END TO END MACHINE LEARNING

FATEMA_NAGORI_19635

TABLE OF CONTENT

- Introduction
- Process
- Conclusion
- Enhancement
- References

INTRODUCTION

End-to-end machine learning refers to the process of building a machine learning system that can take in raw data and produce a desired output, without relying on hand-crafted features or pre-processing steps. In an end-to-end machine learning system, the entire process from data ingestion to output generation is automated.

The goal of end-to-end machine learning is to simplify and streamline the machine learning workflow, reducing the amount of manual intervention required and making it easier to develop and deploy machine learning models.

Process

A: LOOK AT THE BIG PICTURE

- Frame the problem

It is clearly a typical supervised learning task since you are given labeled training examples (each instance comes with the expected output, i.e., the district's median housing price).

It is also a typical regression task, since you are asked to predict a value.

More specifically, this is a multivariate regression problem since the system will use multiple features to make a prediction (it will use the district's population, the median income, etc.).

There is no continuous flow of data coming in the system, there is no particular need to adjust to changing data rapidly, and the data is small enough to fit in memory, so plain batch learning should do just fine.

If the data was huge, you could either split your batch learning work across multiple servers (using the MapReduce technique, as we will see later), or you could use an online learning technique instead.

- Select a performance Measure

	Root Mean Square Error (RMSE) Euclidian distance	Mean Absolute Error (MAE) Manhattan distance
Formula	$\mathrm{RMSE}(\mathbf{X}, h) = \sqrt{\frac{1}{m}\sum_{i=1}^{m}\left(h(\mathbf{x}^{(i)}) - y^{(i)}\right)^2}$	$\mathrm{MAE}(\mathbf{X}, h) = \frac{1}{m}\sum_{i=1}^{m}\left\lvert h(\mathbf{x}^{(i)}) - y^{(i)}\right\rvert$
	• m is the number of instances in the dataset you are measuring. • $x^{(i)}$ is a vector of all the feature values (excluding the label) of the i^{th} instance in the dataset • $y^{(i)}$ is its label (the desired output value for that instance).	
Comment	○ RMSE is a typical performance measure for regression problems	

- ## Check the Assumptions: Classification vs. Regression

Lastly, it is good practice to list and verify the assumptions that have been made so far (by you or others); this can help you catch serious issues early on. For example, the district prices that your system outputs are going to be fed into a downstream Machine Learning system, and you assume that these prices are going to be used as such. But what if the downstream system converts the prices into categories (e.g., “cheap,” “medium,” or “expensive”) and then uses those categories instead of the prices themselves? In this case, getting the price perfectly right is not important at all; your system just needs to get the category right. If that’s so, then the problem should have been framed as a classification task, not a regression task. You don’t want to find this out after working on a regression system for months.

Fortunately, after talking with the team in charge of the downstream system, you are confident that they do indeed need the actual prices, not just categories. Great! You’re all set, the lights are green, and you can start coding now

B: GET THE DATA

- CREATE A WORKSPACE-FOR JUPYTER NOTEBOOK
- DOWNLOAD THE DATA

- First you will need to have Python installed. It is probably already installed on your system. If not, you can get it at

  https://www.python.org/

- Next you need to create a workspace directory for your Machine Learning code and datasets. Open a terminal and type the following commands (after the $ prompts):

```
$ export ML_PATH="$HOME/ml"      # You can change the path if you prefer
$ mkdir -p $ML_PATH
```

- You will need a number of Python modules: Jupyter, NumPy, Pandas, Matplotlib, and Scikit-Learn.
  - If you already have Jupyter running with all these modules installed, you can safely skip to "Download the Data".
  - If you don't have them yet, there are many ways to install them (and their dependencies).

  You can use your system's packaging system (e.g., apt-get on Ubuntu, or MacPorts or HomeBrew on macOS), install a Scientific Python distribution such as Anaconda and use its packaging system, or just use Python's own packaging system, pip, which is included by default with the Python binary installers (since Python 2.7.9).[6] You can check to see if pip is installed by typing the following command:

```
$ pip3 --version
pip 9.0.1 from [...]/lib/python3.5/site-packages (python 3.5)
```

- You should make sure you have a recent version of pip installed, at the very least >1.4 to support binary module installation (a.k.a. wheels). To upgrade the pip module, type:[7]

```
$ pip3 install --upgrade pip
Collecting pip
[...]
Successfully installed pip-9.0.1
```

● TAKE A QUICK LOOK AT THE DATA STRUCTURE

A. The top five rows using Pandas DataFrame's head() method.

```
In [5]: housing = load_housing_data()
        housing.head()
```

Out[5]:

	longitude	latitude	housing_median_age	total_rooms	total_bedrooms	populatio
0	-122.23	37.88	41.0	880.0	129.0	322.0
1	-122.22	37.86	21.0	7099.0	1106.0	2401.0
2	-122.24	37.85	52.0	1467.0	190.0	496.0
3	-122.25	37.85	52.0	1274.0	235.0	558.0
4	-122.25	37.85	52.0	1627.0	280.0	565.0

Note:
- Each row represents one district.
- There are 10 attributes (you can see the first 6 in the screenshot):
  - longitude
  - latitude
  - housing_median_age
  - total_rooms
  - total_bedrooms
  - population
  - households
  - median_income
  - median_house_value
  - ocean_proximity

▪ ocean_proximity

B. The info() method is useful to get a quick description of the data, in particular the total number of rows, and each attribute's type and number of non-null values.

```
In [6]: housing.info()
        <class 'pandas.core.frame.DataFrame'>
        RangeIndex: 20640 entries, 0 to 20639
        Data columns (total 10 columns):
        longitude             20640 non-null float64
        latitude              20640 non-null float64
        housing_median_age    20640 non-null float64
        total_rooms           20640 non-null float64
        total_bedrooms        20433 non-null float64
        population            20640 non-null float64
        households            20640 non-null float64
        median_income         20640 non-null float64
        median_house_value    20640 non-null float64
        ocean_proximity       20640 non-null object
        dtypes: float64(9), object(1)
        memory usage: 1.6+ MB
```

- CREATE A TEST CASE

- How to create a Test Set

```
# The following code is for illustration only.
# - Sklearn has train_test_split()
def split_train_test(data, test_ratio):
    shuffled_indices = np.random.permutation(len(data))
    test_set_size = int(len(data) * test_ratio)
    test_indices = shuffled_indices[:test_set_size]
    train_indices = shuffled_indices[test_set_size:]
    return data.iloc[train_indices], data.iloc[test_indices]

# Randomly select 20% of the dataset as the Test Set
train_set, test_set = split_train_test(housing, 0.2)
print(len(train_set), "train +", len(test_set), "test")
```

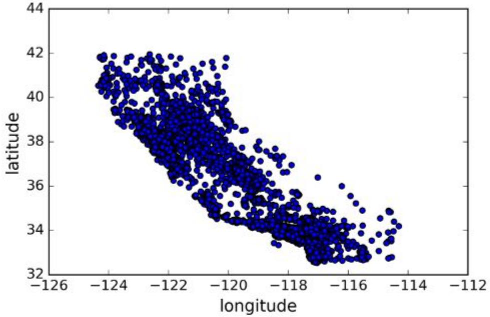
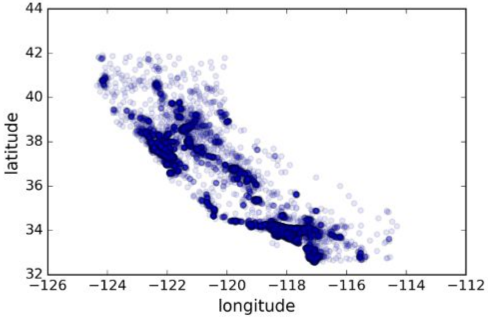
C: Discover and Visualize the Data to Gain Insights

- VISUALISE THE GRAPHICAL DATA

A. **Geographical Information (Latitude and Longitude)**

- **Population Density Information**

:

No Patterns	Has Patterns
`housing.plot(kind="scatter", x="longitude", y="latitude")` 	`housing.plot(kind="scatter", x="longitude", y="latitude", alpha=0.1)`  Note: ◦ Setting the alpha option to 0.1 makes it much easier to visualize the places where there is a high density of data points ▪ the Bay Area ▪ Los Angeles ▪ San Diego ▪ A long line of fairly high density in the Central Valley, in particular around Sacramento and Fresno. ◦ You may need to play around with visualization parameters to make the patterns stand out.

- **House Price Information**

Code	Plot

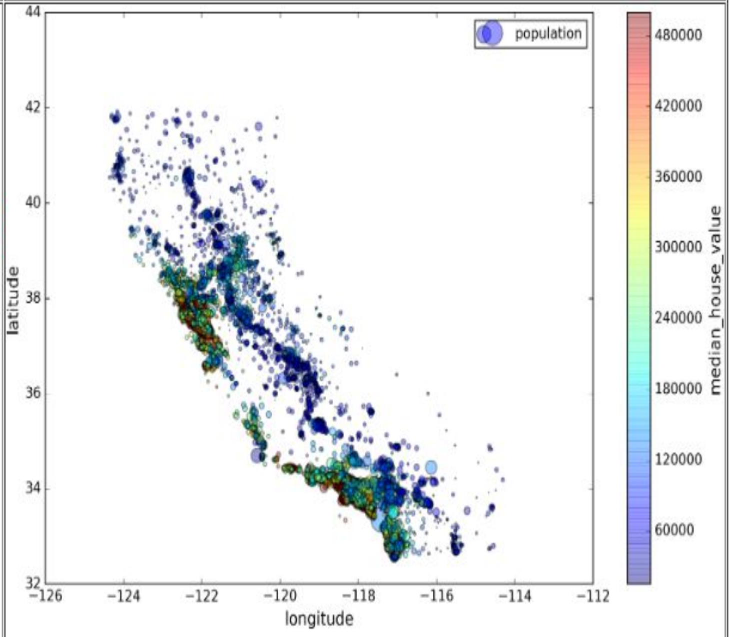
Code

```
#######################################################################
# - Option s: The radius of each circle represents the district's
#             population
# - Option c: The color represents the price
# - Option cmap: A predefined color map called jet, which ranges
#                from blue (low values) to red (high prices)
#######################################################################
housing.plot(kind="scatter", x="longitude", y="latitude", alpha=0.4,
    s=housing["population"]/100, label="population", figsize=(10,7),
    c="median_house_value", cmap=plt.get_cmap("jet"), colorbar=True,
)
plt.legend()
```

Note:

- This image tells you that the housing prices are very much related to the location (e.g., close to the ocean) and to the population density, as you probably knew already.
- It probably be useful to
  - use a clustering algorithm to detect the main clusters, and
  - add new features that measure the proximity to the cluster centers.
  - add a new feature ocean proximity
    - Although in Northern California the housing prices in coastal districts are not too high, so it is not a

Plot



● LOOKING FOR CORRELATION:

- **Approach 1: Computing Standard Correlation Coefficient to find the correlations**

Code

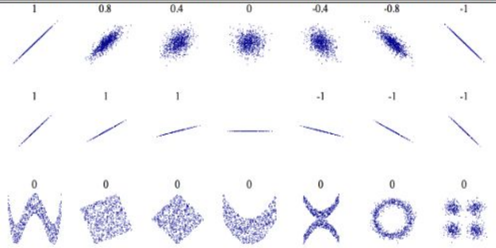
```
# Since the dataset is not too large, you can easily
# compute the standard correlation coefficient (also
# called Pearson's r) between every pair of attributes
# using the corr() method:
corr_matrix = housing.corr()

# How much each attribute correlates with the median
# house value:
>>> corr_matrix["median_house_value"].
          sort_values(ascending=False)
median_house_value    1.000000
median_income         0.687170
total_rooms           0.135231
housing_median_age    0.114220
households            0.064702
total_bedrooms        0.047865
population           -0.026699
longitude            -0.047279
latitude             -0.142826
Name: median_house_value, dtype: float64
```

Note:

- The correlation coefficient only measures linear correlations ("if *x* goes up, then *y* generally goes up/down").
  - It may completely miss out on nonlinear relationships (e.g., "if *x* is close to zero then *y* generally goes up").
  - How all the plots of the bottom row have a correlation coefficient equal to zero despite the fact that their axes are clearly not independent: these are examples of nonlinear relationships.
  - The second row shows examples where the correlation coefficient is equal to 1 or –1; notice that this has nothing to do with the slope.
    - For example, your height in inches has a correlation coefficient of 1 with your height in feet or in nanometers.

Standard correlation coefficient of various datasets (Wikipedia)



Note:

- **The correlation coefficient ranges from –1 to 1.**
  - **~ 1 ==> strong positive correlation**
    - **For example, the median house value tends to go up when the median income goes up.**
  - **~ –1 ==> strong negative correlation**
    - **For example, a small negative correlation between the latitude and the median house value (i.e., prices have a slight tendency to go down when you go north).**
  - **~0 ==> no linear correlation.**

- **Approach 2: Using Pandas' scatter_matrix to find the correlations visually.**
  - Since there are now 11 numerical attributes, you would get $11^2 = 121$ plots, which would not fit on a page, so let's just focus on a few promising attributes that seem most correlated with the median housing value (Figure 2-15):

```
from pandas.tools.plotting import scatter_matrix

attributes = ["median_house_value", "median_income", "total_rooms",
              "housing_median_age"]
scatter_matrix(housing[attributes], figsize=(12, 8))
```

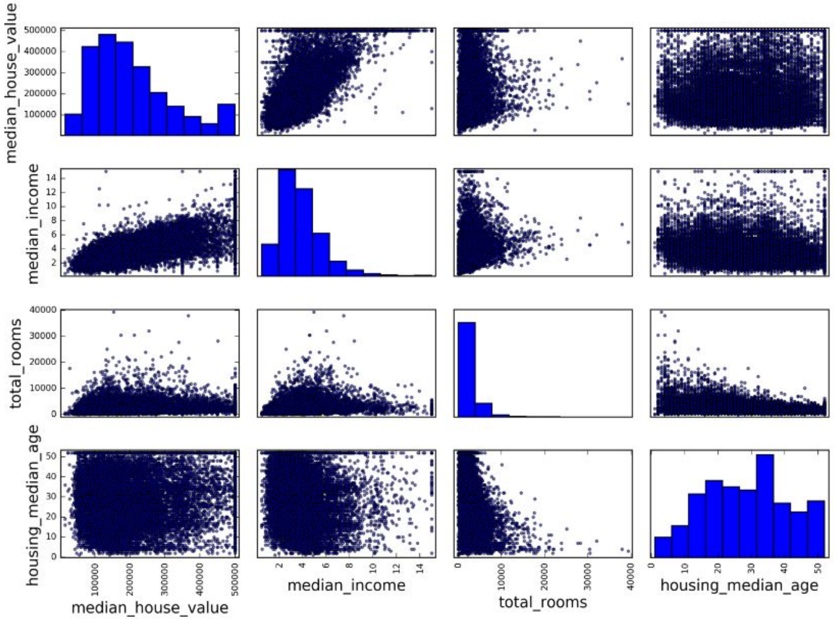


Figure 2-15. Scatter matrix

EXPERIMENT WITH ATTRIBUTE COMBINATION

- **Step 1: Review What you learned in the previous sections:**
  - Identified a few data quirks
    - You may want to clean up before feeding the data to a Machine Learning algorithm.
  - Found interesting correlations between attributes, in particular with the target attribute (i.e., median house price).
  - Found that some attributes have a tail-heavy distribution
    - So you may want to transform them (e.g., by computing their logarithm).
- **Step 2: One last thing you may want to do before actually preparing the data for Machine Learning algorithms is to try out various attribute combinations.**
  - **The total number of rooms in a district is not very useful if you don't know how many households there are.**
    - What you really want is the number of rooms per household.
      housing["rooms_per_household"] = housing["total_rooms"]/housing["households"]
  - **The total number of bedrooms by itself is not very useful:**
    - you probably want to compare it to the number of rooms.
      housing["bedrooms_per_room"] = housing["total_bedrooms"]/housing["total_rooms"]
  - **The population per household also seems like an interesting attribute combination to look at.**
- housing["population_per_household"]=housing["population"]/housing["households"]

Step 3: And now let's look at the correlation matrix again:

Before Attribute Combinations

```
>>> corr_matrix["median_house_value"].
          sort_values(ascending=False)
median_house_value     1.000000
median_income          0.687170
total_rooms            0.135231
housing_median_age     0.114220
households             0.064702
total_bedrooms         0.047865
population            -0.026699
longitude             -0.047279
latitude              -0.142826
Name: median_house_value, dtype: float64
```

After Attribute Combinations

```
>>> corr_matrix = housing.corr()
>>> corr_matrix["median_house_value"].sort_values(ascending=False)
median_house_value            1.000000
median_income                 0.687160
rooms_per_household           0.146285
total_rooms                   0.135097
housing_median_age            0.114110
households                    0.064506
total_bedrooms                0.047689
population_per_household     -0.021985
population                   -0.026920
longitude                    -0.047432
latitude                     -0.142724
bedrooms_per_room            -0.259984
Name: median_house_value, dtype: float64
```

Note:

- **The new bedrooms_per_room attribute is much more correlated with the median house value than the total number of rooms or bedrooms.**
  - Apparently houses with a lower bedroom/room ratio tend to be more expensive.
  - The number of rooms per household is also more informative than the total number of rooms in a district - obviously the larger the houses, the more expensive they are.

Step 4: Continue from Step 1 to 3

This is an iterative process:

Once you get a prototype up and running, you can analyze its output to gain more insights and come back to this exploration step.

D: PREPARE THE DATA FOR MACHINE LEARNING ALGORITHM

- ## DATA CLEANING

Fix missing features

1. **Pandas' Approach to replace the missing values of only one attribute: DataFrame's dropna(), drop(), and fillna() methods**
    - You noticed earlier that the total_bedrooms attribute has some missing values, you can fix the issue by using DataFrame's dropna(), drop(), and fillna() methods:

207 (= 20640 - 20433) districts are missing the value of total_bedrooms	Fix the missing value

<pre>housing.info()
 	
RangeIndex: 20640 entries, 0 to 20639
Data columns (total 10 columns):
longitude   20640 non-null float64
latitude   20640 non-null float64
housing_median_age   20640 non-null float64
total_rooms   20640 non-null float64
total_bedrooms   20433 non-null float64
population   20640 non-null float64
households   20640 non-null float64
median_income   20640 non-null float64
median_house_value   20640 non-null float64
ocean_proximity   20640 non-null object
dtypes: float64(9), object(1)
memory usage: 1.6+ MB</pre>
	<pre># Option 1: Get rid of the corresponding districts.
	housing.dropna(subset=["total_bedrooms"])
	
	# Option 2: Get rid of the whole attribute.
	housing.drop("total_bedrooms", axis=1)
	
	# Option 3: Set the missing values to some value (zero,
	# the mean, the median, etc.) on the training
	# set
	median = housing["total_bedrooms"].median()
	housing["total_bedrooms"].fillna(median, inplace=True)</pre>
	Note:
	▪ Don't forget to save the median value that you have computed.
	  ∘ You will need it later to replace missing values in the test set when you want to evaluate your system, and also once the system goes live to replace missing values in new data.

1. **Sciki-Learn's Approach to replace missing values of all attributes: Imputer**

Step 1: Create an Imputer instance, specifying that you want to replace each attribute's missing values with the median of that attribute:
from sklearn.preprocessing import Imputer

imputer = Imputer(strategy="median")

- Step 2: Since the median can only be computed on numerical attributes, we need to create a copy of the data without the text attribute ocean_proximity:
housing_num = housing.drop("ocean_proximity", axis=1)
- Step 3: Fit the imputer instance to the training data using the fit() method:
imputer.fit(housing_num)
- Step 4: The imputer has simply computed the median of each attribute and stored the result in its statistics_ instance variable.
  - **Only the total_bedrooms attribute had missing values, but we cannot be sure that there won't be any missing values in new data after the system goes live, so it is safer to apply the imputer to all the numerical attributes:**

>>> imputer.statistics_

array([-118.51 , 34.26 , 29. , 2119.5 , 433. , 1164. , 408. , 3.5409])

>>> housing_num.median().values

- array([-118.51 , 34.26 , 29. , 2119.5 , 433. , 1164. , 408. , 3.5409])
- Step 5: Use this "trained" imputer to transform the training set by replacing missing values by the learned medians:
X = imputer.transform(housing_num)
- Step 6: The result is a plain Numpy array containing the transformed features. If you want to put it back into a Pandas DataFrame, it's simple:
housing_tr = pd.DataFrame(X, columns=housing_num.columns)

- HANDLING TEXT AND CATEGORICAL ATTRIBUTE
- **Convert <u>text categorical</u> attribute <u>ocean_proximity</u> to be able to compute its median**
    - **Step 1: Convert from text categories to integer categories**
      Note:
        - **One issue with Categorical Value representation is that ML algorithms will assume that two nearby values are more similar than two distant values.**
    - **Step 2: Convert from integer categories to <u>One-Hot Vectors</u>**

CUSTOM TRANSFORMER

- **Although Scikit-Learn provides many useful transformers, you will need to write your own for tasks such as *<u>custom cleanup operations</u>* or *<u>combining specific attributes</u>* .**
    - **You will want your transformer to work seamlessly with Scikit-Learn functionalities (such as pipelines),**
        - **Since Scikit-Learn relies on <u>duck typing</u> (not inheritance), all you need is to create a class and implement three methods:**
            - **fit() (returning self)**
            - **transform()**
            - **fit_transform()**
                - **You can get fit_transform() for free by simply adding TransformerMixin as a base class.**
    - **Also, if you add BaseEstimator as a base class (and avoid *args and **kargs in your constructor) you will get two extra methods (get_params() and set_params()) that will be useful for automatic hyperparameter tuning.**
-

- FEATURE SCALING
- One of the most important transformations you need to apply to your data is feature scaling.
  - With few exceptions, Machine Learning algorithms don't perform well when the input numerical attributes have very different scales.
    - This is the case for the housing data:
      The total number of rooms ranges from about 6 to 39,320, while the median incomes only range from 0 to 15. Note that scaling the target values is generally not required.
- There are two common ways to get all attributes to have the same scale:
  - Min-Max Scaling
    - Min-max scaling (many people call this normalization) is quite simple:
      - values are shifted and rescaled so that they end up ranging from 0 to 1.
        - We do this by subtracting the min value and dividing by the max minus the min.
        - Scikit-Learn provides a transformer called MinMaxScaler for this. It has a feature_range hyperparameter that lets you change the range if you don't want 0–1 for some reason.
  - Standardization.
    - Scikit-Learn provides a transformer called StandardScaler for standardization.
    - Process
      - Step 1: It subtracts the mean value (so standardized values always have a zero mean).
      - Step 2: It divides by the variance so that the resulting distribution has unit variance.
    - Con: Unlike Min-Max Scaling, standardization does not bound values to a specific range, which may be a problem for some algorithms (e.g., neural networks often expect an input value ranging from 0 to 1).
    - Pro: Standardization is much less affected by outliers.
      - For example, suppose a district had a median income equal to 100 (by mistake).
        - Min-max scaling would then crush all the other values from 0–15 down to 0–0.15,
        - Standardization would not be much affected.
- As with all the transformations, it is important to fit the scalers to the training data only, not to the full dataset (including the test set).
  - Only then can you use them to transform the training set and the test set (and new data).

TRANSFORMATION PIPELINES

- Scikit-Learn provides the Pipeline class to help with sequences of transformations.

```
#################################################
# A small pipeline for the numerical attributes:
#################################################
from sklearn.pipeline import Pipeline
from sklearn.preprocessing import StandardScaler

num_pipeline = Pipeline([
        # For Data Cleaning
        ('imputer', Imputer(strategy="median")),
        # For Custom Transformer
        ('attribs_adder', CombinedAttributesAdder()),
        # For Feature Scaling
        ('std_scaler', StandardScaler()),
    ])

housing_num_tr = num_pipeline.fit_transform(housing_num)
```

Note:

- The Pipeline constructor takes a list of name/estimator pairs defining a sequence of steps.

E: SELECT AND TRAIN A MODEL

- Process
  1. Framed the problem
  2. Got the data and explored it
     - you sampled a training set and a test set
  3. Created transformation pipelines to clean up and prepare your data for Machine Learning algorithms automatically.
  4. Select and train a Machine Learning model.

- **TRAINING AND EVALUATING ON THE TRAINING SET**

Option 1: Linear Regression

Option 2: Decision Tree

Better Evaluation Using Cross-Validation - Paramter vs. Hyperparamter

- Use Overfitting To Evaluate Different Models - extra
- KNN Cross-Validation - including K-Fold cross-validation
- Cross-Validation — Introduction

F: Fine-Tune Your Model

Let’s assume that you now have a shortlist of promising models. You now need to fine-tune them. Let’s look at a few ways you can do that.

1. Find hyperparameter values
    1. Grid Search vs. Random Search
        1. Grid Search
            - Confusion Matrix - extra
        2. Randomized Search
    2. Ensemble Methods
2. Analyze the Best Models and Their Errors
3. Evaluate Your System on the Test Set

G: Launch Monitor and Maintain Your System

After getting approval to launch

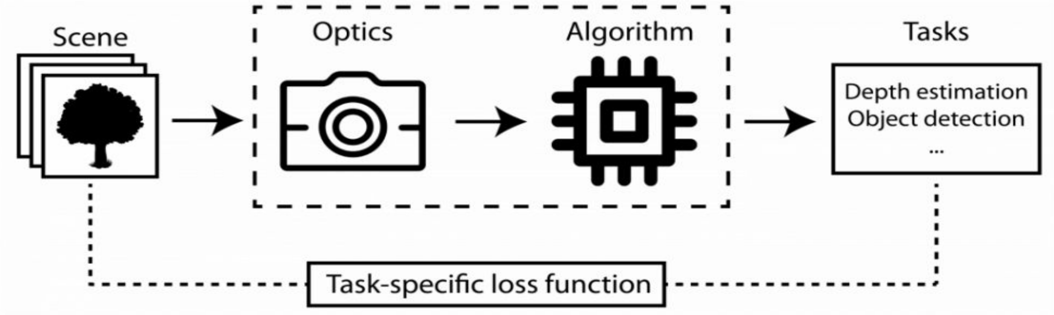
- **Step 1: You need to get your solution ready for production.**
- **Step 2: Write monitoring code to check your system's live performance at regular intervals and trigger alerts when it drops.**
  - **May need to get assistance from field experts.**
- **Step 3: You should also make sure you evaluate the system's input data quality.**

CONCLUSION:

End-to-end is indisputably a great tool for solving elaborate tasks. The idea of using a single model that can specialize to predict the outputs directly from the inputs allows the development of otherwise extremely complex systems that can be considered state-of-the-art. However, every enhancement comes with a price: while consecrated in the academic field, the industry is still reluctant to use E2E to solve its problems due to the need for a large amount of training data and the difficulty of validation.

ENHANCEMENT:

End-to-End Optimization, Computer Vision and Machine Learning



Computational imaging systems involve both optics and algorithm designs. Instead of optimizing these two components separately and sequentially, we treat the entire system as one neural network and develop an end-to-end optimization framework. Specifically, the first layer of the network corresponds to physical optical elements, and all subsequent layers represent the computational algorithm. All the parameters are learned based on task-specific loss over a large dataset. Such a learning-based framework can potentially go beyond the limits imposed by model-based methods and create better sensor systems.

REFERENCES:

Chapter 2 – End-to-end Machine Learning project

Google Slides presentation to the GitHub.

Labs step-by-step by going through the process of each section

THANK YOU