HW1 – Data Exploration and Preparation

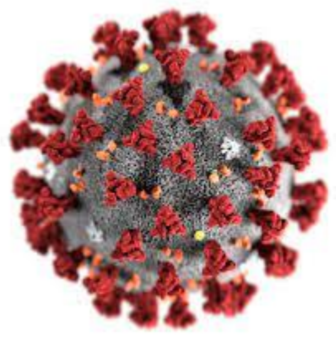
Goal

This exercise is the first of three mini-projects that will guide you in your task of stopping the spread of disease around the globe.

Here we present the **Virus Test Challenge Dataset (VTC)** containing labeled information from patients suffering from all sorts of diseases. Your goal is to understand this dataset and prepare it for prediction.

While we will soon learn several methods for training prediction models, none of them will work without us first observing, understanding, and cleaning our data. So, in this exercise you are asked to perform standard data preparation practices, which will push you towards understanding regularities, and especially irregularities, in your data.

Good Luck!



Instructions

- **Submission**
    - **Submit by:** Sunday, 08.05.2022, 23:59
        - Late submissions will be penalized 5 points per day, up to 2 days.
    - Submissions in pairs only.
    - Submitted on 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 are allowed to 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.
    - May 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 and aesthetics.
  - Tables should include feature names and suitable titles.
  - Plots:
    - Must be clear, readable, and coherent.
    - Should have suitable titles, axis labels, and legends (if needed).
    - Should have grid lines (except maybe heatmaps).
    - We recommend adjusting the default font sizes of matplotlib at the beginning of your notebook. You can use the following code snippet:

```
from matplotlib import pylab
params = {'xtick.labelsize': 18,
          'ytick.labelsize': 18,
          'axes.titlesize' : 22,
          'axes.labelsize' : 20,
          'legend.fontsize': 18,
          'legend.title_fontsize': 22,
         'figure.titlesize': 24
         }
pylab.rcParams.update(params)
```

- **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 (choose the relevant options for you):
    - Working with jupyter: a notebook with your code, *id1-id2.ipynb*.
    - Your completed kNN module (=class) from Part 2, *kNN.py*
    - Working with a “traditional” IDE: one clear main script, *id1-id2.py*, and any additional files required for running the main script.
    - Data preparation function *prepare.py* (from Part 6).
  - Do not submit csv files.

Part 1: Data Loading and First Look

The VTC dataset is available on the course website under the name `virus_data.csv`. You should load the CSV file and explore it using the pandas library. There you will find many features that are both interesting and relevant for our prediction tasks, along with their ground-truth labels for our target variables: **`covid`** (whether a patient is afflicted with COVID-19), **`spread`** (the patient's potential to spread COVID-19), and **`risk`** (whether a patient is at risk for serious illness). All your decisions in the data preparation process should be made with these targets in mind.

Unfortunately, as with any real-world dataset, VTC includes many redundancies, missing data, and plain garbage. These irregularities will surely harm your models' performance in the assignments. Throughout this exercise, we will try to minimize these unwanted properties in our data. To do so, we should understand what features are available to us.

Note: The dataset is completely synthetic (i.e., made up).
It is possible that some statistics do not match statistics in the real world.

(Q1) Load the dataset into a Pandas `DataFrame`.
Answer (in your report): how many rows and columns are in the dataset?

(Q2) Print the `value_counts` of the `num_of_siblings` feature (see Tutorial 01). Add the obtained output to your report. Moreover, describe in one short sentence what you think this feature refers to in the real world. State what type (continuous, categorical, ordinal, or neither) you believe this feature to be. Explain.
Note: Ordinal variables are categorical with a natural order (e.g., year of birth).
Remember to clearly write the number of the question next to your answer.

(Q3) In your report, write a table describing each feature. The columns must be:

a. Feature name: the name of the feature as it is written in the dataset.
b. Description: a short description (1 short sentence) of your understanding as to the feature's meaning in the real world.
c. Type: Continuous, Categorical, Ordinal, or Other.

NOTE: do not to include the target columns ("`covid`", "`spread`" and "`risk`").

(Q4) For each feature you marked as type "Ordinal" or "Other" in the previous question, explain why you think that is the correct typing for them (no more than 2 sentences per feature).

