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:** Wednesday, 30.11.2022, 23:59.
    We'll receive late submissions for additional 24 hours, deducing 5 points from the grade.
  - 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.
  - 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):
  - 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.
- **Failing to follow any of the instructions above may 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 `num_of_siblings` 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 to 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 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.

[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`, `PCR_02`, and `PCR_09`.

(Q5) Compute the correlation between `spread` and each of 3 aforementioned PCR features (see Tutorial 01). Attach the correlations to your report.

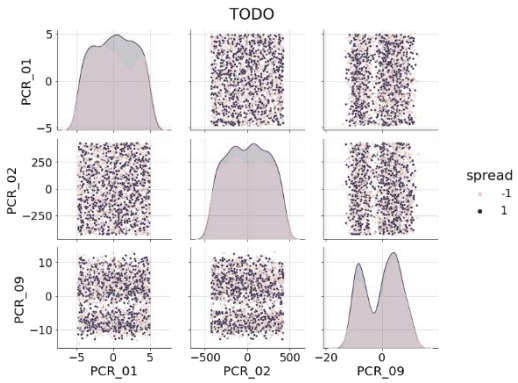
Task: Create a `seaborn.pairplot` of the 3 aforementioned PCR features. Use the `hue` parameter to color the different (train) data points according to their `spread`. Make sure your plots are readable and clear, and that they have proper titles, grid lines, axis labels, etc.

Following is a code snippet that can help you start, and an example of the resulted 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)
```



(Q6) Attach the figure to your report.

According to your figure, which 2 of the 3 features are most useful to predict the `spread`?

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 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.

We will now test your implementation.

Task: Create a temporary `DataFrame` by taking only the two features you chose in **0** from the training set. Train a `kNN` model (with `k=1`) on this subset (to fit the `spread` label). Use the provided `visualize_clf` function to visualize the decision regions of both models (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 $\{\text{True}, \text{False}\}$.

(Q7) 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.

(Q8) 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 **(Q7)** for the same $k = 1$ model on the raw data. Use these results to explain why normalization is important for nearest neighbor models.

(Q9) Using the normalized dataset, train another kNN model with $k = 7$. 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 **(Q8)**. 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.

(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 that normally-distributed feature using min-max scaling a bad idea? (Assume the number of samples is large and think about 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.

(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).

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 three more general blood "groups". That is, instead of having a separate Boolean feature for each blood type, we would have only three Boolean features representing the different groups.

(Q12) Like we did in Tutorial 01 (in code-block [17]), plot a normalized cross tabulation plot (`crosstab`) of the `risk` (not `spread`!) target variable (y-axis) and the different blood types (x-axis).

Don't forget to have suitable titles, axis labels, and grid lines in all your figures!

Attach the plot to your report.

Based only on this plot, how would you partition the blood types into three blood groups? (Specify the groups and briefly explain your logic.)

There might be more than one correct partition.

Task: According to the groups you suggested, create three new Boolean features in your `DataFrame`. 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(["A+", "O-"])
```

During this exercise, you will extract new information out of existing features.

(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: 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 = 2
ROWS = 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 you chose on 0). 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.

(Q16) The following code snippet computes the correlation between `risk` label and the rest of the features. Attach to the report the 10 most correlated features to `risk`.

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

We will now perform some bivariate analysis.

The following snippet performs basic bivariate analysis for the `PCR` features, conditioned on the `risk` variable.

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

(Q17) Use the above snippet and look at the bivariate analysis for the `PCR` features. Choose two pairs of features that (together) form "interesting" structures (with or without regard to the `risk` variable).

Specify these pairs of features and copy their joint scatter plots to your report. Does any of these pairs (by itself) seem to explain the `risk` variable to a large extent?

(Q18) For each of the two feature-pairs you chose in the previous question, create three `jointplot`s (see Tutorial 01), one for each blood group you created in **(Q12)**, conditioned on the `risk` variable.

For instance, if you chose two pairs (PCR_01, PCR_02) and (PCR_01, PCR_03), and the blood groups are A, B, and C; then one of the jointplots you create should show PCR_01 vs. PCR_02, setting the color (hue) according to the risk variable, and showing on ly data points from blood group A. And so on.

Attach the 6 resulting plots to your report. Remember to have grids, titles, and axis-labels.

At least one of the plots in the last question should look informative for predicting the risk (for a specific blood group, using only two features). If not, re-do the last two questions with other pairs of features.

(Q19) Did the features from the informative pairs of (Q18) exhibit a "high" correlation to the risk in (Q16)? Explain why.

(Q20) Considering everything that you just saw, which type of model would be most suitable for predicting the risk – kNN, decision trees, or linear model (with no mappings)? Explain your answer in detail (4-6 sentences).

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: 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 **Error! Reference source not found.** you are asked to specify the normalization method you chose for each feature.

(Q21) In your report, show a univariate analysis visualization before and after normalization for the `weight` 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.

Above we manually looked for informative features.

We will now briefly experience with finding such features automatically.

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.

To make things clearer: in forward feature selection we start from an empty subset of features. Then, we choose a single feature that maximizes the accuracy of a given learning algorithm (when using only that feature). We add the best feature to the subset. Then, we continue this process recursively and greedily (choosing the feature that contributes the most to the learned model at each iteration), until we have enough features in our subset.

In backward selection, we start from all features and then recursively remove one feature at each iteration. The feature we remove is the one whose removal interferes the least (or contributes the most) to the accuracy of the learned model. We stop when once we removed sufficiently many features.

(Q22) Assume that we have $d_1 \in \mathbb{N}$ features in the dataset, and we wish to find a subset of $1 \leq d_2 < d_1$ features.

How does d_1, d_2 affect the complexity of the number of the models we would have to train when performing (a) forward feature selection, and (b) backward feature selection? Answer for both cases. Ignore other factors (like the time needed to train each model etc.). Explain your answers briefly.

Use forward SequentialFeatureSelector (see example within) to find **three** features that (supposedly) best predict the `spread` label. Like in the example, use sklearn's `KNeighborsClassifier` (but use $k = 5$ 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).

(Q23) What are the three features that the selection process found? Do these features include the ones you chose on (Q6)? Do they include the feature you chose manually in (Q14)? If not – are the features found here correlated to the ones from the other questions?

(Q24) Generally, why is it important to perform the normalization step before performing sequential feature selection?

(Q25) Generally, does the choice of a learning algorithm (here we chose 5-NN) matter in a sequential feature selection process? If so, how? If not, why?

Explain your answer in 4-5 sentences.

Part 6: 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.

(Q26) 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 to 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: Write a module[2] called `prepare.py` containing a function with the following signature:

```
def preprare_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 (Q26). 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.