Tutorial_PGMPY_statement

October 4, 2024

1 Bayesian Networks

This TP provides an excellent opportunity to gain further exposure to relevant topics and applications of Bayesian Networks (diagnosis, prediction, etc.).

Objectives: Students will be able to: - Create Bayesian Networks. - Learn Bayesian Networks from real-world problems and datasets. - Determine inferences from Bayesian Networks.

Technology Requirements: - Linux (windows user may install virtual machines) - Python 3.8 or higher (due to pgmpy compatibility) - Download and install pip and then install pgmpy: `$ pip install pgmpy` - If the previous did not work, this can be done manually after downloading the code source:

```
$ git clone https://github.com/pgmpy/pgmpy
$ cd pgmpy/
$ sudo pip install -r requirements.txt
$ sudo python setup.py install
```

*Note: if you encountered problems installing pip or pgmpy, refer to the pgmpy Installation Page: https://pgmpy.org/started/install.html

[ ]:
```
!pip install pgmpy
```

[ ]:
```
import pgmpy
import pgmpy.models
import pgmpy.inference
import pgmpy.estimators
import pgmpy.inference

import networkx as nx
import matplotlib.pyplot
import xgboost

import pandas as pd
import time

from sklearn.metrics import ConfusionMatrixDisplay, accuracy_score
```

1.1 Part 0: Building a model from scratch: Monty Hall Problem

You might know about the famous Monty Hall, or three doors problem (https://en.wikipedia.org/wiki/Monty_Hall_problem). This is a TV game where a contestant *Contestant* interacts with a host *Host* to find a price *Price* hidden behind one of three closed doors. The contestant makes a first choice of one door they would like to open. The host will then help the contestant by opening one of the doors, leaving the contestant to choose again amongst 2 doors. If the chosen door hides the price, the constestant wins.

The host rules for choosing a door to open are: - never open the door behind which the price is hidden - never open the door chosen by the contestant

A mathematician states: "Selecting in the second step the door that wasn't chosen in the first step, has a 66% probability of granting the price to the contestant". We want to use Bayesian Network modeling to affirm or infirm this statement.

We will model this problem with a three states Bayesian Network. - *Contestant* is a random variable with 3 possible values (the first chosen door by the contestant) - *Price* is a random variable with 3 possible values (the price door) - *Host* is a random variable with 3 possible values (the opened door by the host)

The price is placed before the contestant make its first choice, and the contestant has no indication on its location.

Question: From your understanding of the statement, write the independence and conditional independence relations you expect about $Contestant, Price, Host$, in the form "X indep Y" or "X indep Y | Z". (*A relation "X indep Y" means that X and Y value are set independently, without any link or interaction. A relation "X indep Y | Z" means that X and Y have been chosen with respect to a same information which is the value of Z, but other than that, there is no further link between the value of X and Y.*)

TODO 0.1 Instantiate an object you will name `monty_hall`, from the class `pgmpy.models.BayesianNetwork` (https://pgmpy.org/models/bayesiannetwork.html). Add edges with its `.add_edge` method, corresponding to each edge in the Monty Hall graph.

Run the cell after to visualize the network.

[ ]:
```python
# TODO
```

[ ]:
```python
# RUN THIS CELL

def plot_bayesian_network(bn):

    # Convert to a directed graph for visualization
    G = nx.DiGraph(bn.edges())

    # Draw the graph
    nx.draw(G, with_labels=True, pos=nx.circular_layout(G))

plot_bayesian_network(monty_hall)
```

TODO 0.2: BayesianNetwork objects have a method that allows to find automatically the independence relations: `.get_independencies` (https://pgmpy.org/base/base.html). Verify that the relations encoded in the network match your answer to the first question.

If the relations differ, try to explain your reasoning (no point will be deducted for an explained incorrect answer).

[ ]:
```
# TODO
```

Since we have specified the structure of the bayesian network in `monty_hall`, we must now provide the conditional probability distribution.

We assume that both *Contestant* and *Price* are chosen randomly (with probability uniformly $\frac{1}{3}$).

TODO 0.3 Use `pgmpy.factors.discrete.CPD.TabularCPD` (https://pgmpy.org/factors/discrete.html) to specify the conditional probability distribution of *Contestant* and of *Price*. Name the TabularCPD of "Contestant" as `cpd_c` and "Price" as `cpd_p`.

Hint: a coin toss can be modeled by `TabularCPD("coin_toss", 2, [[0.5], [0.5]])`

The name of each variable (in first argument) must be identical to the corresponding node in the BayesianNetwork graph `montly_hall`.

[ ]:
```
# TODO
```

Run the following cell.

[ ]:
```
# RUN THIS CELL

cpd_h = pgmpy.factors.discrete.CPD.TabularCPD("Host",3,
    [
        [0, 0, 0, 0, 0.5, 1, 0, 1, 0.5],   #Host=0
        [0.5, 0, 1, 0, 0, 0, 1, 0, 0.5],   #Host=1
        [0.5, 1, 0, 1, 0.5, 0, 0, 0, 0],   #Host=2
    ],
    evidence=["Contestant", "Price"],
    evidence_card=[3, 3])

cpd_h._truncate_strtable = lambda x: x  # prevents table truncating
print(cpd_h)
```

Question: From the table, what is the value of $\mathbb{P}(Host = 0|Contestant = 1, Price = 2)$? How do you interpret this value with respect to the problem setting?

Question: How many numerical parameters in total were created to build this Bayesian Network? How many would be needed to create the entire joint probability table of $\mathbb{P}(Contestant, Price, Host)$?

Run the following cell to add the probability tables to the `monty_hall` object.

[ ]:
```
# RUN THIS CELL
# We now add the conditional probability distributions (cpd) to the bayesian␣
↪network.

monty_hall.add_cpds(cpd_c, cpd_p, cpd_h)

# Verify that the model is correctly specified

assert monty_hall.check_model()
```

If the previous cell throws an error, you may have mispecified either the monty_hall graph (todo 0.1) or the CPD of *Contestant* and *Price* (todo 0.3).

You can check how `check_model` verifies the BayesianNetwork in its documentation (https://pgmpy.org/models/bayesiannetwork.html)

1.1.1 Application

Suppose that you, as a contestant, have chosen $Contestant = 0$ as your first guess.

The host might open door $Host = 1$, or $Host = 2$ as a result (but never door 0 as it was chosen by the contestant).

TODO 0.4 Using `pgmpy.inference.ExactInference.VariableElimination` (https://pgmpy.org/exact_infer/ve.html), compute the exact conditional probability distribution of $\mathbb{P}(Price|Contestant = 0, Host = 1)$ and $\mathbb{P}(Price|Contestant = 0, Host = 2)$.

Once a VariableElimination object has been set, you can use the class method `VariableElimination.query` with adequate `variables` and `evidence` parameters. The remaining arguments can be set to their default values.

[ ]:
```
inference = pgmpy.inference.ExactInference.VariableElimination(monty_hall)

# TODO
```

Question: According to the two result tables, should you maintain your choice of door $Contestant = 0$, or select the remaining door?

Question: Even though the contestant final choice has one of two doors, why are the probability of success not (0.5, 0.5)?

1.2 Part 1: Bayesian Modeling of a credit risk dataset

This part is dedicated to learning a Bayesian model (graph and distribution) to model Credit risk in Germany. In this part, we will not specify the structure by hand, instead it will be learned from the data.

We will see how to: - open and inspect a dataset as a DataFrame - handle missing values with and without domain knowledge - apply binning to variables with lots of possible values with and without domain knowledge - separate data into training and validation sets

First, run the following cell to open the data in the form of a `pandas.DataFrame`.

```
[ ]: german_credit = pd.read_csv("https://raw.githubusercontent.com/vidhi-chugh/
       ↪tds_articles/master/pgmpy_WICDS/german_credit.csv")
     german_credit
```

Question: What do rows represent? What do columns represent?

Question: What is the name (in the data) of the variable that we want to model?

Question: Are there categorical variables? Continuous? Discrete?

1.2.1 Data Preparation

Before we can use the dataset to model credit risk, we must prepare the data to ensure that it can be used properly. This generally involves three steps. - Ensuring that there are no missing values - Ensuring that all columns have a type that allows Bayesian Network structure learning and estimation. Specifically, all variables should be discrete, with a sufficiently low number of possible values (https://pgmpy.org/index.html). - Separing the data into training and validation sets. The training set is used for structure learning and parameter estimation. The validation set is used to compare different learned models on a prediction task.

We will first handle missing (NaN) values. - A domain expert tells us that missing values in `"Saving accounts"` are valid. A NaN values show that the credit seeker does not have a saving account. We will replace this value by `"no account"`.

```
[ ]: # RUN THIS CELL

     # in this cell, german_credit['Saving accounts'].isnull() provides the index␣
       ↪table of all null saving account rows.
     # then, german_credit.loc[... ,'Saving accounts'] select the cells to be set to␣
       ↪the value "no account"

     german_credit.loc[german_credit['Saving accounts'].isnull(), 'Saving accounts']␣
       ↪= 'no account'
```

TODO 1.1: Is there any other column with missing values? You can use the `DataFrame.isnull()` method to check if a value is null, and the `DataFrame.sum(axis=0)` method to get the number of nonzero values in each column.

(https://pandas.pydata.org/docs/reference/api/pandas.DataFrame.isnull.html)
(https://pandas.pydata.org/docs/reference/api/pandas.DataFrame.sum.html)

If there is any missing value remaining, remove the corresponding rows from the DataFrame, using the `.dropna()` method.

(https://pandas.pydata.org/docs/reference/api/pandas.DataFrame.dropna.html)

```
[ ]: # TODO
```

Now that we have dealt with missing values, we must ensure that variables are discrete with low number of possible values. We will arbitrarily say that **above 10 values** is too many.

When a variable has too many values, we must apply binning to it. Binning means grouping together values under new names, into a small number of categories. There are different strategy to binning. - A domain expert might create meaningful binning from prior knowledge. For instance, age is usually grouped in 18-25, 26-45, 46-65, 65+ categories. - When no prior knowledge is available, a common strategy on ordered data (where a "lesser than" notion makes sense) is to create bins of equal population with ordered values. For instance, age could be grouped in three groups from 18 to 30, 31 to 37, 37+ if each group correspond to roughly the same number of observations (rows).

The following cell iterates over the columns of the dataset, and for each of them, prints the number of unique values contained in the column. It uses `pd.DataFrame.columns` (https://pandas.pydata.org/docs/reference/frame.html) and `pd.Series.unique` (https://pandas.pydata.org/docs/reference/series.html).

```
[ ]: # RUN THIS CELL
     for variable in german_credit.columns:
         print("Variable '{}' has {} possible values".format(variable,␣
      ↪len(german_credit[variable].unique())))
```

Question Which columns seem to have too many unique values to handle like categorical variables?

TODO 1.2: Apply the function `pandas.qcut` (https://pandas.pydata.org/docs/reference/api/pandas.qcut.html), with quantiles `q=[0, .33, .66, 1]` to make three equally populated bins, for `Age`,`Credit amount`, and `Duration`.

