CPSC 330 - Applied Machine Learning

Homework 3: Preprocessing

Associated lectures: Lectures 4, 5, 6

Due date: Saturday, May 28, 2022 at 18:00

Table of Contents

Imports

In [28]:
```
import matplotlib.pyplot as plt
import numpy as np
import pandas as pd
from sklearn.compose import ColumnTransformer, make_column_transformer
from sklearn.dummy import DummyClassifier
from sklearn.impute import SimpleImputer
from sklearn.model_selection import cross_val_score, cross_validate, train_test_split
from sklearn.neighbors import KNeighborsClassifier
from sklearn.pipeline import Pipeline, make_pipeline
from sklearn.preprocessing import OneHotEncoder, StandardScaler, OrdinalEncoder
from sklearn.svm import SVC
from sklearn.tree import DecisionTreeClassifier
from pandas_profiling import ProfileReport
```

Instructions

---

rubric={points:6}

Follow the homework submission instructions.

You may work with a partner on this homework and submit your assignment as a group. Below are some instructions on working as a group.

- The maximum group size is 2.
- Use group work as an opportunity to collaborate and learn new things from each other.
- Be respectful to each other and make sure you understand all the concepts in the assignment well.

- It's your responsibility to make sure that the assignment is submitted by one of the group members before the deadline.
- You can find the instructions on how to do group submission on Gradescope here.

Note: The assignments will get gradually more open-ended as we progress through the course. In many cases, there won't be a single correct solution. Sometimes you will have to make your own choices and your own decisions (for example, on what parameter values to use when they are not explicitly provided in the instructions). Use your own judgment in such cases and justify your choices, if necessary.

Introduction

A crucial step when using machine learning algorithms on real-world datasets is preprocessing. This homework will give you some practice of data preprocessing and building a supervised machine learning pipeline on a real-world dataset.

Exercise 1: Introducing the dataset

In this lab, you will be working on the adult census dataset. Download the CSV and save it as `adult.csv` locally in this homework folder.

This is a classification dataset and the classification task is to predict whether income exceeds 50K per year or not based on the census data. You can find more information on the dataset and features here.

The starter code below loads the data CSV (assuming that it is saved as `adult.csv` in this folder).

Note that many popular datasets have sex as a feature where the possible values are male and female. This representation reflects how the data were collected and is not meant to imply that, for example, gender is binary.

In [2]:
```
census_df = pd.read_csv("adult.csv")
census_df.shape
```

Out[2]:
```
(32561, 15)
```

1.1 Data splitting

rubric={points:4}

In order to avoid violation of the golden rule, the first step before we do anything is splitting the data.

Your tasks:

1. Split the data into `train_df` (50%) and `test_df` (50%) with `random_state = 10`. Keep the target column (`income`) in the splits so that we can use it in the exploratory data analysis.

Usually having more data for training is a good idea. But here I'm using 50%/50% split because this is kind of a big dataset for a modest laptop. A smaller training data means it won't take too long to train the model on your laptop. A side advantage of this would be that with a bigger test split, we'll have a more reliable estimate of the deployment performance!

In [3]:
```
train_df, test_df = train_test_split(census_df, test_size=0.5, random_state=10)
```

Exercise 2: Exploratory data analysis (EDA)

Let's examine our `train_df`.

In [4]:
```
train_df.sort_index()
```

Out[4]:

	age	workclass	fnlwgt	education	education.num	marital.status	occupation	relationship	race	s
1	82	Private	132870	HS-grad	9	Widowed	Exec-managerial	Not-in-family	White	Fema
2	66	?	186061	Some-college	10	Widowed	?	Unmarried	Black	Fema
3	54	Private	140359	7th-8th	4	Divorced	Machine-op-inspct	Unmarried	White	Fema
4	41	Private	264663	Some-college	10	Separated	Prof-specialty	Own-child	White	Fema
6	38	Private	150601	10th	6	Separated	Adm-clerical	Unmarried	White	Ma
...	...	...	...	...	...	...	...	...	...	
32544	72	?	129912	HS-grad	9	Married-civ-spouse	?	Husband	White	Ma
32545	45	Local-gov	119199	Assoc-acdm	12	Divorced	Prof-specialty	Unmarried	White	Fema
32547	39	Local-gov	111499	Assoc-acdm	12	Married-civ-spouse	Adm-clerical	Wife	White	Fema
32551	43	Self-emp-not-inc	27242	Some-college	10	Married-civ-spouse	Craft-repair	Husband	White	Ma
32554	32	Private	116138	Masters	14	Never-married	Tech-support	Not-in-family	Asian-Pac-Islander	Ma

16280 rows × 15 columns

We see some missing values represented with a "?". Probably these were the questions not answered by some people during the census. Usually `.describe()` or `.info()` methods would give you information

on missing values. But here, they won't pick "?" as missing values as they are encoded as strings instead of an actual NaN in Python. So let's replace them with `np.nan` before we carry out EDA. If you do not do it, you'll encounter an error later on when you try to pass this data to a classifier.

In [5]:
```
train_df_nan = train_df.replace("?", np.nan)
test_df_nan = test_df.replace("?", np.nan)
train_df_nan.shape
```

Out[5]:
```
(16280, 15)
```

In [6]:
```
train_df_nan.sort_index()
```

Out[6]:

	age	workclass	fnlwgt	education	education.num	marital.status	occupation	relationship	race	s
1	82	Private	132870	HS-grad	9	Widowed	Exec-managerial	Not-in-family	White	Fema
2	66	NaN	186061	Some-college	10	Widowed	NaN	Unmarried	Black	Fema
3	54	Private	140359	7th-8th	4	Divorced	Machine-op-inspct	Unmarried	White	Fema
4	41	Private	264663	Some-college	10	Separated	Prof-specialty	Own-child	White	Fema
6	38	Private	150601	10th	6	Separated	Adm-clerical	Unmarried	White	Ma
...	...	...	...	...	...	...	...	...	...	
32544	72	NaN	129912	HS-grad	9	Married-civ-spouse	NaN	Husband	White	Ma
32545	45	Local-gov	119199	Assoc-acdm	12	Divorced	Prof-specialty	Unmarried	White	Fema
32547	39	Local-gov	111499	Assoc-acdm	12	Married-civ-spouse	Adm-clerical	Wife	White	Fema
32551	43	Self-emp-not-inc	27242	Some-college	10	Married-civ-spouse	Craft-repair	Husband	White	Ma
32554	32	Private	116138	Masters	14	Never-married	Tech-support	Not-in-family	Asian-Pac-Islander	Ma

16280 rows × 15 columns

The "?" symbols are now replaced with NaN values.

2.1 Visualizing features

rubric={points:10}

Your tasks:

1. Examine the information given by `train_df_nan.info()` and `train_df_nan.describe()` methods. In case of `.describe()`, you can optimally use the `include="all"` argument to show summary statistics of all features.
2. Visualize the histograms of numeric features.
3. From the visualizations, which features seem relevant for the given prediction task?

> Note: (Optional) If you're feeling excited about this you are welcome to use `pandas_profiling` for more elaborate visualization and EDA.

In [7]:
```
train_df_nan.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 16280 entries, 27978 to 17673
Data columns (total 15 columns):
 #   Column          Non-Null Count  Dtype 
---  ------          --------------  ----- 
 0   age             16280 non-null  int64 
 1   workclass       15403 non-null  object
 2   fnlwgt          16280 non-null  int64 
 3   education       16280 non-null  object
 4   education.num   16280 non-null  int64 
 5   marital.status  16280 non-null  object
 6   occupation      15398 non-null  object
 7   relationship    16280 non-null  object
 8   race            16280 non-null  object
 9   sex             16280 non-null  object
 10  capital.gain    16280 non-null  int64 
 11  capital.loss    16280 non-null  int64 
 12  hours.per.week  16280 non-null  int64 
 13  native.country  15998 non-null  object
 14  income          16280 non-null  object
dtypes: int64(6), object(9)
memory usage: 2.0+ MB
```

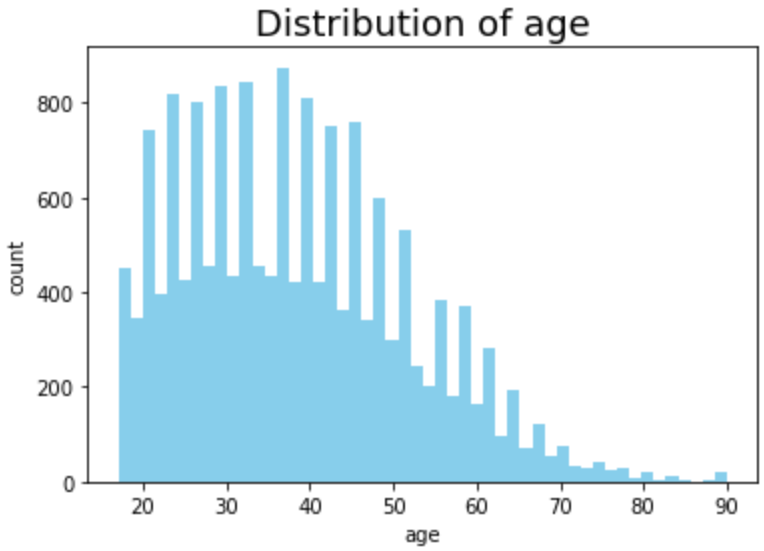
In [8]:
```
train_df_nan.describe(include="all")
```

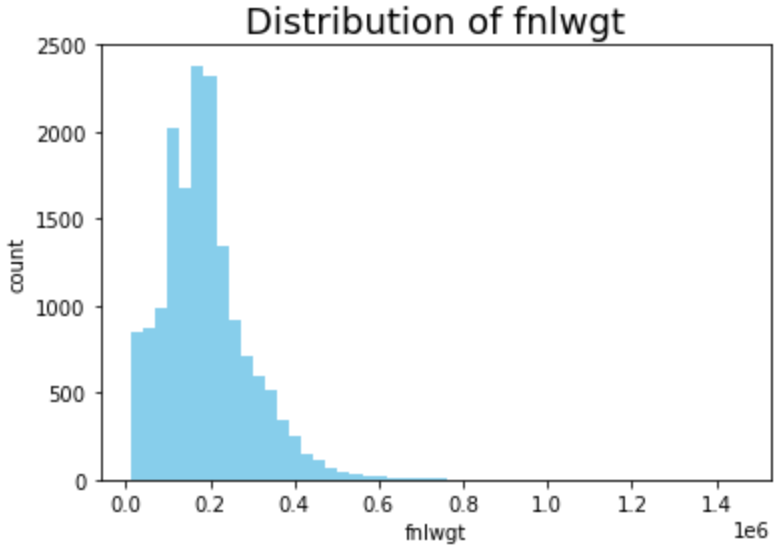
Out[8]:

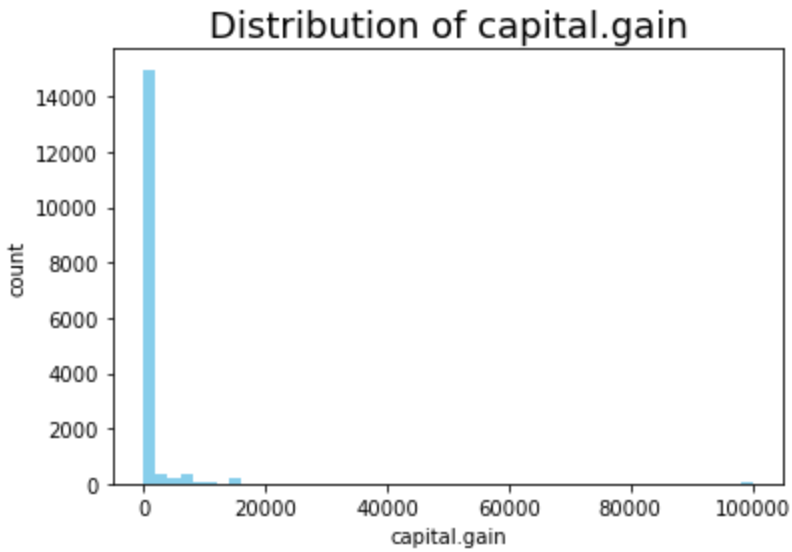
	age	workclass	fnlwgt	education	education.num	marital.status	occupation	relationship
count	16280.00000	15403	1.628000e+04	16280	16280.000000	16280	15398	16280
unique	NaN	8	NaN	16	NaN	7	14	6
top	NaN	Private	NaN	HS-grad	NaN	Married-civ-spouse	Prof-specialty	Husband
freq	NaN	11404	NaN	5259	NaN	7498	2076	6582
mean	38.55731	NaN	1.896666e+05	NaN	10.095885	NaN	NaN	NaN
std	13.53856	NaN	1.047738e+05	NaN	2.573179	NaN	NaN	NaN
min	17.00000	NaN	1.228500e+04	NaN	1.000000	NaN	NaN	NaN
25%	28.00000	NaN	1.178330e+05	NaN	9.000000	NaN	NaN	NaN
50%	37.00000	NaN	1.780250e+05	NaN	10.000000	NaN	NaN	NaN
75%	48.00000	NaN	2.369410e+05	NaN	12.000000	NaN	NaN	NaN
max	90.00000	NaN	1.455435e+06	NaN	16.000000	NaN	NaN	NaN

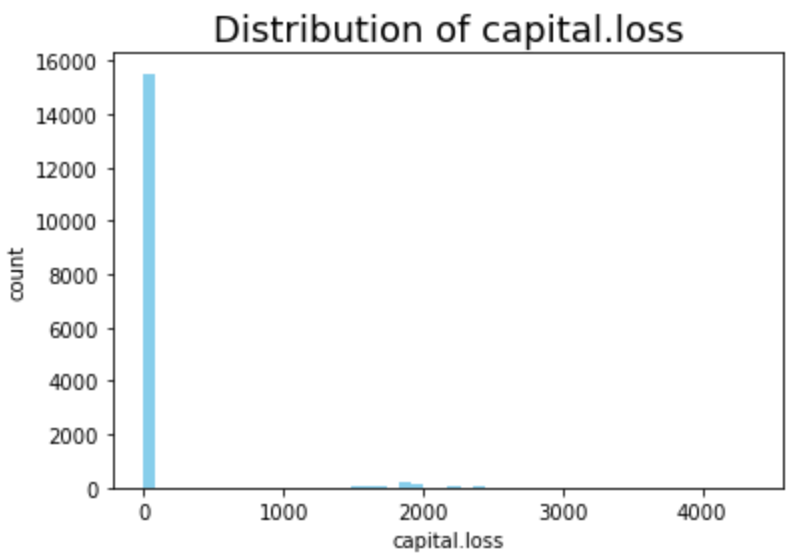
In [9]:
```
for feature in ['age', 'fnlwgt', 'capital.gain', 'capital.loss', 'hours.per.week']:
```

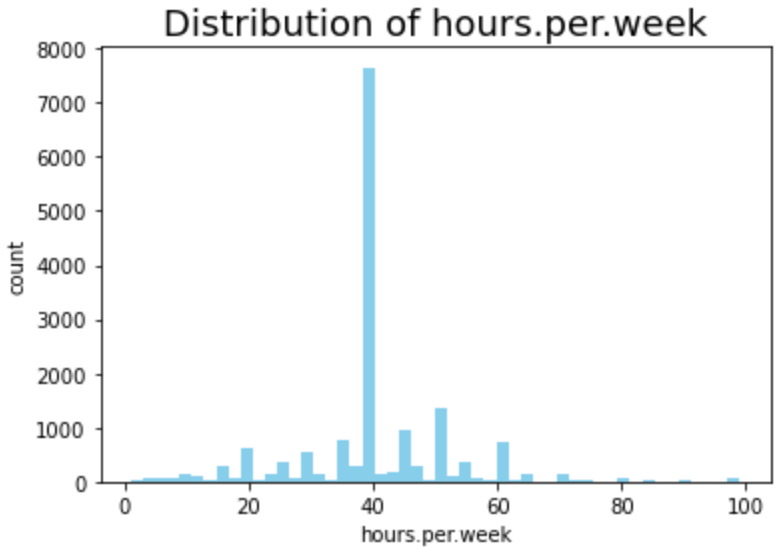
```
    plt.xlabel(feature)
    plt.ylabel('count')
    plt.title(f'Distribution of {feature}', fontsize=18)
    plt.hist(train_df_nan[[feature]], bins=50, facecolor='skyblue', alpha=1, label="0")
    plt.show()
```











In [10]:
```
# ProfileReport(train_df_nan, title="Pandas Profiling Report")
```

Based on the profile report, they all seem relevant except for `fnlwgt`. The seemingly relevant numeric features are age, hours.per.week, capital.gain, and capital.loss.

2.2 Identify transformations to apply

rubric={points:18}

Your tasks:

1. Identify the sequence of transformations that you would apply on each column in the dataset and fill in the table below accordingly. An example of the sequence of transformations to be applied on the `occupation` feature is shown in the table below. You may decide not to apply any transformations on a certain column or entirely drop a column from your model. That's totally fine.
2. Are there common transformations you would like to apply on certain types of features? Identify different feature types for applying different transformations. In particular, fill in the lists below.
3. Is including the `race` feature for predicting income ethically a good idea? Briefly discuss.

> Note: This question is a bit open-ended and there is no single correct solution.

Feature	Transformation
occupation	imputation, OHE
age	scale
workclass	imputation, OHE
fnlwgt	drop
education	drop
education.num	ordinal
marital.status	OHE
relationship	OHE
race	OHE
sex	OHE
capital.gain	scale
capital.loss	scale
hours.per.week	scale
native.country	drop

In [11]:
```
# Fill in the lists below.
# It's OK to keep some of the lists empty or add new lists.
numeric_features = ['age', 'hours.per.week', 'capital.gain', 'capital.loss']
categorical_features = ['occupation', 'workclass', 'marital.status', 'relationship', 'race']
ordinal_features = ['education.num']
binary_features = ['sex']
drop_features = ['fnlwgt', 'education', 'native.country']
passthrough_features = []
target = "income"
```

`fnlwgt` is a consensus number that doesn't seem useful based on the profile report. `education` is redundant because `education.num` is its ordinal form. `native.country` has a high cardinality, a low correlation with `income`, and likely share a confounding variable with `race`.

Including 'race' as a feature in the model is not unethical in itself. For example, studies in socioeconomics in regards to racism would require modeling with race as a feature in order to quantify the effects. What would be unethical, however, to use such model in hiring practices, and thus to perpetuate racial biases that the model demonstrates.

2.3 Separating feature vectors and targets

rubric={points:4}

Your tasks:

1. Create `X_train`, `y_train`, `X_test`, `y_test` from `train_df_nan` and `test_df_nan`.

2. At this point, if you train `sklearn`'s `SVC` model on `X_train` and `y_train` would it work? Why or why not?

In [12]:
```
X_train = train_df_nan.drop(columns=[target])
y_train = train_df_nan[target]
X_test = test_df_nan.drop(columns=[target])
y_test = test_df_nan[target]
```

No, the data has missing values, and not encoded, and there is no pipelines set up yet.

Exercise 3: Preprocessing

3.1 Preprocessing using `sklearn`'s `ColumnTransformer` and `Pipeline`

rubric={points:18}

Let's carry out preprocessing using `sklearn`'s `ColumnTransformer` and `Pipeline`. Note that you can define pipelines in two ways:

- by using `Pipeline` and explicitly providing named steps
- by using `make_pipeline`, which automatically names the steps in the pipeline with their class names.

Similarly you can create a column transformer in two ways:

- by using `ColumnTransformer`
- by using `make_column_transformer`

You may use the method of your choice but `make_pipeline` and `make_column_transformer` are highly recommended.

Your tasks:

1. Create a column transformer `preprocessor` based on transformations you want to apply on the data from 2.2.
2. Transform the data by calling `fit_transform` on the training set. What's the shape of the transformed data?
3. Why do we need to use a column transformer in this case? Briefly explain.

In [13]:
```
# from lecture 6
preprocessor = make_column_transformer(
    (make_pipeline(SimpleImputer(strategy="median"), StandardScaler()), numeric_features),
    (OrdinalEncoder(dtype=int), ordinal_features), # no need for categories because the categori
    (OneHotEncoder(handle_unknown="ignore", drop="if_binary", sparse=False, dtype=int), binary_f
    (make_pipeline(
        SimpleImputer(strategy="most_frequent"),
        OneHotEncoder(handle_unknown="ignore", dtype=int, sparse=False)),
     categorical_features),
```

```
    ("passthrough", passthrough_features),
    ("drop", drop_features),
)
```

In [14]:
```
X_train_pp = preprocessor.fit_transform(X_train, y_train)
X_train_pp.shape
```

Out[14]:
```
(16280, 46)
```

In [15]:
```
# Confirming ordering of encoded categories
preprocessor.named_transformers_['ordinalencoder'].fit(X_train[['education.num']])
preprocessor.named_transformers_['ordinalencoder'].categories_
```

Out[15]:
```
[array([ 1,  2,  3,  4,  5,  6,  7,  8,  9, 10, 11, 12, 13, 14, 15, 16],
       dtype=int64)]
```

The features require various types of transformation, and could not be processed with a single pipeline. The numeric features need to be scaled, and the categorical features need to be one-hot encoded, but not vice versa.

Exercise 4: Building models

Now that we have preprocessed features, we are ready to build models. Below, I'm providing the function we used in class which returns mean cross-validation score along with standard deviation for a given model. Feel free to use it to keep track of your results if you like.

In [16]:
```
results_dict = {}  # dictionary to store all the results
```

In [17]:
```
def mean_std_cross_val_scores(model, X_train, y_train, **kwargs):
    """
    Returns mean and std of cross validation

    Parameters
    ----------
    model :
        scikit-learn model
    X_train : numpy array or pandas DataFrame
        X in the training data
    y_train :
        y in the training data

    Returns
    ----------
        pandas Series with mean scores from 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)
```

4.1 Baseline model

rubric={points:6}

Your tasks:

1. Define a pipeline with two steps: `preprocessor` from 3.1 and `scikit-learn`'s `DummyClassifier` with `strategy="prior"` as your classifier.
2. Carry out 5-fold cross-validation with the pipeline. Store the results in `results_dict` above. Display the results as a pandas DataFrame.

> You may use the function `mean_std_cross_val_scores` above to carry out cross-validation and storing results. Refer to the class notes if you are unsure about how to use it.

In [18]:
```
dummy_pipeline = make_pipeline(preprocessor, DummyClassifier(strategy="prior"))
results_dict['dummy'] = mean_std_cross_val_scores(dummy_pipeline, X_train, y_train, cv=5, return

pd.DataFrame(results_dict)
```

Out[18]:

	dummy
fit_time	0.043 (+/- 0.004)
score_time	0.018 (+/- 0.008)
test_score	0.759 (+/- 0.000)
train_score	0.759 (+/- 0.000)

4.2 Trying different classifiers

rubric={points:14}

Your tasks:

1. For each of the models in the starter code below:
   - Define a pipeline with two steps: `preprocessor` from 3.1 and the model as your classifier.
   - Carry out 5-fold cross-validation with the pipeline.
   - Store the results in `results_dict`.
2. Display all the results so far as a pandas dataframe.
3. Compare the train and validation accuracies and `fit` and `score` times in each case. How do the the validation accuracies compare to the baseline model from 4.1? Which model has the best validation

accuracy? Which model is the fastest one?

> Note that this might take a while to run.
>
> You may use the function above `mean_std_cross_val_scores` to carry out cross-validation and storing results. Refer to the class notes if you are unsure about how to use it.

In [19]:
```
models = {
    "decision tree": DecisionTreeClassifier(),
    "kNN": KNeighborsClassifier(),
    "RBF SVM": SVC(),
}
```

In [20]:
```
for model_name, model in models.items():
    pipeline = make_pipeline(preprocessor, model)
    results_dict[model_name] = mean_std_cross_val_scores(pipeline, X_train, y_train, cv=5, retur
pd.DataFrame(results_dict)
```

Out[20]:

	dummy	decision tree	kNN	RBF SVM
fit_time	0.043 (+/- 0.004)	0.095 (+/- 0.002)	0.050 (+/- 0.006)	4.086 (+/- 0.552)
score_time	0.018 (+/- 0.008)	0.014 (+/- 0.000)	0.908 (+/- 0.043)	1.596 (+/- 0.058)
test_score	0.759 (+/- 0.000)	0.814 (+/- 0.003)	0.835 (+/- 0.007)	0.851 (+/- 0.008)
train_score	0.759 (+/- 0.000)	0.985 (+/- 0.001)	0.882 (+/- 0.002)	0.852 (+/- 0.002)

The models have higher accuracies compared to the baseline model. The RBF SVM model has the best validation accuracy. The decision tree model is the fastest (aside from the dummy model).

(optional) 4.3 Exploring importance of scaling

rubric={points:1}

In this exercise you'll examine whether scaling helps in case of KNNs and SVM RBFs.

Your tasks:

1. Create a column transformer without the `StandardScaler` step for `numeric_features`.
2. Repeat the steps in 4.2 with this new column transformer.
3. Compare the results of scaled numeric features with unscaled numeric features. Is scaling necessary for decision trees? Why or why not?

In [21]:
```
results_dict_no_scaling = {}
```

In [22]:
```
preprocessor_no_scaling = make_column_transformer(
    (make_pipeline(SimpleImputer()), numeric_features),
    (OrdinalEncoder(dtype=int), ordinal_features),
    (OneHotEncoder(handle_unknown="ignore", drop="if_binary", sparse=False, dtype=int), binary_f
    (make_pipeline(
        SimpleImputer(strategy="most_frequent"),
        OneHotEncoder(handle_unknown="ignore", dtype=int, sparse=False)),
     categorical_features),
```

```
    ("passthrough", passthrough_features),
    ("drop", drop_features),
)
```

In [23]:
```
for model_name, model in models.items():
    pipeline = make_pipeline(preprocessor_no_scaling, model)
    results_dict_no_scaling[model_name] = mean_std_cross_val_scores(pipeline, X_train, y_train,
pd.DataFrame(results_dict_no_scaling)
```

Out[23]:

	decision tree	kNN	RBF SVM
fit_time	0.093 (+/- 0.003)	0.047 (+/- 0.008)	4.765 (+/- 0.218)
score_time	0.015 (+/- 0.001)	0.820 (+/- 0.097)	1.800 (+/- 0.029)
test_score	0.814 (+/- 0.004)	0.840 (+/- 0.010)	0.803 (+/- 0.003)
train_score	0.985 (+/- 0.001)	0.889 (+/- 0.003)	0.803 (+/- 0.001)

Scaling improves the RBF SVM model's accuracy. Not much difference for the KNN model, and the model's accuracy is even a little better without scaling. It made no difference for the decision tree model. Scaling is not necessary for decision trees because scaling does not change the overall shapes of distributions of feature values.

4.4 Hyperparameter optimization

rubric={points:10}

In this exercise, you'll carry out hyperparameter optimization for the hyperparameter `C` of SVC RBF classifier. In practice you'll carry out hyperparameter optimization for all different hyperparameters for the most promising classifiers. For the purpose of this assignment, we'll only do it for the `SVC` classifier with one hyperparameter: `C`.

Your tasks:

1. For each `C` value in the `param_grid` in the starter code below:
   - Create a pipeline object with two steps: preprocessor from 3.1 and `SVC` classifier with the value of `C`.
   - Carry out 5-fold cross validation with the pipeline.
   - Store the results in `results_dict` and display results as a pandas DataFrame.
2. The hyperparameter `C = 1.0` is the default value in `scikit-learn`. Does this default value (`C = 1.0`) seem reasonable, meaning it is performing the best or comparable to the best `C` values?

> Note: Running this might take a while.

In [24]:
```
param_grid = {"C": np.logspace(-2, 2, 5)}
param_grid
```

Out[24]:
```
{'C': array([1.e-02, 1.e-01, 1.e+00, 1.e+01, 1.e+02])}
```

In [25]:
```
for C in param_grid["C"]:
    pipeline = make_pipeline(preprocessor, SVC(C=C))
    results_dict[f"SVC C={C}"] = mean_std_cross_val_scores(pipeline, X_train, y_train, cv=5, ret
```

In [26]:
```
pd.DataFrame(results_dict)
```

Out[26]:

	dummy	decision tree	kNN	RBF SVM	SVC C=0.01	SVC C=0.1	SVC C=1.0	SVC C=10.0	SVC C=100.0
fit_time	0.043 (+/- 0.004)	0.095 (+/- 0.002)	0.050 (+/- 0.006)	4.086 (+/- 0.552)	7.009 (+/- 0.974)	5.087 (+/- 0.615)	4.222 (+/- 0.270)	5.008 (+/- 0.214)	6.942 (+/- 0.135)
score_time	0.018 (+/- 0.008)	0.014 (+/- 0.000)	0.908 (+/- 0.043)	1.596 (+/- 0.058)	2.368 (+/- 0.197)	1.938 (+/- 0.128)	1.660 (+/- 0.126)	1.747 (+/- 0.182)	1.563 (+/- 0.037)
test_score	0.759 (+/- 0.000)	0.814 (+/- 0.003)	0.835 (+/- 0.007)	0.851 (+/- 0.008)	0.770 (+/- 0.002)	0.839 (+/- 0.006)	0.851 (+/- 0.008)	0.853 (+/- 0.007)	0.854 (+/- 0.003)
train_score	0.759 (+/- 0.000)	0.985 (+/- 0.001)	0.882 (+/- 0.002)	0.852 (+/- 0.002)	0.771 (+/- 0.001)	0.839 (+/- 0.002)	0.852 (+/- 0.002)	0.858 (+/- 0.001)	0.865 (+/- 0.001)

C = 1.0 seems comparable to the best performing model, and seems reasonable in this case.

Exercise 5: Evaluating on the test set

Now that we have a best performing model, it's time to assess our model on the set aside test set. In this exercise you'll examine whether the results you obtained using cross-validation on the train set are consistent with the results on the test set.

5.1 Scoring on the unseen test set

rubric={points:10}

Your tasks:

1. Train the best performing model on the entire training set.
2. Report the results of this model on `X_test`.
3. Are the cross-validation results and test results consistent?

In [27]:
```
pipeline = make_pipeline(preprocessor, SVC(C=100.0))
pipeline.fit(X_train, y_train)
pipeline.score(X_test, y_test)
```

Out[27]:
```
0.854861494994165
```

Yes, they are consistent.

Submission instructions

PLEASE READ: When you are ready to submit your assignment do the following:

1. Run all cells in your notebook to make sure there are no errors by doing `Kernel -> Restart Kernel and Clear All Outputs` and then `Run -> Run All Cells`.
2. Notebooks with cell execution numbers out of order or not starting from "1" will have marks deducted. Notebooks without the output displayed may not be graded at all (because we need to see the output in order to grade your work).
3. Upload the assignment using Gradescope's drag and drop tool. Check out this Gradescope Student Guide if you need help with Gradescope submission.

In [38]:
```
df = pd.DataFrame(dict(
    x=[1, 2, np.nan],
    y=[2, np.nan, 0]
))

df = SimpleImputer().fit_transform(df)
StandardScaler().fit_transform(df)
```

Out[38]:
```
array([[-1.22474487,  1.22474487],
       [ 1.22474487,  0.        ],
       [ 0.        , -1.22474487]])
```

In [54]:
```
enc = OneHotEncoder(handle_unknown='ignore')
X = [['Male', 1], ['Female', 3], ['Female', 2]]
print(enc.fit_transform(X).toarray())
OneHotEncoder(handle_unknown='ignore')
print(enc.categories_)
print(enc.inverse_transform(enc.transform([['Female', 1], ['Male', 4]]).toarray()))
print(enc.inverse_transform([[0, 1, 1, 0, 0], [0, 1, 0, 1, 0], [1, 0, 0, 1, 0], [0, 1, 0, 1, 0]]
enc.get_feature_names_out(['gender', 'group'])
```

```
[[0. 1. 1. 0. 0.]
 [1. 0. 0. 0. 1.]
 [1. 0. 0. 1. 0.]]
[array(['Female', 'Male'], dtype=object), array([1, 2, 3], dtype=object)]
[['Female' 1]
 ['Male' None]]
[['Male' 1]
 ['Male' 2]
 ['Female' 2]
 ['Male' 2]]
```

Out[54]:
```
array(['gender_Female', 'gender_Male', 'group_1', 'group_2', 'group_3'],
      dtype=object)
```

In [52]:
```
print(enc.transform([['NB', 6], ['Male', 6]]).toarray())
```

```
[[0. 0. 0. 0. 0.]
 [0. 1. 0. 0. 0.]]
```

In [76]:
```
from sklearn.feature_extraction.text import CountVectorizer
corpus = [
    'This is the first document. document. document.',
    'This document is the second document.',
    'And this is the third one.',
    'Is this the first document?',
]

vectorizer = CountVectorizer()
X = vectorizer.fit_transform(corpus)
pd.DataFrame(X.toarray(), columns=vectorizer.get_feature_names())
```

```
C:\Users\Kenny\miniconda3\envs\cpsc330\lib\site-packages\sklearn\utils\deprecation.py:87: FutureWarning: Function get_feature_names is deprecated; get_feature_names is deprecated in 1.0 and will be removed in 1.2. Please use get_feature_names_out instead.
  warnings.warn(msg, category=FutureWarning)
```

Out[76]:

	and	document	first	is	one	second	the	third	this
0	0	3	1	1	0	0	1	0	1
1	0	2	0	1	0	1	1	0	1
2	1	0	0	1	1	0	1	1	1
3	0	1	1	1	0	0	1	0	1

Congratulations on finishing the homework! This was a tricky one but I hope you are feeling good after working on it. You are now ready to build a simple supervised machine learning pipeline on real-world datasets! Well done!

In [ ]: