Assignment 3: Classification

Deadline: 28 April 2025

Packages: NumPy, Pandas, Scikit-learn

Name: YOUR_NAME

Matriculation Number: YOUR_MATRICULATION_NUMBER

Submission Instructions: Upload your Jupyter notebook and a PDF export (with results) on Moodle. Furthermore, upload the exported KNIME workflow (with your pickle file in the data directory) for Task 1.4 on Moodle. Go to the corresponding checkmark list to indicate which tasks you have completed and feel confident to explain in class. The checkmark list will be the basis for grading. If you fail to explain your submission you will be awarded 0 points for the entire assignment; after the second such incident you would fail the course.

Write your solutions in the code cells for the different tasks. You may also add additional code cells as well as markdown cells if you want to write down additional explanations, observations, or assumptions.

There are also questions that require you to write textual answers into markdown cells.

If anything is unclear, ask in the forum on Moodle and/or make reasonable assumptions. Document any such assumptions in the Jupyter notebook and the PDF report.

Case 1: Bank Loans

Files: loans.csv

You have a dataset of loan documentation. The goal is to train a model that will predict whether a loan is good or bad based on various indicators.

Task 1.1: Data Preprocessing

Our analysis will require a class variable to distinguish good loans from bad loans. We can derive that from the detailed loan status described by the *loan_status* variable. Add a variable *class*, derived from *loan_status* as follows:

- Assign *class* to 'good' if *loan_status* is 'Fully Paid' or 'Does not meet the credit policy. Status:Fully Paid'.
- Assign *class* to 'bad' if *loan_status* is 'Default', 'Charged Off', or 'Does not meet the credit policy. Status:Charged Off'.

Our analysis will require that each loan's class is known. Include only inactive loans for which the class is known, which are loans with a *loan_status* value mapped to class 'good' or 'bad'. Other *loan_status* values indicate that a loan is still active.

Our analysis will require variables to be in numerical representation, though the dataset includes potentially useful information in categorical representation. Convert categorical variables to numerical representation as follows:

- From *term*, remove the word 'months'.
- From *emp_length*, remove 'year*', change '< 1' to 0, change '10+' to 10, change 'n/a' to null value.
- Change *grade*, *sub_grade*, *home_ownership*, and *purpose* to numerical values by index coding, i.e., transforming the string categories into numeric values, where each category corresponds to an integer value.

Transform the dataset for analysis. Filter out (i.e., remove) some of the variables like this:

- Identification variables, like *id* and *member_id* are not predictive, so we do not include them.
- Leaky variables are those that contain information that could only be known when the class is already known. Since we are ultimately interested in predicting the class before the class is actually known, we do not include leaky variables *recoveries*, *collection_recovery_fee*, or *collections_12_mths_ex_med*.
- Empty variables, i.e., columns with only null values, are missing any information at all, so we do not include them.
- No-variance variables are missing any information at all, so we do not include them.
- Sparse variables, i.e., those with more than half of their values missing, might be too difficult to sensibly impute, so we do not include them.

For convenience, do not include non-numerical variables, except for the class variable.

Impute by simply substituting zeros for missing values.

Normalize the variables using z-score normalization.

Convert to principal component representation and filter out the low-variance principal components. Ultimately, you should end up with only the first two principal components (and the class).

See https://scikit-learn.org/stable/modules/generated/sklearn.decomposition.PCA.html for more information on principal component analysis (PCA) using scikit-learn.

The provided KNIME workflow specifies the necessary preprocessing steps. If you double-click on the **Data Cleaning** and **Apply PCA** nodes, you can view the workflow of these components.

```
In [ ]:
```

Task 1.2: Classifier Construction

Use the dataset as a k-nearest neighbors (KNN) classifier. Use different values for the hyper-parameter *k*, e.g., three, four, five, and six, to fit a KNN classifier. Use different hyper-parameter values for the construction of the classifier.

You may use the KNIME workflow to obtain the training data if you cannot complete the previous task. To export a CSV file with the preprocessed training data from the KNIME workflow, you have to change the *folder* field in the **Create File/Folder Variables** of the provided KNIME workflow in accordance with your directory hierarchy.

See https://scikit-learn.org/stable/modules/generated/sklearn.neighbors.KNeighborsClassifier.html for k-nearest neighbors in scikit-learn.

```
In [ ]:
```

Task 1.3: Classifier Evaluation

Conduct five-fold cross-validation to obtain estimates of the performance of the different knn classifiers.

See https://scikit-learn.org/stable/modules/cross_validation.html for more information cross-validation using scikit-learn.

```
In [ ]:
```

Task 1.4: Deployment

Export one of the previously constructed classifiers as a **pickle** file named "classifier.pkl".

See https://scikit-learn.org/stable/model_persistence.html for options on model persistence and refer to the pickle module documentation.

Install the **KNIME Python Integration** and **KNIME Conda Integration** (optional, if you use Anaconda/Conda) extensions in KNIME. Integrate the pickle file in the provided KNIME workflow and use your model to classify new observations. Note that you have to load the exported pickle file into the data directory of the KNIME workflow (inside the project's directory in your KNIME workspace).

We will take the loans with unknown class from the original dataset and use the trained model to predict the class. Take a look at the preprocessing steps in the KNIME workflow. Notice that the preprocessing of the "new" data to be classified uses the same models for normalization and PCA than the training data, i.e., we use the normalization model and the PCA weight matrix obtained from the training data. Run the KNIME workflow and look at the obtained predictions in the output table of the **Python Script** node.

Note: Take a look at the Python script classifiers. You have to be able to explain what happens there.

Export the updated workflow and upload the file on Moodle.

Case 2: Truck Fleet Maintenance

Files: trucks_training.csv; trucks_test.csv, trucks_full.csv

A transportation company manages a fleet of trucks, each of which is equipped with hundreds of sensors that measure the operating conditions of several components while out on the road.

The Air Pressure System (APS) is one of the components of interest. If a truck's APS is suspected to fail soon, the truck can be proactively called in for maintenance service at a relatively low but non-zero cost. Conversely, if a truck's APS is assumed to be working properly but does fail, the truck must be repaired in the field at relatively high cost.

While the many sensors of a truck's APS do not indicate explicitly whether or not the APS will fail soon, we might be able to identify patterns that would allow for predicting the health of a truck's APS in advance of a failure. Using a suitable predictive model for APS failure would potentially allow the company to reduce maintenance costs by calling in trucks early, before a relatively costly field maintenance is required due to a failure.

Your goal is to construct a predictive model for supporting the following decision:

Which trucks should be called in for APS maintenance service?

Task 2.1: Data Loading

Use pandas to load the dataset **trucks_training.csv** of APS sensor measurements into a dataframe. Each observation describes the sensor measurements for a unique truck. The 170 predictor variables represent the 170 types of sensors. A *class* variable indicates whether a truck's APS has failed (**pos**) or not (**neg**). Rename the class labels: **neg** becomes **ok** and **pos** becomes **failure**.

```
In [ ]:
```

Task 2.2: Data Understanding (I)

Familiarize yourself with the dataset. In particular, look at the count and relative frequency of the observations for **failure** and **ok** classes, respectively. Furthermore, determine the number of missing values for each observation, and determine the number of missing values for each variable. Determine the variance of each variable. Look at the correlation between the class and the different predictor variables. Furthermore, look at the correlation between different predictor variables.

```
In [ ]:
```

Task 2.3: Data Cleaning

Filter out (i.e., remove) low-/zero-variance variables. Filter out (i.e., remove) variables and rows with too many missing values. Otherwise, impute missing values with the variable mean. For each of those variables, keep the variable means; we will need them for the preprocessing after deployment.

```
In [ ]:
```

Task 2.4: Data Understanding (II)

Represent the dataset in principal component form. Normalize the predictor variables, use them to compute a weight matrix, and apply the weight matrix to the predictor variables to compute the principal components of the observations. Note that the transformed dataset still represents exactly the same observations, though they are now expressed in terms of different predictor variables.

Identify the variables (principal components) that comprise just over 50 % of the total variance. To identify those variables, obtain the variance and the cumulative variance per principal component. Relate the variance and the cumulative variance per principal component with the total variance (sum of variance) to obtain the proportion of the variance and the cumulative proportion of the variance per principal component.

```
In [ ]:
```

Task 2.5: Data Preprocessing

Keep only the principal components that comprise just over 50 % of the total variance. Keep all the **ok** observations from the dataset and perform bootstrapping, i.e., random sampling with replacement, of the minority (**failure**) class to obtain a balanced dataset with as many **failure** observations as **ok** observations. Finally, split the dataset into *training set* and *validation set*.

```
In [ ]:
```

Task 2.6: Classifier Training

Use scikit-learn to train a support vector machine (SVM) classifier on the training set. Set the *probability* parameter to **true** in order for the classifier to obtain probabilities for the observations to belong to a certain class.

See https://scikit-learn.org/stable/modules/generated/sklearn.svm.SVC.html#sklearn.svm.SVC for more information on training an SVM classifier in scikit-learn.

```
In [ ]:
```

Task 2.7: Classifier Tuning

Train SVM classifiers with different hyper-parameter settings, e.g., setting the *probability* parameter to **false** (but try also varying the values for other hyper-parameters).

See https://scikit-learn.org/stable/modules/generated/sklearn.svm.SVC.html#sklearn.svm.SVC for more information on the hyper-parameters.

Use different cutoff values for classifying the observations in the validation set as **failure** or **ok**, e.g., classify an observation as **failure** if the predicted probability of that observation belonging to the class is at least 25 %, 50 %, 75 %, 90 %.

Compare precision, recall, and accuracy of using classifiers with different hyper-parameters and cutoff values. Use the **validation set** to test the accuracy.

```
In [ ]:
```

Task 2.8: Evaluation

Now train the classifier on the concatenation of training and validation set using your choice of hyper-parameter settings and remember the cutoff value (if the probabilistic SVM classifier was your choice).

Load the **trucks_test.csv** dataset for the evaluation. To match the original representation of the dataset used to train the classifier, perform the same preprocessing as for the training. In particular, rename the class labels in the new dataset, impute missing values in the **new** dataset with variable means **previously calculated from the original dataset**, and remove variables that were also previously removed. Use the PCA weight matrix **previously calculated from the original dataset** and transform the new dataset into principal component form. Keep only the same variables as in the original dataset. You do not need to balance the dataset.

We are going to evaluate the classifier based on the cost savings with respect to the baseline scenarios. Assume that the cost of maintenance service per truck is 100 euros, the cost of field repair per truck that has failed is 5,000 euros.

Compare the performance of the baseline scenarios---i.e., calling in all trucks for maintenance service and calling in no trucks for maintenance service, respectively---with the chosen classifier in terms of accuracy, precision, and recall as well as total maintenance costs in euros. *Calling in all trucks for maintenance service* corresponds to a classifier that always predicts **failure** for any observation whereas *calling in no trucks for maintenance service* corresponds to a classifier that always predicts **ok** for any observation.

Note that in the following, we treat **ok** trucks as the positive class and **failure** trucks as the negative class. We use the following terminology:

- The proportion of incorrectly predicted trucks as **failure** from among those that are actually good is the **false negative rate**.
- The proportion of correctly predicted trucks as **failure** from among those that are actually **failure** is the **true negative rate**.

- The proportion of incorrectly predicted trucks as good from among those that are actually **failure** is the **false positive rate**.

The total maintenance costs are as follows:

costs = (*costs per maintenance service* × *# trucks for necessary maintenance*) + (*costs per maintenance service* × *# trucks for unnecessary maintenance*) + (*costs per field repair* × *# trucks for field repair*) + (0 × *# other trucks*)

where

- *# trucks for necessary maintenance* = *true negative rate* × *# trucks actually failing*,
- *# trucks for unnecessary maintenance* = *false negative rate* × *# trucks actually not failing*,
- *# trucks for field repair* = *false positive rate* × *# trucks actually failing*, and
- *# other trucks* = *true positive rate* × *# trucks actually not failing*.

```
In [ ]:
```

How do you judge the usefulness of the predictive model?

INSERT YOUR ANSWER HERE!

Task 2.9: Data Preprocessing and Modeling (Revisited)

Go back to the *Data Preprocessing* and *Modeling* stages of the CRISP-DM. Use the **trucks_full.csv** dataset for data mining. You are now free to use different preprocessing techniques (you are *not* limited to principal component analysis) and different algorithms (you are *not* limited to SVM classifiers). Evaluate the performance of each classifier. Choose suitable sampling/validation strategies.