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!

Source: xkcd

Instructions

- **Submission**
  - Submit by Tuesday, 20.02.24 (23:59) in groups of 2-3 students.
  - 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 (בדיקה מדגמית).

- **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.**
    - Will be partially reviewed and graded (בדיקה מדגמית).
  - Answer the questions in this instruction file according to their numbering.
  - Add concise explanations, figures (outputs of your code), tables, etc.
  - 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)
```

  - You are evaluated for your answers but also for readability, clarity, and aesthetics.
- **Submit a zip file containing** (please use hyphens, not underscores):
  - Define *<filename>* as your dash-separated IDs, i.e., *id1-id2* or *id1-id2-id3*.
  - The zip file's name should be *<filename>.zip* (e.g., *123456789-200002211.zip*).
  - **Only one group member should submit the assignment to the webcourse!**
  - The report PDF file with all your answers (but not your code!), named *<filename>.pdf*.
  - Your code (choose the relevant options for you):
    - Working with jupyter: your notebook, *<filename>.ipynb*.
    - Your completed kNN module (=class) from Part 2, *kNN.py*
    - Working with a "traditional" IDE: one clear main script, *<filename>.py*, and any additional files required for running the main script.
    - Data preparation function *prepare.py* (from Part 6).
  - Do not submit csv files.
- **Failing to follow any of the instructions above will lead to point deduction!**

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: **`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 and noise. These irregularities will surely harm your models' performance in the assignments. Throughout this exercise, we will try to minimize these unwanted properties in the 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?

Before we continue, let us define the "ordinal" variable type. Ordinal variables are categorical with a natural order (e.g., year of birth), and are somewhere between continuous and categorical variables.

(Q2) Print the `value_counts` of the `conversations_per_day` feature (see Tutorial 01). Copy the obtained output to your report. Moreover, describe in one short sentence what you think this feature refers to in the real world.
This feature's type is "ordinal". Explain briefly why.
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 sentence with your understanding of the feature's meaning in the real world.
c. Type: Continuous, Categorical, Ordinal, or Other.
Don't overthink this (especially the "ordinal" type), some variable may be suitable for two types.

Note: do not include the target columns ("`spread`" and "`risk`").

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, but for now the most important thing to remember is that **you may only use the training set when making decisions about the data**.

(Q4) Split the data into a training set (80%) and a test set (20%). As the `random_state`, use the sum of the last two digits of each of your IDs[1] (two or three IDs).
The random state will ensure that you get the same split every time. Answer: 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 complete the rest of the assignment.

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

[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 `spread` 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

Our medical experts suspect the `spread` is largely determined by `PCR_01` and `PCR_03`.

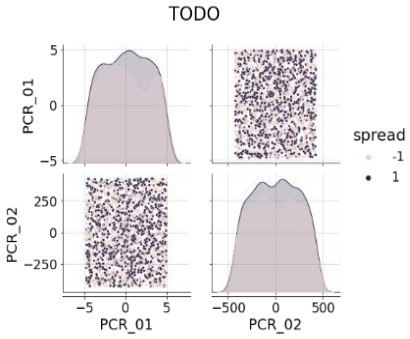
Task A: Create a `seaborn.pairplot` of the 2 aforementioned PCR features. Use the `hue` parameter to color the different (train) data points according to their `spread`. Make sure your plots are readable and clear, and that they have proper titles, grid lines, axis labels, etc. Any missing requirement will lead to a points deduction!

Following is a code snippet that can help you start, and an example of the resulting figure (with different data):

```
g=sns.pairplot(TODO, plot_kws={"s": 12})
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)
```



(Q5) Attach the figure to your report.
Based on it, are those features useful in predicting the `spread`? Explain briefly.

(Q6) Compute the correlation between `spread` and each of the 2 aforementioned PCR features (see Tutorial 01). Attach the correlations to your report. How do these findings settle with the ones from the previous question?

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

```
from sklearn.base import BaseEstimator,ClassifierMixin

class kNN(BaseEstimator, ClassifierMixin):
    def __init__(self, n_neighbors:int = 3):
        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
```

Tip: Read about `scipy…cdist`, `np.copy`, and `np.argsort` (or better: `np.argpartition` [1],[2]). Avoid using `for` loops, `list`, `map`, `lambda`, etc.

(Q7) What is the time complexity of the prediction function you wrote, applied on a single test datapoint, in terms of the number of neighbors k, the number of training datapoints m and the data dimension d? Explain. It is okay to "estimate" the complexity of python library functions. For instance, if you use `np.argsort` on n elements, then its complexity should be $O(n \log n)$. Use your reason and CS knowledge.

We will now test your implementation.

Task C: Create a temporary `DataFrame` by taking only the two features `PCR_01` and `PCR_03` from the training set. Train a `1-NN` model (with `k=1`) on this subset (to fit the `spread` label). Use the provided `visualize_clf` function to visualize the decision regions of the model (send only the training set to this function, so that only the training examples will be scattered on the plot).

Compute the training accuracy and test accuracy of the model by calling its `score` method, e.g., call `h.score(Xtrain, Ytrain)`.

Make sure that all labels in your notebook (the ones in the dataset and the ones your model return) are ± 1, and not $\{0,1\}$ or $\{\text{True}, \text{False}\}$.

(Q8) Attach the figure to your report. Specify the model's training and test accuracies. (The plot should exhibit a bizarre behavior which we will discuss next.)

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 now focus on two normalization techniques: Standardization (Z-score) and min-max scaling (read the explanations in the links). Implementations can be found here and here.

Notice: we often ask questions in the exams regarding reading materials that appear in the assignments.

(Q9) Use min-max scaling (between $[-1,1]$) to normalize the two features in the temporary `DataFrame` you created before, and train a new kNN model ($k = 1$) on the normalized dataset.

Compute the new training and test accuracies and draw the decision regions of the model. Attach the results to your report and compare them to those from **(Q8)** for the same $k = 1$ model on the raw data. Use these results to explain why normalization is important for nearest neighbor models. Provide mathematical justifications.

(Q10) Using the normalized dataset, train another kNN model with $k = 5$. Compute the training and test accuracy and draw the decision regions of this model.

Attach the results to your report and compare them to those from **(Q9)**. Use these results to briefly explain the effect of k on the decision regions.

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

(Q11) Assume a dataset with two features, one randomly sampled (i.i.d.) from a uniform continuous distribution on the range $[2,5]$ and the other randomly sampled (i.i.d.) from a chi-squared distribution $\chi^2(k = 2)$ (see in Wikipedia).

(The labels are determined by some unknown function of these two features.)

Why is normalizing both features using min-max scaling to $[-1,1]$ a bad idea? Explain in detail.

Hint: think how this affects kNN's performance.

We are now ready to start the preprocessing stage for the rest of the features!

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.

(Q12) 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).
How many Boolean features are needed to create such a representation for the `blood_type` feature?

Our medical experts suspect that different blood types have common effects on the `risk` target variable. They suggest "merging" blood types into two more general blood "groups": `{O+, B+}` and `{O-, A-, A+, B-, AB+, AB-}`. That is, instead of having a separate Boolean feature for each blood type, we would have only one Boolean feature representing the different groups.

Task D: According to the suggested groups, create a new Boolean feature called `SpecialProperty` in your `DataFrame`, indicating whether the specific data point has a blood type in `{O+, B+}` or not. Then, remove the original `blood_type` feature from the `DataFrame`.

Technical: You can use the following snippet as a starting point to create a Boolean series according to a subset of the values of a feature:

```
df["blood_type"].isin(["O+", "B+"])
```

(Q13) 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).

Task E: For any feature that you classified as "categorical" or "other" in **(Q3)**, determine whether useful information can be extracted from it. Transform these features accordingly and/or craft new features from them.

At this point, 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 two target variables (`risk` 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 snippet 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 = ['PCR_01', 'num_of_siblings']
COLS, ROWS = (2, len(COL_NAME))

plt.figure(figsize=(5 * COLS, 4 * ROWS))
for row in range(ROWS):
  column = COL_NAME[row]

  for j, cls in enumerate(["risk", "spread"]):
    plt.subplot(ROWS,COLS, row * COLS + 1 + j)

    isContinuous = "float" in df[column].dtype.name
    sns.histplot(data=df, x=column, hue=cls, line_kws={"linewidth": 3},
                 kde=isContinuous, multiple="layer" if isContinuous else "dodge")
    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 (`risk`, `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) According to the univariate analysis, name one feature that seems informative for predicting the `spread` target variable (other than the 2 features from **Q6**).
Attach the appropriate univariate plot and briefly explain (2-3 sentences) why this plot makes you think that feature is informative.

(Q15) According to the univariate analysis, name one feature that seems informative for predicting the `risk` target variable (other than the blood groups).
Attach the appropriate univariate plot and briefly explain (2-3 sentences) why this plot makes you think that feature is informative.

We will now perform some bivariate analysis.

The following snippet performs basic bivariate analysis for the `PCR` features, conditioned on the `risk` variable. This snippet is a good reference for the next question.

```
sns.pairplot(df[df.filter(like='PCR').columns.tolist() + ["risk"]],
             plot_kws={"s": 3}, hue="risk")
```

Our medical experts believe that there exists a pair of `PCR` features that might be helpful for predicting the `risk`, but only after splitting the data according to blood groups.

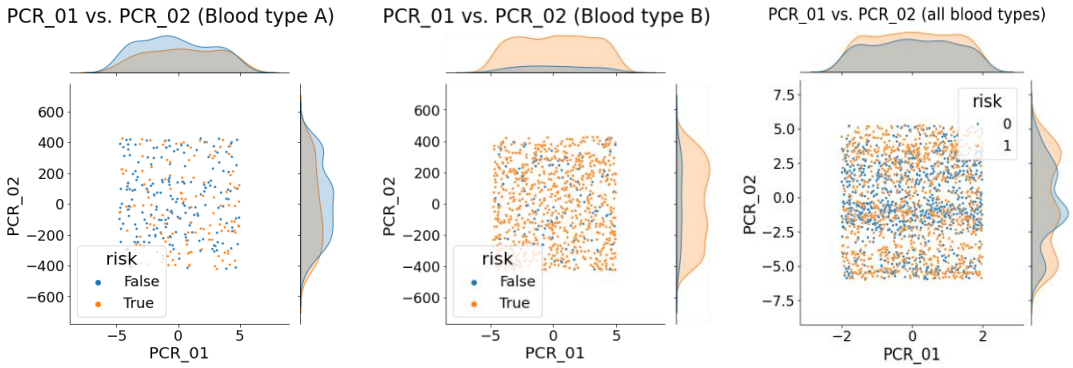
(Q16) Split the (training) data based on the binary `SpecialProperty` feature created in **(Task D)**. For each split, perform a bivariate analysis for the PCR features**,** in relation to the `risk`. This means you should produce two "matrices" of plots, one for each blood group. Each matrix should contain 10x10 subplots, representing all possible combinations of PCR features. Do not include these plots in your report.

According to those plots, choose a pair of `PCR` features that could be helpful for predicting the `risk` (with the partition according to the `SpecialProperty`). What `PCR` features did you choose? And why?

(Q17) For the pair of `PCR` features you chose, create three `jointplot`s (see Tutorial 01), all conditioned on the `risk` variable. The first `jointplot` should include only the data in the first blood group you created in (Task D), `{O+, B+}`. The second `jointplot` should include only the data in the other blood group. The third `jointplot` should be for the full data, without partitioning to blood groups.

Attach the 3 resulting plots to your report. Remember to have grids, titles, and axis-labels.

Example (for one hypothetic pair):



Notice that the three plots above here do not look informative for predicting the risk in any of the blood groups (look at the marginal distributions for instance).

The following snippet demonstrates a call to the provided `plot3d` function:

```
plot3d(df, "PCR_01", "SpecialProperty", "PCR_02", title="TODO", hue="risk", s=5)
```

(Q18) Use the provided function `plot3d` to plot the pair of `PCR` features (axes X and Z) and the `SpecialProperty` feature (axis Y), colored by the `risk` label. Make sure that the plot is clear & readable and that it has a proper title. Attach the plot to your report.

In the following questions, we will provide an analytical explanation of the ability of various models to fit the training data. These models will use three features: the two `PCR` features and the `SpecialProperty` feature, with the aim of fitting the `risk`. Explain your answers in detail (2-5 sentences for each model).

No code execution is required for this section. These questions focus solely on modeling aspects (whether exists a model which can fit the training data). They do not revolve around generalization (i.e., test data) or optimization (how to find such a model).

(Q19) How well will a decision tree of `max-depth=3` be able to fit the training data? Explain briefly.

(Q20) How well will a decision tree of `max-depth=30` be able to fit the training data? Explain briefly.

(Q21) How well will a `1-NN` model be able to fit the training data? Note that in this question, a point in the training set is not considered its own neighbor (i.e., when making a prediction for a training data point, the model won't use the same point for prediction, but only the nearest point in the remaining training set).

Hint: consider the scale of the features in your answer.

Part 4: More Data Normalization

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

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

In (Q24) you are asked to specify the normalization method you chose for each feature.

Hint: Think about (Q11) when deciding which scaler to use for each feature.

(Q22) In your report, show a univariate analysis visualization before and after normalization for the `age` feature. No need to explain anything.

(Q23) What will be the effects of data normalization on your answers in **(Q19)**, **(Q20)**, **(Q21)**?

Part 5: Data Preparation Pipeline

We have finished exploring and preparing our data. Throughout this assignment, you transformed existing features into new ones, made sure all features are numeric, normalized the data, and so on.

(Q24) 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 written in the dataset.
   Names of new features should be meaningful!
b. **Keep**: "V" if the feature is kept, "X" otherwise (e.g., `blood_type` is removed).
c. **New**: "V" if the feature was handcrafted using other feature(s), "X" otherwise.
d. **Normalization method**.
e. **Explanation (reason)**: 1-3 short sentences describing why you made these choices for this feature.

Note: do not include the target variables "`spread`" and "`risk`" in the table.

Now it is time to create an automatic data preparation pipeline that will prepare any incoming data point for prediction by our models.

Remember that this pipeline **should only reflect the training set**. For example, assuming you normalized some feature with the standard scaler (which computes the mean and the std of the given data), then the pipeline should normalize the same feature in the test set using the same mean and std that were computed on the training set.

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

```
def prepare_data(training_data, new_data)
```

The `new_data` parameter is the `DataFrame` to be prepared and `training_data` is the training set `DataFrame` used during data exploration. Your function should perform as described in (Q24). The output is a copy of `data` (the original parameter should remain unchanged), after it has been preprocessed according to the provided `training_data`.

You are required to submit `prepare.py`.

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

```
# Prepare training set according to itself
train_df_prepared = prepare_data(train_df, train_df)

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

Save your two preprocessed `DataFrame`s as CSV files and keep them for the next assignment. Do not submit any CSV files!

Important: Return to the instructions at the beginning of the document and make sure that you submit all the required 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.