In [1]:
```
class Obj:
    def __init__(self):
        pass

    def fit(self):
        self.x = 0
```

In [2]:
```
o = Obj()
```

In [3]:
```
o.x
```

```
---------------------------------------------------------------------------
AttributeError                            Traceback (most recent call last)
Input In [3], in <cell line: 1>()
----> 1 o.x

AttributeError: 'Obj' object has no attribute 'x'
```

In [ ]:
```
o.fit()
```

In [ ]:
```
o.y = 1
```

In [ ]:
```
o.y
```

In [ ]:
```
print(o)
```

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 [ ]:
```
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
```

In [ ]:
```
y = np.random.randint(10, size=(10))
x = np.random.randint(10, size=(10))
```

In [ ]:
```
plt.plot(x, y/5, 'o')
```

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

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 [ ]:
```
### BEGIN SOLUTION

train_df, test_df = train_test_split(census_df, test_size=0.5, random_state=10)

### END SOLUTION
```

Exercise 2: Exploratory data analysis (EDA)

Let's examine our `train_df`.

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

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

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

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.

BEGIN SOLUTION

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

In [ ]:
```
train_df_nan.describe()
```

In [ ]:
```
numeric_features = [
    "age",
    "fnlwgt",
    "capital.gain",
    "capital.loss",
    "hours.per.week",
]
train_df_nan_by_income = train_df_nan.groupby("income")
for feat in numeric_features:
    ax = train_df_nan_by_income[feat].plot.hist(bins=20, alpha=0.4, legend=True)
    plt.xlabel(feat)
    plt.title("Histogram of " + feat)
    plt.show()
```

Age feature seem to be a promising feature.

END SOLUTION

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	
workclass	
fnlwgt	
education	
education.num	
marital.status	
relationship	
race	
sex	
capital.gain	
capital.loss	
hours.per.week	
native.country	

In [ ]:
```
# Fill in the lists below.
# It's OK to keep some of the lists empty or add new lists.
numeric_features = []
categorical_features = []
ordinal_features = []
binary_features = []
drop_features = []
passthrough_features = []
target = "income"
```

BEGIN SOLUTION

Feature	Transformation
age	scaling

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

It's fine to apply imputation for all features to make things simpler. Also it's a good strategy to deal with missing values in test data.

In [ ]:
```
# Let's identify numeric and categorical features

numeric_features = [
    "age",
    "fnlwgt",
    "capital.gain",
    "capital.loss",
    "hours.per.week",
]

categorical_features = [
    "workclass",
    "marital.status",
    "occupation",
    "relationship",
    # "race",
    "native.country",
]

ordinal_features = ["education"]
binary_features = ["sex"]
drop_features = ["education.num"]
target = "income"
```

In [ ]:
```
train_df["education"].unique()
```

In [ ]:
```
education_levels = [
    "Preschool",
    "1st-4th",
    "5th-6th",
    "7th-8th",
    "9th",
    "10th",
    "11th",
    "12th",
```

```
    "HS-grad",
    "Prof-school",
    "Assoc-voc",
    "Assoc-acdm",
    "Some-college",
    "Bachelors",
    "Masters",
    "Doctorate",
]
```

In [ ]:
```
assert set(education_levels) == set(train_df["education"].unique())
```

It might not be a good idea to include the race feature to predict income. Such systems get used in applications which can affect real people. For example, this prediction might be used in deciding whether to approve a loan application or not. Influencing this decision by race feature might harm people belonging to certain race.

END SOLUTION

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?

BEGIN SOLUTION

In [ ]:
```
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]
```

In [ ]:
```
y_train
```

It won't work at this point because our data is not preprocessed yet; we have some categorical columns and some NaN values in numeric columns. We need to preprocess it first before feeding it into ML algorithms.

END SOLUTION

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 [ ]:
```
preprocessor = None
```

BEGIN SOLUTION

In [ ]:
```
numeric_transformer = make_pipeline(StandardScaler())

ordinal_transformer = OrdinalEncoder(categories=[education_levels], dtype=int)

categorical_transformer = make_pipeline(
    SimpleImputer(strategy="constant", fill_value="missing"),
    OneHotEncoder(handle_unknown="ignore", sparse=False),
)

binary_transformer = make_pipeline(
    SimpleImputer(strategy="constant", fill_value="missing"),
    OneHotEncoder(drop="if_binary", dtype=int),
)

preprocessor = make_column_transformer(
    (numeric_transformer, numeric_features),
    (ordinal_transformer, ordinal_features),
    (binary_transformer, binary_features),
    (categorical_transformer, categorical_features),
    ("drop", drop_features),
)
```

In [ ]:
```
preprocessor
```

In [ ]:
```
transformed = preprocessor.fit_transform(X_train, y_train)
```

In [ ]:
```
transformed.shape
```

We need column transformer because we want to apply different transformations on different columns.

END SOLUTION

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

In [ ]:
```
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.

BEGIN SOLUTION

In [ ]:
```
# Baseline model

from sklearn.dummy import DummyClassifier

dummy = DummyClassifier(strategy="prior")
pipe = make_pipeline(preprocessor, dummy)
results_dict["dummy"] = mean_std_cross_val_scores(
    pipe, X_train, y_train, cv=5, return_train_score=True
)
pd.DataFrame(results_dict)
```

END SOLUTION

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 [ ]:

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

BEGIN SOLUTION

In [ ]:
```
for model_name, model in models.items():
    print(model_name, ":")
    pipe = make_pipeline(preprocessor, model)
    results_dict[model_name] = mean_std_cross_val_scores(
        pipe, X_train, y_train, cv=5, return_train_score=True
    )
```

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

Observations

- SVC has the best validation performance, followed by KNN and then decision tree.
- Decision tree is clearly overfitting.
- kNN and SVM seem to be too slow compared to the decision tree.

END SOLUTION

(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?

BEGIN SOLUTION

In [ ]:
```
preprocessor_unscaled = make_column_transformer(
    (ordinal_transformer, ordinal_features),
    (binary_transformer, binary_features),
    (categorical_transformer, categorical_features),
    ("drop", drop_features),
)

for model_name, model in models.items():
    # print(model_name, ":")
```

```
    pipe = make_pipeline(preprocessor_unscaled, model)
    results_dict[model_name + "_unscaled"] = mean_std_cross_val_scores(
        pipe, X_train, y_train, cv=5, return_train_score=True
    )
```

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

In [ ]:
```
X_train.shape
```

Observations

- Scaling is clearly helping the distance-based classifiers kNN and SVM with RBF kernel.
- It doesn't seem to have much impact on the decision tree classifier, which makes sense. In principle, tree-based algorithms should be indifferent to scaling and should be able to deal with nominal features. But the `scikit-learn` implementation of the decision tree algorithm requires numeric values for branching.

END SOLUTION

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

BEGIN SOLUTION

In [ ]:
```
for param in param_grid["C"]:
    model_name = "RBF SVC"
    pipe = make_pipeline(preprocessor, SVC(C=param))

    key = model_name + "(C= " + str(param) + ")"
    results_dict[key] = mean_std_cross_val_scores(
```

```
        pipe, X_train, y_train, cv=5, return_train_score=True
    )
```

In [ ]:
```
results_df = pd.DataFrame(results_dict).T
results_df
```

END SOLUTION

BEGIN SOLUTION

Yes, the hyperparameter C = 1.0 is giving good, comparable results.

END SOLUTION

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 [ ]:
```
X_test.shape
```

In [ ]:
```
y_test.shape
```

In [ ]:
```
### BEGIN SOLUTION

final_pipe = make_pipeline(preprocessor, SVC(C=1.0))
final_pipe.fit(X_train, y_train)
final_pipe.score(X_test, y_test)
```

- The test results are more or less consistent with the validation results, which is great!!

END SOLUTION

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.

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!