HW2: Algorithm Implementation and Basic Model Selection

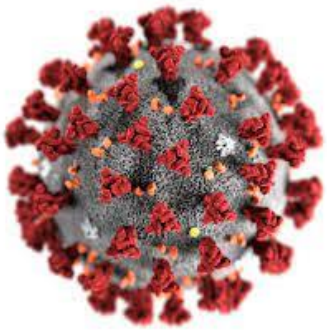
Goal

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 two algorithms that we learned: k-NN and Soft-SVM. Moreover, you will practice basic hyperparameter tuning (model selection) using three algorithms: k-NN, ID3, and Soft-SVM.

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

Good Luck!



Instructions

- **Submission**
  - **Submit by:** Thursday, 30.12.2021, 23:59.
    - Late submissions will be accepted until 02.01.2022 and penalized 5pts per day.
  - Submissions in pairs only in the webcourse.

- **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).

- **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:
      - *kNN.py* : your completed kNN module (=class) from Part 1.
      - *SoftSVM.py* : your completed SVM module (=class) from Part 3.
      - *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.

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`, `PCR_02`, `PCR_03`, `PCR_07`, `PCR_10`, `num_of_siblings`, `sugar_levels`, and `household_income`. Note that there are other important features for this assignment and the next. 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 you deleted the `Job` feature.

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

Part 1: k-Nearest Neighbors (model implementation and selection)

We will start our first step of defeating COVID with one of the simplest models we know, "k-Nearest Neighbors". K-NN classifies an unseen datapoint based on the k training points "closest" to it. In this part we will implement k-NN that uses the Euclidean distance and use it to build the complete learning pipeline.

Implementation

Our first step is to implement a basic k-NN classifier. We will inherit the `BaseEstimator` class from sklearn for compatibility with scikit-learn API. We will also inherit `ClassifierMixin` which will automatically add accuracy scoring function to our model.

Task: Implement k-NN estimator using the code template below (don't change method signatures):

Tip: Read about `scipy…cdist`, `np.copy`, and `np.argsort` (or even better: `np.argpartition`). Avoid using `for` loops.

```
from sklearn.base import BaseEstimator,ClassifierMixin

class kNN(BaseEstimator, ClassifierMixin):
    def __init__(self, n_neighbors:int = 5):
        self.n_neighbors = n_neighbors

    def fit(self, X, y):
        # TODO: complete
        return self

    def predict(self, X):
        # Note: You can use self.n_neighbors here
        predictions = None
        # TODO: compute the predicted labels (+1 or -1)
        return predictions
```

Verifying your implementation

Before moving on, we wish to perform a sanity check to our model.

We will do this by plotting the decision boundaries for a toy data set and making sure they meet our expectations

Task: Copy the function from the given *visualize_clf.py* into your notebook / project.

(Q1) Use the following code snippet to generate a toy dataset. Use the sum of the last digit of your i.d and your partner's i.d for `random_state` (as in HW1).

```
X_toy, y_toy = make_classification(n_samples=100, n_features=2,
                                   random_state=TODO, flip_y = 0.1,
                                   n_informative=2, n_redundant=0)
```

Use the provided `visualize_clf` function to visualize the decision boundaries of your model. To your report, attach the plots for `k=1` and `k=9` on the generated set. Compare these two learned models from a learning perspective (e.g., overfitting vs. underfitting).

Some more exploration

(Q2) The following code snippet computes the correlation between `spread` label and the rest of the features. Attach to the report the 10 most correlated features to `spread`.

```
s = df.corr().spread.abs()
s.sort_values(kind="quicksort", ascending=False)
```

(Q3) Plot a 3d scatterplot using the `PCR_03`, `PCR_07`, and `PCR_10` features (train set only). Use two different colors to color the points according to their `spread` class.
Add the plot to your report.
Reminder: all your plots should have suitable axis labels and titles.

(Q4) Describe the relation between `spread` and the above features. Describe the decision boundary mathematically (you may denote some unknown constants and use them in the formula).

Create a temporary `DataFrame` containing only `PCR_03`, `PCR_07`, and `PCR_10`.

(Q5) Train your `k-NN` model with `k=11` on the temporary `DataFrame` to fit the `spread` class. What is the training accuracy?

Tip: use `kNN.score` to evaluate the accuracy of your model.

Data Normalization

The data we deal with is often not organized optimally for learning algorithms. That is why we transform it to better capture the information ingrained within. In line with what you have seen in the previous assignment, there are also many techniques for data transformation. In this section, we will focus on two normalization techniques: Standardization (Z-score) and min-max scaling (implementations here and here).

(Q6) Briefly explain the difference between Z-score and min-max scaling. In your answer, consider their effect on a feature's distribution and explain when we might choose one transformation over the other.

(Q7) Normalize `PCR_03`, `PCR_07`, and `PCR_10` in the temporary `DataFrame` you created. Train a new k-NN model ($k = 11$) on the normalized `DataFrame` like in (Q5).
What is the updated training accuracy? Compare your results to (Q5) results. Can you explain the difference in accuracies?

By now we understand data normalization is an important step in the ML pipeline for several models. To ensure we get best results in our experiments, add a normalization step to the data preparation pipeline from HW1 (i.e., `prepare_data` function). The choice of normalization method for each feature should be based on the data exploration you performed in HW1.

(Q8) Choose two features that you decided to normalize (other than `PCR_03, 07, 10`), one with Z-score and one with min-max scaling and explain (in no more than 4 sentences each) why you chose that normalization method for each of these two features. In your report, include the histogram plots of these features before and after the transformation.

Model selection (hyperparameter tuning)

Most ML models are characterized by a set of parameters that control the learning process which are not optimized during training (e.g., `k` in k-NN and `max_depth` in decision trees). As we saw in the tutorials, these hyperparameters can change the learning dynamics dramatically. Tuning hyperparameters is done using k-fold-cross-validation, where we split the training data into k_v folds – each time we train on $k_v - 1$ folds and use the last fold for validation (i.e., performance evaluation).

This procedure gives us an estimation of the model performance on unseen data; thus, we can estimate the optimal hyperparameters.

Note: do not use the test set for hyperparameter tuning.

(Q9) Use sklearn.model_selection.cross_validate to find the best k (neighbours) value in `range(1,61,2)` for predicting the `spread` class with the 3 features we isolated in the normalized temporary `DataFrame` from (Q7).

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.

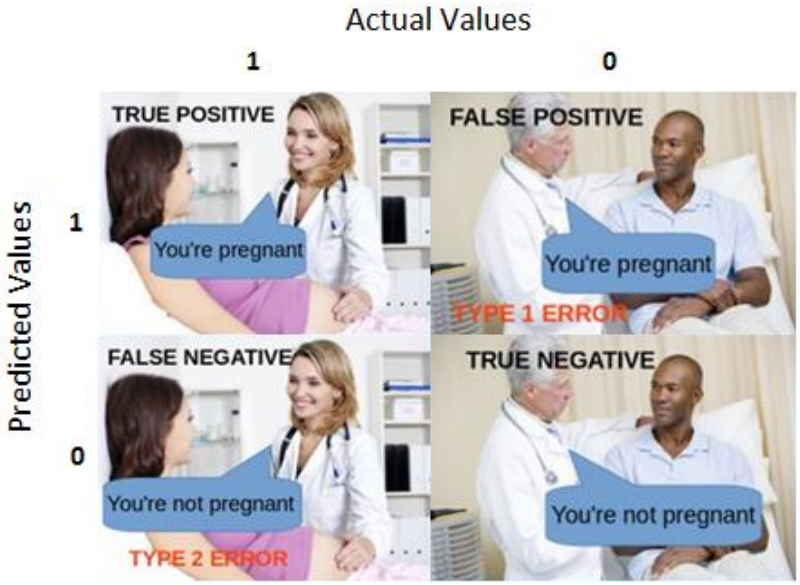
Explain: which k value is the best? What are its mean training and validation accuracies?

Submit: Copy your completed `kNN` module (=class) into a separate file called *`kNN.py`*.

Submit this file at the end of your work.

Error analysis

The accuracy measure gives us a basic idea on the model performance. However, to better understand the model behavior we will use confusion matrix. A confusion matrix compares the predictions of the model against the actual labels of the data.



Using confusion matrices, we can better understand the types of mistakes our model makes. We wish to create a confusion matrix for a model trained using the best value of the hyperparameter (i.e., k) that you found earlier. Remember to train using only the 3 useful features that you used earlier.

It is too early to use the test set, so instead we will use only the training set and create cross-validated estimates (using 8-folds) for each training point. See the following answer from stack overflow for a code example.

(Q10) Using the optimal hyperparameter value, compute a cross-validated 2×2 confusion matrix for the `spread` class. Fill and attach an appropriate table to your report. Does your model tend more to false positives / false negatives?

Actual labels

Predicted		**Positive**	**Negative**
	Positive		
	Negative		

Part 2: Decision trees (model visualization and selection)

In this part we will focus on predicting the `risk` 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

(Q11) Return to your report / notebook for HW1. Based on your analysis in HW1, name 3 features that seemed important for predicting the `risk` class. Attach the corresponding plots that made you believe these features are indeed important.

(Q12) Compute the 10 most correlated features to the `risk` class and attach them to your report.

(Q13) Train a model with ID3 and `max_depth=4` (not including the root level).
What is the training accuracy? Visualize the trained tree using `plot_tree` (provide appropriate `feature/class_names`) and attach the plot to your report.

(Q14) Is the "importance" of the features you named in (Q11) noticeable from the correlation computation in (Q12)?
Were these features used by ID3 in the decision tree built in (Q13)?
Compare and discuss your findings.

Model selection

It is time to search for the best tree to fight covid!

