stress level of video game players. This time however, we are asked to use the **Bayesian estimation method** since we are given the **Probability Density Function** of $\theta$:

$$p(\theta) = \frac{1}{10\pi\left(1 + \left(\frac{\theta}{10}\right)^2\right)}$$

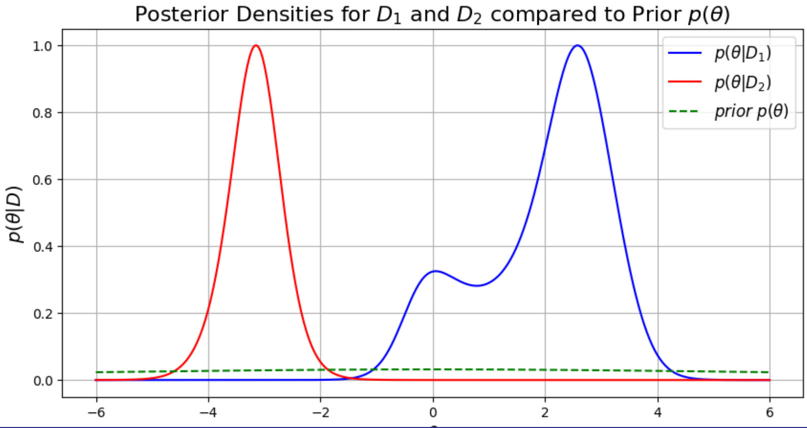
## Part B

This means that we are now able to calculate the **a posteriori probability** of $\theta$, $P(\theta|D)$ like so:

$$P(\theta|D) = \frac{P(\theta|D)p(\theta)}{\int_{-\infty}^{\infty} P(\theta|D)p(\theta)\, d\theta}$$

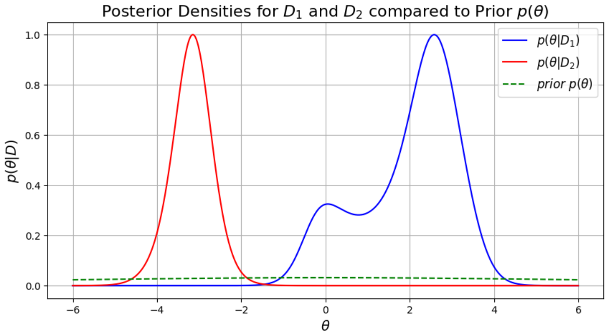
## Part B1

Plotting the **a posteriori probability** for each dataset $P(\theta|D_1)$ and $P(\theta|D_2)$ we can observe that:



## Part B1

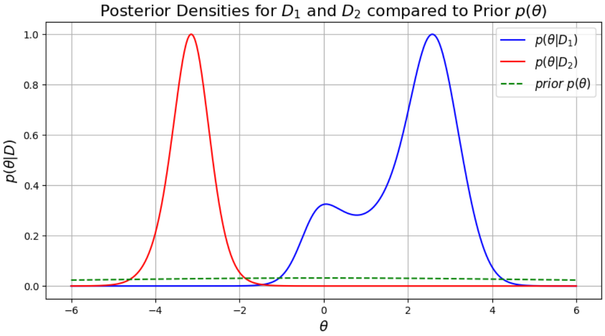
Observations:



The **location of the peaks** in the posterior distributions $P(\theta|D_1)$ and $P(\theta|D_2)$ shows the impact of each dataset ($D_1$ and $D_2$).

## Part B1

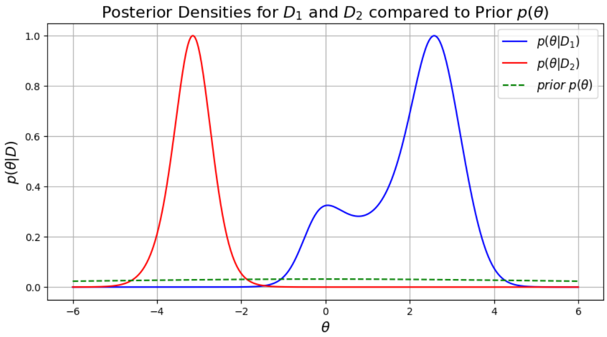
Observations:



The **prior** $p(\theta)$ is **broad** and **evenly distributed** across the range of $\theta$ values, indicating **little preference for any particular** $\theta$.

