HW2: Algorithm Implementation and Basic Model Selection

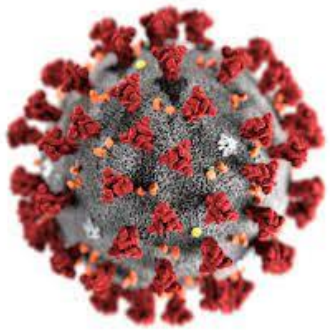
Goal

This assignment is the second of three mini projects that will guide you in your task of stopping the spread of disease around the globe!

In this assignment you will implement Soft-SVM using gradient descent. You will also practice basic hyperparameter tuning (model selection) using three algorithms: k-NN, Decision Trees (with ID3), and Soft-SVM.

Many techniques and ideas from this assignment will greatly help you in Major HW3.

Good Luck!



Instructions

- **Submission**
  - **Submit by:** Monday, 06.06.2022, 23:59.
    - Late submissions will be accepted until 07.06.2022 and penalized 5pts.

- **Python environments and more**
  - We recommend using jupyter notebooks. Google colab can be very convenient since it does not require installing anything on your local computer. It will also help you to collaborate with your partner online.
  - Initial notebook here.
    - Demonstrates how to upload a dataset to Google colab and how to download files from Google colab.
    - You can save a copy of this notebook to your Google drive.
  - However, you can use any Python IDE you choose. For working locally with an IDE, we recommend first installing conda for package management (with Python 3.6 or 3.8), and then installing an IDE like PyCharm or Spyder.

- **Your code**
  - Should be clearly and briefly documented.
  - Variables/classes/functions should have meaningful names.
  - Will be partially reviewed and graded.

- **Final report**
  - Should be written in a word processor (Office Word, Google docs, etc.).
    - Should not contain the code itself.
    - Do not submit jupyter notebooks as PDFs.
  - Can be in Hebrew, English, or both.
  - **You are primarily assessed based on your written report.**
  - Answer the questions in this instruction file according to their numbering.
  - Add concise explanations, figures (outputs of your code), tables, etc.
  - You are evaluated for your answers but also for clarity, readability, and aesthetics.
  - **Tables** should include feature names and suitable titles.
  - **Plots** should have suitable titles, axis labels, legends, and grid lines (when applicable)
    - We recommend adjusting the default font sizes of matplotlib (see snippet in HW1).

- **Submit a zip file containing** (please use hyphens, not underscores):
  - The report PDF file with all your answers, named *id1-id2.pdf*.
    - Do not include your code in your report.
  - Your code:
    - Working with jupyter: a notebook with your code, *id1-id2.ipynb*.
    - Working with a "traditional" IDE: all relevant python scripts.
    - The following files:
      - *SoftSVM.py* : your completed SVM module (=class).
      - *prepare.py* : your updated data preparation pipeline (including normalization).
    - Do not submit csv files.

Preliminary: Data Loading

Task: Follow the procedure below.

a. Start by **loading** the preprocessed data from the previous assignment.
   Note: in Lecture 08 we explain why some preprocess steps (e.g., normalization), should be applied to the validation folds according to statistics computed on the train fold. Here, for simplicity only, you compute these statistics according to the all the training samples (before splitting it to train and validation folds).

b. Make sure the data is **partitioned** correctly to train and test, according to the instructions in the previous assignment.
   The train-test partitions **must** be identical to the ones you used in HW1.

c. ~~**Important:** do not use the test set in this assignment.~~

d. Make sure you did not delete the following features: `PCR_{01, 04, 05, 08, 10}`, `sport_activity`, and `sugar_levels`. Note that there are other important features. Here we chose to specify only some of them. If you deleted some of them, you should edit your preprocessing pipeline accordingly.

e. Make sure all target variables follow the `{+1,-1}` convention (rather than `{1,0}` or `{True,False}`).

Part 1: Basic model selection with k-Nearest Neighbors

Like in the previous HW, we start with the simple k-NN model and use it to practice some new concepts, like model selection. In this part we will use k-NN on `PCR_08` and `PCR_10` to predict the `risk`, similarly to HW1. You should use sklearn.neighbors.KNeighborsClassifier rather than your custom implementation from the previous HW.

Visualization and basic analysis

Task: Create a temporary `DataFrame` containing only `PCR_08` and `PCR_10`.

Reminder: Use the preprocessed (normalized) training set.

(Q1) Attach a `jointplot` of `PCR_08`, `PCR_10` (use the `risk` class as the `hue` argument).

(Q2) Train a k-NN model using $k = 1$ on your training set and use the updated `visualize_clf` method (given in this HW) to visualize the resulted decision regions. Make sure to have appropriate title and labels.

Model selection

Most ML models are characterized by a set of parameters that control the learning process and are not optimized during training (e.g., `k` in k-NN and `max_depth` in decision trees). As we learned, these hyperparameters can change the learning dynamics dramatically. Hyperparameters are often tuned using k-fold-cross-validation, where we split the training data into k_v folds and train k_v models – each trained on $k_v - 1$ folds and use the last fold for validation (i.e., performance evaluation). This procedure estimates the model's performance on unseen data; thus, we can find the (estimated) optimal hyperparameters.

Note: DO NOT use the test set for hyperparameter tuning.

(Q3) Use sklearn.model_selection.cross_validate to find the best k (neighbours) value in `list(range(1, 20, 1)) + list(range(20, 300, 10))` for predicting the `risk` class using `PCR_08` and `PCR_10`.

If you wish to see clearer trends, you **can** use higher values than `300`, e.g., `500`.

Use the (default) accuracy metric and 8-folds to perform cross-validation.

Using the outputs of `cross_validate`, plot a *validation curve*, i.e., the (mean) training and validation accuracies (y-axis) as a function of the k values (x-axis).

Add the plot to your report.

Answer: which k value is the best? What are its mean training and validation accuracies (computed during the cross validation)? Also explain which k values cause overfitting and underfitting (and why is that).

(Q4) Use the optimal hyperparameter combination you found and retrain a k-NN model on all the training samples. In your report: plot the decision regions of this final model and write its test accuracy (computed on the separate test split) of this model.

(Q5) Use the updated `visualize_clf` method to visualize the `risk` decision regions for a k-NN model with $k = 1$ and one with the best k value from the previous question. Compare these two k-values and discuss the results (2-3 sentences).

(Q6) Repeat (Q3) but use all the features in your training dataset (instead of only `PCR_08` and `PCR_10`). Add the validation curve to your report. Discuss in detail the differences between the results here to those of (Q3) and try to explain them in light of the mechanism of k-NN models.

Part 2: Decision trees

In this part we will focus on predicting the `covid` class using decision trees. Rather than implementing the models by yourself, you will use sklearn's DecisionTreeClassifier with entropy as a splitting criterion (ID3) and focus on hyperparameter tuning and visualization.

Visualization and basic analysis

(Q7) Return to your report / notebook for HW1. Based on your analysis there (especially in Q28), specify here 4-5 features that seemed important for predicting the `covid` class. Attach here the plots that made you believe these features are indeed helpful.

(Q8) Compute the 10 most correlated features to the `covid` class and attach them to your report.

(Q9) Train a model with ID3 and `max_depth=4` (not including the root level; use the entire training set, i.e., all the features after preprocessing from all the training samples). What is the training accuracy?

Visualize the trained tree using `plot_tree` (provide feature and class names; use `filled=True`) and attach the plot to your report. The plot should be readable!

(Q10) Is the "importance" of the features you named in (Q7) noticeable from the correlation computation in (Q8)?

Were these same features used by ID3 in the decision tree built in (Q9)?

Compare and discuss your findings (about ½ a page, including appropriate small figures).

Model selection

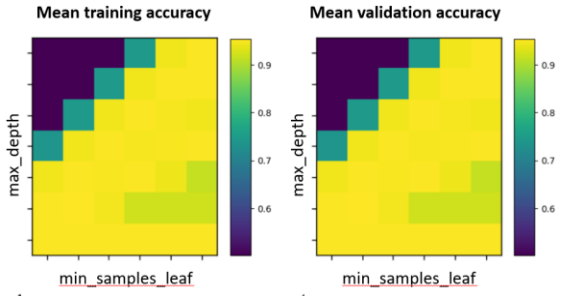
It is time to search for the best tree to fight covid! Using the DecisionTreeClassifier documentation, understand how the `min_samples_leaf` argument can mitigate overfitting. You will now tune two hyperparameters simultaneously – both `min_samples_leaf` and `max_depth`. You need to look for the combination of these two hyperparameters that lead to the best validation performance. There are many approaches for tuning multiple hyperparameters, and here we take the grid search approach (shortly explained here).

(Q11) Using 8-fold cross-validation, tune the two aforementioned hyperparameters by performing a grid search (see GridSearchCV). Find the combination yielding the best validation error for predicting the `covid` class. You should:

a. Choose appropriate ranges for both hyperparameters. This may require a few attempts. To make things quicker when trying to find appropriate hyperparameter ranges, you can start by using only 2 folds.

b. Since we tune two hyperparameters, instead of a validation curve, plot two heatmaps (seaborn / pyplot), one for the cross-validated training accuracy and one for the cross-validated validation accuracy.

These heatmaps should roughly have the following style / structure:



Make sure to plot the appropriate "ticks" on both axes and use annotations (`annot=True`) to explicitly write the accuracies inside the heatmap cells.

Important: The plots should be readable and informative!

c. Add the 2 plots to your report and specify which hyperparameter combination is optimal.

d. Write a hyperparameter-combination that causes underfitting.

e. Write a hyperparameter-combination that causes overfitting.

f. Add a short discussion regarding why each specific hyperparameter-combination from sub-questions 'd' and 'e' resulted in under/over-fitting.

(Q12) Use the optimal hyperparameter combination you found and retrain a decision tree on all the training samples. In your report: ~~plot the decision regions of this final model~~ and write its test accuracy of this model.

Part 3: Linear SVM and the Polynomial kernel

In this part we will implement a Soft-SVM classifier to better understand this model. We will use gradient-based optimization to find the optimal parameter. Recall that using the whole dataset to perform one step update is costly. To mitigate this problem, we will implement the Stochastic Gradient Descent (SGD) algorithm.

Implementation of the loss and its gradient

Recall the Soft-SVM formulation:

$$\underset{\boldsymbol{w}\in\mathbb{R}^d, b\in\mathbb{R}}{\operatorname{argmin}} \underbrace{\|\boldsymbol{w}\|_2^2 + C\sum_{i=1}^{m}\max\{0, 1-y_i(\boldsymbol{w}^\top \boldsymbol{x}_i + b)\}}_{\triangleq p_C(\boldsymbol{w},b)}$$

Following is the **analytic** sub-gradient of the objective function above

$$\nabla_w p_C(\boldsymbol{w}, b) = 2\boldsymbol{w} + C\sum_{i=1}^{m} f\big(y_i(\boldsymbol{w}^\top \boldsymbol{x}_i + b)\big) y_i \boldsymbol{x}_i \text{ , where } f(z) = \begin{cases} -1, & z < 1 \\ 0, & z \geq 1 \end{cases}$$

$$\frac{\partial}{\partial b} p_C(\boldsymbol{w}, b) = C\sum_{i=1}^{m} f\big(y_i(\boldsymbol{w}^\top \boldsymbol{x}_i + b)\big) y_i$$

Task: Copy the `SoftSVM` module from the given *`SoftSVM.py`* into your notebook / project.

Task: Complete the (static) `SoftSVM.loss` method in the module, so that it computes the objective loss $p_C(\boldsymbol{w}, b)$ on a given dataset.

Remember: `w` is a vector and `b` is a scalar.

Tip: When possible, prefer vector operations (e.g., `np.sum`, `np.sign`, `np.maximum`). Avoid using `for` loops.

Task: Complete the (static) `SoftSVM.subgradient` method in the module, so that it computes the analytic sub-gradients $\nabla_w p_C(\boldsymbol{w}, b)$ and $\frac{\partial}{\partial b} p_C(\boldsymbol{w}, b)$ described above.

Tip: When possible, prefer vector operations (e.g., `np.sum`, `np.sign`, `np.maximum`). Avoid using `for` loops.

Loading a synthetic dataset for basic experiments

For the few following questions, we will need a toy dataset which is almost linearly separable.

