Assignment 1, FIT5201, S1 2024

Due Date: 11:55PM on 19th Apr 2024

Version 1.1 (March 25, 2024)

Objectives

This assignment assesses your understanding of

1. model complexity,
2. model selection,
3. uncertainty in prediction with bootstrapping,
4. probabilistic machine learning, and
5. linear models for regression and classification,

covered in Modules 1, 2, and 3 in written (Jupyter Notebook) and oral (interview) form. In your Jupyter notebook you can use any import from the libraries `numpy`, `scipy`, `matplotlib`, and `scikit-learn` to solve the tasks except those imports that would render a task trivial. Imports from other external libraries are not allowed. The maximum number of marks of this assignment is 100 (arithmetic mean of up to 100 marks for written and up to 100 marks for oral part). This assignment constitutes 25% of your final mark for this unit. **Make sure you read and understand not only all the tasks but also the notes at the end of this document about submission and interview, assessment criteria, and penalties. Good luck with your assignment!**

1 Model Complexity and Model Selection

In this section, you study the effect of model complexity on the training and testing error. You also demonstrate your programming skills by developing a regression algorithm and a cross-validation technique that will be used to select the models with the most effective complexity.

Background. A KNN regressor is similar to a KNN classifier (covered in Activity 1.1) in that it finds the K nearest neighbors and estimates the value of the given test point based on the values of its neighbours. The main difference between KNN regression and KNN classification is that a KNN classifier returns the label that has the majority vote in the neighborhood, whilst KNN regressor returns the average of the neighbors' values. In Activity 1 of Module 1, we use the number of misclassifications as the measurement of training and testing errors in a KNN classifier. For KNN regressor, you need to choose another error function (e.g., the sum of the squares of the errors) as the measurement of training errors and testing errors.

Question 1 [KNN Regressor, 5+5=10 Marks]

I Implement a KNN regressor using the `scikit-learn` conventions, i.e., in a class with the following skeleton.

```
from sklearn.base import BaseEstimator

class KnnRegressor(BaseEstimator):

    def __init__(self):  # ADD PARAMETERS AS REQUIRED
        # YOUR CODE HERE

    def fit(self, x, y):
        # YOUR CODE HERE
        return self

    def predict(self, x):
        # YOUR CODE HERE
```

Hint: You can closely follow the implementation from Activity 1.1 of the KNN classifier. You cannot use `sklearn.neighbors.KNeighborsRegressor` to solve this task.

Note: Inheriting from `BaseEstimator` is not strictly required, but allows to use the implementation with utility functions like `sklearn.model_selection.cross_validate`.

II To test your implementation, load the datasets `diabetes` and `california housing` through the functions `load_diabetes` and `fetch_california_housing`, both of which are available in the module `sklearn.datasets`. For both datasets, perform a training/test split (using a fraction of 0.6 of the data as training data), fit your KNN regressor to the training portion (using some guess for a good value of k), and report the training and test errors.

Question 2 [L-fold Cross Validation, 5+5+5=15 Marks]

I Implement a L-Fold Cross Validation (CV) scheme using the `scikit-learn` convention for data splitters, i.e., using the following skeleton. *Note that this is usually referred to as K-fold cross-validation. We are simply using the symbol L here to differentiate the cross validation parameter from the number of neighbours in K-nearest neighbours.*

```
class LFold:

    def __init__(self): # ADD PARAMETERS AS REQUIRED
        # YOUR CODE HERE

    def get_n_splits(self, x=None, y=None, groups=None):
        # YOUR CODE HERE

    def split(self, x, y=None, groups=None):
        # YOUR CODE HERE
```

```
            yield train_idx, test_idx
```

Hint: As usual, there are many ways to implement the desired solution. For one of the ways, the function `np.concatenate` can be useful to build up the array if the train indices.

Test your implementation for correctness by running a simple example like the following.

```
for idx_train, idx_test in LFold(5).split(list(range(20))):
    print(idx_train, idx_test)
```

You cannot use `sklearn.model_selection.KFold` to solve this task.

II For both datasets from Question 1, use your L-fold CV implementation to systematically test the effect of the KNN parameter K by testing all options from 1 to 50 and, for each K, instead of only performing a single training/test split run your L-Fold CV. For each K compute the mean and standard deviation of the mean squared error (training and test) across the L folds and report the K for which you found the best test performance (for both datasets).

Hint: To avoid code duplication and name clashes, consider creating a function or class that, given some generic input dataset, encapsulates the experiment of performing the cross validation for each candidate K and stores/returns the results in an appropriate data structure.

III For both datasets, plot the mean training and test errors against the choice of K with error bars (that represent 95%-confidence intervals). You can compute the confidence intervals as

$$m \pm 1.96 s/\sqrt{L}$$

where m is the sample mean and s is the sample standard deviation of the error across the L folds (the quantity $s/\sqrt{L}$ is also referred to as the *standard error of the mean*). Based on this plot, comment on

- The effect of the parameter K. For both datasets, identify regions of overfitting and underfitting for the KNN model.
- The effect of the parameter L of the CV procedure. HINT: You might want to repeat the above process with different values for L to get an intuition of its effect.

Hint: Think about what values for K make the model more or less flexible. Again, it might be useful to create a function for plotting to avoid code duplication and name clashes.

Question 3 [Automatic Model Selection, 5 + 5 = 10 Marks]

I Implement a version of the KNN regressor that automatically chooses an appropriate value of K from a list of options by performing an *internal* cross-validation on the training set at fitting time. As usually, use the `scikit-learn` paradigm, i.e., use the following template.

```
from sklearn.base import BaseEstimator

class KnnRegressorCV(BaseEstimator):
```

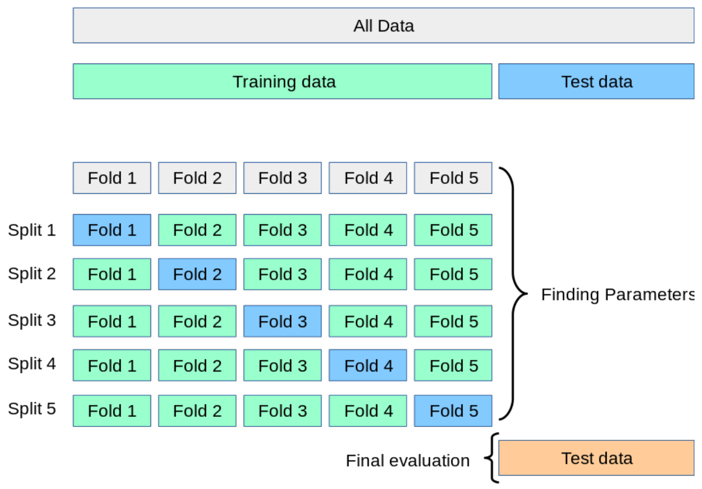

Figure 1: Illustration of data use for Question 3.II with a simple train/test split (and the "inner" cross validation performed on the training data by `KnnRegressorCV`). A better test would replace the training/test split with an additional "outer" cross validation to check multiple times what K has been selected. This scheme would be called a nested cross validation. See `https://scikit-learn.org/stable/modules/cross_validation.html`.

```
def __init__(self, ks=list(range(1, 21)), cv=LFold(5)):
    # YOUR CODE HERE

def fit(self, x, y):
    # YOUR CODE HERE
    self.k_ = # YOUR CODE HERE
    return self

def predict(self, x):
    # YOUR CODE HERE
```

Hint: You might want to store the optimal value for K that has been determined by the internal cross-validation (as indicated in template) for ease of access in the next step. Also you can again use inheritance from `BaseEstimator` to allow usage of `cross_validate` in the next step.

II For both datasets from the previous questions, test your KNN regressor with internal CV by using either an *outer* single train/test-split or, ideally, with an *outer* cross-validation (resulting in a so-called nested cross-validation scheme). See Fig. 1 for a further explanations.

Report on the (mean) k value that is chosen by the KNN regressor with internal cross-validation and whether it corresponds to the best k-value with respect to the outer test sets. Comment on what factors determine whether the internal cross-validation procedure is successful in approximately selecting the best model.

2 Probability

In this section, you show your knowledge about the basics of probability theory by solving simple but basic probabilistic modelling and inference problems. Solve the problems based on the probability concepts you have learned in Modules 1 and 2.

Question 4 [Bayes Rule, 5+5=10 Marks] Suppose we have one red, one blue, and one yellow box with the following content:

- In the red box we have 3 apples and 8 oranges,
- in the blue box we have 4 apples and 4 oranges, and
- in the yellow box we have 9 apples and 1 orange.

Now suppose we selected one of the boxes uniformly at random and then, in a second step, picked a fruit from it, again uniformly at random.

I Implement a Python function that simulates the above experiment (using a suitable method of a `numpy` random number generator obtained via `numpy.random.get_default_rng`). For instance you could name the function `fruit_experiment` and it could take a parameter for the number of repeated simulations. When calling `fruit_experiment(4)` the output could then look like:

```
array(['red', 'blue', 'blue', 'yellow'], dtype='object'),
array(['orange', 'orange', 'apple', 'apple', dtype='object'))
```

Hint: Depending on your implementation, the method `integers` of the random number generator could be useful.

II Answer the following question by a formal derivation in a markdown cell (ideally using Latex for clean typesetting): If the picked fruit is an apple, what is the probability that it was picked from the yellow box?

Hint: Formalise this problem using the notions in the "Random Variable" paragraph in Appendix A of Module 1. You might want to use your simulation function from Part I to check your answer.

Question 5 [Expected Values, 5+5+5=15 Marks] Consider the following simple one-player game: the player first rolls two fair six-sided dice and then she determines her score as the sum of the outcomes of a number of additional die roles, where the number of additionally rolled dice is equal to the sum of numbers rolled with the first two dice. Formally, we can describe this game with a set of random variables:

- X_1 and X_2, the outcome of the first and second die roll, respectively.
- Y_i for $i = 1, \dots 12$, the outcome of the i-th subsequent die role if $i \leq X_1 + X_2$ or 0 otherwise.
- $Z = Y_1 + \cdots + Y_{12}$, the final score of the player.

We are interested in experimentally and analytically determining the score that a player can expect to achieve in this game on average, i.e., the expected value $\mathbb{E}[Z]$.

I Implement a Python function `die_experiment` that simulates the above game for a desired number of repetitions and returns the array of scores achieved by the player for each repetition.

II Estimate the expected player score $\mathbb{E}[Z]$ by performing 10,000 repetitions and provide an error margin of this estimate with 95% certainty (you can use the formula of Q2 Part III for this).

III Analytically derive the expected value $\mathbb{E}[Z]$ in a markdown cell (using Latex formulae for clean typesetting).

Hint: Determine first the conditional expectation $\mathbb{E}[Z|X_1 = x_1, X_2 = x_2]$ given specific values x_1 for X_1 and x_2 for X_2. Then use the rule that the marginal expectation $\mathbb{E}[Z]$ can be computed as the average of $\mathbb{E}[Z|X_1 = x_1, X_2 = x_2]$ over all possible values x_1 and x_2 weighted by their probability (this is the "rule of total expectation" as used in the analysis for the bias/variance trade-off).

3 Ridge Regression

In this section, you develop Ridge Regression by adding the L2 norm regularization to the linear regression (covered in Activity 2.1 of Module 2) and study the effect of the L2 norm regularization on the training and testing errors. This section assesses your mathematical skills (derivation), programming, and analytical skills.

Question 6 [Ridge Regression, 10+5+5=20 Marks]

I Derive the weight update steps of stochastic gradient descent (SGD) for linear regression with L2 regularisation norm or a system of linear equations that uniquely determine the minimum of the regularised error function. Give this derivation with enough explanation in a markdown cell (ideally using Latex for readable math typesetting). The starting point is the definition of the regularised error function and the end result is either the weight update step for this function in (stochastic) gradient descent or a system of linear equations described in matrix/vector notation. In both cases, you have to derive the gradient as an intermediate step.

Hint: Recall that for linear regression we defined the error function E. For this assignment, you only need to add an L2 regularization term to the error function (error term plus the regularization term). This question is similar to Activity 2.1 of Module 2.

II Using the analytically derived gradient from Step I, implement either a direct or a (stochastic) gradient descent algorithm for Ridge Regression (use again the usual template with `__init__`, `fit`, and `predict` methods. You cannot use any import from `sklearn.linear_model` for this task.

III Study the effect of the L2-regularization on the training and testing errors, using the synthetic data generator from Activity 2.3. i.e., where data is generated according to

$$X \sim \text{Uniform}(-0.3, 0.3)$$
$$Y = \frac{\sin(5\pi x)}{1+2x} + \epsilon$$
$$\epsilon \sim \text{N}(0, 0.1)$$

a Consider the ridge regression model for each λ in $\{10^{-10+9i/100}, \ldots, 10^{-1} : 0 \leq i \leq 100\}$ by creating a pipeline of your implemented ridge regressor with a polynomial feature transformer with degree 5.

  Hint: You can create an array with the above choices for λ via `numpy.geomspace(10**-10, 0.1, 101, endpoint=True)`.

b Fit each model at least ten times (resampling a training dataset of size 20 each time) for all choices of λ. To reduce the variance of the experiment make sure that for each repetition all models use the same training dataset (i.e., make the repetitions the outer loop and the choices of λ the inner loop, and sample only one training set per outer loop).

c Create a plot of mean squared errors (use different colors for the training and testing errors), where the x-axis is log lambda and y-axis is the log mean squared error. Discuss λ, model complexity, and error rates, corresponding to underfitting and overfitting, by observing your plot.

  Hint: For log-scaling an axis you can use, e.g., `pyplot.x_scale('log')`. Moreover, note that, as we have a synthetic data source here, you can simply sample a large amount of independent test data and re-use that to approximate the generalisation error for all fits.

4 Logistic Regression versus Bayes Classifier

This task assesses your analytical skills. You need to study the performance of two well-known generative and probabilistic models, i.e. Bayesian classifier and logistic regression, as the size of the training set increases. Then, you show your understanding of the behavior of learning curves of typical generative and probabilistic models.

Question 7 [Discriminative vs Generative Models, 5+5+5+5=20 Marks]

I Load the `breast_cancer` dataset via `load_breast_cancer` in `sklearn.datasets`, import `LogisticRegression` from `sklearn.linear_model`, and copy the code from Activity 3.3. for the Bayes classifier (BC). For the Bayes classifier consider the Naive Bayes variant (without shared covariance) as well as the variants with full covariance (shared and not shared). Perform a training/test split (with `train_size` equal to 0.8) and report which of the models performs best in terms of train and test performance.

  Note: for logistic regression you can also use the code for the variant with regularisation from Activity 3.2, but this option requires a more careful calibration of the classifier parameters (batch size, max iterations and error tolerance).

II Implement an experiment where you test the performance for increasing training sizes of $N = 5, 10, \ldots, 500$. For each N sample 10 training sets of the corresponding size, fit all models, and record training and test errors.

   Hints: You can use `training_test_split` from `sklearn.model_selection` with an integer parameter for `train_size` (do not forget to use `shuffle=True`). Again make the repetitions the inner loop to assure that all models are trained on the same training set for a given repetition and sample size.

III Create suitable plots that compare the mean train and test performances of all models as a function of training size. There is no need to include error bars if that makes the plot too hard to read.

IV Formulate answers to the following questions:

   a What happens to each classifiers train and test performance when the number of training data points is increased?

   b Which classifier is best suited when the training set is small, and which is best suited when the training set is big?

   c Justify your observations by providing some speculations and possible reasons.

   Hint: Think about model complexity and the fundamental concepts of machine learning covered in Module 1. In particular think of the number of parameters that each model has to learn and what assumptions the models make about the data which could be violated.

Submission and Interview

The assignment is assessed based on file(s) submitted via Moodle as well as an interview after the submission deadline. Not submitting the file or not attending the interview will both result in 0 marks for the assignment.

Submission Please submit one zip-file named according to the schema

`STUD-ID_FIRSTNAME_LASTNAME_assignment1.zip`

that contains two versions (the actual "ipynb"-file and a "pdf"-export) of five Jupyter Notebooks—one for each section of this assignment ("1 Model Complexity and Model Selection", "2 Probability", etc.). The files contained in the zip file should be named as follows:

```
STUD-ID_FIRSTNAME_LASTNAME_a1_sec1.ipynb
STUD-ID_FIRSTNAME_LASTNAME_a1_sec1.pdf
STUD-ID_FIRSTNAME_LASTNAME_a1_sec2.ipynb
STUD-ID_FIRSTNAME_LASTNAME_a1_sec2.pdf
...
```

Overall, there should be eight files within the zip-file corresponding to the four sections of the assignment. Furthermore, make sure that notebooks for each question contain

- your name and student ID in a leading markdown cell followed by

- a structure that clearly separates between questions (with markdown headlines and sub-headlines) and then for each question
- all required code,
- all required figures,
- and your answers to all question parts that require a free-text answer (in markdown cells).

Note that depending on your system it might be necessary to first generate an html export and then save that with your web browser as pdf-file. The submission must be received via Moodle by the due date mentioned on the top of this document.

Interview In addition to the submission, you will be asked to meet (online or on-campus) with a member of the teaching staff for an interview of about ten minutes. If the interview is held online: (i) please activate your camera and have a photo id ready to identify yourself and (ii) make sure to be situated in an adequate environment (stable internet connection, no noise pollution). You are supposed to attend your interview alone and answer questions speaking freely, in particular, without the help any AI tools.

Notes Please note that,

1. One second delay will be penalized as one day delay. So please submit your assignment in advance (considering the possible Internet delay) and do not wait until the last minute.
2. We will not accept any resubmitted version. So please double check your assignment before the submission.
3. If you do not attend your scheduled interview, you will not be given a second interview opportunity (outside of clearly documented medical emergencies etc.).

Assessment Criteria

Your final mark for the assignment is the average between the mark for your submitted files and the mark achieved in your interview. This means that your interview performance is an integral part of your assessment and not primarily an academic integrity test (although an academic integrity investigation can be triggered if evidence arises during the interview that, e.g., you are not the author of your submission file).

The following outlines the criteria which your **submitted notebooks** will be assessed against:

1. Working code: The code executes without errors and produces correct results.
2. Understandable code: Usage of clear ideoms, function and variable names, and comments where useful.
3. Quality of your written answers: Completeness and logical coherence of arguments, usage of proper technical terms, citation of external sources when used.

4. Marks can be deducted if relevant information is hard to access because of the inclusion of irrelevant and redundant information. For your information: The overall marking time for an individual assignment is set to 20 minutes. If the marker cannot make sufficient sense of your submission during that timefrime they will typically deduct marks.

In addition, the following outlines the assessment criteria during the **interview**: The purpose of the interview is to assess your ability to verbally explain your work in a coherent and stringent way. During the interview you will be asked a number of questions that you answer to demonstrate this ability. This involves several levels starting from a basic understanding (e.g., 'what does this line of code do') up to the conceptual level (e.g., 'what are general considerations to choose this parameter'). The grade for an individual answer will depend on the coherence of explanations of code, mathematical derivation, and the fundamental concepts of machine learning used.

At the end of the interview your interviewer will give you some brief feedback about how you did and tell you your interview mark (0 to 100).

Penalties

- Late submission (students who submit an assessment task after the due date will receive a late-penalty of 10% of the available marks in that task per calendar day. Assessment submitted more than 7 calendar days after the due date will receive a mark of zero (0) for that assessment task.)
- Any file not properly named (-5%)
- Lack of structure in Jupyter Notebook, e.g., no section title for questions (-5%)