## Part B1

Observations:



The **posteriors** are much **more concentrated** compared to the prior, showing how the data refines the prior belief and provides more accurate estimates of $p(\theta)$.

## Part B2

At this point, we are tasked to classify the two datasets using **this discriminant function**:

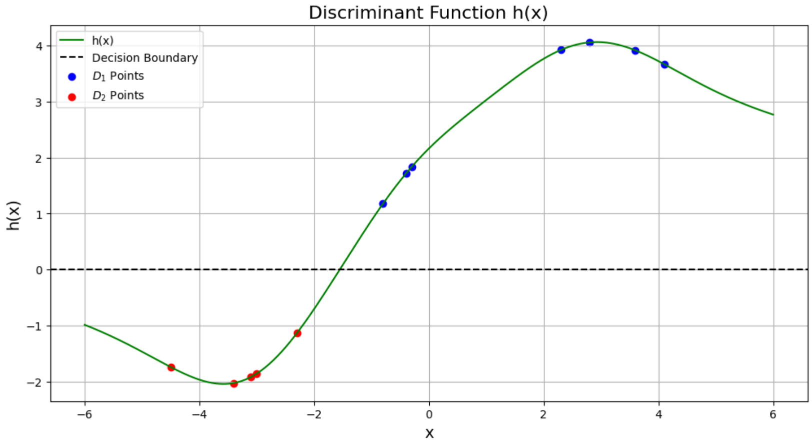
$$h(x) = \log P(x|D_1) - \log P(x|D_2) + \log P(\omega_1) - \log P(\omega_2)$$

In order to implement this, we declare this **posterior predictive distribution** $P(x|D)$ as:

$$P(x|D) = \int P(x|\theta)P(\theta|D)\, d\theta$$

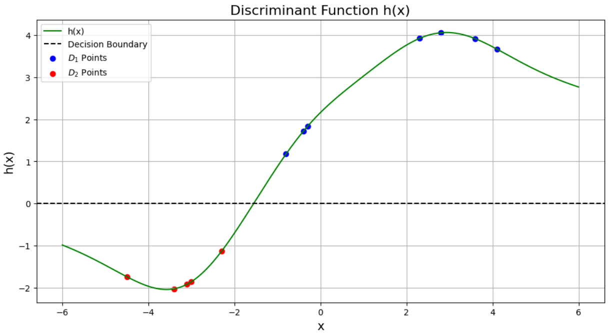
## Part B2

Now, we can visualize $h(x)$:



## Part B2

Observations:



The **decision boundary** is at $h(x) = 0$, and **correctly**, all the points of $D_1$ are classified **above** the decision boundary and all the points of $D_2$ are classified **below** the decision boundary.

## Part B2

**Bayesian Estimation** vs **Maximum Likelihood** method:

In this specific example, the Bayesian Parameter Estimation method is preferred over the Maximum Likelihood method, since it classifies the data with **100%** accuracy, as opposed to the Maximum Likelihood approach **91.7%**. Most likely, this discrepancy is due to the fact that in part B we have prior knowledge about the **Probability Density Function** $\theta$.

## Part C

In this part, we are tasked to train a **Decision Tree Classifier** on three specific **Iris species** (Iris setosa, Iris versicolor, and Iris virginica) using the sepal length and width of each plant.

Meaning, we have **two features** and **three classes**.

## Part C1

First, we **load** the Iris dataset and **isolate** only the first two features. Subsequently, we **split** the dataset into a training and a testing set **(50%:50%)** and **define** random state in order to ensure reproducibility.

Then, we **train** the Decision Tree Classifier and **evaluate** its performance for **various** depths.

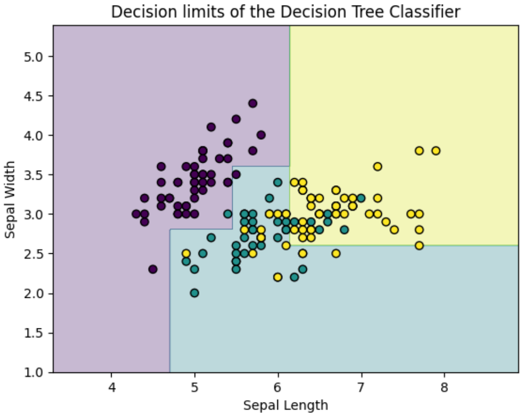
## Part C1

We aim to find the **best depth** of Decision Tree Classifier in regard to **accuracy**.

After testing for depth in a range of up to 10 to avoid overfitting, we conclude that the optimal depth is **3** with **78.67%** accuracy.

## Part C1

Therefore, we use the **best model** (with a depth of 3) to classify our data:



## Part C2

Now, we are tasked to **create** 100 new training sets from 50% of the original training set each time and **train a Random Forest classifier** with 100 trees using the **Bootstrap** technique.

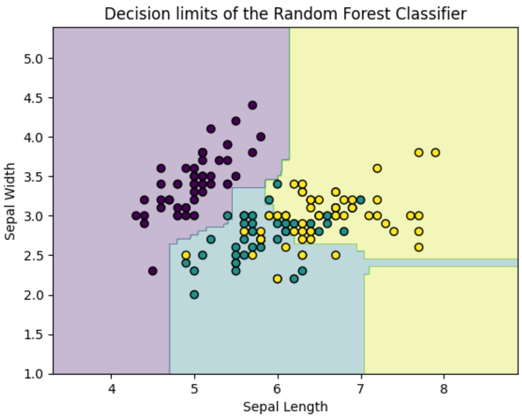
## Part C2

First, we will need to create the **100 samples** for the Bootstrap technique ($\gamma = 50\%$).

Then, we can **train** the Random Forest Classifier and **evaluate** its performance for **various** depths. Similar to before, we are testing for depth in a range of up to 10 to avoid overfitting.

## Part C2

We conclude that the optimal depth is **2** with **82.67%** accuracy:



## Part C2

**Random Forest** vs **Decision Tree** classifier:

We remark that the **Random Forest** slightly outperforms the **Decision Tree** classifier. More specifically, we observe a **5% improvement** in terms of accuracy. This discrepancy is due to the nature of Decision Trees, which create **rough decision boundaries** (hard splits) and often result in **overfitting** to the training data. In contrast, Random Forests produce **smoother decision boundaries** and **less overfitting** because they **average** the predictions from multiple trees. This has the effect of mitigating the overfitting typically seen in a single Decision Tree and enabling **better generalization**.

## Part C2

How does the **percentage** $\gamma$ affect the performance of the algorithm?

From the analysis we did, we observe that:

- for **lower** $\gamma$ **values** (0.1, 0.2, 0.3) the accuracy varies between **0.80** and **0.81**, showing slight improvements as $\gamma$ increases.
- For **higher values** (0.4 and above), accuracy **stabilizes** around **0.83**, suggesting diminishing returns in performance improvement with increasing $\gamma$.

## Part C2

In general, **smaller $\gamma$ values** result in **higher bias** because each bootstrap set includes less information about the entire dataset.

Conversely, **larger $\gamma$ values** reduce diversity among the bootstrap sets, which can **limit the ensemble's ability to mitigate variance** effectively.

In summary, increasing $\gamma$ **up to 0.4** enhances the algorithm's performance, but **further increases have no significant impact** on accuracy.

## Part D

In the final part of the assignment, we are tasked with training a classification model of our choice using a provided dataset **(datasetTV.csv)** and later make predictions on a test dataset **(datasetTest.csv)**. The TV dataset consists of:

- **8743** samples.
- **224** features per sample.

While the test dataset has no labels as it is only meant to test the trained model on and has:

- **6955** samples.
- **224** features per sample as well.

## Part D - Data Scaling

Before fitting a model to our datasets, we must first **scale** the data of both the training and test sets. This is an essential step as this ensures that all features contribute equally to the model's decision-making process, as features with larger scales won't be able to dominate the learning process.

## Part D

In order to choose a suitable model, we tested several different classifiers and their prediction accuracy using **5-Fold cross validation**. More specifically we tested:

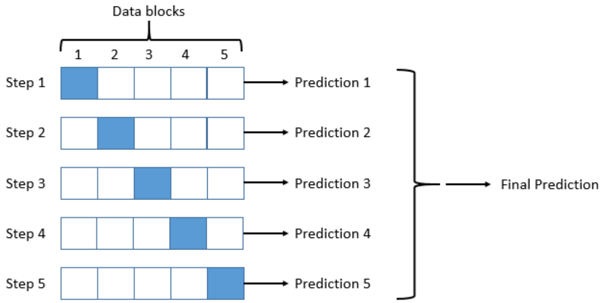
- A **k-NN** classifier with a score of **0.81**
- A **Secure Vector Machine** classifier with a score of **0.83**
- A **Random Forest** classifier with a score of **0.80**
- A **Decision Tree** classifier with a score of **0.72**

## Part D - Validation Score

**5-Fold cross validation** is a technique commonly used for evaluating the performance of a decision model, as it is great at assessing its performance on unseen data since:

- The dataset is divided randomly in **5** equal sized sub-sets (folds), where in each of the 5 iterations, 4 out of the 5 folds are used for training the model and the other is used as a test set.
- After all 5 iterations are complete, we calculate the mean accuracy across all iterations.

## Part D - Validation Score



## Part D

The classification model we ended up choosing was a **Support Vector Machine (SVM)** classifier as it had the best 5-Fold cross validation accuracy among all the other classification models we tested. The **SVM** classifier was the best option, as it is able to handle high-dimensional spaces like the one in our dataset effectively, while also being less prone to overfitting. Scaling the data like previously mentioned, is especially important as it ensures that the optimization algorithm moves at a consistent rate for all features.

## Part D - Parameter Tuning

After deciding on a model, we need to tune its parameters in order to maximize its prediction accuracy. In order to do this performed a **Grid Search** as well as a **Random Search**, since calculating the accuracy of all possible combinations of parameters is very computationally expensive.

## Part D - Parameter Tuning

Beginning with a **Grid Search**, the parameters that were tested, as well as their possible values are seen below:

- Kernel: **rbf**, **linear**.
- C: **1, 10, 100**.
- Gamma value $\gamma$: **scale, auto**.

## Part D - Parameter Tuning

**Grid Search** on its own has some disadvantages. Since every possible combination of parameters is being tested, testing a large number of possible parameters can get very computationally expensive since we are also testing combinations of parameters that are unlikely to yield acceptable results.

## Part D - Parameter Tuning

This is why we also used the **Random Search** technique, which allows us to define a range of possible values and test them by randomly sampling values for each parameter. While this method can be efficient, it may not always find the 'optimal' result because it doesn't exhaustively explore the parameter space. For this reason, it is used in conjunction with **Grid Search**, which systematically tests all possible combinations of parameters.

## Part D - Parameter Tuning

Combining the results of both techniques, the optimal values that we used for our classification model where:

- Kernel: **rbf**.
- C: **4**.
- : Gamma value $\gamma$: **scale**.

## Part D - Final Results

Finally, our **Support Vector Machine** classification model with specified parameters, yields acceptable results, with **5-Fold cross validation** scores of: [0.864, 0.858, 0.851, 0.850, 0.839], and a mean score of **0.85**. The relatively small variance in scores across each iteration of the cross-validation assures us that there is no overfitting in our trained model.