General constraints for code submissions Please adhere to these rules to make our and your life easier! We will deduct points if your solution does not fulfill the following:

- If not stated otherwise, we will use exclusively Python 3.5.
- If not stated otherwise, we expect a Python script, which we will invoke exactly as stated on the exercise sheet.
- Your solution exactly returns the required output (neither less nor more) – you can implement a `--verbose` option to increase the verbosity level for developing.
- Add comments and docstrings, so we can understand your solution.
- (If applicable) The `README` describes how to install requirements or provides addition information.
- (If applicable) Add required additional packages to `requirements.txt`. Explain in your `README` what this package does, why you use that package and provide a link to it's documentation or GitHub page.
- (If applicable) All prepared unittests have to pass.
- (If applicable) You can (and sometimes have to) reuse code from previous exercises.

The goal of this exercise is to get you familiar with a simple supervised machine learning algorithm for classification and the influence of its hyperparameters on the generalization performance. You will implement the k-nearest neighbor (KNN) classifier, which is a non-parametric model, i.e. it does not make any assumptions about the data generating distribution.

All tasks include the submission of some results (besides the code). To submit these results, please submit a PDF with all the results and your name(s).
We provide a simple Makefile which you can use to install all packages listed in your requirements file (`make init`).

1. **Implementing your own KNN classifier** [1 point]

   Consider the classification of the examples in the Iris Flower Dataset, which consists of 3 species/classes (*Iris setosa, Iris virginica and Iris versicolor*) and each example has 4 features (*sepal lenght, sepal width, petal length, petal width*). There is a total of 150 examples.

   In a nutshell, KNN assigns a class to an unseen example based on the majority vote of the K most similar instances. We will use two distance metrics as a similarity metric in this exercise:

   - Euclidian distance $d_E(x, x') = \sqrt{\sum_{i=1}^{n} (x_i - x'_i)^2}$
   - Manhattan distance $d_M(x, x') = \sum_{i=1}^{n} |x_i - x'_i|$

   where n is the number of features. When a new test example x is fed to the KNN classifier it performs the two following steps:

   - Computes the distance d between this example and all the examples in the training set.
   - For each class j estimate the conditional probability for the example being assigned to that class based on the K closest examples:

     $$P(y = j|X = x) = \frac{1}{K} \sum_{i \in A} \mathbb{I}(y^{(i)} = j),$$

     where A is the set of K closest training examples and $\mathbb{I}$ is the indicator function. This way, the KNN classifier assigns to x the class with the highest probability.

   Your first task is to implement by yourself the KNN algorithm in Python so it can learn to classify the examples in the Iris Dataset. For starters select a random value for K as the default one. It should be possible to call your program as `python knn_classifier.py [K value] -d [distance metric]`.

   Run the code for the two different similarity metrics and report their results in the pdf.

2. **Tuning the hyperparameters of your KNN classifier** [1 point]

   Now we want to tune the value of K and the distance metric d using 10-fold cross-validation as a cost metric. Provide in the pdf what type of hyperparameters these two are. You have to determine the range of values that K will take $[K_{min}, K_{max}]$ based on the size of your training dataset and plot the accuracy of your model depending on the K value (similar to the plot in slide 12 of Lecture 1) for each distance metric type. It should be possible to call your program as `python exhaustive_search.py`. After that 2 plots will be generated (one for each distance metric).

   Afterwards you should report in the pdf the optimal value(s) of K and d that you find by this exhaustive search (its called exhaustive since you evaluate all possible configurations in your design space), as well as the accuracy of the classifier on the test set with these values.

   In the end, you should write a short justification about the chosen range for K and interpret why the cross-validation accuracy goes down for larger/smaller values of K than the optimal one(s).

3. **Feedback** [Bonus: 1 points]

   For each question in this assignment, state:

   - How long you worked on it.
   - What you learned.
   - Anything you would improve in this question if you were teaching the course.

Submit your solution for the tasks by uploading a PDF to your groups BitBucket repository. The PDF has to include the name of the submitter(s). Teams of at most 2 students are allowed.