Partitioning the data

During the learning process, we measure our models' performance on two disjoint sets: **training** and **test**. A training set is a subset of the dataset from which the machine learning algorithm learns relationships between features and target variables. The test set provides a final estimate of the machine learning model's performance after it has been trained. Test sets should never be used to make decisions about which algorithms to use or for improving or tuning algorithms.
Note: later in the course, we will use another data subset, called the validation set.

Here we will split our full dataset into a training set containing a random sample of 80% of the data, and a test set containing the remaining 20%.

We will explore why this data partitioning is important later in the course (when we have models to evaluate), but for now the most important thing to remember is that **you may only use the training set when making decisions about the data**.

(Q5) Split the data into training and test sets. Take 80% of the data as the training set and the remaining 20% as the test set. As the `random_state`, use the sum of the last two digits of your i.d and your partner's i.d[1].
The random state will ensure that you get the same split every time. Why is it important that we use the exact same split for all our analyses?

Note: it could be easier for you to answer this question after you completed the rest of the assignment.

In the following, perform the data preparation actions ONLY on the training set and leave the test set untouched (for now).

[1] i.e., if my i.d is 200033035 and my partner's is 300011016, then the random state should be 35+16=51.

Part 2: Warming up with k-Nearest Neighbors

In this part, we focus on the `risk` target variable and start with one of the simplest models we know, "k-Nearest Neighbors". We will use this part as a warmup for the rest of the assignment.

Reminder: we use only the training set for now.

Basic data exploration

A medical expert suspects the `risk` is largely determined by `PCR_04`, `PCR_08`, and `PCR_10`.

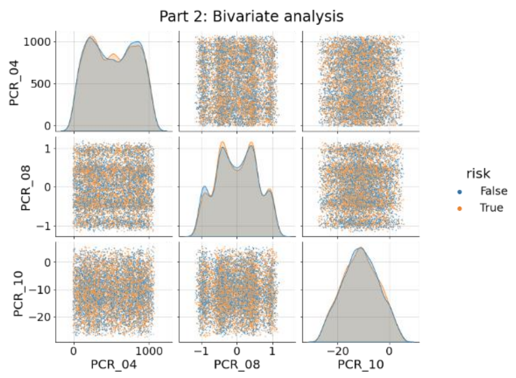
(Q6) Compute the correlation between `risk` and each of 3 aforementioned PCR features (see Tutorial 01). Attach the correlations to your report.
Are these features highly correlated with the `risk`?

Task: Create a `seaborn.pairplot` of the 3 aforementioned PCR features. Use the `hue` parameter to color the different (train) data points according to their `risk`. Make sure your plots have proper titles, grid lines, axis labels, etc. Following is a code snippet that can help you start, and an example for the resulted figure (with different data):

```
g=sns.pairplot(TODO, plot_kws={"s": 3})
g.fig.suptitle("TODO", y=1.04)

for ax in np.ravel(g.axes):
  ax.grid(alpha=0.5)

g.fig.set_size_inches(12,8)
```



(Q7) Attach the figure to your report.
According to your figure, which 2 of the 3 features can be useful to predict the `risk`?
How does your answer here settle with the answer for the previous question? Explain briefly.

k-NN 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 better: `np.argpartition` [1],[2]). Avoid using `for` loops, `list`, `map`, `lambda`, etc.

```
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
```

We will now test your implementation.

Task: Create a temporary `DataFrame` by taking only the two features you chose in (Q7) from the training set. Train two `kNN` models on this subset – one with `k=1` and one with `k=9`. Use the provided `visualize_clf` function to visualize the decision boundaries of both models.

Compute the training accuracy of both models by calling their `score` methods, e.g., call `h.score(Xtrain, Ytrain).`

(Q8) Attach the two figures to your report. Specify the training accuracies.

a. Which model has a better training accuracy? Explain briefly.

b. According to the decision regions, which of the two models seems better in a generalization sense? Explain briefly.

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. We will focus on two normalization techniques: Standardization (Z-score) and min-max scaling (implementations here and here).

(Q9) Use min-max scaling (between -1 and $+1$) to normalize the two features in the temporary `DataFrame` you created before and train a new kNN model ($k = 9$) on the normalized dataset.
Compute the updated training accuracy and draw the updated decision boundaries. Attach the results to your report and compare them to those from **(Q8)** for the same $k = 9$ model on the raw data. Use these results to explain why normalization is important for nearest neighbor models.

Note: The following question is general and does not deal with the given dataset.

(Q10) Assume one of the features in a dataset is randomly sampled (i.i.d.) from $\mathcal{N}(0,1)$, and assume the distributions of the other features are unknown.
Why is normalizing the normally-distributed feature using min-max scaling a bad idea? (Think about kNN's performance.)

We are now ready to start the preprocessing stage using the rest of the features. For the rest of this assignment, we consider the problem of predicting the `risk` target variable as solved and focus only on the `covid` and `spread` target variables.

Part 3: Data Exploration

We've already taken a first look at our features and understood their purpose and typing. We now dive deeper into our features. Our first step will be to extract as many features as we can from the data, and then explore every one of them individually.
The learning algorithms we learn in this course often only accept numbers as inputs. Thus, we must transform non-numeric features using meaningful numeric representations.

(Q11) The `blood_type` feature has several categories that are combinations of A, B, AB, O characters and the +,- symbols. These string values are meaningless in terms of separation. A common solution is to use one-hot-encoding (OHE). This is a bitmap indicating the one single category to which the sample belongs. For instance, three categories ("X", "Y", "Z") are encoded by three Booleans into (001,010,100). Pandas provides us with this functionality. Use it to generate an OHE to replace the `blood_type` feature. What is the length of this vector?

During this exercise, you will extract new information out of existing features. If that data is categorical (with a reasonable number of categories), consider transforming it to an OHE vector.

(Q12) In **(Q3)** you found features that are neither categorical nor continuous. One of them should have been `symptoms` because it holds string values with multiple categorical values per entry. Can we extract information from this feature that may be useful for our prediction task, i.e., can we craft new features using the `symptoms` feature that are more informative? If so, add these newly crafted features to your `DataFrame` and briefly describe the extraction method you used in your report. If not, explain why that is (2-3 sentences).

(Q13) For each feature that you classified as "other" in **(Q3)**, determine whether useful information can be extracted from this feature. If so, craft new features and add them to your `DataFrame` as you did in **(Q11)**. For every new feature added to the dataset, explain (in 1-2 sentence) why you think this feature is important. Furthermore, transform any other categorical data into OHE vectors.

At this stage, you should make sure you are left with only numeric features (i.e., integers or floats).

You will now carry out most of the univariate analysis in your notebook (or IDE). You should not add all the plots to the report, only the ones we specifically request.

For each feature (including extracted features), plot two histograms, one for each of the remaining target variables (`covid` and `spread`). In each histogram use the `hue` keyword to "split" the histogram by the target variable's value (e.g., high/low spread potential). For continuous/ordinal features you should also use the `kde` keyword to draw the estimated distribution curve (see Tutorial 01).

The following code example generates a 2-column figure of histograms of the features in the `COL_NAME` list. You may use this as a template to generate meaningful plots. Refer to the seaborn documentation to understand more on `histplot`'s keyword arguments. We encourage you to explore these options deeply as they will help you generate informative graphs.

```
COL_NAME = ['blood_type', 'num_of_siblings']
COLS = 2
ROWS = len(COL_NAME)

plt.figure(figsize=(5 * COLS, 4 * ROWS))
for row in range(ROWS):
  column = COL_NAME[row]
  plt.subplot(ROWS,COLS, row * COLS + 1)
  sns.histplot(data=df, x=column, hue="covid", line_kws={"linewidth": 3},
               kde=("float" in df[column].dtype.name))
  plt.grid(alpha=0.5)

  plt.subplot(ROWS,COLS, row * COLS + 2)
  sns.histplot(data=df, x=column, hue="spread", line_kws={"linewidth": 3},
               kde=("float" in df[column].dtype.name))
  plt.grid(alpha=0.5)

plt.tight_layout()
```

To clarify: in your jupyter notebook you should generate 2 histograms for every feature. Each histogram corresponds to one target feature (`covid`, `spread`), where the different labels are counted separately and colored differently. Continuous variable histograms should also have estimated distribution curves (using the `kde` argument).

(Q14) In your report, add the two histograms you got for `PCR_05` and the two histograms you got for `PCR_07` (only). Can these features help predict `covid` / `spread`? Describe these findings in detail (5-6 sentences).

We will now perhorm some bivariate analysis.

(Q15) 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)
```

(Q16) Relying on <u>all</u> the above, find a combination of <u>two</u> features that best explains the `spread` label. You should use bivariate-analysis plots like `pairplot` or `jointplot`. In your report, explain your process of finding these two features and attach a proper `jointplot` demonstrating their relation to the `spread` label (use the `hue` parameter, make sure the marginal distributions are shown, and that the main plot has suitable grid lines).

Part 4: Preparing Data

Missing data

As you can see, there are many missing values for multiple features. Missing data is a common phenomenon in machine learning that can cause all kinds of problems. There are several strategies for dealing with missing data, including:

- Removing data points (examples) with missing features
  - Very extreme and **should not be done in this exercise**.
- Adding features that indicate '*data is missing here*'
  - Could be useful when the reason the data is missing is informative (such as missing grades for students that do not submit homework).
  - Letting the ML model use it if it wants to, ignore it otherwise
- **Imputation** (the assignment of a value to an attribute by inference)
  - Making assumptions that allow educated estimation of missing values from data.
  - Such assumptions are usually statistical in nature, such as assuming data come from a specific distribution.

In this section, you will substitute missing values with imputed data.

This blog post contains descriptions and examples of basic imputation techniques for both continuous and categorical features.

(Q18) Give one advantage and one disadvantage in choosing median imputation for continuous variables (1 short sentence each).

(Q19) What do we gain when we impute missing categorical data using the "missing category" technique?

(Q20) Analyze `happiness_score` and perform imputation on this feature (do not use "missing category"). What imputation technique did you choose and why do you think it is suitable in this case?

(Q21) Impute all missing values in the dataset (do not use "missing category"). In your report, show univariate analysis visualization before and after imputation for the features `sex` and `weight`. In (Q28) you'll be asked to specify the imputation method you chose for each feature. Note that not all features have missing values!

Data Normalization

We will now complete the normalization process for all numeric features (notice that all features should be numeric at this point).

Task: Use the univariate analysis above to choose an appropriate normalization method (see Part 2) for each feature in your `DataFrame`. Accordingly, apply sklearn's StandardScaler and MinMaxScaler on all features.

In (Q28) you'll be asked to specify the normalization method you chose for each feature.

(Q22) In your report, show univariate analysis visualization before and after normalization for the `sugar_levels` feature.

Part 5: Feature Selection

Inserting non-informative features into our learning algorithms can harm the learning process and introduce unnecessary noise. To prevent this, we commonly look for a subset of features that are most informative for the task we aim to solve. The process of eliminating features is called **feature selection** or **feature pruning**.

Feature selection is a Search / Optimization problem, where the goal is to find an informative feature subset. Such a subset can either be the "optimal" subset, but we often aim for a "good enough" subset. Generally, finding an optimal feature set for an arbitrary target concept is NP-hard. There is a need for a measure for assessing the "goodness" of a feature subset (scoring function) and/or a good heuristic that will (effectively) prune the space of possible feature subsets and will guide the search.

We will now briefly experience with finding redundant features.

Manual Feature Selection

Create a temporary `DataFrame` with only the 10 PCR features only (see how to filter here). Use the given `plot_corr_matrix` function to plot a correlation matrix of the 10 PCR features. Also create a `pairplot` of these 10 features (try `plot_kws={"s": 2}` to make the scattered points smaller). No need to attach the plots to your report.

(Q23) According to the `pairplot`, which PCR feature interacts most strongly with `PCR_01`? What is the correlation between these two features? Does this make sense? Following these observations, can we remove one of these two features from our dataset (when our goal is to predict some target variable)? Explain.

(Q24) Find a PCR feature that is (highly) correlated with two other PCR features. Use `plot3d` to plot these three features in 3d. Attach the plot to your report.

a. Can you detect a structure in these features?

b. When using a linear classifier (to predict some target variable), does it make sense to remove some of these 3 features? If so – which should be removed and why? If not – why?

Automatic Feature Selection

We distinguish between three different techniques to perform automatic feature selection:

- **Filter methods:** Rank feature subsets independently of a classifier using statistical tools (uni/bi-variate analysis, correlation, and many more).
- **Wrapper methods (sequential feature selection):** Use classifiers to assess feature subsets.
- **Embedded methods:** Perform variable selection (explicitly or implicitly) during training (e.g., like in decision trees; also later in the course).

We will now perform a basic feature selection process.

Read sklearn's short explanation on sequential feature selection.
Use forward SequentialFeatureSelector (see example within) to find two features that (supposedly) best predict the `spread` label. Like in the example, use sklearn's `KNeighborsClassifier` (with 5 neighbors) model as a classifier for the selection process (do not use the `kNN` model you wrote earlier). We do not recommend changing other parameters on this assignment (specifically, do not alter the `scoring` or `cv` parameters).

(Q25) What are the two features that the selection process found? Are these the same features you found manually in **(Q16)**?

(Q26) Generally, why is it important to perform the imputation and normalization steps before performing sequential feature selection?

Final Feature Selection

At this point, you know everything you need to properly clean your data. In the following questions, you will be required to use your newly acquired skills to conduct a meticulous analysis of your data and complete the feature selection processes.

You are required to select features that are informative for predicting `spread` / `covid`.
Note that we already found two suitable features for predicting `risk` at (Q7). Make sure you keep these two features.

Here are a few tips and guidelines:

- Make sure you imputed all missing data.
- Do not leave raw features with values that are not meaningful to the algorithms (e.g., strings).
- Use the univariate analysis you performed in Part 3.
- Scatter plots in bivariate analysis are often the most revealing
  - Seaborn's `pairplot` is a great starting point for bivariate analysis. This function also has many advanced options that we suggest you explore (especially the `vars`, `hue`, and the `kind` keywords).
  - Play around with seaborn's `jointplot` function with different `hue` splits for target and non-target features and explore other keyword arguments.
    - Use the marginal histograms on the axes to better understand feature importance.
- You can use manual or sequential (forward or backward) feature selection like we did earlier.

(Q27) Complete the feature selection process using techniques presented in class and in this exercise according to the above guidelines. In your report, describe all interesting findings and your conclusions from those findings. Add plots to support your claims. Your answer should be 1-3 pages long including (reasonably sized) plots.

No need to exaggerate – this part should take at most 1-2 days
Extra points will be given to few submissions having interesting and original analyses.

(Q28) Write a table summarizing the data preparation process you created.
The columns of the table must be:

a. **Feature name**: the name of the feature as it is written in the dataset.
Names of new features should be meaningful!
b. **Keep**: "V" if the feature is kept, "X" otherwise.
c. **New**: "V" if the feature was handcrafted using other feature(s), "X" otherwise.
d. **Imputation method**.
e. **Normalization method**.
f. **Explanation (reason)**: 1-3 short sentences describing why you made these choices for this feature.

NOTE: do not to include the target variables "`covid`", "`spread`", and "`risk`" in the table.

Part 6: Data Preparation Pipeline

Now that we have finished exploring, cleaning, and pruning our data, it is time to create a data preparation pipeline that will allow us to transform any incoming data point into a "clean" data entry ready for prediction.

Remember that this pipeline **should only reflect the training set**. For example, assuming you imputed some feature with the median value of the training set, then the pipeline should impute missing data from the test set using the same median value, i.e., use the training set median as the "filler value" for all missing data in that feature for all subsets.

Task: Write a python module[2] called `prepare.py` containing a single function with the following signature:

```
def preprare_data(data, training_data)
```

The `data` parameter is the `DataFrame` to be cleaned and `training_data` is the training set `DataFrame` used during data exploration. Your function should perform the data cleaning process as summarized in **(Q28)**, including normalization and imputation. The output is a copy of `data` (the original parameter should remain unchanged), after it has been cleaned according to the provided `training_data`.

If your function uses stochastic steps, make sure to set a seed at its beginning.

You are required to submit `prepare.py`.

Apply the function to both the train and test sets like so:

```
# Clean training set according to itself
train_df_clean = prepare_data(train_df, train_df)

# Clean test set according to the raw training set
test_df_clean = prepare_data(test_df, train_df)
```

Save your two clean dataframes as CSV files and keep them for the next assignment.

Do not submit any CSV files!

[2] If you are using jupyter notebook or Colab and have issues importing external modules, you can simply write the function in your notebook and copy it later to the `prepare.py` file using your preferred text editor. Do not forget to copy the relevant `import` statements that are required for your function to run.