You should replace the previous column with the new value: syntax `dataframe[column] = pd.qcut(...)`.

```
[ ]: # TODO
```

```
[ ]: # RUN THIS CELL
     # verify that Age, Credit amount and Duration have been correctly prepared
     for variable in ["Age", "Credit amount", "Duration"]:
         assert len(german_credit[variable].unique())==3
         print("Variable '{}' has the following list of possible values: {}".
      ↪format(variable, german_credit[variable].unique()))
```

TODO 1.3: To separate our Train/Validation sets, use the method `DataFrame.sample` (https://pandas.pydata.org/docs/reference/api/pandas.DataFrame.sample.html) to sample without replacement a fraction 0.9 of the dataset german_credit, called `train_set`. Use the method `DataFrame.drop` (https://pandas.pydata.org/docs/reference/api/pandas.DataFrame.drop.html) with indexes `train_set.index` to create a validation set `val_set`. **IMPORTANT** for reproducibility, make sure to seed your sample selection with parameter `random_state`.

```
[ ]: # TODO
```

```
[ ]: # RUN THIS CELL

     assert len(train_set)==900
     assert len(val_set)==100
```

We will put aside the validation set for the remainder of part 1 and part 2.

1.2.2 Model learning

To learn a Bayesian Network structure fitting the data, `pgmpy` provides several algorithms. We will use Hill Climb Search (https://pgmpy.org/structure_estimator/hill.html). This algorithm: - is greedy: a scoring function is optimized by making gradual modification to a Directed Acyclic Graph, until no modification can improve the score (hence, hill-climbing). - is heuristic: the scoring function represents some sort of belief of what a best network structure would be. For instance, a best graph would both fit the data well while having a small *description length* (an encoding of this graph should be as small as possible) (BIC score, AIC score). Another best graph concept would rely on fitting the data well while maximizing a probability over a distribution of graphs (K2 score, DBE score…). See http://www.lx.it.pt/~asmc/pub/talks/09-TA/ta_pres.pdf for more details.

TODO 1.4: Instanciate `pgmpy.estimators.HillClimbSearch` on the training set. Apply the method `HillClimbSearch.estimate` on the instance, with `k2score`, and otherwise default parameters. Finally, reuse `plot_bayesian_network` defined in part 0 to plot the learned network.

```
[ ]: # TODO
```

TODO 1.5: The `estimate` method argument `tabu_length` controls how many of the last graph modifications are impossible to modify. Run another graph estimation while setting `tabu_length=1`. Is there any change to the graph?

```
[ ]: # TODO
```

TODO 1.6: As mentionned before, different scoring functions encode different beliefs of what an optimal graph would look like. Apply the `estimate` method with `bicscore` scoring method. The BIC (Bayesian Information Criterion) penalizes graphs with a lot of edges, under the belief that the simplest structure is the most likely (Occam's razor https://en.wikipedia.org/wiki/Occam%27s_razor).

```
[ ]: ## TODO
```

Question To your opinion, which graph is simpler to interpret?

Question To your opinion, which graph is faster to evaluate?

We will now use the graph produced by the `bicscore` method. We now need to learn a set of factor functions to model the joint probability distribution. Precisely, for each node V with parents (in-going edges) $Pa(V)$, we learn $\mathbb{P}(V|Pa(V))$.

The library pgmpy provides several parameter estimation algorithms, each with different particularities. The one we will use is the following: - Maximum Likelihood Estimation (https://pgmpy.org/param_estimator/mle.html): estimates joint probabilities by using the frequency/count of each configuration of values. It needs all variables to be observed in the data.

There are other methods, that do not apply to our case: - Expectation Maximization (https://pgmpy.org/param_estimator/em.html): Used in the presence of *latent* variables, i.e. variables in the graph for which there is no data. Expectation Maximization does not maximize the likelihood of the joint distribution, but the expectation of this likelihood. - Bayesian Estimation (https://pgmpy.org/param_estimator/bayesian_est.html): Used when domain knowledge, or belief, is known beside what is recorded in the data. This knowledge is specified to the algorithm in the form of *priors*. Due to the additional information, parameter estimates typically require less observations than Maximum Likelihood Estimation.

