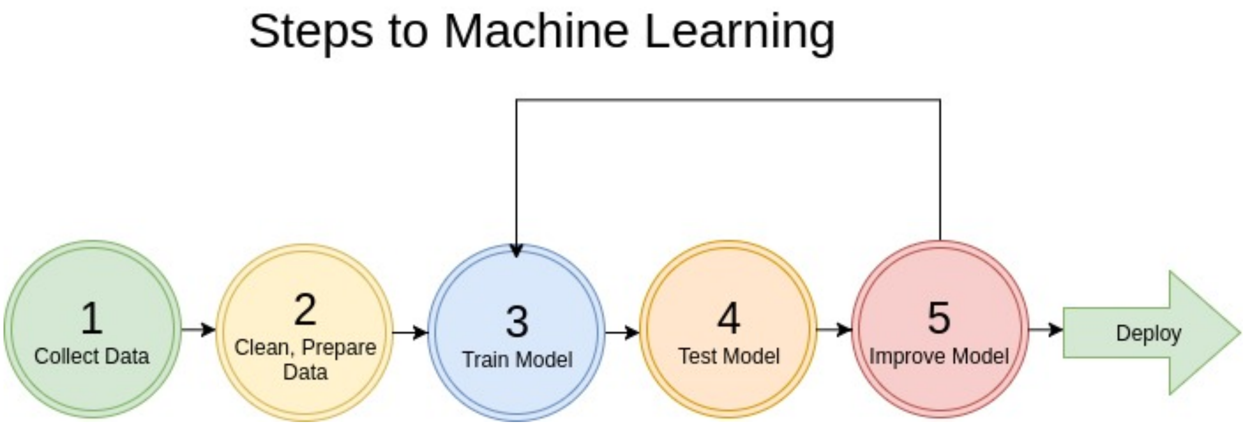
Introduction

Welcome to **CS188 - Data Science Fundamentals!** We plan on having you go through some grueling training so you can start crunching data out there... in today's day and age "data is the new oil" or perhaps "snake oil" nonetheless, there's a lot of it, each with different purity (so pure that perhaps you could feed off it for a life time) or dirty which then at that point you can either decide to dump it or try to weed out something useful (that's where they need you...)

In this project you will work through an example project end to end.

Here are the main steps:

1. Get the data
2. Visualize the data for insights
3. Preprocess the data for your machine learning algorithm
4. Select a model and train
5. Does it meet the requirements? Fine tune the model



Working with Real Data

It is best to experiment with real-data as opposed to aritifical datasets.

There are many different open datasets depending on the type of problems you might be interested in!

Here are a few data repositories you could check out:

- [UCI Datasets (http://archive.ics.uci.edu/ml/)](http://archive.ics.uci.edu/ml/)
- [Kaggle Datasets (kaggle.com)](kaggle.com)
- [AWS Datasets (https://registry.opendata.aws)](https://registry.opendata.aws)

Below we will run through an California Housing example collected from the 1990's.

Setup

In [ ]:

```
import sys
assert sys.version_info >= (3, 5) # python>=3.5
import sklearn
assert sklearn.__version__ >= "0.20" # sklearn >= 0.20

import numpy as np #numerical package in python
import os
%matplotlib inline
import matplotlib.pyplot as plt #plotting package

# to make this notebook's output identical at every run
np.random.seed(42)

#matplotlib magic for inline figures
%matplotlib inline
import matplotlib # plotting library
import matplotlib.pyplot as plt

# Where to save the figures
ROOT_DIR = "."
IMAGES_PATH = os.path.join(ROOT_DIR, "images")
os.makedirs(IMAGES_PATH, exist_ok=True)

def save_fig(fig_name, tight_layout=True, fig_extension="png", resolut
    '''
        plt.savefig wrapper. refer to
        https://matplotlib.org/3.1.1/api/_as_gen/matplotlib.pyplot.sav
    '''
    path = os.path.join(IMAGES_PATH, fig_name + "." + fig_extension)
    print("Saving figure", fig_name)
    if tight_layout:
        plt.tight_layout()
    plt.savefig(path, format=fig_extension, dpi=resolution)
```

In [ ]:

```
import os
import tarfile
import urllib
DATASET_PATH = os.path.join("datasets", "housing")
```

Intro to Data Exploration Using Pandas

In this section we will load the dataset, and visualize different features using different types of plots.

Packages we will use:

- **Pandas (https://pandas.pydata.org):** is a fast, flexibile and expressive data structure widely used for tabular and multidimensional datasets.
- **Matplotlib (https://matplotlib.org)**: is a 2d python plotting library which you can use to create quality figures (you can plot almost anything if you're willing to code it out!)
    - other plotting libraries:seaborn (https://seaborn.pydata.org), ggplot2 (https://ggplot2.tidyverse.org)

```
In [ ]: import pandas as pd

        def load_housing_data(housing_path):
            csv_path = os.path.join(housing_path, "housing.csv")
            return pd.read_csv(csv_path)
```

```
In [ ]: housing = load_housing_data(DATASET_PATH) # we load the pandas datafra
        housing.head() # show the first few elements of the dataframe
                       # typically this is the first thing you do
                       # to see how the dataframe looks like
```

A dataset may have different types of features

- real valued
- Discrete (integers)
- categorical (strings)

The two categorical features are essentialy the same as you can always map a categorical string/character to an integer.

In the dataset example, all our features are real valued floats, except ocean proximity which is categorical.

```
In [ ]: # to see a concise summary of data types, null values, and counts
        # use the info() method on the dataframe
        housing.info()
```

```
In [ ]: # you can access individual columns similarly
        # to accessing elements in a python dict
        housing["ocean_proximity"].head() # added head() to avoid printing mar
```

```
In [ ]: # to access a particular row we can use iloc
        housing.iloc[1]
```

In [ ]:
```
# one other function that might be useful is
# value_counts(), which counts the number of occurences
# for categorical features
housing["ocean_proximity"].value_counts()
```

In [ ]:
```
# The describe function compiles your typical statistics for each
# column
housing.describe()
```

If you want to learn about different ways of accessing elements or other functions it's useful to check out the getting started section here (https://pandas.pydata.org/pandas-docs/stable/getting_started/index.html)

Let's start visualizing the dataset

In [ ]:
```
# We can draw a histogram for each of the dataframes features
# using the hist function
housing.hist(bins=50, figsize=(20,15))
# save_fig("attribute_histogram_plots")
plt.show() # pandas internally uses matplotlib, and to display all the
           # the show() function must be called
```

In [ ]:
```
# if you want to have a histogram on an individual feature:
housing["median_income"].hist()
plt.show()
```

We can convert a floating point feature to a categorical feature by binning or by defining a set of intervals.

For example, to bin the households based on median_income we can use the pd.cut function

In [ ]:
```
# assign each bin a categorical value [1, 2, 3, 4, 5] in this case.
housing["income_cat"] = pd.cut(housing["median_income"],
                               bins=[0., 1.5, 3.0, 4.5, 6., np.inf],
                               labels=[1, 2, 3, 4, 5])

housing["income_cat"].value_counts()
```

In [ ]:
```
housing["income_cat"].hist()
```

Next let's visualize the household incomes based on latitude & longitude coordinates

In [ ]:

```
## here's a not so interestting way plotting it
housing.plot(kind="scatter", x="longitude", y="latitude")
save_fig("bad_visualization_plot")
```

In [ ]:

```
# we can make it look a bit nicer by using the alpha parameter,
# it simply plots less dense areas lighter.
housing.plot(kind="scatter", x="longitude", y="latitude", alpha=0.1)
save_fig("better_visualization_plot")
```

In [ ]:

```
# A more interesting plot is to color code (heatmap) the dots
# based on income. The code below achieves this

# load an image of california
images_path = os.path.join('./', "images")
os.makedirs(images_path, exist_ok=True)
filename = "california.png"

import matplotlib.image as mpimg
california_img=mpimg.imread(os.path.join(images_path, filename))
ax = housing.plot(kind="scatter", x="longitude", y="latitude", figsize
                       s=housing['population']/100, label="Population'
                       c="median_house_value", cmap=plt.get_cmap("jet'
                       colorbar=False, alpha=0.4,
                      )
# overlay the califronia map on the plotted scatter plot
# note: plt.imshow still refers to the most recent figure
# that hasn't been plotted yet.
plt.imshow(california_img, extent=[-124.55, -113.80, 32.45, 42.05], al
           cmap=plt.get_cmap("jet"))
plt.ylabel("Latitude", fontsize=14)
plt.xlabel("Longitude", fontsize=14)

# setting up heatmap colors based on median_house_value feature
prices = housing["median_house_value"]
tick_values = np.linspace(prices.min(), prices.max(), 11)
cb = plt.colorbar()
cb.ax.set_yticklabels(["$%dk"%(round(v/1000)) for v in tick_values], f
cb.set_label('Median House Value', fontsize=16)

plt.legend(fontsize=16)
save_fig("california_housing_prices_plot")
plt.show()
```

Not suprisingly, the most expensive houses are concentrated around the San Francisco/Los Angeles areas.

Up until now we have only visualized feature histograms and basic statistics.

When developing machine learning models the predictiveness of a feature for a particular target of intrest is what's important.

It may be that only a few features are useful for the target at hand, or features may need to be augmented by applying certain transfomrations.

None the less we can explore this using correlation matrices.

In [ ]:

```
corr_matrix = housing.corr()
```

In [ ]:

```
# for example if the target is "median_house_value", most correlated i
# which happens to be "median_income". This also intuitively makes ser
corr_matrix["median_house_value"].sort_values(ascending=False)
```

In [ ]:

```
# the correlation matrix for different attributes/features can also be
# some features may show a positive correlation/negative correlation o
# it may turn out to be completely random!
from pandas.plotting import scatter_matrix
attributes = ["median_house_value", "median_income", "total_rooms",
              "housing_median_age"]
scatter_matrix(housing[attributes], figsize=(12, 8))
save_fig("scatter_matrix_plot")
```

In [ ]:

```
# median income vs median house vlue plot plot 2 in the first row of t
housing.plot(kind="scatter", x="median_income", y="median_house_value"
             alpha=0.1)
plt.axis([0, 16, 0, 550000])
save_fig("income_vs_house_value_scatterplot")
```

Augmenting Features

New features can be created by combining different columns from our data set.

- rooms_per_household = total_rooms / households
- bedrooms_per_room = total_bedrooms / total_rooms
- etc.

In [ ]:

```
housing["rooms_per_household"] = housing["total_rooms"]/housing["house
housing["bedrooms_per_room"] = housing["total_bedrooms"]/housing["tota
housing["population_per_household"]=housing["population"]/housing["hou
```

In [ ]:

```
# obtain new correlations
corr_matrix = housing.corr()
corr_matrix["median_house_value"].sort_values(ascending=False)
```

In [ ]:

```
housing.plot(kind="scatter", x="rooms_per_household", y="median_house_
             alpha=0.2)
plt.axis([0, 5, 0, 520000])
plt.show()
```

In [ ]:

```
housing.describe()
```

Preparing Dastaset for ML

Once we've visualized the data, and have a certain understanding of how the data looks like. It's time to clean!

Most of your time will be spent on this step, although the datasets used in this project are relatively nice and clean... it could get real dirty.

After having cleaned your dataset you're aiming for:

- train set
- test set

In some cases you might also have a validation set as well for tuning hyperparameters (don't worry if you're not familiar with this term yet..)

In supervised learning setting your train set and test set should contain (**feature**, **target**) tuples.

- **feature**: is the input to your model
- **target**: is the ground truth label
    - when target is categorical the task is a classification task
    - when target is floating point the task is a regression task

We will make use of scikit-learn (https://scikit-learn.org/stable/) python package for preprocessing.

Scikit learn is pretty well documented and if you get confused at any point simply look up the function/object!

In [ ]:

```
from sklearn.model_selection import StratifiedShuffleSplit
# let's first start by creating our train and test sets
split = StratifiedShuffleSplit(n_splits=1, test_size=0.2, random_state
for train_index, test_index in split.split(housing, housing["income_ca
    train_set = housing.loc[train_index]
    test_set = housing.loc[test_index]
```

In [ ]:

```
housing = train_set.drop("median_house_value", axis=1) # drop labels f
                                                       # the input to
housing_labels = train_set["median_house_value"].copy()
```

Dealing With Incomplete Data

In [ ]:

```
# have you noticed when looking at the dataframe summary certain rows
# contained null values? we can't just leave them as nulls and expect
# model to handle them for us...
sample_incomplete_rows = housing[housing.isnull().any(axis=1)].head()
sample_incomplete_rows
```

In [ ]:

```
sample_incomplete_rows.dropna(subset=["total_bedrooms"])    # option 1
```

In [ ]:

```
sample_incomplete_rows.drop("total_bedrooms", axis=1)       # option 2
```

In [ ]:

```
median = housing["total_bedrooms"].median()
sample_incomplete_rows["total_bedrooms"].fillna(median, inplace=True)
sample_incomplete_rows
```

Could you think of another plausible imputation for this dataset? (Not graded)

Prepare Data

In [ ]:

```
# This cell implements the complete pipeline for preparing the data
# using sklearns TransformerMixins
# Earlier we mentioned different types of features: categorical, and f
# In the case of floats we might want to convert them to categories.
# On the other hand categories in which are not already represented as
# feeding to the model.

# Additionally, categorical values could either be represented as one-
# Here we encode them using one hot vectors.

from sklearn.impute import SimpleImputer
from sklearn.compose import ColumnTransformer

from sklearn.pipeline import Pipeline
from sklearn.preprocessing import StandardScaler
from sklearn.preprocessing import OneHotEncoder

from sklearn.base import BaseEstimator, TransformerMixin


imputer = SimpleImputer(strategy="median") # use median imputation for
housing_num = housing.drop("ocean_proximity", axis=1) # remove the cat
# column index
```

```
rooms_ix, bedrooms_ix, population_ix, households_ix = 3, 4, 5, 6

#
class AugmentFeatures(BaseEstimator, TransformerMixin):
    '''
    implements the previous features we had defined
    housing["rooms_per_household"] = housing["total_rooms"]/housing["h
    housing["bedrooms_per_room"] = housing["total_bedrooms"]/housing["
    housing["population_per_household"]=housing["population"]/housing[
    '''
    def __init__(self, add_bedrooms_per_room = True):
        self.add_bedrooms_per_room = add_bedrooms_per_room
    def fit(self, X, y=None):
        return self  # nothing else to do
    def transform(self, X):
        rooms_per_household = X[:, rooms_ix] / X[:, households_ix]
        population_per_household = X[:, population_ix] / X[:, househol
        if self.add_bedrooms_per_room:
            bedrooms_per_room = X[:, bedrooms_ix] / X[:, rooms_ix]
            return np.c_[X, rooms_per_household, population_per_househ
                         bedrooms_per_room]
        else:
            return np.c_[X, rooms_per_household, population_per_househ

attr_adder = AugmentFeatures(add_bedrooms_per_room=False)
housing_extra_attribs = attr_adder.transform(housing.values)

num_pipeline = Pipeline([
        ('imputer', SimpleImputer(strategy="median")),
        ('attribs_adder', AugmentFeatures()),
        ('std_scaler', StandardScaler()),
    ])

housing_num_tr = num_pipeline.fit_transform(housing_num)
numerical_features = list(housing_num)
categorical_features = ["ocean_proximity"]

full_pipeline = ColumnTransformer([
        ("num", num_pipeline, numerical_features),
        ("cat", OneHotEncoder(), categorical_features),
    ])

housing_prepared = full_pipeline.fit_transform(housing)
```

Select a model and train

Once we have prepared the dataset it's time to choose a model.

As our task is to predict the median_house_value (a floating value), regression is well suited for this.

In [ ]:

```python
from sklearn.linear_model import LinearRegression

lin_reg = LinearRegression()
lin_reg.fit(housing_prepared, housing_labels)

# let's try the full preprocessing pipeline on a few training instance
data = test_set.iloc[:5]
labels = housing_labels.iloc[:5]
data_prepared = full_pipeline.transform(data)

print("Predictions:", lin_reg.predict(data_prepared))
print("Actual labels:", list(labels))
```

We can evaluate our model using certain metrics, a fitting metric for regresison is the mean-squared-loss

$$L(\hat{Y}, Y) = \sum_{i}^{N} (\hat{y_i} - y_i)^2$$

where $\hat{y}$ is the predicted value, and y is the ground truth label.

In [ ]:

```python
from sklearn.metrics import mean_squared_error

preds = lin_reg.predict(housing_prepared)
mse = mean_squared_error(housing_labels, preds)
rmse = np.sqrt(mse)
rmse
```

TODO: Applying the end-end ML steps to a different dataset.

We will apply what we've learnt to another dataset (airbnb dataset). We will predict airbnb price based on other features.

[25 pts] Visualizing Data

[5 pts] Load the data + statistics

- load the dataset
- display the first few rows of the data
- drop the following columns: name, host_id, host_name, last_review
- display a summary of the statistics of the loaded data
- plot histograms for 3 features of your choice

In [ ]:

[5 pts] Plot total number_of_reviews per neighbourhood_group

In [ ]:

[5 pts] Plot map of airbnbs throughout New York (if it gets too crowded take a subset of the data, and try to make it look nice if you can :)).

In [ ]:

[5 pts] Plot average price of room types who have availability greater than 180 days.

In [ ]:

[5 pts] Plot correlation matrix

- which features have positive correlation?
- which features have negative correlation?

In [ ]:

[25 pts] Prepare the Data

[5 pts] Set aside 20% of the data as test test (80% train, 20% test).

In [ ]:

[5 pts] Augment the dataframe with two other features which you think would be useful

In [ ]:

[5 pts] Impute any missing feature with a method of your choice, and briefly discuss why you chose this imputation method

In [ ]:

[10 pts] Code complete data pipeline using sklearn mixins

In [ ]:

[15 pts] Fit a model of your choice

The task is to predict the price, you could refer to the housing example on how to train and evaluate your model using MSE. Provide both test and train set MSE values.

In [ ]: