In [ ]:

```
import os
import sys
import copy
import numpy as np
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.metrics import roc_auc_score


import warnings
warnings.filterwarnings("ignore")
```

In [ ]:

```
data = pd.read_csv("./../../data/fraud_detection_bank_dataset.csv")
col_names = [f"col_{i}" for i in range(111)]
col_names_naive = [x for x in col_names if len(data[x].value_counts().keys()) < 10]
target = "targets"
```

In [ ]:

```
def accuracy(y, y_hat):

    return (y == y_hat).sum() / len(y)
```

Naive Bayes Model

Bayes theorm

- We considered spam filtering methods based on naïve Bayes:

$$p(y_i = \text{"spam"} \mid x_i) = \frac{p(x_i \mid y_i = \text{"spam"})\, p(y_i = \text{"spam"})}{p(x_i)}$$

- Makes conditional independence assumption to make learning practical:

$$\underbrace{p(\text{hello}=1, \text{vicodin}=0, 340=1 \mid \text{spam})}_{\text{HARD}} \approx p(\underbrace{\text{hello}=1 \mid \text{spam}}_{\text{easy}})\, p(\underbrace{\text{vicodin}=0 \mid \text{spam}}_{\text{easy}})\, p(\underbrace{340=1 \mid \text{spam}}_{\text{easy}})$$

- Predict "spam" if $p(y_i = \text{"spam"} \mid x_i) > p(y_i = \text{"not spam"} \mid x_i)$.
  - We don't need $p(x_i)$ to test this.

Tips

The implement of Bayes is pretty simple, still there are three points pretty important

- Laplace smooth: if variable has K possible values, then $p(x=a|y=0) = \frac{p(x=a,y=0)+\beta}{p(y=0)+K*\beta}$
- What if variable is continious? Does `laplace smooth` help with continuous variables. No, continous variable could have infinite values. **Guassian Bayes or Discretization continous variables**
- Avoiding Underflow: Use log function transfer `multiply` to `add`

In [ ]:

```
train_data, test_data = train_test_split(data, train_size=0.8, random_state=123)
X_train, y_train = train_data[col_names_naive].values, train_data[target].values
X_test, y_test = test_data[col_names_naive].values, test_data[target].values
```

In [ ]:

```
from collections import Counter

class Naive_Bayes():

    @staticmethod
    def transfer(X, y):
        set_label = sorted(set(y))
        data_dict = {l:X[y == l] for l in set_label}
```

```python
        return data_dict

    def __init__(self, laplace_dict):

        self.laplace_dict = laplace_dict

    def fit(self, X, y):

        data_dict = self.transfer(X, y)
        model = {x:{} for x in data_dict.keys()}
        total_sum, K = len(y), len(data_dict.keys())

        for label, X in data_dict.items():

            sample_sum = np.shape(X)[0]

            for idx, feat in enumerate(X.T):
                counter = Counter(feat)
                counter = dict(counter)

                laplace_dict = self.laplace_dict[idx]
                for k in laplace_dict:
                    if k not in counter.keys():
                        counter[k] = 1 / (len(laplace_dict) + sample_sum)
                    else:
                        counter[k] = (counter[k] + 1) / (len(laplace_dict) + sample_sum)

                model[label][idx] = counter

            model[label]["cnt"] = (sample_sum + 1) / (total_sum + K)

        self.model = model

    def predict(self, X_obs):

        likehoods = np.array([])
        labels = np.array([])
        for label in self.model.keys():
            xx = np.array(list(map(lambda idx: self.model[label][idx[0]][idx[1]], enumerate(X_obs))))
            xx = np.append(xx, self.model[label]["cnt"])
            likehood = np.log(xx).sum()
            likehoods = np.append(likehoods, likehood)
            labels = np.append(labels, label)

        return labels[np.argmax(likehoods)]
```

In [ ]:

```
laplace_dict = {}
for idx, feat in enumerate(data[col_names_naive].values.T):
    laplace_dict[idx] = np.unique(feat)

nb = Naive_Bayes(laplace_dict)
nb.fit(X_train, y_train)
```

In [ ]:

```
y_train_hat = np.apply_along_axis(nb.predict, axis=1, arr=X_train)
y_test_hat = np.apply_along_axis(nb.predict, axis=1, arr=X_test)

print("Train accuracy: ", accuracy(y_train, y_train_hat))
print("Test accuracy: ", accuracy(y_test, y_test_hat))
```

```
Train accuracy:  0.8521436423598387
Test accuracy:  0.8600390815828041
```

In [ ]:

```
import matplotlib.pyplot as plt
from sklearn.metrics import confusion_matrix, precision_score, recall_score, f1_score, accuracy_score

print(confusion_matrix(y_test, y_test_hat))
print(precision_score(y_test, y_test_hat))
print(recall_score(y_test, y_test_hat))
print(f1_score(y_test, y_test_hat))
print(accuracy_score(y_test, y_test_hat))
```

```
[[2935  112]
 [ 461  586]]
0.839541547277937
0.559694364851958
0.6716332378223496
0.8600390815828041
```

Guassian Naive Bayes Model

- Guassian Distribution:

$$P(x_i \mid y) = \frac{1}{\sqrt{2\pi\sigma_y^2}} \exp\left(-\frac{(x_i - \mu_y)^2}{2\sigma_y^2}\right)$$

In [ ]:

```python
from collections import Counter

class Guassian_Naive_Bayes():

    @staticmethod
    def transfer(X, y):
        set_label = sorted(set(y))
        data_dict = {l:X[y == l] for l in set_label}
        return data_dict

    @staticmethod
    def get_mean(X):
        return X.mean(axis=0)

    @staticmethod
    def get_var(X):
        return X.var(axis=0) + 1e-6

    def fit(self, X, y):

        data_dict = self.transfer(X, y)
        model = {x:{"mean":self.get_mean(data_dict[x]), "var":self.get_var(data_dict[x])} for x in data_dict.keys()}

        total_sum = len(y)
        for i in data_dict.keys():
            model[i]["cnt"] = np.log(data_dict[i].shape[0] / total_sum)

        self.model = model

    def cal_guassian(self, X_test, y=0):

        mean = self.model[y]["mean"]
        var = self.model[y]["var"]
        return (-((X_test - mean) ** 2) / (2 * var) - np.log(np.sqrt(2 * np.math.pi * var))).sum(axis=1)

    def predict(self, X_test):

        likehoods = []
        labels = np.array([])
        for idx, label in enumerate(self.model.keys()):
            likehood = self.cal_guassian(X_test, y=label) + self.model[label]["cnt"]
            likehoods.append(likehood)
            labels = np.append(labels, label)

        return labels[np.argmax(np.array(likehoods), axis=0)]
```

In [ ]:

```
guassian_nb = Guassian_Naive_Bayes()
guassian_nb.fit(X_train, y_train)
```

In [ ]:

```
y_train_hat = guassian_nb.predict(X_train)
y_test_hat = guassian_nb.predict(X_test)

print("Train accuracy: ", accuracy(y_train, y_train_hat))
print("Test accuracy: ", accuracy(y_test, y_test_hat))
```

```
Train accuracy:  0.8231342372053255
Test accuracy:  0.8270639960918417
```

In [ ]:

```
import matplotlib.pyplot as plt
from sklearn.metrics import confusion_matrix, precision_score, recall_score, f1_score, accuracy_score

print(confusion_matrix(y_test, y_test_hat))
print(precision_score(y_test, y_test_hat))
print(recall_score(y_test, y_test_hat))
print(f1_score(y_test, y_test_hat))
print(accuracy_score(y_test, y_test_hat))
```

```
[[2743  304]
 [ 404  643]]
0.6789862724392819
0.6141356255969437
0.6449348044132397
0.8270639960918417
```

In [ ]: