Starting point

We would like to compare 2 different approaches to classify objects based on predictors.

Chosen approaches:

- kNN classifier
- Naive Bayes classifier

The comparison will be based on Accuracy, Recall, Precision and F1-Value of the models. The goal is to find the best model to predict "income" based on meaningful categorical features in the *census* data.

In [ ]:
```
import pandas as pd
import numpy as np

from matplotlib import pyplot as plt
from sklearn.metrics import (
    confusion_matrix,
    classification_report,
    RocCurveDisplay,
    roc_curve,
    auc
)

from sklearn.model_selection import (
    GridSearchCV,
    train_test_split
)

from sklearn.naive_bayes import GaussianNB, CategoricalNB
```

Data preparation

For both models the census.csv data will be used.

In [179…
```
dat = pd.read_csv("census.csv")
dat.head()
```

Out[179…

	age	workclass	fnlwgt	education	education.num	marital.status	occupation
0	39	State-gov	77516	Bachelors	13	Never-married	Adm-clerical
1	50	Self-emp-not-inc	83311	Bachelors	13	Married-civ-spouse	Exec-managerial
2	38	Private	215646	HS-grad	9	Divorced	Handlers-cleaners
3	53	Private	234721	11th	7	Married-civ-spouse	Handlers-cleaners
4	28	Private	338409	Bachelors	13	Married-civ-spouse	Prof-specialty

The target is **Income**

In [180…

```
dat[["income"]].value_counts()
```

Out[180…

```
income
<=50K     24720
>50K       7841
Name: count, dtype: int64
```

Our target variable doesnt show the balanced distribution within the groups. So we should care about creating stratified data samples during spliting on train-test samples.

We need to choose a few meaningful categorical features as predictors.

In [181…

```
def analyze_categorical_columns(df):
    categorical_cols = df.select_dtypes(include=['object', 'category']).c

    if len(categorical_cols) == 0:
        print("No categorical columns in DataFrame.")
        return

    for col in categorical_cols:
        print(f"Number of unique values: {df[col].nunique()}")
        print(f"{df[col].value_counts()}")
        print("-" * 30)

analyze_categorical_columns(dat)
```

```
Number of unique values: 9
workclass
Private             22696
Self-emp-not-inc     2541
Local-gov            2093
?                    1836
State-gov            1298
Self-emp-inc         1116
Federal-gov           960
Without-pay            14
Never-worked            7
Name: count, dtype: int64
------------------------------
Number of unique values: 16
education
HS-grad         10501
Some-college     7291
Bachelors        5355
Masters          1723
Assoc-voc        1382
11th             1175
Assoc-acdm       1067
10th              933
7th-8th           646
Prof-school       576
9th               514
12th              433
Doctorate         413
5th-6th           333
1st-4th           168
Preschool          51
Name: count, dtype: int64
------------------------------
Number of unique values: 7
marital.status
Married-civ-spouse       14976
Never-married            10683
Divorced                  4443
Separated                 1025
Widowed                    993
Married-spouse-absent      418
Married-AF-spouse           23
Name: count, dtype: int64
------------------------------
Number of unique values: 15
occupation
Prof-specialty       4140
Craft-repair         4099
Exec-managerial      4066
Adm-clerical         3770
Sales                3650
Other-service        3295
Machine-op-inspct    2002
?                    1843
Transport-moving     1597
Handlers-cleaners    1370
Farming-fishing       994
Tech-support          928
Protective-serv       649
Priv-house-serv       149
```

```
Armed-Forces           9
Name: count, dtype: int64
------------------------------
Number of unique values: 6
relationship
Husband           13193
Not-in-family      8305
Own-child          5068
Unmarried          3446
Wife               1568
Other-relative      981
Name: count, dtype: int64
------------------------------
Number of unique values: 5
race
White                 27816
Black                  3124
Asian-Pac-Islander     1039
Amer-Indian-Eskimo      311
Other                   271
Name: count, dtype: int64
------------------------------
Number of unique values: 2
sex
Male      21790
Female    10771
Name: count, dtype: int64
------------------------------
Number of unique values: 42
native.country
United-States                 29170
Mexico                          643
?                               583
Philippines                     198
Germany                         137
Canada                          121
Puerto-Rico                     114
El-Salvador                     106
India                           100
Cuba                             95
England                          90
Jamaica                          81
South                            80
China                            75
Italy                            73
Dominican-Republic               70
Vietnam                          67
Guatemala                        64
Japan                            62
Poland                           60
Columbia                         59
Taiwan                           51
Haiti                            44
Iran                             43
Portugal                         37
Nicaragua                        34
Peru                             31
France                           29
Greece                           29
Ecuador                          28
```

```
Ireland                          24
Hong                             20
Cambodia                         19
Trinadad&Tobago                  19
Laos                             18
Thailand                         18
Yugoslavia                       16
Outlying-US(Guam-USVI-etc)       14
Honduras                         13
Hungary                          13
Scotland                         12
Holand-Netherlands                1
Name: count, dtype: int64
------------------------------
Number of unique values: 2
income
<=50K    24720
>50K      7841
Name: count, dtype: int64
------------------------------
```

Based on information above the set of these categorical feachures were selected:

- workclass (9 classes)
- education (16 classes)
- marital.status (7 classes)
- occupation (15 classes)
- sex (2 classes)

In [182…

```
whole_dataset = dat[["workclass","education","marital.status", "occupatio
whole_dataset.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 32561 entries, 0 to 32560
Data columns (total 6 columns):
 #   Column          Non-Null Count  Dtype 
---  ------          --------------  ----- 
 0   workclass       32561 non-null  object
 1   education       32561 non-null  object
 2   marital.status  32561 non-null  object
 3   occupation      32561 non-null  object
 4   sex             32561 non-null  object
 5   income          32561 non-null  object
dtypes: object(6)
memory usage: 1.5+ MB
```

In [183…

```
whole_dataset.isna().sum()
```

Out[183…

```
workclass         0
education         0
marital.status    0
occupation        0
sex               0
income            0
dtype: int64
```

There are no any missing values, good.

Onehot encoding

In [184…

```
# Use pd.get_dummies() to one-hot encode the categorical columns
ds_encoded = pd.get_dummies(whole_dataset, columns=["workclass","educatio
ds_encoded.iloc[:,:5].head(1)
```

Out[184…

	income	workclass_Federal-gov	workclass_Local-gov	workclass_Never-worked	workclass_Pri
0	<=50K	False	False	False	F

Here we can see just first few columns.

In [185…

```
print(f"Total number of columns: {len(ds_encoded.columns)}")
print(ds_encoded.columns)
```

```
Total number of columns: 45
Index(['income', 'workclass_Federal-gov', 'workclass_Local-gov',
       'workclass_Never-worked', 'workclass_Private', 'workclass_Self-emp-
inc',
       'workclass_Self-emp-not-inc', 'workclass_State-gov',
       'workclass_Without-pay', 'education_11th', 'education_12th',
       'education_1st-4th', 'education_5th-6th', 'education_7th-8th',
       'education_9th', 'education_Assoc-acdm', 'education_Assoc-voc',
       'education_Bachelors', 'education_Doctorate', 'education_HS-grad',
       'education_Masters', 'education_Preschool', 'education_Prof-schoo
l',
       'education_Some-college', 'marital.status_Married-AF-spouse',
       'marital.status_Married-civ-spouse',
       'marital.status_Married-spouse-absent', 'marital.status_Never-marri
ed',
       'marital.status_Separated', 'marital.status_Widowed',
       'occupation_Adm-clerical', 'occupation_Armed-Forces',
       'occupation_Craft-repair', 'occupation_Exec-managerial',
       'occupation_Farming-fishing', 'occupation_Handlers-cleaners',
       'occupation_Machine-op-inspct', 'occupation_Other-service',
       'occupation_Priv-house-serv', 'occupation_Prof-specialty',
       'occupation_Protective-serv', 'occupation_Sales',
       'occupation_Tech-support', 'occupation_Transport-moving', 'sex_Mal
e'],
      dtype='object')
```

The total number of columns is 45.

Now the data is ready to be splitted on test-train samples.

Train-test split

In [186…

```
X = ds_encoded.iloc[:,1:]
y = ds_encoded.iloc[:,0]

X_train, X_test, y_train, y_test = train_test_split(
```

```
    X, y, test_size=1/3, random_state=471
)
```

In [187…

```
pd.concat({"train": y_train.value_counts(), "test": y_test.value_counts()
# y_test.value_counts().merge(y_train.value_counts(), how = "inner")
```

Out[187…

	train	test
income		
<=50K	16518	8202
>50K	5189	2652

Looks quite stratified.

k-NN-model

As in previous homework we can use grid search to find good k and determine the best kNN model. We already know, that this method perform pretty well because it uses cross validation (in our case with 10 folds).

In [188…

```
gs = GridSearchCV(estimator = knn,
        param_grid = {'n_neighbors' : list(range(1,10))},
        scoring = 'accuracy',
        cv = 10,
        refit = True)
best_knn_model = gs.fit(X_train, y_train).best_estimator_
print('Best k : %d' % best_knn_model.get_params()['n_neighbors'])
```

```
Best k : 9
```

Grid search tells us that the best k is 9.

We can calculate some performance metrics to check how good the model actually.

In [189…

```
pred = best_knn_model.predict(X_test)
print(confusion_matrix(y_test, pred))
```

```
[[7442  760]
 [1284 1368]]
```

There are 7442 and 1368 correctly detected "<=50K" and ">50K" classes respectivvely.

But the number of false predictions is ratherly big (760 and 1284). Seems like this kNN model strugle to determine rich class (>50K), because the second row in the confusion matrix represented by numbers close to 1300, whereas the total number of observations with ">50K" in test data is around 2600. That means just half of the rich class was detected.

We can explore classification_report to get more informations about model.

In [190…

```
print(classification_report(y_test, pred))
```

```
              precision    recall  f1-score   support

       <=50K       0.85      0.91      0.88      8202
        >50K       0.64      0.52      0.57      2652

    accuracy                           0.81     10854
   macro avg       0.75      0.71      0.73     10854
weighted avg       0.80      0.81      0.80     10854
```

As mentioned before the class marked as ">50K" is problematic for the model, thus recall is just 0.52. Also the precision for this category is rather small showing 0.64.

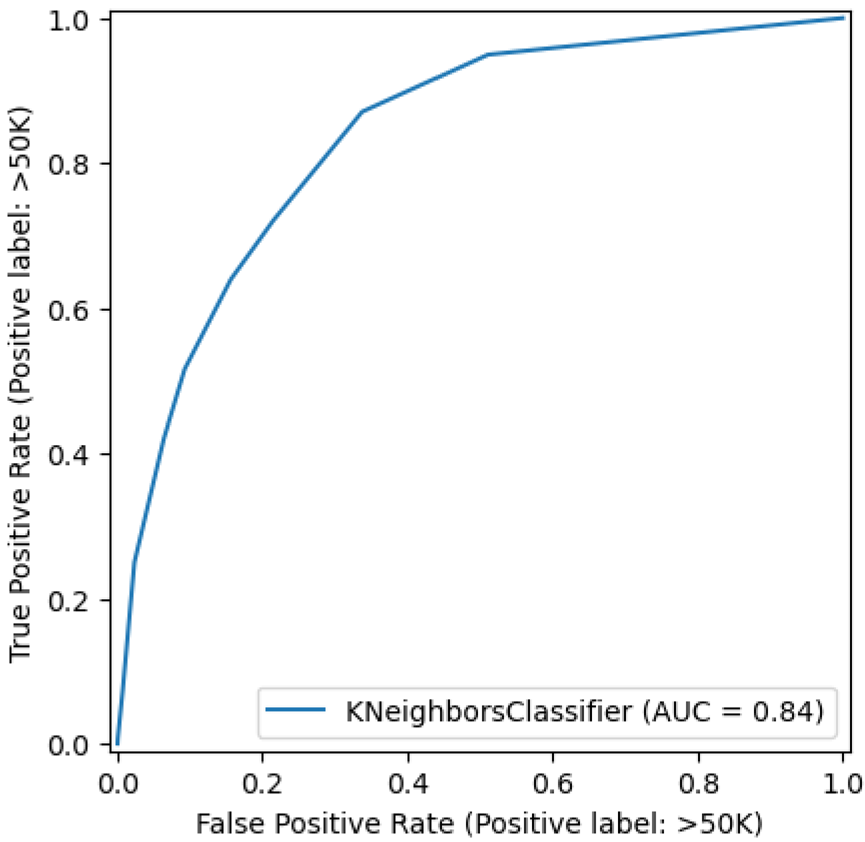
The class with larger number of observations shows better performance with respect to precision (0.85) and recall (0.91), and thus the f1-score (0.88) as well.

The accuracy is showing 0.81, but such a big number reaches because of nice performance of larger class with about 80% of observations (<=50K), while smaller class (>50K) shows bad performance.

Combined performance metrics (macro avg & weighted avg) showcase around 0.73 and 0.80 respectively, which is not so good.

In [191…

```
RocCurveDisplay.from_estimator(best_knn_model, X_test, y_test)
plt.show()
```



NaiveBayes model

Naive Bayes classifier assumes that the effect of a particular feature in a class is independent of other features

There are 2 types of Naive Bayes functions that exists in sklearn.

GaussianNB:

- Assumes continuous features follow a Gaussian (normal) distribution.
- Used for classification with continuous features.

CategoricalNB:

- Assumes categorical features follow a categorical distribution.
- Used for classification with categorical features.

In our case, we only have categorical features, so it's an easy decision to choose the latter method. However, it's still interesting to see how the former method might perform in this kind of task, because there was no *CategoricalNB* in previous versions of Scikit-learn. Additionally, I would like to test it because **ChatGPT** suggested using Gaussian NB, which seems strange when we only have categorial variables.

Gaussian Naive Bayes

In [192…

```
modelNB1 = GaussianNB()

modelNB1.fit(X_train, y_train)

predNB1 = modelNB1.predict(X_test)
```

In [193…

```
print(confusion_matrix(y_test, predNB1))
```

```
[[1966 6236]
 [  91 2561]]
```

Looks very bad, as expected because of use wrong Naive Bayes method (Gaussian instead of Categorical)

In [194…

```
print(classification_report(y_test, predNB1))
```

```
              precision    recall  f1-score   support

       <=50K       0.96      0.24      0.38      8202
        >50K       0.29      0.97      0.45      2652

    accuracy                           0.42     10854
   macro avg       0.62      0.60      0.42     10854
weighted avg       0.79      0.42      0.40     10854
```

Classification report also proof that this model performs bad with *weghted avg f1-score* around 0.4.

Categorical Naive Bayes

In [195…

```
modelNB2 = CategoricalNB()
```

```
modelNB2.fit(X_train, y_train)

predNB2 = modelNB2.predict(X_test)
```

In [196…

```
print(confusion_matrix(y_test, predNB2))
```

```
[[6979 1223]
 [ 931 1721]]
```

This matrix looks much better compared to modelNB1.

But we would like to compare performance metrics with knn_midel.

In [197…

```
print(classification_report(y_test, predNB2))
```

```
              precision    recall  f1-score   support

        <=50K       0.88      0.85      0.87      8202
         >50K       0.58      0.65      0.62      2652

     accuracy                           0.80     10854
    macro avg       0.73      0.75      0.74     10854
 weighted avg       0.81      0.80      0.80     10854
```

Usage of Categorical Naive Bayes brings us to model, that show simmilar patterns as knn_model, but slightly different.

As well as knn_model this NB_model shows better performance regarding larger class (<=50K) with precision at 0.88 and recall at 0.85. In contrast to knn_model the later metric is lower, but not significantly (0.91 -> 0.85 still high compared to other numbers).

As for smalle class (>50K) this model performs in another way, because the precision (0.64 -> 0.58) and recall (0.52 -> 0.65) swapped their positions in relative numbers. The f1-score for the larger group sligthly decreased (0.88 -> 0.87), while this for smaller group increased (0.57 -> 0.62).

To understand which model is better knn_model or NB_model, we should define what is more important for us: precision or recall.

In my personal oppinion the recall of samller class should be more important in this case, **therefore NB_model is slightly better for this task**, while still the result of this model is 80% in terms of accuracy.

Threshhold tuning

In [198…

```
X_train_part, X_val, y_train_part, y_val = train_test_split(
    X_train, y_train, test_size=1/3, random_state=1)
```

We splitted the train dataset, that we used befor, on 2 new datasets: train_part and val.

Actual parts of the whole dataset:

- 1/3 test data
- 4/9 train_part data
- 2/9 val data

In [199…

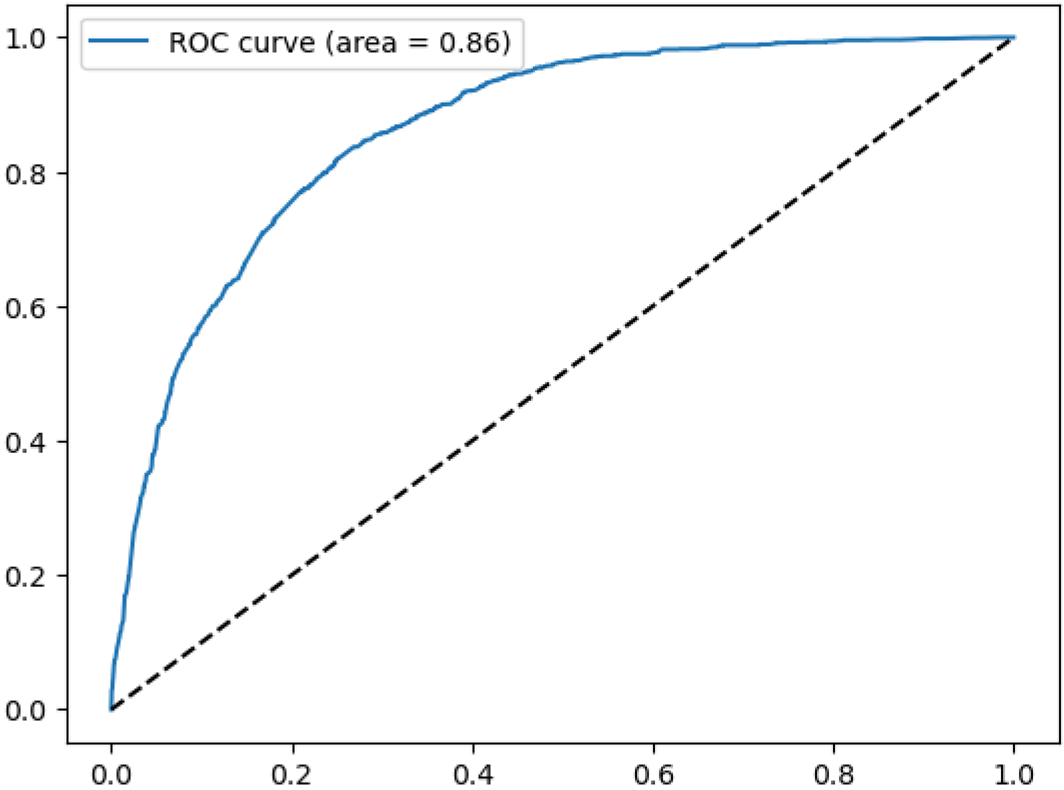
```
modelNB3 = CategoricalNB()

modelNB3.fit(X_train_part, y_train_part)

y_prob_val = modelNB3.predict_proba(X_val)[:, 1]
fpr, tpr, thresholds = roc_curve(y_val, y_prob_val, pos_label='>50K')

roc_auc = auc(fpr, tpr)

plt.plot(fpr, tpr, label=f'ROC curve (area = {roc_auc:.2f})')
plt.plot([0, 1], [0, 1], 'k--')
plt.legend()
plt.show()
```



Another method to plot Receiver Operating Characteristic (ROC) curve using manually calculated FalsePositiveRate (fpr) and TruePositiveRate (tpr).

To define the best threshold we can use Youden's J Statistic, which is simply difference between *tpr* and *fpr*. That means that we try to find threshold that helps to get the model with better recall of the chousen class (in this case ">50K").

In [200…

```
youden_j = tpr - fpr

best_threshold = thresholds[np.argmax(youden_j)]

y_prob_test = modelNB3.predict_proba(X_test)[:, 1]
pred_threshold = (y_prob_test >= best_threshold).astype(int)
```

```
print("threshold:", best_threshold)
prediction = ['>50K' if p == 1 else '<=50K' for p in pred_threshold]
```

```
threshold: 0.2461130910160678
```

New threshhold is 0.246, which is significantly lower then basic one (0.5). We can assume the model will pick ">50K" more often, because the threshold becomes lower.

In [201…

```
print(confusion_matrix(y_test, prediction))
```

```
[[6057 2145]
 [ 477 2175]]
```

As mentioned before the model increased predictions of ">50K" in total, bringing us to more false positive predictions (1223 -> 2145), while the number of true positive predictions increases slightly (1721 -> 2175).

TP and FP are almost equal, which means that the model produce 50% incorrect predictions for small class.

In [202…

```
print(classification_report(y_test, prediction))
```

```
              precision    recall  f1-score   support

       <=50K       0.93      0.74      0.82      8202
        >50K       0.50      0.82      0.62      2652

    accuracy                           0.76     10854
   macro avg       0.72      0.78      0.72     10854
weighted avg       0.82      0.76      0.77     10854
```

As expected, the recall for small class becomes higher, but we loose in precision: 50% is low.

Interesting thing that the f1-score for ">50K" doesn't change (0.62 -> 0.62).

Also, accuracy is slightly decreased, compared to model with initial threshold. As well as overall performance metrics becomes lower.

Conclusion

Overall, all the models that we tested for this task can be reasonably used, except the model based on GaussianNB. Some of them have slightly better recall of small class, while others perform better in terms of precision for that. Also, the kNN model demonstrates that it requires much more computational power than NaiveBayes in this case, because we have a lot of dimentions (>40) after onehote encoding.

I would say that the final model should be ClassificationNaiveBayes based on initial (0.5) threshold, because it shows balanced performance.