Task: Create and visualize a moon dataset using the following code snippet. For the random state (highlighted), use the same seed as you used in HW1 for splitting the data (created from your IDs).

```
from sklearn.datasets import make_moons
X_moons, y_moons = make_moons(n_samples=1000, shuffle=True, noise=0.05, random_state=TODO)
y_moons = ((2 * y_moons) - 1)[:, None]
print(f"{X_moons.shape}, {y_moons.shape}")

plt.figure(), plt.grid(alpha=0.5), plt.title("Synthetic moon dataset")
_ = sns.scatterplot(x=X_moons[:, 0], y=X_moons[:, 1], hue=y_moons[:, 0])
```

(Q13) Attach the resulted plot to your report.

Verifying your implementation: Numerical vs. analytical gradients

Recall from your calculus course, the definition of the derivative is:

$$f'(x_0) = \lim_{\delta \to 0} \frac{f(x_0 + \delta) - f(x_0)}{\delta}$$

Thus, we can deduce a method to compute the **numerical** partial derivative w.r.t. w_i by approximating the limit expression with a finite δ:

$$\forall i = 1, \ldots, d: \ \frac{\partial p_C}{\partial w_i} \approx \frac{p_C(\boldsymbol{w} + \delta e_i, b) - p_C(\boldsymbol{w}, b)}{\delta} \triangleq u_i, \ \text{ where } e_i = [0, \ldots, 0, \underbrace{1}_{i-th}, 0, \ldots, 0]$$

Denote the **numerical** sub-gradient by $\boldsymbol{u}_\delta(\boldsymbol{w}, b) = \begin{bmatrix} u_1 \\ \vdots \\ u_d \end{bmatrix}$.

Using the numerical sub-gradient, we will now verify the correctness of your implementation for the loss and its analytic sub-gradient.

We will plot the residuals $\| \underbrace{\nabla_w p_C(\boldsymbol{w}, b)}_{\text{analytic}} - \underbrace{\boldsymbol{u}_\delta(\boldsymbol{w}, b)}_{\text{numeric}} \|_2$ over many repeats as a function of δ.

Task: Copy the functions from the given `verify_gradients.py` into your notebook / project. Read and understand these functions but do not edit them.

(Q14) Using the synthetic moon dataset, generate a plot that compares the numerical gradients to the analytic gradients. Do this by running the following command:

```
compare_gradients(X_moons, y_moons, deltas=np.logspace(-9, -1, 12))
```

Important: Make sure `y_moons` follows the `{+1,-1}` convention.

Attach the plot to your report. Briefly discuss and justify the demonstrated behavior.

Solving Soft SVM problems using Stochastic Gradient Descent (SGD)

Task: Complete the given `SoftSVM.predict` method according to the decision rule of linear classifiers (return the predicted labels using the `{+1,-1}` convention).

Tip: prefer vector operations (e.g., `np.dot` and `np.sign`) when possible.

Avoid using `for` loops.

Task: Read and understand the given `SoftSVM.fit_with_logs` method.

Complete the code inside the loop to perform a gradient step (compute `g_w` and `g_b` and use them to update `w` and `b`).

SGD is an iterative learning algorithm. A common method to analyze such algorithms is to plot a *learning curve*, i.e., plot the accuracy and/or loss of the model over time (i.e., steps).

(Q15) The following snippet trains a `SoftSVM` model on the synthetic moon dataset and plots the learning curves. Make sure your model converges. For that, you may need to adjust the `lr` and `max_iter` values. However, you may not change the given `C` value. Add suitable labels and titles. Finally, add the plot(s) to your report. Also attach a plot of the decision regions of your model (at the end of training).

Answer: What is the maximal training accuracy and the minimal training loss achieved by your model? Are they attained at the same step? If so – why? If not – how is it possible?

```
clf = SoftSVM(C=1, lr=2e-4)
losses, accuracies = clf.fit_with_logs(X_moons, y_moons, max_iter=3000)
plt.figure(figsize=(13, 6))
plt.subplot(121), plt.grid(alpha=0.5), plt.title ("TODO")
plt.semilogy(losses), plt.xlabel("TODO"), plt.ylabel("TODO")
plt.subplot(122), plt.grid(alpha=0.5), plt.title ("TODO")
plt.plot(accuracies), plt.xlabel("TODO"), plt.ylabel("TODO")
plt.tight_layout()
```

Submit: Copy your completed `SoftSVM` module into a separate file called *`SoftSVM.py`*.

Submit this file at the end of your work.

Using a feature mapping

We now want to understand the polynomial feature mapping and its corresponding kernel. To make our following explanation clearer, assume we want to use a 2nd-degree polynomial feature mapping on a 2-dimensional data (i.e., having two raw features).

Recap: solving SVMs with feature mappings

We have two ways to use this feature mapping (we present Hard-SVM for simplicity):

1. *Explicitly apply the feature mapping* $\phi(\boldsymbol{x}) = \left[1, \sqrt{2}x_1, \sqrt{2}x_2, \sqrt{2}x_1x_2, x_1^2, x_2^2\right]$ *and solve the primal problem in the new 6-dimensional space:*

$$\underset{\boldsymbol{w}\in\mathbb{R}^6}{\operatorname{argmin}} \|\boldsymbol{w}\|_2^2 \quad \text{s.t.} \quad y_i \cdot \boldsymbol{w}^\top \phi(\boldsymbol{x}_i) \geq 1, \ \ \forall i \in [m]$$

2. *Use an appropriate kernel function, i.e.,* $K(\boldsymbol{u}, \boldsymbol{v}) = \phi(\boldsymbol{u})^\top \phi(\boldsymbol{v}) = (\boldsymbol{u}^\top \boldsymbol{v} + 1)^2$*, and solve the dual problem in an* m*-dimensional space (without explicitly computing the feature mappings):*

$$\max_{\boldsymbol{\alpha}\in\mathbb{R}_+^m} \sum_{i=1}^m \alpha_i - \frac{1}{2}\sum_{i,j=1}^m y_i y_j \alpha_i \alpha_j \underbrace{\phi(\boldsymbol{x}_i)^\top \phi(\boldsymbol{x}_j)}_{=K(\boldsymbol{x}_i, \boldsymbol{x}_j)}$$

We will now use the polynomial feature mapping to predict to predict the `spread` variable using the two features we found on HW1: `PCR_05` and `sugar_levels`.
Since you only implemented the primal (Soft-)SVM problem, we will use the **first** way and explicitly transform the aforementioned 2 features into 6 polynomial features.

Task: Use your training set to create a temporary `DataFrame` containing only `PCR_05`, `sugar_levels`, and `spread`.

(Q16) Plot a `jointplot` of `PCR_05` and `sugar_levels`, using `spread` as the `hue`.

We now wish to create a single model that: (1) transforms the 2 features into 6 polynomial features and then (2) trains your custom `SoftSVM` module on these 6 features. To do so, we will use a Pipeline.

The following code snippet creates such a pipeline:

```
svm_clf = Pipeline([('feature_mapping', TODO),
                    ('SVM', SoftSVM(C=1e4, lr=1e-3, batch_size=32))])
```

Task: Complete the first step in the pipeline above to make it apply a 2nd-degree PolynomialFeatures transformation.

(Q17) Train such a model 5 times (on all the training samples), using the hyperparameters written in the snippet above. Set the `max_iter` to 10,000. Since you work with a pipeline, use the following command to specify a hyperparameter of an inner model inside a pipeline: `svm_clf.fit(X_train, y_train, SVM__max_iter=10000)`.
In your report:
(a) Write the 5 resulting train accuracies, their mean, and their standard deviation.
(b) Plot the 5 resulting decision regions (use `visualize_clf` on the training set).
(c) Answer: what is (are) the source(s) of the variability in the resulting models (refer also to the convexity of this problem)?

(Q18) Use 10-folds cross-validation to tune the `C` hyperparameter. You should choose suitable hyperparameter ranges (tip: use `np.logspace`) on your own. Keep the `lr`, `max_iter`, and `batch_size` fixed as they are in the snippet above.
In your report:
(a) Specify the range you chose for the tuning process.
(b) Attach a suitable *validation curve* to your report (plot both train and validation accuracies). The x-axis should appear in a logarithmic scale. Fully discuss and explain the behavior demonstrated in the plot.
(c) Specify the optimal `C` value and its corresponding (cross validated) train and validation accuracies.

(Q19) Train your model on all the training samples using the optimal `C` value you found.
In your report: plot the decision regions of this final model and write its test accuracy.

Part 4: The RBF kernel

Before we start: what is the Radial Basis Function kernel?

In Lectures 04 and 05 we presented the RBF kernel (also called the Gaussian kernel). This kernel is often defined as $K(\boldsymbol{u},\boldsymbol{v}) = \exp\left\{-\frac{1}{2\sigma^2}\|\boldsymbol{u}-\boldsymbol{v}\|_2^2\right\}$ *or* $\exp\{-\gamma\|\boldsymbol{u}-\boldsymbol{v}\|_2^2\}$*, which can be decomposed using an infinite-dimensional feature mapping (see the lecture).*

Note: If you wish, you can already practice this mapping by solving Q4 in Exam A from the previous semester.

After solving the appropriate 2 optimization problem, we get a dual solution $\boldsymbol{\alpha} \in \mathbb{R}_+^m$*. Like we saw in the lecture, given a new datapoint* $\boldsymbol{x} \in \mathbb{R}^d$ *for prediction, the model predicts* $h(\boldsymbol{x}) = \text{sign}(\sum_{i=1}^m \alpha_i y_i K(\boldsymbol{x},\boldsymbol{x}_i)) = \text{sign}\left(\sum_{i\in[m],\alpha_i>0} \alpha_i y_i K(\boldsymbol{x},\boldsymbol{x}_i)\right)$*.*

This rule acts similarly to a weighted nearest-neighbor algorithm (on the training set), where "neighborhoods" of datapoints are computed using the kernel function.

For instance, consider a 1-dimensional training dataset: $\{\underbrace{(0,-1)}_{(x_1,y_1)}, \underbrace{(2,+1)}_{(x_2,y_2)}, \underbrace{(-1,+1)}_{(x_2,y_2)}\}$*.*

Assume the dual SVM solution is $\boldsymbol{\alpha} = (1,1,2)$*.*

When $\sigma = 1$*, we get the (weighted) Gaussians on Figure (a).*

Given a new datapoint $x = 0.5$*, the model predicts:*

$$h(\boldsymbol{x}) = \text{sign}\sum_i \alpha_i y_i K(\boldsymbol{x},\boldsymbol{x}_i) = \text{sign}\big(1\cdot(-1)\cdot K(0.5,0) + 1\cdot 1\cdot K(0.5,2) + 2\cdot 1\cdot K(0.5,-1)\big)$$

$$= \text{sign}\left(-\exp\left\{-\frac{\|0.5-0\|_2^2}{2}\right\} + \exp\left\{-\frac{\|0.5-2\|_2^2}{2}\right\} + 2\exp\left\{-\frac{\|0.5-(-1)\|_2^2}{2}\right\}\right)$$

$$= \text{sign}\left(-\exp\left\{-\frac{0.25}{2}\right\} + \exp\left\{-\frac{2.25}{2}\right\} + 2\exp\left\{-\frac{2.25}{2}\right\}\right) \approx \text{sign}(-0.88 + 0.32 + 0.65) = +1$$

In fact, we can create an even clearer visualization by drawing the weighted sum of positive points' Gaussians (blue) and the one of negative ones (red). See Figure (b).

Finally, Figure (c) shows the case resulted Gaussians when $\sigma = \frac{1}{10}$*. Notice how each training point influences only a small environment around it. We explain this below.*

Figure (a)

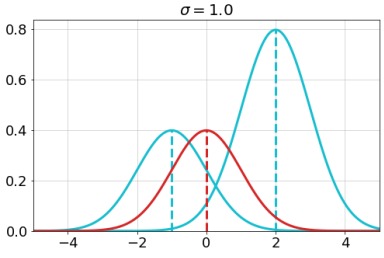
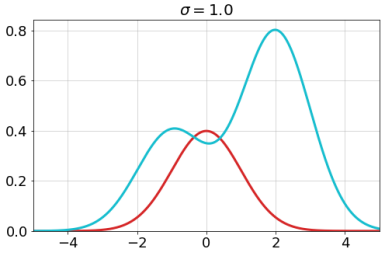
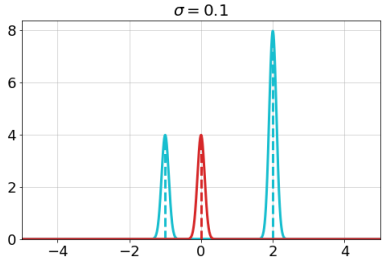


Figure (b)



Only very closely to $x = 0$*, the model predicts* $y = -1$*.*

Figure (c)



Each points effects only a small environment around it

As another example, consider a (finite) solution vector $\boldsymbol{\alpha} \in \mathbb{R}_+^m$ in the extreme case where $\sigma^2 \to 0$ and $\gamma \to \infty$, it can be easily shown that we should get a similar behavior (perhaps up to edge cases) to the 1-nearest-neighbor algorithm on the support vectors, i.e., the vectors with corresponding nonzero α weights:

$$h(\boldsymbol{x}) = \operatorname{sign}\left(\sum_{i\in[m],\alpha_i>0} \alpha_i y_i K(\boldsymbol{x},\boldsymbol{x}_i)\right) = \lim_{\gamma\to\infty} \operatorname{sign}\left(\sum_{i\in[m],\alpha_i>0} \alpha_i y_i \exp\{-\gamma\|\boldsymbol{x}-\boldsymbol{x}_i\|_2^2\}\right)$$
$$= y_{i^*} \text{ where } i^* = \operatorname{argmin}_{i\in[m],\alpha_i>0}\|\boldsymbol{x}-\boldsymbol{x}_i\|_2^2$$

To complete our explanations, read this blogpost and then watch this short video. While doing this, keep in mind the prediction rule $h(\boldsymbol{x}) = \operatorname{sign}\left(\sum_{i\in[m],\alpha_i>0} \alpha_i y_i K(\boldsymbol{x},\boldsymbol{x}_i)\right)$.

Like we did in Part 1, we now wish to predict the `risk` variable using `PCR_08` and `PCR_10`. This time, we shall use a Kernel SVM with an RBF kernel. Since the corresponding feature mapping is of an infinite dimensionality, we will have to use the kernel trick and solve the dual problem, like we explained 2. Our custom SoftSVM class is not suitable for solving the dual formulation, hence we will use sklearn's implementation.

Task: Use sklearn.svm.SVC to train an SVM with an RBF kernel on the two features and the `risk` variable. We first wish to visualize the regularizing effect of γ, so you should begin by fixing `C=1e4` and train 10 different models (on the entire training set, i.e., all training samples, narrowed down to the two features) with all γ values in `np.logspace(-5, 4, 10)`.

(Q20) Use our visualization method to visualize the ten models' decision regions (use the `marker_size` argument to adjust the plots for small decision regions).
Attach the ten plots to your report (resize them to fit in a single page of your report). Remember that all plots should have suitable titles.

We recommend that you try to understand the exhibited behaviors in these ten plots in light of the explanations at the beginning of this part. You are not required to explain these behaviors in your report.

(Q21) Compare the decision regions of the k-NN model with $k = 1$ from (Q2) to those of the RBF model with $\gamma = 10^4$ from (Q20). We expected the two models to be very similar, but we still see significant differences. Try to explain the differences (try to give several reasons, there isn't a single correct answer).

(Q22) Now, given the problem at hand (i.e., predicting the `risk` using an SVM with the RBF kernel on `PCR_08` and `PCR_10`), we wish to find an optimal combination of C and γ. To this end, repeat the grid search steps from (Q11) and perform 8-fold cross validation on C and γ. Remember: to save time, you can use only 2 folds when looking for appropriate ranges for these hyperparameters.
Don't forget to attach here the required plots and explanations, as in (Q11).

(Q23) Use the optimal hyperparameter combination you found and retrain an RBF model on the all the training samples. In your report: plot the decision regions of this final model and write its test accuracy. Compare these results to those from (Q4).
Based on the test accuracy, which model is better for the task of predicting the `risk`?

Submitting the files

Return to the instructions on Pages 2-3 and make sure you submit **all** required files.