Mudcard

- **what is purpose of the number in the random_state, or is that not really important?**
  - It is extremely important to use a number, it's not really important what number you use
  - Rerun the cell a couple of times when the random_state argument is removed and check which points are in the training set
  - Then fix the random state to be some number, and rerun the cells again.
  - Then use another number as the random state, and rerun the cells again.
- **Why do we use the same test data in final evaluation.**
  - I assume this is fot k-fold splitting
  - You need to use the test set only once, after you are done with cross-validation
  - Having said that, when you change the random state in train_test_split (and you will be required to do so), different points will be in the test set each time
- **Is there a more systematic alternative to random shuffling that ensures even representation of all classes? What if the dataset is imbalanced?**
  - Yes, you can do a stratified split, we will talk about this during the second half of the term.
- **Also, is it possible to ensure all "types of feature matrices" are well-represented in all sets?**
  - Usually that's not a requirement. You want to make sure the target variable is evenly represented.
- **I'm still sort of confused what the difference is between validation and testing sets, and why both are needed**
  - I hope all of this will be clear in less than two weeks!
- **I would like to go over the parameters of the train test split method. I understand conceptually be would like to have a breakdown of how the use the function.**
  - Write some test code and experiment with all the arguments. I only have time to discuss what I think are the most important arguments in class.
- **Once you train a model with your training data, then validate and test it, do you then make a 'final' model trained on all the data?**
  - You can retrain the model on X_other and y_other.
  - You usually don't use X_test and y_test when you retrain the model.
- **Why does k-fold without shuffling exist if it makes an iid dataset less random?**
  - Because non-iid datasets also exist and for those, it makes sense to not shuffle sometimes
- **What does it mean to set aside one feature for classifying and use the other categories as info for the model?**

    - I'm not sure what you are refering to. Please post on the course forum or talk to me during my office hours.
- **I was a bit confused about why we need to do two train-test splits.**
    - Because we want three sets: train, validation, and test
    - Train_test_split only splits a dataset into two part, not three.
    - So it needs to be applied twice.
- **The 0.75 value specifically**
- **Can we review how we calculate the split for the training set? (Quiz 2 answer)**
- **I am confused about the fractions aspect of this. Is there a resource to study that?**
    - this is high school math so I don't really have good recommendations.
    - Read through the quiz again carefully to work out the fractions.
    - Come to the office hours if you need help.
- **"When should we prefer K-Fold cross-validation over a simple train/validation/test split?**
    - train/val/test split is usually used for large datasets because you only need to train one model per hyperparameter.
    - kfold is better suited for small to medium datasets when you care less about computational efficiency.
    - in kfold, you'll train k number of models for each hyperparameter
- **How the KFold object works?**
    - work with the code provided in the lecture notes to figure it out

Lecture 6: Data preprocessing

By the end of this lecture, you will be able to

- apply one-hot encoding on categorical features
- apply ordinal encoding on ordinal features
- apply scaling and normalization to continuous variables

The supervised ML pipeline

0. Data collection/manipulation: you might have multiple data sources and/or you might have more data than you need

- you need to be able to read in datasets from various sources (like csv, excel, SQL, parquet, etc)
- you need to be able to filter the columns/rows you need for your ML model
- you need to be able to combine the datasets into one dataframe

1. Exploratory Data Analysis (EDA): you need to understand your data and verify that it doesn't contain errors

- do as much EDA as you can!

2. Split the data into different sets: most often the sets are train, validation, and test (or holdout)

- practitioners often make errors in this step!
- you can split the data randomly, based on groups, based on time, or any other non-standard way if necessary to answer your ML question

3. Preprocess the data: ML models only work if X and Y are numbers! Some ML models additionally require each feature to have 0 mean and 1 standard deviation (standardized features)

- often the original features you get contain strings (for example a gender feature would contain 'male', 'female', 'non-binary', 'unknown') which needs to be transformed into numbers
- often the features are not standardized (e.g., age is between 0 and 100) but it needs to be standardized

4. Choose an evaluation metric: depends on the priorities of the stakeholders

- often requires quite a bit of thinking and ethical considerations

5. Choose one or more ML techniques: it is highly recommended that you try multiple models

- start with simple models like linear or logistic regression
- try also more complex models like nearest neighbors, support vector machines, random forest, etc.

6. Tune the hyperparameters of your ML models (aka cross-validation or hyperparameter tuning)

- ML techniques have hyperparameters that you need to optimize to achieve best performance
- for each ML model, decide which parameters to tune and what values to try
- loop through each parameter combination
  - train one model for each parameter combination
  - evaluate how well the model performs on the validation set
- take the parameter combo that gives the best validation score
- evaluate that model on the test set to report how well the model is expected to perform on previously unseen data

7. Interpret your model: black boxes are often not useful

- check if your model uses features that make sense (excellent tool for debugging)
- often model predictions are not enough, you need to be able to explain how the model arrived to a particular prediction (e.g., in health care)

Problem description, why preprocessing is necessary

Data format suitable for ML: 2D numerical values.

X	feature_1	feature_2	...	feature_j	...	feature_m	y
data_point_1	x_11	x_12	...	x_1j	...	x_1m	**y_1**
data_point_2	x_21	x_22	...	x_2j	...	x_2m	**y_2**
...	...	...	...	...	...	...	**...**
data_point_i	x_i1	x_i2	...	x_ij	...	x_im	**y_i**
...	...	...	...	...	...	...	**...**
data_point_n	x_n1	x_n2	...	x_nj	...	x_nm	**y_n**

Data almost never comes in a format that's directly usable in ML.

- let's check the adult data

In [ ]:
```
import pandas as pd
from sklearn.model_selection import train_test_split

df = pd.read_csv('../data/adult_data.csv')

# let's separate the feature matrix X, and target variable y
y = df['gross-income'] # remember, we want to predict who earns more than 50
X = df.loc[:, df.columns != 'gross-income'] # all other columns are features

random_state = 42

# first split to separate out the training set
X_train, X_other, y_train, y_other = train_test_split(X,y,train_size = 0.6,r

# second split to separate out the validation and test sets
X_val, X_test, y_val, y_test = train_test_split(X_other,y_other,train_size =

print('training set')
print(X_train.head()) # lots of strings!
print(y_train.head()) # even our labels are strings and not numbers!
```

scikit-learn transformers to the rescue!

Preprocessing is done with various transformers. All transformes have three methods:

- **fit** method: estimates parameters necessary to do the transformation,
- **transform** method: transforms the data based on the estimated parameters,
- **fit_transform** method: both steps are performed at once, this can be faster than doing the steps separately.

Transformers we cover today

- **OneHotEncoder** - converts categorical features into dummy arrays
- **OrdinalEncoder** - converts ordinal features into an integer array
- **MinMaxScaler** - scales continuous variables to be between 0 and 1
- **StandardScaler** - standardizes continuous features by removing the mean and scaling to unit variance

By the end of this lecture, you will be able to

- **apply one-hot encoding on categorical features**
- apply ordinal encoding on ordinal features
- apply scaling and normalization to continuous variables

Unordered categorical data: one-hot encoder

- some categories cannot be ordered. e.g., workclass, relationship status

In [ ]:
```
from sklearn.preprocessing import OneHotEncoder
help(OneHotEncoder)
```

In [ ]:
```
# toy example
train = {'gender':['Male','Female','Unknown','Male','Female','Female'],\
         'browser':['Safari','Safari','Internet Explorer','Chrome','Chrome',
test = {'gender':['Female','Male','Unknown','Female'],'browser':['Chrome','F

Xtoy_train = pd.DataFrame(train)
Xtoy_test = pd.DataFrame(test)

ftrs = ['gender','browser']

# initialize the encoder
enc = OneHotEncoder(sparse_output=False) # handle_unknown='ignore'  # by def
# fit the training data
enc.fit(Xtoy_train)
print('categories:',enc.categories_)
print('feature names:',enc.get_feature_names_out(ftrs))
# transform X_train
X_train_ohe = enc.transform(Xtoy_train)
#print(X_train_ohe)
# do all of this in one step
X_train_ohe = enc.fit_transform(Xtoy_train)
```

```
print('X_train transformed')
print(X_train_ohe)

# transform X_test
X_test_ohe = enc.transform(Xtoy_test)
print('X_test transformed')
print(X_test_ohe)
```

In [ ]:
```
# apply OHE to the adult dataset

# let's collect all categorical features first
onehot_ftrs = ['workclass','marital-status','occupation','relationship','rac
# initialize the encoder
enc = OneHotEncoder(sparse_output=False,handle_unknown='ignore') # by defaul
# fit the training data
enc.fit(X_train[onehot_ftrs])
print('feature names:',enc.get_feature_names_out(onehot_ftrs))
print(len(enc.get_feature_names_out(onehot_ftrs)))
```

In [ ]:
```
# transform X_train
onehot_train = enc.transform(X_train[onehot_ftrs])
print('transformed train features:')
print(onehot_train)
# transform X_val
onehot_val = enc.transform(X_val[onehot_ftrs])
print('transformed val features:')
print(onehot_val)
# transform X_test
onehot_test = enc.transform(X_test[onehot_ftrs])
print('transformed test features:')
print(onehot_test)
```

By the end of this lecture, you will be able to

- apply one-hot encoding on categorical features
- **apply ordinal encoding on ordinal features**
- apply scaling and normalization to continuous variables

Ordered categorical data: OrdinalEncoder

- use it on categorical features if the categories can be ranked or ordered
  - educational level in the adult dataset
  - reaction to medication is described by words like 'severe', 'no response', 'excellent'
  - any time you know that the categories can be clearly ranked

In [ ]:
```
from sklearn.preprocessing import OrdinalEncoder
help(OrdinalEncoder)
```

In [ ]:
```
# toy example
import pandas as pd

train_edu = {'educational level':['Bachelors','Masters','Bachelors','Doctora
test_edu = {'educational level':['HS-grad','Masters','Masters','College','Ba

Xtoy_train = pd.DataFrame(train_edu)
Xtoy_test = pd.DataFrame(test_edu)

# initialize the encoder
cats = [['HS-grad','College','Bachelors','Masters','Doctorate']]

enc = OrdinalEncoder(categories = cats) # The ordered list of
# categories need to be provided. By default, the categories are alphabetica

# fit the training data
enc.fit(Xtoy_train)
# print the categories - not really important because we manually gave the c
print(enc.categories_)
# transform X_train. We could have used enc.fit_transform(X_train) to combin
X_train_oe = enc.transform(Xtoy_train)
print(X_train_oe)
# transform X_test
X_test_oe = enc.transform(Xtoy_test) # OrdinalEncoder always throws an error
                                     # it encounters an unknown category in tes
print(X_test_oe)
```

In [ ]:
```
# apply OE to the adult dataset
# initialize the encoder
ordinal_ftrs = ['education'] # if you have more than one ordinal feature, ad
ordinal_cats = [[' Preschool',' 1st-4th',' 5th-6th',' 7th-8th',' 9th',' 10th
                 ' Some-college',' Assoc-voc',' Assoc-acdm',' Bachelors',' Ma
# ordinal_cats must contain one list per ordinal feature! each list contains
# of the corresponding feature

enc = OrdinalEncoder(categories = ordinal_cats)   # By default, the categori
                                                  # which is NOT what you

# fit the training data
enc.fit(X_train[ordinal_ftrs])  # the encoder expects a 2D array, that's why

# transform X_train. We could use enc.fit_transform(X_train) to combine fit
ordinal_train = enc.transform(X_train[ordinal_ftrs])
print('transformed train features:')
print(ordinal_train)
# transform X_val
ordinal_val = enc.transform(X_val[ordinal_ftrs])
print('transformed validation features:')
print(ordinal_val)
# transform X_test
ordinal_test = enc.transform(X_test[ordinal_ftrs])
print('transformed test features:')
print(ordinal_test)
```

Quiz 1

Please explain how you would encode the race feature below and what would be the output of the encoder. Do not write code. The goal of this quiz is to test your conceptual understanding so write text and the output array.

race = [' Amer-Indian-Eskimo', 'White', 'Black', 'Asian-Pac-Islander', 'Black', 'White', 'White']

By the end of this lecture, you will be able to

- apply one-hot encoding on categorical features
- apply ordinal encoding on ordinal features
- **apply scaling and normalization to continuous variables**

Continuous features: MinMaxScaler

- If the continuous feature values are reasonably bounded, MinMaxScaler is a good way to scale the features.
- Age is expected to be within the range of 0 and 100.
- Number of hours worked per week is in the range of 0 to 80.
- If unsure, plot the histogram of the feature to verify or just go with the standard scaler!

In [ ]:
```
from sklearn.preprocessing import MinMaxScaler
help(MinMaxScaler)
```

In [ ]:
```
# toy data
# let's assume we have two continuous features:
train = {'age':[32,65,13,68,42,75,32],'number of hours worked':[0,40,10,60,4
test = {'age':[83,26,10,60],'number of hours worked':[0,40,0,60]}

# (value - min) / (max - min), if value is 32, min is 13 and max is 75, ther

Xtoy_train = pd.DataFrame(train)
Xtoy_test = pd.DataFrame(test)

scaler = MinMaxScaler()
scaler.fit(Xtoy_train)
print(scaler.transform(Xtoy_train))
print(scaler.transform(Xtoy_test)) # note how scaled X_test contains values
```

In [ ]:
```
# adult data

minmax_ftrs = ['age','hours-per-week']
```

```
scaler = MinMaxScaler()
scaler.fit(X_train[minmax_ftrs])
print(scaler.transform(X_train[minmax_ftrs]))
print(scaler.transform(X_val[minmax_ftrs]))
print(scaler.transform(X_test[minmax_ftrs]))
```

Continuous features: StandardScaler

- If the continuous feature values follow a tailed distribution, StandardScaler is better to use!
- Salaries are a good example. Most people earn less than 100k but there are a small number of super-rich people.

In [ ]:
```
from sklearn.preprocessing import StandardScaler
help(StandardScaler)
```

In [ ]:
```
# toy data
train = {'salary':[50_000,75_000,40_000,1_000_000,30_000,250_000,35_000,45_0
test = {'salary':[25_000,55_000,1_500_000,60_000]}

Xtoy_train = pd.DataFrame(train)
Xtoy_test = pd.DataFrame(test)

scaler = StandardScaler()
print(scaler.fit_transform(Xtoy_train))
print(scaler.transform(Xtoy_test))
```

In [ ]:
```
# adult data

std_ftrs = ['capital-gain','capital-loss']
scaler = StandardScaler()
print(scaler.fit_transform(X_train[std_ftrs]))
print(scaler.transform(X_val[std_ftrs]))
print(scaler.transform(X_test[std_ftrs]))
```

Quiz 2

Which of these features could be safely preprocessed by the minmax scaler?

- number of minutes spent on the website in a day
- number of days a year spent abroad in a year
- USD donated to charity

How and when to do preprocessing in the ML pipeline?

- **APPLY TRANSFORMER.FIT ONLY ON YOUR TRAINING DATA!** Then transform the validation and test sets.
- One of the most common mistake practitioners make is leaking statistics!
  - fit_transform is applied to the whole dataset, then the data is split into train/validation/test
    - this is wrong because the test set statistics impacts how the training and validation sets are transformed
    - but the test set must be separated from train and val, and val must be separated from train
  - or fit_transform is applied to the train, then fit_transform is applied to the validation set, and fit_transform is applied to the test set
    - this is wrong because the relative position of the points change

 No description has been provided for this image

Scikit-learn's pipelines

- The steps in the ML pipleine can be chained together into a scikit-learn pipeline which consists of transformers and one final estimator which is usually your classifier or regression model.
- It neatly combines the preprocessing steps and it helps to avoid leaking statistics.

https://scikit-learn.org/stable/auto_examples/compose/plot_column_transformer_mixed_types.html

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

from sklearn.compose import ColumnTransformer
from sklearn.pipeline import Pipeline
from sklearn.preprocessing import StandardScaler, OneHotEncoder, OrdinalEncc
from sklearn.model_selection import train_test_split

#np.random.seed(0)

df = pd.read_csv('../data/adult_data.csv')

# let's separate the feature matrix X, and target variable y
y = df['gross-income'] # remember, we want to predict who earns more than 50
X = df.loc[:, df.columns != 'gross-income'] # all other columns are features

random_state = 42

# first split to separate out the training set
X_train, X_other, y_train, y_other = train_test_split(X,y,train_size = 0.6,r

# second split to separate out the validation and test sets
X_val, X_test, y_val, y_test = train_test_split(X_other,y_other,train_size =
```

In [ ]:
```
# collect which encoder to use on each feature
# needs to be done manually
ordinal_ftrs = ['education']
ordinal_cats = [[' Preschool',' 1st-4th',' 5th-6th',' 7th-8th',' 9th',' 10th
                ' Some-college',' Assoc-voc',' Assoc-acdm',' Bachelors',' Ma
onehot_ftrs = ['workclass','marital-status','occupation','relationship','rac
minmax_ftrs = ['age','hours-per-week']
std_ftrs = ['capital-gain','capital-loss']

# collect all the encoders
preprocessor = ColumnTransformer(
    transformers=[
        ('ord', OrdinalEncoder(categories = ordinal_cats), ordinal_ftrs),
        ('onehot', OneHotEncoder(sparse_output=False,handle_unknown='ignore'
        ('minmax', MinMaxScaler(), minmax_ftrs),
        ('std', StandardScaler(), std_ftrs)])

clf = Pipeline(steps=[('preprocessor', preprocessor)]) # for now we only pre
                                                       # later on we will ad

X_train_prep = clf.fit_transform(X_train)
X_val_prep = clf.transform(X_val)
X_test_prep = clf.transform(X_test)

print(X_train.shape)
print(X_train_prep.shape)
print(X_train_prep)
```

Mudcard

In [ ]:
```
```