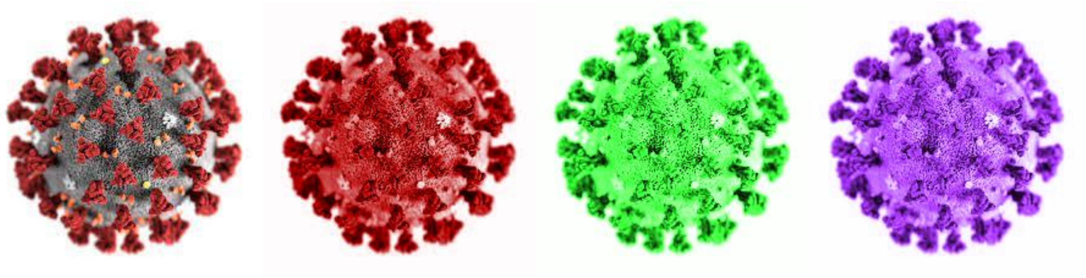
HW2 – Modeling and Classification

Goal

In this exercise we will implement several classification models and dive into their inner workings to better understand them and to draw conclusions about our data. We will be required to explore the sklearn library and its documentation to familiarize ourselves with its many ML models and tools. Our goal is to best classify our data given a specific target.

Classification, even binary classification alone, is a widely used method for a myriad of purposes, from face recognition to election predictions (and a whole lot more). We will attempt to classify important medical information that can help doctors and emergency teams make better decisions, whether it be for medical diagnosis or efficient evacuation and quarantine.

Good Luck!



Instructions

- **Submission:**
  - **Submit by:** 03.06.2021, 23:59
    - Late submissions will be penalized 5 points per day.
  - Submissions in pairs only.

- **Python environments and more**
  - Recommended: jupyter notebooks. Google colab can be very convenient for online collaboration.
    - Initial notebook here (demonstrates how to upload/download to/from Google colab).
  - If you prefer working locally on an IDE, we recommend installing conda for package management, and then an IDE like PyCharm or Spyder.

- **Your code:**
  - Should be clearly and briefly documented.
  - Variables/classes/functions should have meaningful names.
  - Should use the classification models of sklearn.

- **Final report**
  - Answer the questions in this instruction file according to their numbering.
  - Add concise explanations, figures, tables, etc. where needed.
  - You are evaluated for your answers but also for clarity and aesthetics.
  - Plots:
    - Should have suitable titles, axis labels, and legends (if needed).
    - Should have grid lines (except maybe heatmaps).

- **Grading**
  - 20% of the grade is based on your model accuracies and your code:
    - In Part 7 you submit predicted labels of unseen & unlabeled data.
    - You are required to surpass specific accuracy thresholds. We will not publish these thresholds. Aim as high as you can.
    - Remember, some models are simply not suitable for certain tasks. Don’t expect to get good results all the time.
    - Your code may be inspected manually as well, so keep it tidy and well documented.
  - The other 80% is based on your final report. Note that the answers to some questions are dependent on your personal results. Not all questions have a single correct answer.
  - Bonuses - There are several bonus opportunities in this assignment. The final grade cannot exceed 110.
    - Write the assignment in English = 2 pts
    - Classification competitions 1st to 4th places = 4, 3, 2, 1 pts

- **Submitted files**

  One zip file containing:

  - The report PDF file with all your answers.
    - Named `id1_id2.pdf` (with your own ID numbers).
    - Cannot be handwritten.
    - Can be in Hebrew, English, or both.
      - Bonus opportunity: writing only in English = 2 pts
  - Your code:
    - Working with notebooks: a jupyter notebook with all your code.
      - Named `id1_id2.ipynb`.
    - Working with a “traditional” IDE: one clear main python script.
      - Named `id1_id2.py`.
      - Additional files (with classes or functions) which are required for running the main script.
  - Your predictions
    - You are given an unseen, unlabeled dataset `virus_test.csv`.
    - Use your models to predict the labels for each of the three tasks.
      - Save the results in a 4 column CSV file named `pred_<model_type>.csv` for your best model.
        - Column headers should have identical headers to your training data (“`ID`”, “`Virus`”, “`SpreadLevel`”, “`Risk`”).
    - More details in Part 7.

Part 1 – Data preparation

In the previous exercise we cleaned, imputed, transformed, normalized, explored, and reduced our data to the most meaningful representation we could find. We would like to use this improved data to train our classifiers.

In the near future (Part 7), we will receive data that we have never seen, and it will be in the same form as the original data: noisy, feature rich, and containing missing values. Notice that a model trained on a specific subset of features can handle only these features during test time.

In this section, load your cleaned train, validation, and test sets that you saved at the end of the previous exercise. Perform any additional data cleaning that you think is necessary (i.e., you can restore features that you previously deleted etc.). Feel free to play around with your data preparation process while training your models.

NOTE: the following features are crucial to classifying our data for our different tasks. If you removed any of them in the previous assignment, you must bring them back if you hope to achieve high performance. Keep in mind that these are not the *only* features required to truly represent the data distribution.

1. `BMI`
2. `HouseholdExpenseOnPresents`
3. `PCR_7`
4. `PCR_72`
5. `PCR_89`

(Q1) Did you use the validation set for data preparation? Did you use the test set? Explain why or why not.

Part 2 – Evaluation method

Before we rush off to implement every classification model known to man, let us first understand the method by which we measure the performance. We would like this scoring method to be identical for all models so that we can compare them fairly. Furthermore, we would like our evaluation to reflect a model's quality for a given task in the real world.

Accuracy

In class we defined the generalization error to be:

$$L_D(h) = E_{(x,y)\sim D}[\mathbb{I}\{y \neq \hat{y}\}]$$

We will attempt to approximate $1 - L_D$ by calculating the empirical mean of **correct** predictions on unseen data. This scoring method is called **accuracy**.

$$acc\big(h,(X,y)\big) = \frac{1}{m}\sum_{i=1}^{m}\mathbb{I}\{y_i = \hat{y_i}\} = \frac{\#correct\ classifications}{num\ samples}$$

This is a very intuitive metric used in many day-to-day scenarios. For the rest of this assignment, the **best model** out of a list of models is the one that yields the highest accuracy on the validation set.

Note: We will however also use ROC curves in the following parts.

Part 3 – Classification

We are now ready to construct our models. In this section, we will implement several different models that were shown in class.

Every subsection below describes a different classification model. There you will find instructions on how to train and evaluate the model. For each one, repeat the described training process for the following **tasks**:

- Classify the `Virus` – predict the virus on a given sample.
- Classify the `Spread` – predict the spread level on a given sample.
- Classify the `Risk` – predict the risk level on a given sample.

NOTE: you can and should choose different hyperparameters per task.

For each subsection submit:

- Your training code.
- In your report:
  - Loss by hyperparameter plot (train and validation together) for each task.
  - Train and validation accuracies of the final model for each task.
  - Answer the questions at the end of each subsection.

k-NN Baseline

The “k Nearest Neighbors” classifier is often considered the simplest classification model, in that it classifies an example based on previously seen examples that are most “similar” to it.

For this reason, we will use k-NN as our baseline model. The purpose of the baseline model is to have a simple example of how well this task can be solved. We expect our final model to outperform this model significantly!

Similarity can be measured in many ways, but here we will stick with the intuitive Euclidean distance, i.e., example x is more similar to x_1 than x_2, iff $\|x - x_1\|_2 \leq \|x - x_2\|_2$.

Training process:

1. For each number of neighbors $k \in [1, 100]$
    a. Train a kNN model using Euclidean distance with the current k.
    b. Evaluate model accuracies on the train and validation sets (separately).
2. Keep the best hyperparameter, i.e., the one yielding the highest validation accuracy, for later testing (Part 4 and 7).

(Q2) Plot the train and validation accuracies as a function of k.

(Q3) What are the train and validation accuracies of your best kNN model for each task?

(Q4) What is the significance of k? In other words, what happens as k increases/decreases? Why?

(Q5) What is the best k values for each task? How did you determine this?

Decision Trees

Making decisions is hard. Why don't we look at each feature individually and slowly come to a sensible solution based on smaller, simpler decisions?

That's exactly what decision trees do. We will use the entropy criterion we saw in class to threshold the features. We will also tune the model using the tree depth parameter t.

Training process:

1. For each maximal depth $t \in [2, 100]$
    a. Train a decision tree model using entropy with the current t.
    b. Evaluate model accuracies on the train and validation sets (separately).
2. Keep the best hyperparameter, i.e., the one yielding the highest validation accuracy, for later testing (Part 4 and 7).

(Q6) Plot the train and validation accuracies as a function of t.

(Q7) What are the train and validation accuracies of your best tree model for each task?

(Q8) Plot a 6x6 confusion matrix for the `Virus` classification task on the validation set. Use your best performing model (=hyperparameter) for this specific task.

(Q9) Plot the decision tree for `risk` classification with a maximal depth of $t = 7$ (use `sklearn.tree.plot_tree`). According to this tree, which features are the most "important" in predicting risk level? Explain.

(Q10) Repeat the above training process for the 2 most important features from the previous question.

Plot the decision boundaries (see here) twice, once with the training set scattered and once with the validation set scattered (see tutorial 03). Does this model overfit? Underfit? Explain.

(Q11) Plot the ROC curve of your kNN and decision tree models, using the validation set, for "`SpreadLevel`" prediction, where high spread is a positive prediction and medium and low spread are a negative prediction.

(Q12) Based on the ROC curves you plotted in the above question, which model would you choose for classifying high/not high `SpreadLevel` (and why)? On your chosen ROC curve, draw a point where you believe is the optimal TPR and FPR tradeoff. Why did you choose this point? What is the risk of false positives in this scenario?

Support Vector Machine

We have extensively discussed the ins and outs of the SVM classifier in class. Here we will implement a soft-SVM model to try to separate our data.

Training process:

By now you should already see a pattern emerging:

```
for hyp in tested_hyperparameter_values_list:
     # Train model with parameter hyp
     # Record train/validation accuracies
# Plot accuracies as a function of hyperparameters.
# Remember the best hyperparameter for later testing.
```

If you haven’t automated this process by now, you should. This is boilerplate code that you will most likely reuse for other ML projects.

Note: you will want to test many `C` values, some of them small and others very big. We suggest using numpy.logspace to get a wide range of small and big values, Then rerun using numpy.linspace in the region where the best predictions occur.

Technical note: for linear SVM, `sklearn.svm.`**`LinearSVC`** works faster than `sklearn.svm.SVC(kernel=’linear’)`.

(Q13) Plot the train and validation accuracies as a function of the hyperparameter `C`.

(Q14) What are the train and validation accuracies of your best SVM model for each task?

(Q15) Observe the trained weights for each class in your best virus classification SVM model. Which features are most dominant for the “`flue`” (flu) class in the current feature space?

(Q16) Are the different `spread` levels linearly separable? Explain.

Part 4 – Testing your models

VERY IMPORTANT NOTE: do not continue to this section until you have finished implementing the models from part 3 and have chosen their optimal hyperparameters.

Finally, we can let the test set come out and play. Retrain all 3 model types for all 3 tasks on the training set. Use the best hyperparameters you found in Part 3. Evaluate the test accuracies for these 9 models.

(Q17) Draw a table containing the test accuracies for each of your models and for each classification task. Model types to test: knn, decision tree, linear SVM.

(Q18) Which model performed best on the test set for each task? Is it the same model that performed best on the validation set (answer for each task separately)? What does it mean if a model performs well on the validation set but not on the test set?

(Q19) Why is it important that we do not tune our model using the test set?

Part 5 – Non-Linear SVM

Looking at the results of SVM, we can tell that the different viruses are far from linearly separable in the current feature space. In this section, we will reduce our feature set to 3 dimensions and explore kernels and non-linear features that will help us separate the `COVID` and `CMV` viruses.

Data prep and viz:

1. Create subsets of your training and validation sets containing only the patients afflicted with `COVID` or `CMV`.
2. Create a 3-dimensional plot of the entire training set containing only the features `PCR_7`, `PCR_72`, and `PCR_89`. Mark `COVID` in red and `CMV` in blue (and add a plot legend!).

(Q20) Describe the plot and the relationship between the `COVID` and `CMV` viruses in this feature space. Provide several angles of the 3D plot from step (2) that best complement your descriptions.

Training process:

1. Find a non-linear feature mapping or kernel that will allow you to separate `COVID` patients from `CMV` patients in the new 3D feature space. Feel free to use your very own custom kernels or feature mappings.
2. Train an SVM model on this feature space using the kernel or feature mapping you chose in the previous step. Remember that the training process includes tuning hyperparameters using the validation set, including your kernel/feature mapping's hyperparameters if it takes any.

   NOTE: One way to plot the accuracies as a function of 2 hyperparameters is using a 2D heatmap.
3. Not satisfied? Go back to step (3).

(Q21) What kernels and/or feature mappings did you try? Which did you choose and why do you think it is the best non-linear representation for this task? Were you able to make the data linearly separable?

(Q22) Plot train and validation accuracies as a function of the hyperparameters.

(Q23) Using your best non-linear SVM model (with the best combination of kernel/feature mapping and hyperparameters), what are the training and validation accuracies?

Part 6 – Custom models challenge (Bonus)

This is the part where you are given the freedom to choose your own models. Train a classification model (any model) from the sklearn library only. You can train different models per task. Be sure that you are using the best feature space you possibly can for each task (yes, they can be different for each task). You may use any non-linearity or feature mapping you can imagine.

Competition

This section is for extra credit. For every one of the 3 tasks, the 4 teams that achieve the highest accuracy will receive 4, 3, 2, and 1 bonus points respectively for this exercise, meaning **your team can accumulate up to 12 bonus points**!
You may participate in any and all of the 3 tasks (“`Virus`”, “`SpreadLevel`”, “`Risk`”).

Teams who choose to participate in this competition must perform the full training process and provide all the results in their final report, including the model they chose, the hyperparameter tuning graphs, and the final results. If your model includes several hyperparameters, you must find a way to plot the tuning results in an informative way. If you chose to use a model that we have not yet reviewed in class, you must provide a brief description of the model, how it works, how the hyperparameters affect the model, and why it is a good model for the task. Teams that fail to show a competent understanding of their models and the results will be disqualified!

Participants must submit the following for each task they participate in:

- Your training code.
- In your report:
  - Explain your model selection process in detail.
  - Include all plots and visualizations necessary to explain your model choice.
  - The train, validation, and test accuracies of your best model.
- The prediction file `pred_custom.csv` in Part 7.

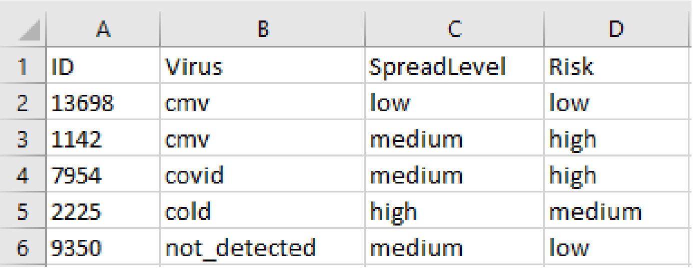
Part 7 – Submitted model predictions

For each task and model you implemented, aside from your code and results, you are expected to submit a single CSV file containing your predictions on *`virus_test.csv`*. Notice that this unseen dataset requires the same preparation as the rest of your data.

Before performing any predictions, train your models on the **entire dataset at your disposal**, including the test set. Now that we've selected and tested our models, there is no harm in using as much data as we can to best predict any unseen data.

For each implemented model type (kNN, decision tree, linear SVM, custom models), submit a file named `pred_<model_type>.csv`. The files should consist of 4 columns with the following headers: "`ID`", "`Virus`", "`SpreadLevel`", "`Risk`". Every row should have the original ID of the predicted input and your model's prediction for each task.

Example for a prediction file (actual file should have 601 rows including the headers):



	A	B	C	D
1	ID	Virus	SpreadLevel	Risk
2	13698	cmv	low	low
3	1142	cmv	medium	high
4	7954	covid	medium	high
5	2225	cold	high	medium
6	9350	not_detected	medium	low

Submit the following files:

- `pred_knn.csv` (mandatory) - best kNN models' predictions.
- `pred_dt.csv` (mandatory) - best decision tree models' predictions.
- `pred_svm.csv` (mandatory) - best linear SVM models' predictions.
- `pred_custom.csv` (bonus competition) - best custom models' predictions.
    - Submit this file only if you wish to participate in the challenge.
    - This should be a (N+1)-columns csv with headers "ID" and the other N out of 3 tasks you chose to participate in ("`Virus`", "`SpreadLevel`", and "`Risk`") containing the predictions for *`virus_test.csv`* of your best model for each task under the appropriate header.

NOTE: do not submit predictions with a false model, e.g., `pred_knn.csv` with decision tree predictions. We know the limitations of the models. Radically positive results will be inquired.