In the DecisionTreeClassifier documentation, read about the `min_samples_leaf` argument and make sure to understand how it can help prevent overfitting.

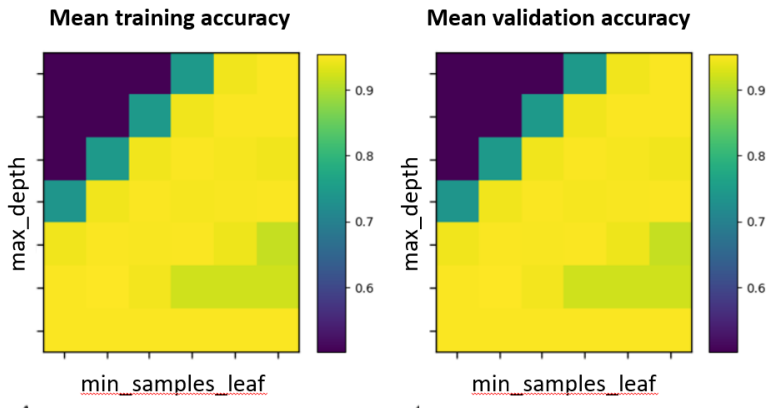
You will now tune on two hyperparameters simultaneously – both `min_samples_leaf` and `max_depth`. You need to look for the combination of these two hyperparameters that lead to the best validation performance. There are many approaches for tuning multiple hyperparameters, and here we take the grid search approach (shortly explained here).

(Q15) 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 `risk` class.

You should:

a. Choose appropriate ranges for both hyperparameters.
   This may require a few attempts.

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 two plots to your report and explain which hyperparameter combination is optimal.

d. Write a hyperparameter-combination that causes underfitting.

e. Write a hyperparameter-combination that causes overfitting.

Part 3: SVM (implementation and optimization)

In this part we will implement a Soft-SVM classifier. 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** subgradient 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 subgradients $\nabla_w p_C(\boldsymbol{w}, b)$ 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.

Verifying your implementation

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** subgradient by $\boldsymbol{u}_\delta(\boldsymbol{w}, b) = \begin{bmatrix} u_1 \\ \vdots \\ u_d \end{bmatrix}$.

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

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.

(Q16) Using the **normalized** dataset and `virus` as our target, generate a plot that compares the numerical gradients to the analytic gradients.
Do this by running the following command:

```
compare_gradients(X_train, y_train, deltas=np.logspace(-5, -1, 9))
```

Important: `X_train` should hold the features of your **normalized** training set.
`y_train` should hold the `virus` training labels, using the `{+1,-1}` convention (rather than `{1,0}` or `{True,False}`).

In your report: Attach the resulting plot.
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 iterative algorithms is by plotting a *learning curve*, i.e., plotting the accuracy and/or loss of the model over time (i.e., steps).

(Q17) Use the following code snippet to train a `SoftSVM` model on the `virus` class (make sure you use the entire normalized training set and `{+1,-1}` labels) and plot a learning curve. You are allowed to slightly change this code if you need to.
Make sure your model converges. For that, you may need to adjust the `lr` and `max_iter` values. However, you may not change the `C` value.
Make sure to add proper labels for the x-axis and both y-axes.
Add a suitable title and a legend. Finally, add the plot to your report.
What is the maximal training accuracy achieved by your model?

```
clf = SoftSVM(C=1e2, lr=1e-5)
losses, accuracies = clf.fit_with_logs(X_train, y_train, max_iter=5000)
fig = plt.figure(figsize=(12, 7))
ax1 = fig.add_subplot(111)
line1 = ax1.semilogy(losses, c='b', label='?')
ax2 = ax1.twinx()
line2 = ax2.plot(accuracies, c='r', label='?')
ax2.grid(alpha=0.5)
plt.show()
```

Submit: Copy your completed `SoftSVM` module into a separate file called *`SoftSVM.py`*.
Submit this file at the end of your work.

Applying a feature mapping

(Q18) Next, we want to train our model to predict the `spread` using the 3-features-dataset from (Q7). Based on the previous analysis, explain why your `SoftSVM` model cannot compete with your `kNN` model (on the `spread` label).

(Q19) Apply a 2nd degree PolynomialFeatures transformation on the 3-features-dataset.
Use 8-folds cross-validation (like in (Q9)) to tune the learning rate, i.e., find the best learning rate in `np.logspace(-10, -1, 10)`. Keep `C=1000` and `max_iter=2000` fixed.
Attach a suitable *validation curve* to your report (plot both train and validation accuracies).
Fully discuss and explain the behavior demonstrated in the plot (from an optimization perspective and a learning perspective).

(Q20) What are the differences between what we did in (Q19) and solving the Kernel SVM problem using the 2nd degree polynomial kernel?

In part 1 you trained a kNN model for predicting the `spread` label using 3 features.
Here, you trained an SVM model for the same purpose using a feature mapping of the same features.

(Q21) Compare the space (=memory) requirements of the two final models (after training ends).

Submitting the files

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