Module 6 - Deployment of predictive models, dissemination of knowledge and additional topics

Assignment overview

For this assignment, it is possible to work in **groups of up to 2 students**. Read the instructions carefully, as they may assign tasks to specific students.

Group members

Leave blanks if group has less than 2 members:

- Student 1: Jingyuan Liu (S.N. 69763183)
- Student 2: Nicholas Tam (S.N. 45695970)

Learning Goals:

After completing this week's lecture and tutorial work, you will be able to:

1. Provide examples of what happens when poorly designed models are deployed
2. Recognize the impact of technologies on at risk populations
3. Recognize and discuss the impacts of addictive online environments that optimize on metrics that do not benefit its users
4. Discuss how data science techniques can be used to mislead
5. Identify how one's identity connects to interpretation of results (e.g., confirmation bias)
6. Discuss the ethics of displaying and reporting findings
7. Explain Goodhart's Law: when the metric becomes the target
8. Understand and assess the environmental impact of data science and AI applications

Goodhart's law - scenario

In class, we cited Goodhart's law:

> *When a measure becomes a target it ceases to be a good measure.*

Let's see how this plays out in a real-life scenario. Start by reading this article by Cathy O'Neil, based on her book *Weapons of Math Destruction*: "How Big Data Transformed Applying to College".

This article describes the importance of a particular metric, the *U.S. News & World Reports* college rankings, for colleges and students applying to those

colleges. These scores are supposed to provide a quantitative, objective measure of the quality of an institution. Colleges know that students and their parents believe in the importance of these rankings, and will go to great lengths to make sure their score is high. This system, however, presents several issues and has had negative consequences on students.

Question 1

Let's start by looking at the algorithm used to produced the rankings. How would you evaluate this algorithm using our FATE lenses? Do you see issues of fairness, accountability, and/or transparency?

- I would argue that this algorithm is not very ethical using the "Common Good Perspective". For example, the article mentions that the algorithms "keep tuitions rising, parents worrying, and kids suffering" and high-school seniors have "more and longer levels of anxiety senior year while waiting for acceptance and rejection letters" because the algorithm is in favor of low acceptance rates. This algorithm puts too much stress, anxiety, and burden on students and parents; the overall well-being of the community do not improve.
- Fairness issue: Apollo Education Group had their advertising for the University of Phoenix designed to target people who were eligible for financial aid, including poor single mothers, recent immigrants, or veterans desperate for a new life. This is unfair to those who were not eligible for financial aid. In addition, for-profit colleges such as the University of Phoenix can spend much more money on marketing and gaming the system; this might be unfair competition for those non-profit colleges.
- Accountability issue: The ranking system being taken as fact leads to many universities and colleges twisting their admissions processes from improving academics to improving their rankings. Who is to blame when gaming of the system is at least in part from the system being taken by many at face value? It is also unclear who should be responsible for when schools lie and cheat to game the system and who should scrutinize the algorithms.
- Transparency issue: The scoring algorithm is unclear, opaque, and changes over time, so it is difficult to explain how certain outcomes come about beyond proxies.

Question 2

List at least 3 ways, cited in the articles, in which colleges are trying to inflate their ranking, and explain why these actions, while boosting a college's score, are not effective at improving the college's quality or the students' experience.

- *"Bucknell University lied about SAT averages from 2006 to 2012, and Emory University sent in biased SAT scores and class ranks for at least 11 years, starting in 2000. Iona College admitted to fudging SAT scores, graduation*

rates, retention rates, acceptance rates, and student-to-faculty ratios in order to move from 50th place to 30th for nine years before it was discovered. Law schools that want to buff up their recent graduate employment numbers have been known to temporarily hire some of those former students or even count nonresponses to employment surveys as 'employed'". Some colleges tried to inflate their ranking by manipulating statistics such as SAT scores, graduation rates, retention rates, student-to-faculty ratios, and graduate employment numbers (employment rates). These manipulated statistics do not reflect the real-world outcomes or educational quality of colleges. For example, these inflated graduate employment numbers (employment rates) are unreliable, even biased, and do not reflect the real-world scenario. Consequently, schools with higher recent graduate employment numbers do not imply that their students will get better employment opportunities after graduation, thus not improving students' experience but diminishing the reliability of schools.

