# Data Mining Project #2

**Total points: 100**

## Objective

The purpose of this project is to implement different decision tree (DT) classifiers to predict whether a transaction is fraudulent (positive class, 1) or not (negative class, 0). It is recommended to use Python to implement the algorithms. If you wish to use another language, your team needs to agree, and you also need to let the instructor know.

Teams: You will be in the same team as in project #1.

## Dataset

The dataset includes a number of continuous features that describe a transaction. Their details have been removed to ensure privacy. You will be provided with two files with input data (dataset _train.csv) and test data (dataset_test.csv). The last column is the class label of the instance.

## Evaluation metrics

You will evaluate the models using Accuracy, Precision, Recall, and F1 score. You can use the built-in functions.

## Classification models

You need to generate three different models named:

1) **predict_DT<teamID>**
2) **predict_DT_bagging<teamID>**
3) **predict_DT_randomforest<teamID>**

Your code needs to:
Step 1) Read the two files provided.
Step 2) Split the training data into two parts: training (80%) and validation (20%) sets. Print out the *statistics*, i.e., number of instances and features in the provided files, number of instances from each class in the training/validation/test sets. Remember to set the random seed to your TeamID, in order to replicate your results as needed.
Step 3) For each different value of hyperparameters:

a) Build the corresponding classification model.
b) Generate predictions for the training, validation, test set.
c) Compute the metrics of the prediction on these sets.

Step 4) Model selection & evaluation: For each model, create a table (and store it in a .csv file) with the columns describing the different hyperparameters tried and the accuracy and F1-Score achieved on the training and validation sets for these specific hyperparameters. For model evaluation, print one line where

you will report the accuracy, recall, precision, and F1-Score on the validation and test set of the best model based on the validation F1-Score. (You will need to put these lines together for each model that you will build to create a table requested in the report.)
Step 5) Generate the necessary plots as outlined below.

The above steps need to be clearly outlined in our code, using comments. You can use the scikit-learn package to build your decision trees, sklearn.tree.DecisionTreeClassifier.

More specifically, for **predict_DT**, you will try the following values and select the value that gives you the best F1-score:

```
nleafnodes = [2, 4, 10, 20, 50, 100, 200, 400, 600, None]
for nleaves in nleafnodes:
      clf = DecisionTreeClassifier(max_leaf_nodes=nleaves)
```

Note that when nleaves=None, you will build the full tree. You can use the default values for anything not mentioned/specified here.

**Plot needed #1**: Create a plot to visualize the **accuracy** of the training/validation/test sets, where the x-axis is the `nleafnodes` and the y-axis is the **accuracy** achieved. What can you notice in the graph?

For **predict_DT_bagging**, you will need to build multiple classifiers, each using a bootstrap sample of the training dataset (create a dataset of the same size with sampling with replacement). You will use simple voting for prediction. Note: you need the bootstrap sampling only for the training data, not the validation or test data. You will need to compare the performance when you have `n_base_classifiers = [2, 5, 10, 15, 20]`. You will need to set:

- `max_leaf_nodes = [20, 50, 100, 200, 400]`
- `random_state = TeamID`

**Plot needed #2**: For **predict_DT_bagging**, plot in one graph the test accuracy for the different number of base classifiers used, where the x-axis is the `n_base_classifiers` and the y-axis is the accuracy achieved. There will be two lines in this graph, one for `max_leaf_nodes=20` and one for `max_leaf_nodes=400`. What can you notice in the graph?

For **predict_DT_randomforest, you will extend the previous model by, additionally,** examining only a subset of the available features in each internal node in order to determine the splitting attribute. To do so, you can set the attribute `max_features` of the base `DecisionTreeClassifier()` function. Similarly to before, you will need to build multiple base classifiers (`n_base_classifiers = [2, 5, 10, 15, 20]`) with the following hyperparameters to check:

- `max_features = [0.1, 0.15, 0.2]` (fraction of features are considered at each split)
- `max_leaf_nodes = [20, 50, 100, 200, 400]`
- `random_state = TeamID`

## What to submit

Your report needs to have the following elements:

- Code files.
- The csv files generated by your code.
- Report:
  1. How did your team split the work?
  2. How did your team communicate and work together? You need to keep track of the dates, times, who was present, and whether you met in person or online for all your interactions regarding the project.
  3. Put together a table, where the rows are the different types of models you built, and the columns are the accuracy, precision, recall, and F1-score of the test set (i.e., the line that your code should print at the end of Step 4). What do you notice on that table? Which model performs the best?
  4. Plots #1 and #2 generated in the last step. What can you notice in the graph?