TODO 1.7: Apply MLE (https://pgmpy.org/param_estimator/mle.html) with its `.get_parameters()` method to estimate the parameters of the Bayesian Network. Add the parameters to a model object (class `pgmpy.models.BayesianNetwork`) built from the graph structure obtained with bicscore. You will name this model `model`. Use the training data only.

```
[ ]: # TODO
```

```
[ ]: # RUN THIS CELL
     assert model.check_model()
```

1.3 Part 2: Bayesian Inference

In this part, we will use the previous model to analyse the mechanisms behind Credit Risk attribution by banks.

We will first examine relations between `Gender` and `Risk`.

TODO 2.1: Use Variable Elimination to print the joint distribution $\mathbb{P}($`Gender`,`Risk`$)$. Use Variable Elimination to print the conditional distribution $\mathbb{P}($`Risk`$|$`Gender`$)$ for both Gender values.

```
[ ]: # TODO
```

Question: Are gender differences associated to lowered risk? What is hidden in the conditional distribution, that is apparent in the joint distribution? What prior would be needed to deduce the joint distribution $\mathbb{P}($`Gender`,`Risk`$)$ from the conditional distributions $\mathbb{P}($`Risk`$|$`Gender`$)$?

TODO 2.2: Examine how some of the other variables are associated to varying Risk (using conditional distributions). Write down your conclusions.

```
[ ]: # TODO
```

BONUS Question Job categories go from "0 - unskilled" to "3 - highly skilled". Does the risk evolution in function of the Job attribute correspond to your expectations? If not, what could be the reason for the difference?

The Bayesian Network associated to german credit has 9 variables. The previous joint and conditional distribution only involve 2 of them. Therefore, the algorithm has eliminated 7 variables.

The computation time of the inference might change depending on the order in which variables are eliminated. The elimination order is controlled by the parameter `elimination_order` of the `.query` method.

TODO 2.3 For an identical query of joint probability $\mathbb{P}(Risk, .)$ between the Risk and an arbitrary variable, measure the time spent when given different elimination orders (`greedy`, `MinFill`, `MinNeighbors`, `MinWeight`). You may use `time.time()` to obtain the current time.

```
[ ]: # TODO
```

The Bayesian Network can help us with *Feature Selection*. The goal of feature selection is to select a subset of variables that contain all the necessary information about `Risk`, i.e., the *Markov Blanket*.

For instance, variable `Job` might have information on `Risk` (i.e., not independent), but this information might already be contained in another variable, making `Job` redundant.

TODO 2.4 To know which variables can be made independent from `Risk` by conditioning on other variables, it is possible to use the graph structure. Find a Markov Blanket of `Risk`. The method `BayesianNetwork.get_markov_blanket` (https://pgmpy.org/base/base.html) is ready made for this task.

```
[ ]: ## TODO
```

Question Interpret the markov blanket information: what is a markov blanket and what does it mean regarding how banks attribute credit risk rating?

BONUS: causal inference   This bonus part covers an application of Bayesian Network not covered during the CM. You may go to part 3 directly then come back to it later.

We will now consider the variable `Job`, for which we observe that Risk is different depending on the category. We can say clearly that Job and Risk aren't **independent**. What we cannot say while looking uniquely at the conditional probability distribution, is whether `Job` influences (is a cause of) `Risk`. Indeed, variables can be associated but not causated.

It is generally difficult to determine causal relationships from observed data. Assuming that causal effects are *linear* (meaning that if `Job` has an influence on `Risk`, then this influence can be correctly modeled by a linear regression model), the causal inference framework (https://pgmpy.org/exact_infer/causal.html) allows to estimate whether two associated variables have a causal influence on each other (Average Treatement Effect https://en.wikipedia.org/wiki/Average_treatment_effect). The question it solves is the following: "if a credit seeker's job category were to increase due to a job change, how much would Risk increase?"

Precisely, the ATE is the difference between the expected Risk when the job stays the same, compared to the expected Risk when the job category increases.

TODO 2.6: Apply `pgmpy.inference.CausalInference` with its method `.estimate_ate` on training data, to infer if `Job` has an influence on `Risk`.

```
[ ]: #TODO
```

Question: Is Job category causaly linked to Risk?

We now consider the variable `Duration`. We can confirm that it is associated to `Risk`, with higher credit amount related to greater proportion of 'bad' risk. Again, we would like to know if this is a cause-effect relation.

We will now use another way of quantifying causal relationships: *do-calculus* with *counterfactuals*. The question it solves is: "if all credit seekers were to be attributed the same duration by the bank, how would risk be distributed?" If Duration and Risk are not causaly linked, then the resulting distribution of Risk will be identical for all durations.

TODO 2.7: Use `CausalInference.query` to estimate the Risk after a do operation on Duration for each category of duration (specify `do={"Duration":value}` for appropriate values).

```
[ ]: #TODO
```

Question: Is Duration causaly linked to Risk?

1.4 Part 3: Bayesian models as predictive models

Now that we have learned a model of the training data, we can apply it to validation data to evaluate how it works as a predictive model. We can use Maximum A Posteriori inference, to find out what Risk is most likely considering the values of the other variables. Formally, for a given validation sample s where Risk is removed, we want to estimate $\arg\max \mathbb{P}(Risk|Gender, ..., Housing = s)$.

TODO 3.1: Use `VariableElimination.map_query` (https://pgmpy.org/exact_infer/ve.html) to predict and store credit risk for each sample in the validation set. You can iterate over samples by using `val_set.to_dict()`, each row being converted to a dictionary. Make sure to not include `Risk` in the evidences.

You will need to store the true risk in a list `y_true` for each sample, and store the corresponding predicted value into a list `y_pred`.

```
[ ]: #TODO
```

Run the next cell to visualize the classification performance.

```
[ ]: # RUN THIS CELL

     print("Classification accuracy:" , accuracy_score(y_true, y_pred))
     ConfusionMatrixDisplay.from_predictions(y_true, y_pred)
```

Question The previous snipet plots a Confusion Matrix. Briefly explain what it means.

Question What is the link between the confusion matrix and the accuracy metric?

Question What would be False Positive and False Negative in this confusion matrix? Which one would you rather minimize if you were to work for a bank?

We would like to compare the previous map_query predictive model to a standard machine learning model.

We will use XGBoost, one of the main libraries of simple learning models based on "boosting" and specifically "gradient boosting" (https://xgboost.readthedocs.io/en/stable/).

XGBoost is a library that can only manipulate numerical values. Columns "Gender", "Housing", "Saving accounts", "Purpose", and "Risk" contain strings, so we need to transform them. For each of these columns, the next cell creates a new column with suffix "_numerical" (i.e. "Gender_numerical"), where each of the string values is replaced by an integer. This can be done with `.apply` (https://pandas.pydata.org/docs/reference/api/pandas.Series.apply.html) and a function, or `.map` (https://pandas.pydata.org/docs/reference/api/pandas.Series.map.html) and a dictionary.

```
[ ]: # Run this cell.

     #method 1
     label_mapping = {'good': 0, 'bad': 1}
     val_set['Risk_numerical'] = val_set['Risk'].map(label_mapping)
     train_set['Risk_numerical'] = train_set['Risk'].map(label_mapping)

     #method 2
     label_mapping = dict([(value,index) for index,value in␣
      ↪enumerate(german_credit["Gender"].unique())])
     val_set['Gender_numerical'] = val_set['Gender'].map(label_mapping)
     train_set['Gender_numerical'] = train_set['Gender'].map(label_mapping)

     #method 3
     label_mapping = lambda x: german_credit["Housing"].unique().tolist().index(x)
     val_set['Housing_numerical'] = val_set['Housing'].apply(label_mapping)
     train_set['Housing_numerical'] = train_set['Housing'].apply(label_mapping)

     label_mapping = lambda x: german_credit["Saving accounts"].unique().tolist().
      ↪index(x)
     val_set['Saving accounts_numerical'] = val_set['Saving accounts'].
      ↪apply(label_mapping)
     train_set['Saving accounts_numerical'] = train_set['Saving accounts'].
      ↪apply(label_mapping)

     label_mapping = lambda x: german_credit["Purpose"].unique().tolist().index(x)
     val_set['Purpose_numerical'] = val_set['Purpose'].apply(label_mapping)
     train_set['Purpose_numerical'] = train_set['Purpose'].apply(label_mapping)
```

Run the next cell. Its purpose is to make XGBoost and Pandas aware of the categorical nature of some columns.

```
[ ]: ##  Run this cell.
     # Make sure that the column names correspond.

     train_set['Purpose_numerical'] = train_set['Purpose_numerical'].
      ↪astype("category")
     val_set['Purpose_numerical'] = val_set['Purpose_numerical'].astype("category")
     train_set['Saving accounts_numerical'] = train_set['Saving accounts_numerical'].
      ↪astype("category")
     val_set['Saving accounts_numerical'] = val_set['Saving accounts_numerical'].
      ↪astype("category")
     train_set['Housing_numerical'] = train_set['Housing_numerical'].
      ↪astype("category")
     val_set['Housing_numerical'] = val_set['Housing_numerical'].astype("category")
     train_set['Gender_numerical'] = train_set['Gender_numerical'].astype("category")
     val_set['Gender_numerical'] = val_set['Gender_numerical'].astype("category")
```

```
train_set['Risk_numerical'] = train_set['Risk_numerical'].astype("category")
val_set['Risk_numerical'] = val_set['Risk_numerical'].astype("category")
train_set['Duration'] = train_set['Duration'].astype("category")
val_set['Duration'] = val_set['Duration'].astype("category")
train_set['Age'] = train_set['Age'].astype("category")
val_set['Age'] = val_set['Age'].astype("category")
train_set['Credit amount'] = train_set['Credit amount'].astype("category")
val_set['Credit amount'] = val_set['Credit amount'].astype("category")
```

TODO 3.2 In the next cell, create an XGBClassifier model (https://xgboost.readthedocs.io/en/stable/python/python_api.html#xgboost.XGBClassifier) with adequate parameter values. Train (`.fit`, https://xgboost.readthedocs.io/en/stable/python/python_api.html on `train_set` with the columns `full_variables` to predict "Risk_numerical". Predict (`.predict`, https://xgboost.readthedocs.io/en/stable/python/python_api.html#xgboost.XGBClassifier.predict) on the validation set `val_set`, and store the result as `y_pred`.

Hint: for categorical values to be handled, a parameter "enable_categorical" must be set to true at model declaration.

[ ]:
```
full_variables = ["Purpose_numerical", "Saving accounts_numerical",␣
 ↪"Housing_numerical", "Gender_numerical", "Credit History", "Age", "Job",␣
 ↪"Credit amount", "Duration"]

## WRITE YOUR CODE BELOW
```

Run the next cell to evaluate the performance of the model.

[ ]:
```
y_true = val_set["Risk_numerical"].values
print("Classification accuracy:" , accuracy_score(y_true, y_pred))
ConfusionMatrixDisplay.from_predictions(y_true, y_pred)
```

TODO 3.3 Train and evaluate another XGBClassifier, this time with input data restricted to the markov blanket found in TODO 2.4.

[ ]:
```
## WRITE YOUR CODE BELOW
```

[ ]:
```
# Run the following

y_true = val_set["Risk_numerical"].values
print("Classification accuracy:" , accuracy_score(y_true, y_pred))
ConfusionMatrixDisplay.from_predictions(y_true, y_pred)
```

Question: Conclude: did removing columns decrease the information on Risk contained by the dataset? Was this an expected behaviour?

Question: Did removing column increase accuracy? Suggest an explanation why.

Question: Conclude on the predictive performance of bayesian modeling versus XGBoost for this dataset.