Risk of cancer prediction for patients with multiple disorders

Scientists have created a model to predict the risk of suffering cancer for people who suffer multiple disorders. This model is expected to save lives.

The same scientist published the details of their research, how the model was built and a detailed description of the data (e.g., the health conditions investigated), the NHS board where the data was collected. The data was deidentified and was not released as it is confidential patient information, and any leak might break existing legislation.

The researchers balanced the benefits and potential risks of the model realease, and it was decided that overall, there is a clear benefit for the population for the model to be made public.

What they didn’t realise, is that the NHS board in question is home to a famous Member of Parliament (MP). This famous MP is a former Prime Minister, and it is of public knowledge he suffered from cancer. Also, it is straightforward for anyone to do an online search and find some other details for this individual (age etc).

Attribute Inference

We will use this example to demonstrate an *attribute inference* attack. In such an attack, an attacker has access to some information about a particular individual but not all, and attempts to use the model to fill in the gaps in their knowledge. In this particular example, some aspects of the MPs health are public knowledgey. We will use that information, and a trained model, to find out information particular to this individual that is not in the public domain, and should remain in the TRE.

Let's get hands on with this example.

The following code imports some standard libraries that we will need.

In [1]:
```
import random
from itertools import product
import numpy as np

np.random.seed(1234)
random.seed(12345)

from scipy.stats import poisson
import pandas as pd
from sklearn.svm import SVC
```

Create the original model

We are assuming that a model is trained within a TRE on real data. However, we do not have access to real data, so we will randomly generate some realistic looking data.

In particular, we will generate data for 200 people: 100 cancer patients, and 100 non-cancer patients. Our MP will be one of the patients in the cancer set.

For each patient, we generate six values that in reality would be extracted from their electronic health records:

1. `diabetes` -- whether or not the patient suffers from diabetes (1 = yes, 0 = no)
2. `asthma` -- whether or not the patient suffers from asthma (1 = yes, 0 = no)
3. `bmi_group` -- the BMI group in which the patient falls (1, 2, 3, or 4)
4. `blood_pressure` -- the blood pressure group in which the patient falls (0, 1, 2, 3, 4, or 5)
5. `smoker` -- whether or not the patient is a smoker (1 = yes, 0 = no)
6. `age` -- the patient's age

Each patient is also associated with a value to indicate whether they are in the cancer group (1) or non-cancer (0).

In [2]:
```python
#1 is cancer, 0 is no cancer, this is our label and what we want to predict.
cancer = [1]*99 + [0]*100

df = pd.DataFrame()

#diabetes 0 no, 1 yes
df['diabetes'] = [[1, 0][random.random()>0.7] for n in range(99)] +
                 [[1, 0][random.random()>0.2] for n in range(100)]

#asthma 0 no, 1 yes
df['asthma'] = [[1, 0][random.random()>0.7] for n in range(99)] +
               [[1, 0][random.random()>0.5] for n in range(100)]

#bmi group 1 under, 2 normal, 3 overweight, 4 obese
df['bmi_group'] = [random.choices([1, 2, 3, 4], weights = [0.5, 5, 7, 5], k = 1)[0
                   for n in range(99)] +
                  [random.choices([1, 2, 3, 4], weights = [1, 7, 4, 1], k = 1)[0]
                   for n in range(100)]

#blood pressure 0 is low, 1 is normal, 5 is extremly high
df['blood_pressure'] = [random.choices([0, 1, 2, 3, 4, 5],
                                       weights = [0.5, 1, 5, 6, 1, 0.5], k = 1)[0]
                        for n in range(99)] +
                       [random.choices([0, 1, 2, 3, 4, 5],
                                       weights = [0.5, 5, 5, 1, 1, 0.5], k = 1)[0]
                        for n in range(100)]

#smoker 0 is non smoker, 1 is smoker
df['smoker'] = [[1, 0][random.random()>0.8] for n in range(99)] +
               [[1, 0][random.random()>0.2] for n in range(100)]

#age
x = np.arange(20,90)
pmf = poisson.pmf(x, 72)
age = [random.choices(x, weights = pmf, k = 1)[0] for n in range(99)]
x = np.arange(20,90)
pmf = poisson.pmf(x, 55)
age2 = [random.choices(x, weights = pmf, k = 1)[0] for n in range(100)]
```

```
df['age'] = age + age2

#Add the data of your MP
cancer = cancer + [1]

#add new row to end of DataFrame
#the order of the list indicates in order diabetes, asthma,
#bmi_group, blood_pressure, smoker, age
df.loc[len(df.index)] = [1, 1, 3, 2, 1, 62]
```

This looks like the kind of data that might exist within a TRE. Here's the first few rows:

In [3]:
```
print(df.head())
```

```
   diabetes  asthma  bmi_group  blood_pressure  smoker  age
0         1       0          4               3       1   72
1         1       1          2               3       0   83
2         0       0          4               3       1   63
3         1       1          4               3       0   77
4         1       1          4               2       1   87
```

Our MP is the final row of the data, here are their values:

In [4]:
```
print(df.iloc[199,:])
```

```
diabetes           1
asthma             1
bmi_group          3
blood_pressure     2
smoker             1
age               62
Name: 199, dtype: int64
```

Model training

The researcher trained a particular machine learning model called a Support Vector Machine (SVM). This is a very popular model for tasks in which we want to assign things (in this case patients) to groups (in this case cancer v non-cancer). The attribute inference attack we will perform is not unique to SVMs, we just use them as a popular example.

Training the model is very straightforward -- just a couple of lines of code (the details are not important).

In [5]:
```
# Train a model
prng = np.random.RandomState(12)
svc = SVC(C=1, gamma=3, probability=True, random_state=prng)
svc.fit(df, cancer)
```

Out[5]:
```
SVC(C=1, gamma=3, probability=True,
    random_state=RandomState(MT19937) at 0x2C05FDD5240)
```

The trained model can be used to make predictions about new individuals. Given data for an individual, it will produce two scores (probabilities). The first is how likely they are to belong to the non-cancer group (higher = more likely) and the second how likely they are to belong to the cancer group. The scores are always positive, and sum to 1.

For example, if we have an individual who has diabetes, has asthma, has a bmi group of 1, blood pressure of 5. is a non-smoker and is 72 years old, we can use the model to predict whether or not they should belong in the cancer or non-cancer groups:

```
In [6]: test_example = pd.DataFrame(
            {
                'diabetes': 1,
                'asthma': 1,
                'bmi_group': 1,
                'blood_pressure': 5,
                'smoker': 1,
                'age': 72
            }, index=[1]
        )
        predictions = svc.predict_proba(test_example)
        print(f'non-cancer score = {predictions[0][0]:.2f}')
        print(f'cancer score = {predictions[0][1]:.2f}')
```

```
non-cancer score = 0.49
cancer score = 0.51
```

The attack

We now assume the role of the attacker. The attacker is allowed to make predictions as we have just done.

As we are interested in a famous individual, some information is available in the public domain. In particular, the attacker knows the following:

- The MP is a smoker
- The MP is aged 62
- The MP has diabetes
- The MP has asthma

The attacker does not know the MP's `bmi_group` or `blood_pressure` and it is those that they are trying to determine through the attack.

The attacker does know the possible values that these two variables can take -- `bmi_group` is 1, 2, 3, or 4 and `blood_pressure` is 0, 1, 2, 3, 4, or 5.

How does the attack work?

Recall that when we used the model to make predictions, the model provided two scores -- the cancer and non-cancer scores. The more extreme these scores become (e.g one is close to 1 and the other to 0 (recall that they have to add up to 1)), the more *confident* the model is in assigning that example. It is not uncommon for models to have higher confidence for examples that they were trained on than examples that they haven't seen before. It is this property that the attacker will make use of.

In particular, the attacker will query the model with the known values and all combinations of the unknown values (i.e. in total, they will make $4\times 6 = 24$ queries for the four bmi groups and 6 blood pressure values). For each query, the attacker will record the score for the cancer group. The attacker will assume that the higher this score (i.e. the more confident the model), the more likely that the values are correct. Note that we used the confidence in the cancer group (rather than non-cancer) because the attacker *knows* that the MP had cancer and would therefore have been in the cancer group.

The following code goes through all values and computes the model's predictions.

In [7]:
```
feature_vals = {
    'diabetes': [1],
    'asthma': [1],
    'bmi_group': [1, 2, 3, 4],
    'blood_pressure': [0, 1, 2, 3, 4, 5],
    'smoker': [1],
    'age': [62]
}

all_combinations = product(*feature_vals.values())
print(all_combinations)
g = {}
for _, combination in enumerate(all_combinations):
    # Turn this particular combination into a dictionary
    g[_] = {n: v for n, v in zip(feature_vals.keys(), combination)}
attack_inputs = pd.DataFrame(g).T

probs = svc.predict_proba(attack_inputs)

# Add the prob cancer to the dataframe
attack_values = attack_inputs.copy()
attack_values['confidence'] = probs[:, 1]
sorted_attack_values = attack_values.sort_values(by='confidence',
                                                 ascending=False)[['bmi_group',
                                                                   'blood_pressure
```

```
<itertools.product object at 0x000002C060E51240>
```

The attacker now has a handy table of all possible values of the unknown variables for the MP and the confidence that the model gives each one. Here are the five values with the highest confidence:

In [8]:
```
print(sorted_attack_values.head())
```

```
    bmi_group  blood_pressure  confidence
14          3               2    0.938385
8           2               2    0.577007
9           2               3    0.545445
15          3               3    0.542361
20          4               2    0.542322
```

The attacker can see that there is a combination that has a *much* higher confidence that the others -- when `bmi_group = 3` and `blood_pressure = 2`, the model places the example in the cancer group with a very high score (0.95), whereas the next highest score is 0.53. To the attacker, this is very strong evidence that the first example were the values that the model was trained on (i.e. are the correct values) and the others have not been seen by the model before.

The attacker can therefore confidently predict that these are the correct values. And, if they did so, they would be correct. This represents a breah from the TRE -- the MPs bmi group and blood pressure constitute personal information that shouldn't be allowed out of the TRE.

Mitigation

An important question for TREs is how can an attack like this be mitigated?

When our researcher trained the SVM, they configured it by setting a particular parameter (`gamma`) to the value 3. Tuning this parameter can lead to a model that is much less susceptible to attack. For example, let's re-train the model, but this time with `gamma = 0.1` and try the attack again:

In [9]:
```
svc = SVC(C=1, gamma=0.1, probability=True)
svc.fit(df, cancer)
probs = svc.predict_proba(attack_inputs)

# Add the prob cancer to the dataframe
attack_values = attack_inputs.copy()
attack_values['confidence'] = probs[:, 1]
sorted_attack_values = attack_values.sort_values(by='confidence',
                                                 ascending=False)[['bmi_group',
                                                                   'blood_pressure

print(sorted_attack_values.head(n=24))
```

```
    bmi_group  blood_pressure  confidence
15          3               3    0.863876
21          4               3    0.847483
9           2               3    0.835714
14          3               2    0.834142
20          4               2    0.812857
8           2               2    0.803217
22          4               4    0.798077
16          3               4    0.797924
3           1               3    0.767166
10          2               4    0.754672
2           1               2    0.727526
23          4               5    0.686306
4           1               4    0.684278
13          3               1    0.683948
19          4               1    0.676913
17          3               5    0.654893
7           2               1    0.638978
11          2               5    0.593411
1           1               1    0.559613
5           1               5    0.534109
18          4               0    0.500000
12          3               0    0.473850
6           2               0    0.421279
0           1               0    0.378039
```

The attacker now sees a very different picture - there is no single combination that for which the model is much more confident than others. The attacker can therefore not make any clear prediction about these values -- the model is much safer.

Conclusions

With this example, we have demonstrated how an attacker who has some information about an individual in the model's training set and is allowed to query the model can potentially learn information about that individual. This is known as an *attribute inference* attack, and it makes use of the fact that under some configurations, models can be more confident on examples they've seen during training than on examples that they haven't.

The susceptibility of a model to attack depends, to a significant degree, on their configuration. In the case of an SVM, changing the `gamma` parameter can control how safe the model is. Although the details of an SVM's `gamma` parameter are not important, it's important to see how small changes in a model's configuration can have dramatic changes in their vulnerability.