Examination:
Programming for Data Science (ID2214)

Course code: ID2214
Course name: Programming for data science
Literature and tools:
It is allowed to consult literature, other documents, including lecture slides, notes, etc. It is not allowed to search for solutions online or to communicate in any way with any other individual or forum to either get or provide help to answer the questions both during the exam and before the follow-up Zoom session after the exam. It is allowed to use any text editor/word processor (on a computer, tablet or phone) for writing answers to the questions, and any tool, such as an editor, Jupyter notebook or IDE may be used for developing the programs on part II.
Date and time: Jan. 1, 2021, 13:00-17:00
Examiner: Henrik Boström

Requirements to pass: 5 points on part I and 10 points on part II

On part I, keep the text short and to the point.
On part II, only the Python standard library, the NumPy and pandas libraries may be assumed, in addition to functions explicitly stated in the tasks.
All answers (including blank ones) should be numbered and ordered, and compiled into one single pdf-document that should be uploaded according to the instructions in Canvas.
It is recommended that the provided Jupyter notebook template is used for producing the pdf-document.
Unreadable answers will be ignored.
Good luck!

Part I (Theory, 10 points)

1a. Methodology, 2 points

Assume that we have obtained a training and test set, both sampled independently from the same (unknown) underlying distribution. We have decided to use ten different learning algorithms and employ hyperparameter tuning using cross-validation on the training set. The model with the highest average performance over all folds is selected for each learning algorithm, which is then evaluated on the test set. The result of the ten models is then compared and the best model is finally selected. Will the performance estimate of the best

model obtained in this way be biased, or can we expect to see about the same performance on a new dataset sampled from the same underlying distribution?

1b. Data preparation, 2 points

Assume that we want to use a learning algorithm that requires categorical features only with no missing values, while the dataset we have at hand contains numerical features for which a large portion of the values are missing. What are the various options we may consider for preparing the dataset to meet the requirements of the algorithm? Specify the order in which the various preparation steps may be taken.

1c. Naïve Bayes, 2 points

Assume that we are facing a binary classification task, where a positive class label (+) is observed when the binary features f_1 and f_2 both have a value of 0 or 1, and a negative label (−) is observed in all other cases, i.e., when $f_1 \neq f_2$. Explain why Naïve Bayes has difficulties in learning an accurate model for this task.

1d. Performance metrics, 2 points

Assume that we have generated a binary classification model, which outputs class probability estimates for each test instance, where the corresponding class label for which the estimated probability is higher than 0.5 will be the predicted label. Assume further that we have observed a perfect area under the ROC curve (AUC = 1.0) on a validation set, but with a less than perfect accuracy. Is there any way in which we can modify the procedure to obtain the predicted labels to reach also a perfect accuracy (without retraining the model)? Explain your reasoning.

1e. Combining models, 2 points

Explain why AdaBoost may benefit from using decision stumps (decision trees of depth 1) rather than fully grown decision trees.

Part II (Programming, 20 points)

2a. Data preparation, 10 points

Your task is to define the following Python function to replace a set of columns in a pandas dataframe, with a single column in which each value corresponds to a combination of the values in the different columns:

```
create_combined_feature(df,f)
```

which given a dataframe `df`, where the columns correspond to features (except for a column named `CLASS`), with the same feature possibly spanning over multiple columns (the columns may share the same feature name), and the rows correspond to instances, and a feature name `f`, returns a new data frame, which

is a copy of the original but all columns with the name `f` has been replaced with a single column with the same name, containing a unique value (string) for each combination of the original values (which are assumed to be strings only, except for missing values, np.nan). Missing values should be ignored, unless all the values for the feature is missing, in which case the combined value should be just np.nan.
For example, assuming the dataframe `df`:

```
CLASS F      F      F
+     a      b      nan
+     c      nan    nan
+     nan    nan    nan
-     b      b      c
-     c      b      nan
```

```
create_combined_feature(df,"F")
```

should result in:

```
CLASS F
+     ['a', 'b']
+     c
+     nan
-     ['b', 'c']
-     ['b', 'c']
```

Hint: You may consider checking that a certain value is of the type string by `type(value)==str` and using `str(sorted(set(values)))` to obtain a string representing the combination of the unique, ordered, strings in `values`.

2b. Clustering, 10 points

Assume that a cluster `C` is represented by a list of indexes, where each index refers to a row of a numpy array `X` (of which the columns correspond to features). Your task is to complete the following program for top-down (divisive) clustering:

```
def top_down_clustering(C,X,min_cluster=5):
   if no_split(C,min_cluster):
      return C
   else:
      left_C, right_C = split_cluster(C,X)
      return [top_down_clustering(left_C,X,min_cluster),
              top_down_clustering(right_C,X,min_cluster)]
```

where `no_split(C,min_cluster)` should evaluate to true if and only if `C` contains less than `min_cluster` instances, and where `split_cluster(C,X)` should return the indexes of two clusters obtained by k-means clustering (hence with $k = 2$). Rather than implementing k-means clustering yourself, you may rely on the implementation of scikit-learn and use it in the following way, where `n_clusters` refers to the number of clusters (k) and `X` is assumed to be a numpy array as described above:

```
from sklearn.cluster import KMeans

kmeans = KMeans(n_clusters=2).fit(X)
```

The labels of the clusters in which the instances in `X` are placed can be found with:

```
kmeans.labels_
```

which returns a vector with a label (0 or 1) for each instance in `X`.

In other words, you need to provide definitions for the above functions `no_split(C,min_cluster)` and `split_cluster(C,X)`. This should allow for `top_down_clustering` to be called with, e.g.,

```
import numpy as np

C = np.arange(10)
X = np.random.rand(10,5)
```

The call `top_down_clustering(C,X,4)` may then (depending on the randomly generated values in `X` and the initialization of k-means clustering) produce a hierarchical cluster, like the following:

```
[[[3, 6, 9], [0, 8]], [[2, 4, 7], [1, 5]]]
```