Examination:
Programming for Data Science (ID2214)

Course code: ID2214
Course name: Programming for data science
Literature and tools:
The following rules apply to both parts of the examination:
Literature, other documents, including lecture slides, notes, etc. are not allowed.
Computers, tablets, phones, etc. are not allowed.
Date and time: January 9, 2020, 14:00-18:00
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.
The answers (including blank ones) should be numbered and ordered.
Unreadable answers will be ignored.
Good luck!

Part I (Theory, 10 points)

1a. Methodology, 2 points

Assume that we want to compare a new algorithm to a baseline algorithm, for some classification task. As we are not sure what hyper-parameter settings to use for the new algorithm, we will investigate 100 different settings for that, while we use a standard hyper-parameter setting for the baseline algorithm. We first randomly split a given dataset into two equal-sized halves; one for model building and one for testing. We then employ 10-fold cross-validation using the first half of the data, measuring the accuracy of each model generated from an algorithm and hyper-parameter setting. Assume that the best performing hyper-parameter setting for the new algorithm results in a higher (cross-validation) accuracy than the baseline algorithm. Should we expect to see the same relative performance, i.e., the new algorithm (with the best-performing hyper-parameter setting) outperforming the baseline (with the standard hyper-parameter setting), when the two models (trained on the entire first half) are evaluated on the second half of the data? Explain your reasoning.

1b. Data preparation, 2 points

Will the use of min-max normalization prior to the use of equal-sized binning have any effect on model building compared to just using equal-sized binning (without normalization)? Explain your reasoning.

1c. Performance metrics, 2 points

Assume that we have evaluated a binary clasification model on a test set with 5000 instances; 4000 belonging to the majority class and 1000 to the minority class. Assume that we have measured the accuracy and AUC, and also observed a much higher precision for the majority class than for the minority class. If we would evaluate the model on a class-balanced test set, which has been obtained from the first by keeping all instances from the minority class and sampling (without replacement) 1000 instances from the majority class, should we expect to see about the same accuracy and AUC as previously observed? Explain your reasoning.

1d. Combining models, 2 points

Assume that we want to apply both random forests and the gradient boosting machine to a regression task, where all regression values in the training set fall into the range [0,1]. Could any of the resulting models predict values outside this range? Explain your reasoning.

1e. Clustering, 2 points

Agglomerative clustering is deterministic, unless we handle encountered ties (regarding what clusters to merge) randomly. If we have chosen to employ single-linkage, will such a random selection of a pair of clusters to merge (from the alternative mergings that result in the same score) have any effect on subsequent mergings that result in higher scores? Explain your reasoning.

Part II (Programming, 20 points)

2a. Data preparation, 10 points

Your task is to define the following Python function that converts a list of text paragraphs (lists of words) into an array using a *bag-of-words* representation:

```
bag_of_words(word_lists)
```

which takes as input a list `word_lists` of lists of words (strings), and returns a pair `(mapping,matrix)`, where `mapping` is a dictionary, which from each unique word in `word_lists` returns an index in the range `[0,n-1]` where `n` is the number of unique words, and where `matrix` is a NumPy array of the shape `(p,n)`, where `p` is the number of paragraphs (elements) in `word_lists`. Each element `matrix[r,col]` should be 1 if `word_lists[r]` contains a word `w` which is mapped to the index `col`, i.e., `mapping[w] == col`, and 0 otherwise.

For example,

```
bag_of_words([["Python"],["Python","Julia"],["R"]])
```

should return a pair, consisting of the mapping:

```
{"Python":0, "Julia":1, "R":2}
```

and the matrix:

```
array([[ 1,  0,  0],
       [ 1,  1,  0],
       [ 0,  0,  1]])
```

Hint: Given a dictionary `d`, `d.get(w)` will return `None` if `w` is not a key in the dictionary (in contrast to `d[w]`), and otherwise return `d[w]`.

2b. Combining models, 10 points

Your task is to define the following Python function:

```
variable_importance(df,model)
```

which given a pandas dataframe `df`, where the columns correspond to features, except for a column named `CLASS`, which contains the class values, and where the rows correspond to instances, and a classification model `model` (see below), will return a list `[(f1,v1), ..., (fn,vn)]`, where `f1, ..., fn` are the feature names (all column names in `df` except `CLASS`) and each `vi` is the (relative) variable importance of feature `fi`.

The variable importances are calculated by for each feature randomly permuting its values and calculating the corresponding accuracy reduction (see below), and finally dividing each accuracy reduction by the sum of the accuracy reductions for all features.

The given model `model` is assumed to provide predictions (generate a list of class labels) for `df` (one label for each instance and ignoring the `CLASS` column when making the predictions) by:

```
predictions = model.predict(df)
```

You may also assume that you have been provided with a definition of the following function:

```
accuracy(predictions,labels)
```

which given a list of predictions and (true) class labels, returns the accuracy.

The accuracy reduction obtained by permuting the values of a feature is the accuracy of the model with respect to the original dataframe minus the accuracy of the model with respect to the dataframe where the values of the feature have been permuted. In the (unlikely) case of an increase of accuracy after the permutation, the accuracy reduction is defined to be zero (rather than negative).

Hint: You may obtain a randomly permuted sequence from a data series or array `values` by `numpy.random.permutation(values)`.