- *"Things like extremely plush living quarters and fancy athletic facilities—some even equipped with lazy rivers—are becoming par for the course. There's nothing inherently academic about most of these changes, but they cost money, both in their construction and in the number of administrators it takes to arrange it all"*. Some colleges tried to inflate their ranking by building many luxurious facilities such as "plush living quarters" and "fancy athletic facilities", and "lazy rivers" in order to attract high-achieving students. These changes are mostly irrelevant to the college's academic quality but cost a lot of money. Consequently, colleges significantly increase costs, leading to a significant increase in tuition that burdens students without improving the college's academic quality or the students' experience.
- *"Rankings go up if colleges appear exclusive, which is determined by various metrics including what proportion of applicants get accepted (better to be low), what proportion of accepted applicants actually enroll (better to be high)"*. Some colleges tried to inflate their rankings by lowering acceptance rates and increasing enrollment rates of accepted students. The increasing prestige leads to more applications and thus rejections, then to more exclusivity, which is not effective at improving the college's quality or the students' experience. The exclusivity does not imply college's quality or the students' experience but puts stress on students and parents.

Question 3

The article mentions that metrics boosting a college's ranking include low acceptance rates (a sign of exclusivity) and high enrollment rates from accepted students (a sign of high desirability). It also explains that this resulted in an increase in applications to different schools (*"the number of students applying to at least seven has tripled since 1990, from 9 percent to 29 percent"*) and in the *death of the concept of safety school*. What do you think the author means by this? What is the connection between the U.S. News rankings and the chances of being invited to apply to a "safety school"?

For clarity, here is a definition of "safety school", according to the Princeton Review: *"A safety school is one where your academic credentials exceed the school's range for the average first-year student. You should be reasonably certain that you will be admitted to your safety schools"*.

I think the author means that the traditional concept of "safety school" no longer holds under this ranking algorithm. Colleges prefer exclusivity (low acceptance rates) and desirability (high enrollment rates) to improve their rankings. Consequently, some less competitive schools become more selective because they want to keep high enrollment rates, making students much less confident about their admissions to their safe schools. For example, if a less competitive school believes that a high-standing student is unlikely to accept the offer because he/she has better choices, then this less competitive school may not give the student an offer because they don't want to see a low possibility for enrollment, even though the student may be high-achieving.

U.S. News rankings focus on exclusivity (low acceptance rates) and desirability (high enrollment rates) for higher rankings. Therefore, even some low-ranking and less competitive traditional safety schools start appearing more selective by reducing the number of offers to those over-standard students unlikely to enroll to maintain high enrollment rates. Consequently, we no longer have any real "safe schools" because those "safety schools" also aim to improve their rankings, thus also having low acceptance rates. The "safety schools" are no longer "safe" after the application of U.S. News rankings algorithm.

Safety schools are designed to allow for high acceptance rates, yet the U.S. News & World Reports ranking system encourages lower acceptance rates for better rankings. Better rankings also add to a school's desirability, and with students applying for a larger number of schools, the possibility of accepted students not enrolling in safety schools increases, which further degrades the ranking of said safety schools. As a consequence, safety schools cannot persist in the long term due to the cycle of poor ranking, poor desirability, and poor enrollment, unless they decrease their acceptance rates to increase their ranking and ironically diverge from the definition of safety schools in the first place.

Addictive Online Environments

The addictive power of some technologies and online environments is becoming widely known. In class, we talked about some of the typical features a developer may include in their app or website to make them more addictive. The list includes:

- endless scrolling
- pull-to-refresh
- never-ending auto-play videos

- push notifications
- temporarily available stories
- likes
- read-receipts

As part of this exercise, we would like each member of the group to go through their phone and find one app with addictive features and one without, and describe them here. Make sure to pick apps you feel comfortable talking about. Answer the question by editing the templates below (one for each student - if you are working alone, you can delete the second template or leave it blank).

Student 1: Jingyuan Liu

App with addictive features

App name: TikTok

App main functionality/goal: Short-form video hosting software and social media platform

Decription of addictive features: Endless scrolling (continuously shows videos with no stopping point as long as you scroll down.), pull-to-refresh (easy to refersh to show new videos), never-ending auto-play videos, push notifications (for recommended videos), likes and reposts

App frequency of use: Daily, sometimes even hourly (weekly? daily? hourly?)

If you are comfortable doing so, describe your use of the app. Do you open it only when you need to complete a specific task? Or also in an unplanned, more spontaneous manner? How long do you stay on the app after opening it? Do you find some app features compelling to open it or to use it for a longer time (e.g. push notifications)?

In general, I open TikTok in an unplanned, frequent, and spontaneous manner, rather than for a specific task. My time spent on the app varies; it could be as short as less than a minute to as long as about an hour. I find the "endless scrolling" feature particularly compelling. Once I start watching short-form videos, it becomes difficult to stop because I am always curious about what the next video will be.

App without addictive features

App name: Canvas

App main functionality/goal: Teaching and learning software, for assignment submission, study materials viewing, etc.

App frequency of use: Several times a day (weekly? daily? hourly?)

If you are comfortable doing so, describe your use of the app. Do you open it only when you need to complete a specific task? Or also in an unplanned, more spontaneous manner? How long do you stay on the app after opening it? Do you find some app features compelling to open it or to use it for a longer time, which were not listed as addictive features?

I usually use this app for study purpose. For example, I use it to submit assignments, view my grades and feedback, check course materials and announcements, etc. I only open it when I need to complete a specific task (i.e. submit an assignment). I did not find many features particularly compelling to open it or to use it for a longer time. If I would say one, I think it is the multifunctionality of this app. For example, I can access different platforms such as Piazza, Gradescope, iClickerCloud and complete various tasks on this Canvas.

Student 2: Nicholas Tam

App with addictive features

App name: Instagram

App main functionality/goal: Social media platform

Decription of addictive features: Impermanence, infinite scrolling mechanism, variable rewards, personalisation, fear of missing out.

App frequency of use: Hourly (weekly? daily? hourly?)

If you are comfortable doing so, describe your use of the app. Do you open it only when you need to complete a specific task? Or also in an unplanned, more spontaneous manner? How long do you stay on the app after opening it? Do you find some app features compelling to open it or to use it for a longer time (e.g. push notifications)?

Opened frequently and spontaneously. Usually opened for a few minutes at a time each, unless using it to contact people. Sharing posts is often compelling, especially due to the posts being impermanent.

App without addictive features

App name: Gmail

App main functionality/goal: Electronic mail service

App frequency of use: Several times a day (weekly? daily? hourly?)

If you are comfortable doing so, describe your use of the app. Do you open it only when you need to complete a specific task? Or also in an unplanned, more spontaneous manner? How long do you stay on the app after opening it? Do you find some app features compelling to open it or to use it for a longer time, which were not listed as addictive features?

Often only for specific task, or if message of interest pops up. A while, depending on what tasks I am doing.

Misleading Data Science Techniques

There are many ways in which data science may lead to the wrong conclusions, either by error or intentionally. A very good summary of such techniques is offered by Carl Bergstrom and Jevin West in their course "Calling Bullshit in the Age of Big Data", offered at the University of Washington and with recordings available online.

We are going to review some of these techniques here (most of them have to do with visualization):

- Dataviz in the popular media.
- Misleading axes.
- Manipulating bin sizes.
- Dataviz ducks.
- Glass slippers.
- The principle of proportional ink.
- Correlation vs. Causation

Now that you have familiarized with some misleading data science techniques, let's try to evaluate a real scenario. Here is an article, published on Nature in 2004, claiming that trends in the winning times of the men's and women's Olympic 100-meters sprint show that women are closing the time gap, and will one day (some time around 2156, to be exact) complete the race faster than men athletes.

Question 4

Do you think that the claim made by the article is true? If not, what mistake(s) did the authors commit that led them to this wrong conclusion?

The claim is not true. They have committed an erroneous extrapolation, assuming that the linear trend will continue beyond the scope of data provided from 1900 to 2000. They have also assumed that the correlation of the difference in winning times and the years imply that the year directly influences the decrease in winning time difference and fail to consider other factors.

AI and the environment

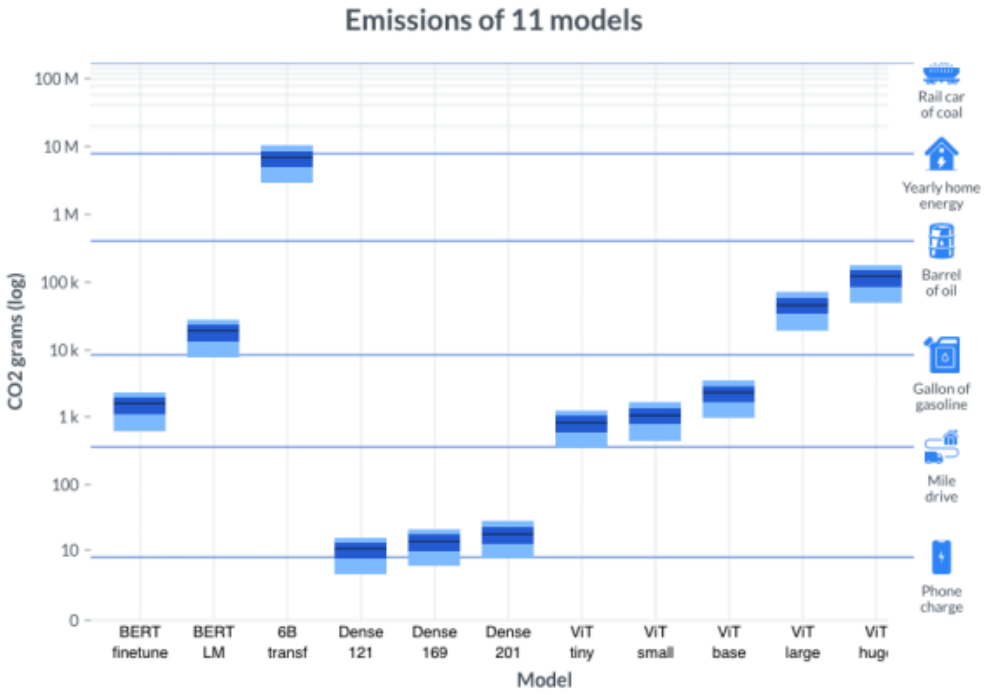
Source: The Carbon Footprint of Artificial Intelligence

AI applications are becoming ubiquitous in society, from recommendation algorithms on streaming services and social media, to semi-autonomous vehicles, large language models and more. Few people, however, including few of the

developers of these systems, spend time thinking about their environmental impact. Even fewer try to evaluate this impact.

The cost of training and maintaining AI algorithms is not immediately evident. The most obvious is the massive amount of electricity that they require. Depending on how that electricity is produced, there is also a variable cost in terms of CO2 released in the athmosphere. And we should not forget the cost of building and maintaining the hardware on which AI algorithms run, from the extraction of the raw materials, to the manifacturing and transportation of this hardware, concluding with the cost of disposing of it once it reached the end of its lifespan. Recent estimates of AI's carbon footprint range from 2.1% to 3.9% of the total greenhouse gas emissions. Still a small fraction of what is produced by more polluting industries such as manufacturing (24%) and transportation (27%), but not insignificant.

In April 2022, AI startup Hugging Face released an article with an estimate of the carbon footprint of one of its own AI models, comprehensive of all emissions produced during the model's whole life cycle. They estimated that the training alone of their model resulted in 25 metric tons of CO2 emissions. When they included the emissions produced by the manufacturing of the hardware and the broader computing infrastructure, and the energy required to run the trained model, the esitmate doubled to 50 metric tons of CO2 emissions, similar to the amount of CO2 produced by 11 gasoline-powered cars driven for one year.



Training neural network models takes more time and energy than simply using them for a single prediction. Here are some estimates for the emissions of 11 popular NLP and computer vision models.

Jesse Dodge, a research scientist at the Allen Institute for AI, said: "When AI developers create new systems, they're trying to push performance, accuracy, or the model's capabilities, and they focus less on efficiency". And it turns out that this focus on accuracy above everything else is costly. Here is a quote from a 2022 paper by Mill et al.:

There is a recognised trade-off between model accuracy and energy efficiency. In fact, the relationship has been shown to be logarithmic. That is, in order to achieve a linear improvement in accuracy, an exponentially larger model is required. A recent study [...] confirmed the existence of the accuracy-energy trade off [...] and indicated a 30-50% saving in energy for training related to a 1% reduction in accuracy.

In this portion of the assignment, we will increase our awareness of the carbon footprint of our algorithms, thanks to a software tool called CodeCarbon. Let's start by installing the required packages in your environment (you can delete or comment out this cell after you are done).

In [2]:
```
# !pip install codecarbon
# !pip install transformers
```

This next cell is for importing CodeCarbon, as well as other libraries we will need for this exercise.

In [1]:
```
from codecarbon import EmissionsTracker
from transformers import pipeline

import matplotlib.pyplot as plt
import numpy as np
import pandas as pd
import xgboost as xgb
from sklearn.compose import ColumnTransformer, make_column_transformer
from sklearn.ensemble import RandomForestClassifier
from sklearn.impute import SimpleImputer
from xgboost import XGBClassifier
from scipy.stats import lognorm, loguniform, randint
from sklearn.model_selection import (
    GridSearchCV,
    RandomizedSearchCV,
    cross_val_score,
    cross_validate,
    train_test_split,
)
from sklearn.pipeline import Pipeline, make_pipeline
from sklearn.preprocessing import OneHotEncoder, StandardScaler
from sklearn.svm import SVC
```

```
C:\Users\lcm57\miniconda3\envs\dsci430\lib\site-packages\tqdm\auto.py:21: TqdmWar
ning: IProgress not found. Please update jupyter and ipywidgets. See https://ipyw
idgets.readthedocs.io/en/stable/user_install.html
  from .autonotebook import tqdm as notebook_tqdm
```

Let's see how the tool works in a toy example. In this cell, we are first creating an EmissionTracker. Then, we can call start() to start tracking the power required by

following operations, thus producing an estimate of the associated carbon emissions. Stop tracking by calling stop() at the end of the operations you want to produce an estimate for. In this case, we are estimating the cost of 100,000 simple additions.

In [7]:
```
tracker = EmissionsTracker(log_level='error')
tracker.start()
try:
    for i in range(100000):
        _ = 1 + 1
finally:
     tracker.stop()
```

In [8]:
```
# Visualize the results
tracker.final_emissions_data
```

Out[8]:
```
EmissionsData(timestamp='2024-11-29T21:42:24', project_name='codecarbon', run_i
d='6cf064a9-453f-41f1-a571-157a1317aaa7', experiment_id='5b0fa12a-3dd7-45bb-976
6-cc326314d9f1', duration=0.05149010004242882, emissions=9.44973804533264e-09,
emissions_rate=1.8352533860967209e-07, cpu_power=42.5, gpu_power=0.0, ram_power
=11.883976936340332, cpu_energy=6.061845833755797e-07, gpu_energy=0, ram_energy
=7.369782284457321e-08, energy_consumed=6.798824062201529e-07, country_name='Ca
nada', country_iso_code='CAN', region='british columbia', cloud_provider='', cl
oud_region='', os='Windows-10-10.0.22631-SP0', python_version='3.10.14', codeca
rbon_version='2.8.0', cpu_count=14, cpu_model='Intel(R) Core(TM) Ultra 7 155U',
gpu_count=None, gpu_model=None, longitude=-123.1924, latitude=49.2492, ram_tota
l_size=31.69060516357422, tracking_mode='machine', on_cloud='N', pue=1.0)
```

In [9]:
```
tracker.final_emissions_data.emissions
```

Out[9]:
```
9.44973804533264e-09
```

Let's now try a more complex task: question answering. Question answering is a task in Natural Language Processing (NLP) which, as the name suggests, aims at producing an answer to a question asked in natural language. For this test will use the framework created by Hugging Face (transformers).

In [10]:
```
tracker = EmissionsTracker(log_level='error')
tracker.start()  # Starting the tracker
try:
    nlp = pipeline('question-answering')  # transformer; for more info, check ht
    print(nlp({
    'question': 'What was on the steps?',
    'context': "He walked down the steps from the train station in a bit of a hu
finally:
     tracker.stop()  # Stopping the tracker
```

```
No model was supplied, defaulted to distilbert/distilbert-base-cased-distilled-sq
uad and revision 564e9b5 (https://huggingface.co/distilbert/distilbert-base-cased
-distilled-squad).
Using a pipeline without specifying a model name and revision in production is no
t recommended.
C:\Users\lcm57\miniconda3\envs\dsci430\lib\site-packages\transformers\pipelines\q
uestion_answering.py:391: FutureWarning: Passing a list of SQuAD examples to the
pipeline is deprecated and will be removed in v5. Inputs should be passed using t
he `question` and `context` keyword arguments instead.
  warnings.warn(
```

```
{'score': 0.6361867785453796, 'start': 258, 'end': 286, 'answer': 'a pair of old
worn-out shoes'}
```

In [11]:
```
tracker.final_emissions_data
```

Out[11]:
```
EmissionsData(timestamp='2024-11-29T21:42:46', project_name='codecarbon', run_i
d='676444bd-54b4-4453-83dd-58ee84ef78be', experiment_id='5b0fa12a-3dd7-45bb-976
6-cc326314d9f1', duration=0.6422978999908082, emissions=1.3387885757545032e-07,
emissions_rate=2.0843732725479288e-07, cpu_power=42.5, gpu_power=0.0, ram_power
=11.883976936340332, cpu_energy=7.581371943969215e-06, gpu_energy=0, ram_energy
=2.0508400172247036e-06, energy_consumed=9.63221196119392e-06, country_name='Ca
nada', country_iso_code='CAN', region='british columbia', cloud_provider='', cl
oud_region='', os='Windows-10-10.0.22631-SP0', python_version='3.10.14', codeca
rbon_version='2.8.0', cpu_count=14, cpu_model='Intel(R) Core(TM) Ultra 7 155U',
gpu_count=None, gpu_model=None, longitude=-123.1924, latitude=49.2492, ram_tota
l_size=31.69060516357422, tracking_mode='machine', on_cloud='N', pue=1.0)
```

In [12]:
```
tracker.final_emissions_data.emissions
```

Out[12]:
```
1.3387885757545032e-07
```

Question 5

Report here the CO2 emissions generated by the two tasks, in grams (`emissions` in the output):

- Additions: $9.44973804533264\times10^{-9}$ grams
- Question answering: $1.3387885757545032\times10^{-7}$ grams
- The results change in each run, but CO2 emissions generated by the second task is usually larger than the CO2 emissions generated by the first task.

Question 6

In a 2009 estimate, Google said each query on its site generates 0.2 g of carbon dioxide, much higher than the toy examples above. In 2023, it was estimated that there are 99000 search queries per second run on the site. Assuming these numbers are accurate, how many **kilograms of carbon dioxide** are generated in a year by Google searches?

- We assume we have 365 days in a year and ignore the leap years with 366 days.
- $(0.2\text{g/queries})\times(99000\text{ queries/s})*(3600\text{ s/hr})*(24\text{ hr/day})*(365\text{day/year})\div(1000\text{g/kg}) = 624412800$kg. Therefore, 624412800 kilograms of carbon dioxide are generated in a year by Google searches.

Question 7

The result of the previous question may not be very meaningful to a non-expert in environmental sciences. Thankfully, the U.S. Environmental Protection Agency (EPA) has released a tool, called Greenhouse Gas Equivalencies Calculator, to convert these numbers in more understandable equivalencies. Try inserting your

result from the previous question in the tool, and answer this question: **how many pounds of coal should be burned to produce the same amount of CO2 as a year of Google searches?**

$693,607,040$ pounds of coal should be burned to produce the same amount of CO2 as a year of Google searches.

Finally, let's evaluate the carbon footprint of one of the models used in this course. In the cells below, you can find the code used to train a random forest model on the NIJ recidivism data used in Assignment 3. Look at this code and add two EmissionTracker, one to track the emissions produced when using the random forest model without fine tuning, and another one to track the emissions produced when fine-tuning the random forest model using 20 combinations of parameters. Include training and scoring in your tracking (but not preprocessing). Comments in the code should help you find the right place to insert the trackers.

In [13]:
```
# Importing the training and test sets (you may need to adjust this code to matc

train_df = pd.read_csv("training_set.csv")
test_df = pd.read_csv("testing_set.csv")

# Separating features and targets

X_train, y_train = (
    train_df.drop(columns=["Recidivism_Within_3years"]),
    train_df["Recidivism_Within_3years"],
)
X_test, y_test = (
    test_df.drop(columns=["Recidivism_Within_3years"]),
    test_df["Recidivism_Within_3years"],
)
```

In [14]:
```
# Feature preprocessing

numeric_feats = [
    "Residence_PUMA",
    "Supervision_Risk_Score_First",
    "Avg_Days_per_DrugTest",
    "DrugTests_THC_Positive",
    "DrugTests_Cocaine_Positive",
    "DrugTests_Meth_Positive",
    "DrugTests_Other_Positive",
    "Percent_Days_Employed",
    "Jobs_Per_Year"
]  # apply scaling and imputation
categorical_feats = ["Gender",
                     "Race",
                     "Age_at_Release",
                     "Gang_Affiliated",
                     "Supervision_Level_First",
                     "Education_Level",
                     "Dependents",
                     "Prison_Offense",
                     "Prison_Years",
                     "Prior_Arrest_Episodes_Felony",
```

```
                      "Prior_Arrest_Episodes_Misd",
                      "Prior_Arrest_Episodes_Violent",
                      "Prior_Arrest_Episodes_Property",
                      "Prior_Arrest_Episodes_Drug",
                      "Prior_Arrest_Episodes_PPViolationCharges",
                      "Prior_Conviction_Episodes_Felony",
                      "Prior_Conviction_Episodes_Misd",
                      "Prior_Conviction_Episodes_Prop",
                      "Prior_Conviction_Episodes_Drug",
                      "Delinquency_Reports",
                      "Program_Attendances",
                      "Program_UnexcusedAbsences",
                      "Residence_Changes",
                      ]  # apply one-hot encoding
passthrough_feats = ["Prior_Arrest_Episodes_DVCharges",
                "Prior_Arrest_Episodes_GunCharges",
                "Prior_Conviction_Episodes_Viol",
                "Prior_Conviction_Episodes_PPViolationCharges",
                "Prior_Conviction_Episodes_DomesticViolenceCharges",
                "Prior_Conviction_Episodes_GunCharges",
                "Prior_Revocations_Parole",
                "Prior_Revocations_Probation",
                "Condition_MH_SA",
                "Condition_Cog_Ed",
                "Condition_Other",
                "Violations_ElectronicMonitoring",
                "Violations_Instruction",
                "Violations_FailToReport",
                "Violations_MoveWithoutPermission"
                ]  # Already binary
drop_feats = ["ID",
              "Recidivism_Arrest_Year1",
              "Recidivism_Arrest_Year2",
              "Recidivism_Arrest_Year3",
              "Training_Sample"]   # features not of interest
target = "Recidivism_Within_3years"

ct = make_column_transformer(
    (
        make_pipeline(SimpleImputer(), StandardScaler()),
        numeric_feats,
    ),  # scaling on numeric features
    (
        make_pipeline(SimpleImputer(strategy="most_frequent"),OneHotEncoder(hand
        categorical_feats,
    ),  # OHE on categorical features
    ("passthrough", passthrough_feats),  # no transformations on the binary feat
    ("drop", drop_feats),  # drop the drop features
)
```

In [15]:

```
# Fitting feature transformer and transforming training features

X_train_transformed = ct.fit_transform(X_train)
column_names = numeric_feats + list(
    ct.named_transformers_["pipeline-2"].get_feature_names_out(
        categorical_feats
    )
) + passthrough_feats
```

In [21]:
```
results = {}  # Dictionary to collect results
emissions = []  # List to collect emission results from the tracker
                # You can add a result to this list after the tracker has stoppe
                # emissions.append(tracker.final_emissions_data.emissions)

# We will use this function to fit and evaluate the models using cross-validatio
def mean_std_cross_val_scores(model, X_train, y_train, **kwargs):
    """
    Returns mean and std of cross validation
    """
    scores = cross_validate(model, X_train, y_train, **kwargs)

    mean_scores = pd.DataFrame(scores).mean()
    std_scores = pd.DataFrame(scores).std()
    out_col = []

    for i in range(len(mean_scores)):
        out_col.append((f"%0.3f (+/- %0.3f)" % (mean_scores[i], std_scores[i])))

    return pd.Series(data=out_col, index=mean_scores.index)
```

In [22]:
```
# Pipeline including the column transformer and the random forest classifier

pipe_forest = Pipeline([
    ('ct', ct),
    ('rf', RandomForestClassifier(class_weight="balanced", random_state=2))])

tracker1 = EmissionsTracker(log_level='error')
tracker1.start()  # Starting the tracker
try:
    # Calling mean_std_cross_val_scores to fit and evaluate the model, and add t

    results["random_forest"] = mean_std_cross_val_scores(
    pipe_forest, X_train, y_train, return_train_score=True
    )
finally:
    tracker1.stop()  # Stopping the tracker
    emissions.append(tracker1.final_emissions_data.emissions)
```

```
C:\Users\lcm57\AppData\Local\Temp\ipykernel_8368\1274509884.py:18: FutureWarning:
Series.__getitem__ treating keys as positions is deprecated. In a future version,
integer keys will always be treated as labels (consistent with DataFrame behavio
r). To access a value by position, use `ser.iloc[pos]`
  out_col.append((f"%0.3f (+/- %0.3f)" % (mean_scores[i], std_scores[i])))
```

In [23]:
```
# Now, we are going to tune the random forest model across 2 parameters: max_dep

param_grid_rf = {
    "rf__n_estimators": randint(low=10, high=100),
    "rf__max_depth": randint(low=2, high=20),
}

random_search_rf = RandomizedSearchCV(
    pipe_forest,
    param_grid_rf,
    n_iter=20,   # This parameter specifies the number of combinations to try (2
    verbose=1,
    n_jobs=-1,
    random_state=123,
```

```
    return_train_score=True,
)

tracker2 = EmissionsTracker(log_level='error')
tracker2.start()  # Starting the tracker
try:
    # Fine-tuning the random forest model
    random_search_rf.fit(X_train, y_train);
    best_rf_model = random_search_rf.best_estimator_

    # Final fit and evaluation of the random forest model using the best paramat
    results["random forest (tuned)"] = mean_std_cross_val_scores(
        best_rf_model, X_train, y_train, return_train_score=True
    )
finally:
    tracker2.stop()  # Stopping the tracker
    emissions.append(tracker2.final_emissions_data.emissions)
```

```
Fitting 5 folds for each of 20 candidates, totalling 100 fits
```

```
C:\Users\lcm57\AppData\Local\Temp\ipykernel_8368\1274509884.py:18: FutureWarning:
Series.__getitem__ treating keys as positions is deprecated. In a future version,
integer keys will always be treated as labels (consistent with DataFrame behavio
r). To access a value by position, use `ser.iloc[pos]`
  out_col.append((f"%0.3f (+/- %0.3f)" % (mean_scores[i], std_scores[i])))
```

In [24]:
```
# Run this cell to visualize some information on the models (including training
results_df = pd.DataFrame(results)
results_df.loc['emissions (g of CO2)'] = emissions
results_df
```

Out[24]:

	random_forest	random forest (tuned)
fit_time	7.143 (+/- 0.499)	4.511 (+/- 0.411)
score_time	0.316 (+/- 0.047)	0.241 (+/- 0.045)
test_score	0.723 (+/- 0.010)	0.725 (+/- 0.009)
train_score	1.000 (+/- 0.000)	0.966 (+/- 0.002)
emissions (g of CO2)	0.000009	0.000015

Question 8

- What was the improvement in accuracy (on the test set) after fine tuning?
- How many more grams of CO2 were produced for this improvement? You can answer in the form "Fine tuning produces X many times more CO2"

- The mean accuracy on the test set after the fine-tuning slightly increased from 0.723 to o.725. The improvement in accuracy is not very significant.
- The CO2 emission increased from 0.000009g to 0.000015g after the fine-tuning. Fine-tuning produces about 1.67 many times more CO2.

Final thoughts

1. If you have completed this assignment in a group, please write a detailed description of how you divided the work and how you helped each other completing it:

We worked on the assignment separately, each taking turns to answer all parts and modifying the responses down the line.

2. Have you used ChatGPT or a similar Large Language Model (LLM) to complete this homework? Please describe how you used the tool. We will never deduct points for using LLMs for completing homework assignments, but this helps us understand how you are using the tool and advise you in case we believe you are using it incorrectly.

- Jingyuan's response: I did not use ChatGPT for this homework.
- Nicholas' response: I did not use ChatGPT for this homework.

3. Have you struggled with some parts (or all) of this homework? Do you have pending questions you would like to ask? Write them down here!

- Jingyuan's response: I am still a bit confused about the content discussed in the video The principle of proportional ink.
- Nicholas' response: I do not have any pending questions for this module.

In [ ]: