End to End Machine Learning Project

Eng. Mohammad R. KATBY

Agenda

build intuition about machine learning	Types of Machine Learning
	Build first ML Model
	Main Challenges
	Life Satisfaction
Machine Learning Project	Look at the big picture
	Get the data
	Discover and visualize the data to gain insights
	Prepare the data for Machine Learning Algorithms
	Select a model and train it
	Fine-tune your model
	Present your solution
	Lunch, monitor, and maintain your system

Mohammad Reda Adnan KATBY

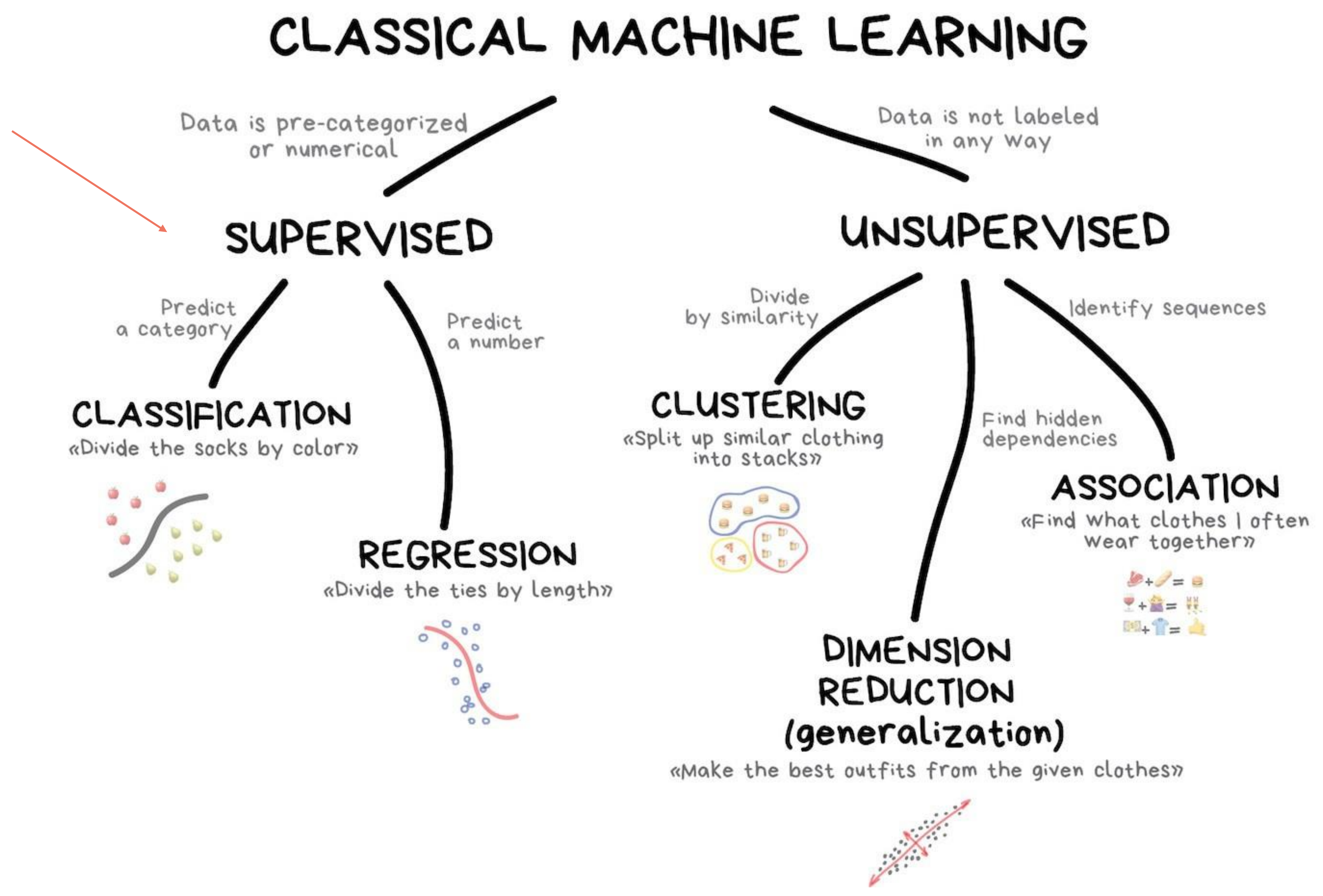
- Technology, Principal Advisor
- 10+ Years experience is software field
- MIT Artificial Intelligence Certified
- Aleppo University Informatics Engineering

Session Objects

- To simplify how Machine Learning works
- To familiarize with Machine Learning Project Checklist
- To learn how to study data from Data Scientist point of view
- To build Machine Learning Model

Types of Machine Learning

Classical Machine Learning - Supervised


CLASSICAL MACHINE LEARNING
Data is pre-categorized or numerical
Data is not labeled in any way
SUPERVISED
UNSUPERVISED
Predict a category
Predict a number
Divide by similarity
Identify sequences
Find hidden dependencies
CLASSIFICATION
«Divide the socks by color»
REGRESSION
«Divide the ties by length»
CLUSTERING
«Split up similar clothing into stacks»
ASSOCIATION
«Find what clothes I often wear together»
DIMENSION REDUCTION (generalization)
«Make the best outfits from the given clothes»

تعلم الآلة للجميع

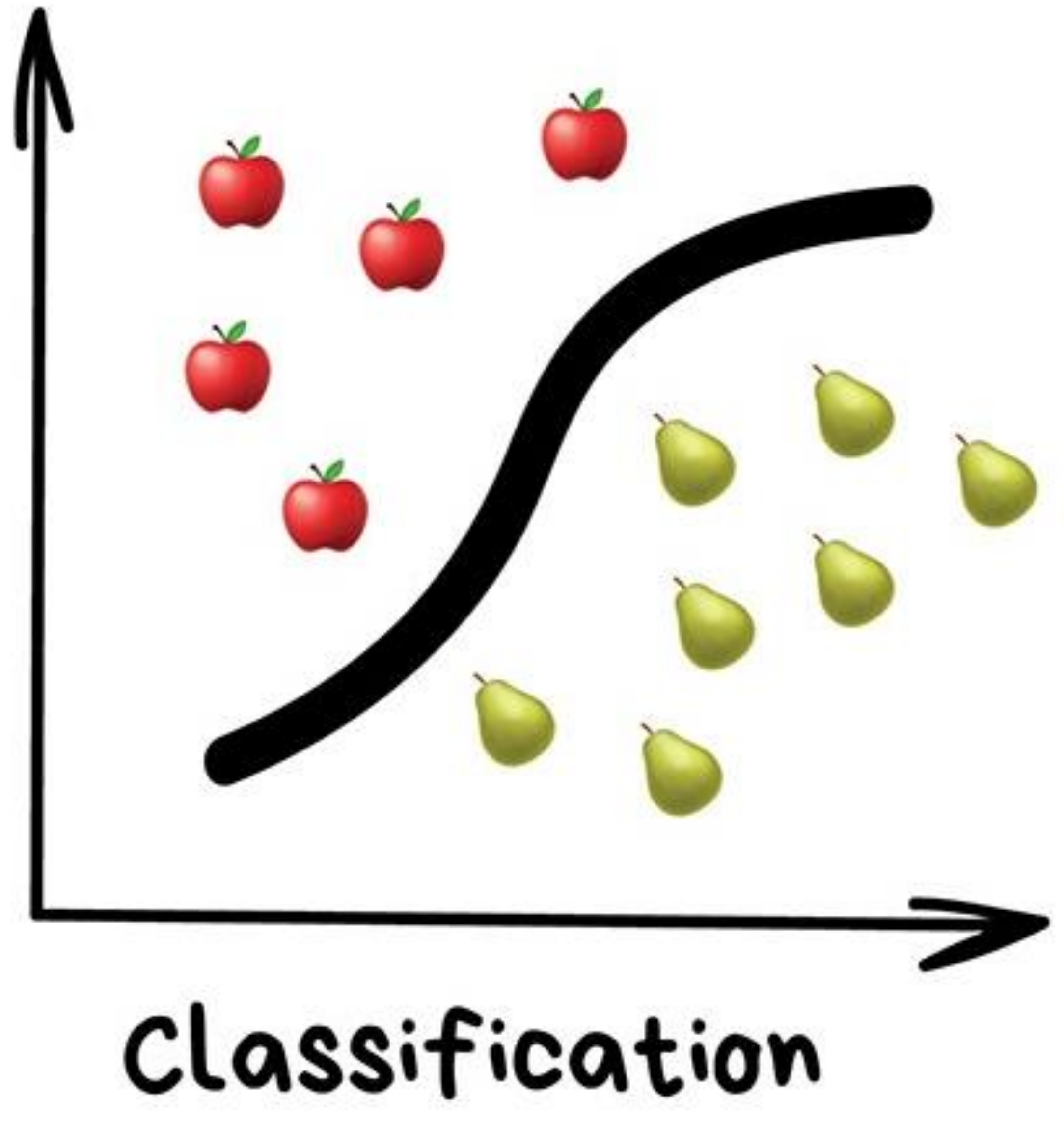
Supervised – Classifications

"Splits objects based at one of the attributes known beforehand. Separate socks based on color, documents based on language, music by genre"

Today used for:

- Spam filtering
- Language detection
- A search of similar documents
- Sentiment analysis
- Recognition of handwritten characters and numbers
- Fraud detection

Popular algorithms: Naive Bayes, Decision Tree, Logistic Regression, K-Nearest Neighbours, Support Vector Machine



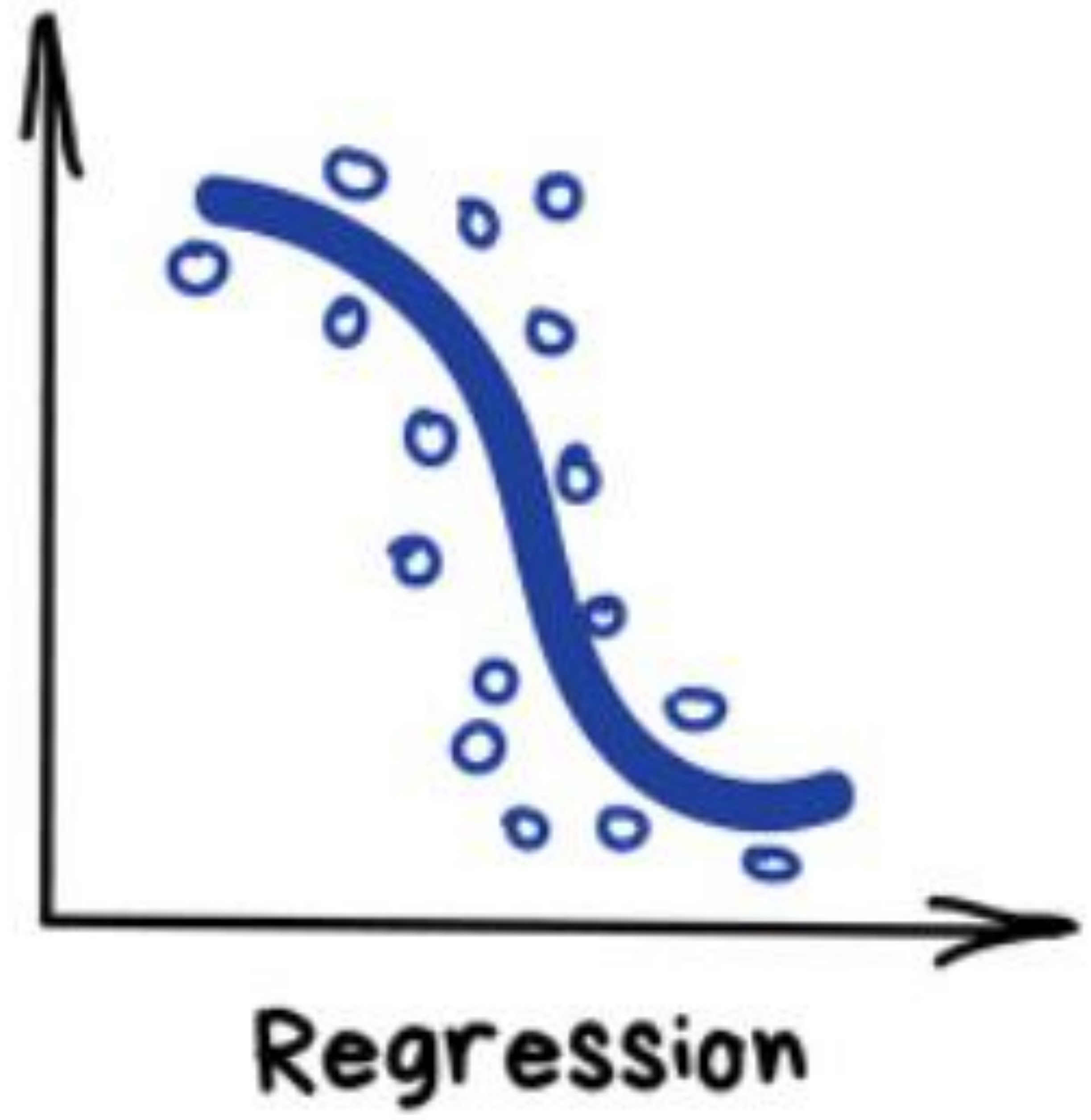
Supervised – Regression

"Draw a line through these dots. Yep, that's the machine learning"

Today used for:

- Stock price forecasts
- Demand and sales volume analysis
- Medical diagnosis
- Any number-time correlations

Popular algorithms: Linear, and Polynomial



Main Challenges of Machine Learning

Bad Data

Bad Algorithms

Insufficient Quantity of Training Data

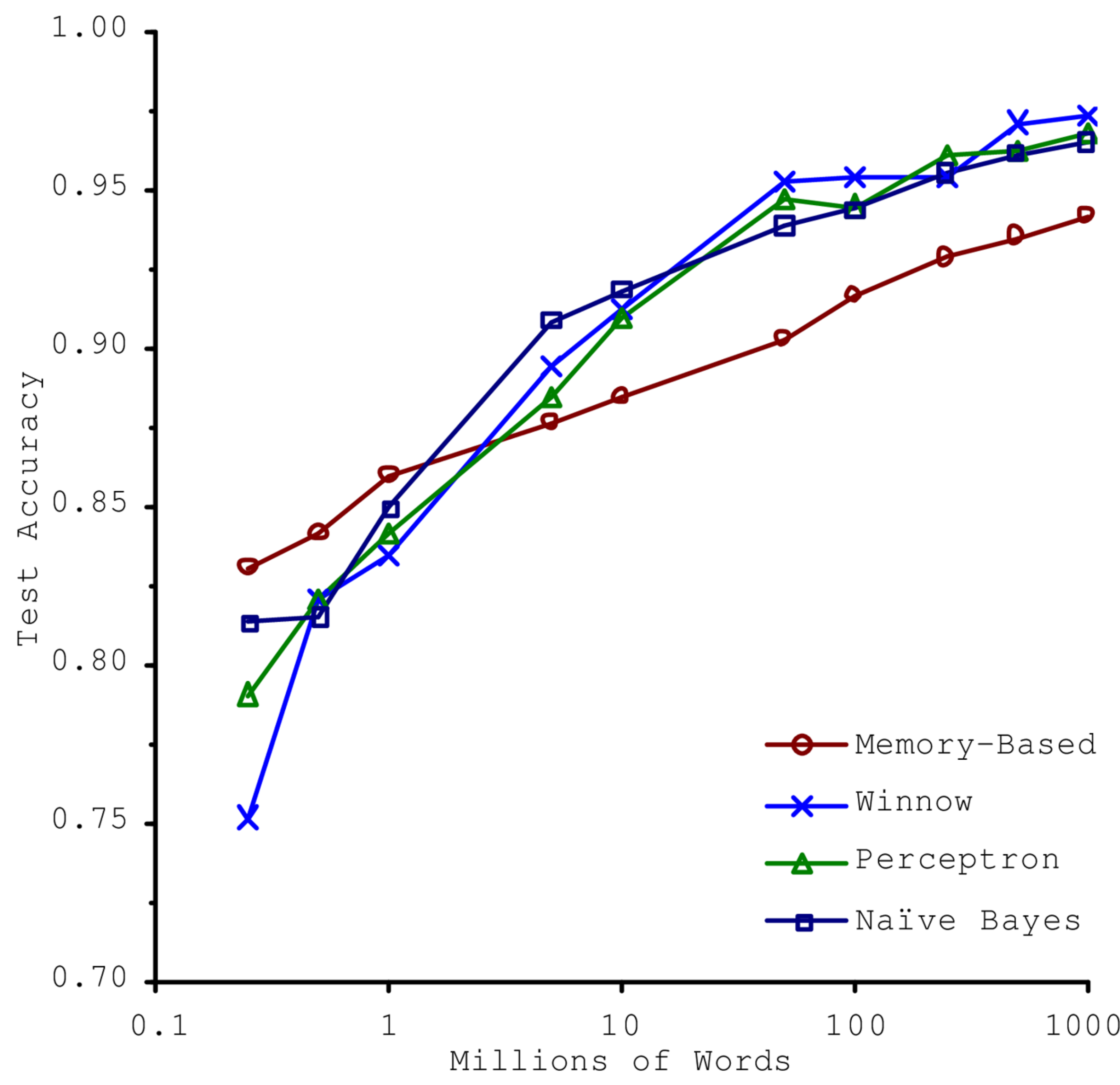
Bad Data

- To learn a baby what an apple is, all you need is to point to an apply and repeat apple few times.
- Now baby able to recognize apple in sizes, colors and shapes **GENIUS**

- **Machine Learning** is not there yet.
- For complex problem such as **image or speech recognition** you may need **millions** of examples

The Unreasonable Effectiveness of Data

- In 2001, Microsoft researchers showed that algorithms performed almost identically well on a complex problem of natural language disambiguation once they were given enough data
- results suggest that time and money should be spent on developing data corpus instead of algorithm



Nonrepresentative Training Data

Bad Data

- It is crucial to use a training set that is representative of the cases you want to generalize to. This is often harder than it sounds:
- if the sample is too small, you will have *sampling noise*
- if the sample is too large, samples can be nonrepresentative if the sampling method is flawed. This is called *sampling bias*.

Landon against Roosevelt US election 1936

- Analysis company predicted high confidence that Landon would get 57% of the votes.
- But Roosevelt won with 62% of the votes. The flaw was in sampling method:

1. The company used telephone directories, list of magazine subscribers, club membership lists etc., and all of these lists tend to favor wealthier people who most likely vote to Republican (Landon)
2. Less than 25% of people received the poll answered. Sampling bias

Poor-Quality Data

Bad Data

if your training data is full of errors, outliers, and noise, it will be harder for the system to detect patterns, so your system is less likely to perform well.

The truth is most data scientists spend a significant part of their time cleaning up training data set

1. Clear, Discard or fix the errors of outliers' row.
2. When some rows missing up some features. You can Ignore, attributes or rows, fill in missing values, traning multiple models with and without missing features

Irrelevant Features

Bad Data

Your system will only be capable of learning if the training data contains enough relevant features and not too many irrelevant ones.

A critical part of the success of a Machine Learning project is coming up with a good set of features to train on

This process, called feature engineering

1. **Feature selection**: selecting the most useful features to train on among existing features.
2. **Feature extraction**: combining existing features to produce a more useful one
3. **Creating new features** by gathering new data

Overfitting the Training Data

Bad Algorithm

Say you are visiting a foreign country and the taxi driver rips you off.

Overfitting: it means that the model performs well on the training data, but it does not generalize well.

Happens when the model is too complex relative to the amount and noisiness of the training data.

The possible solutions are:

1. To simplify the model (Reducing the number of attributes in the training data or by constraining the model)
2. To gather more training data
3. To reduce the noise in the training data (e.g., fix data errors and remove outliers)

Underfitting the Training Data

Bad Algorithm

Is the opposite of overfitting: it occurs when your model is too simple to learn the underlying structure of the data.

Reality is just more complex than the model, so its predictions are bound to be inaccurate, even on the training examples.

The main options to fix this problem are

1. Selecting a more powerful model, with more parameters
2. Feeding better features to the learning algorithm (feature engineering)
3. Reducing the constraints on the model (e.g., reducing the regularization hyperparameter)

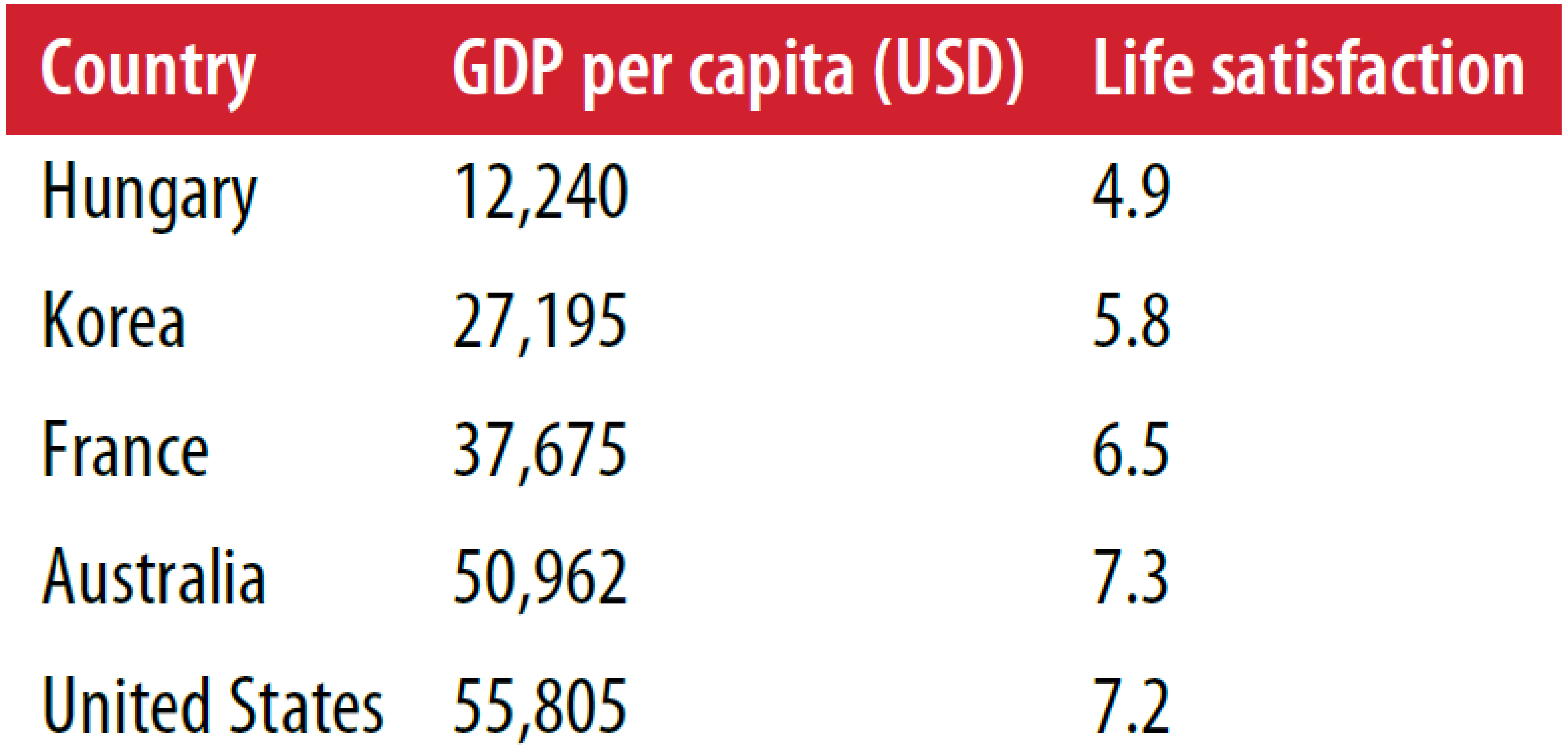
Do you think money makes people happy?

Life Satisfaction

1-Get Data

1. *Better Life Index*: OECD's WebSite will get "Life Satisfaction"
2. *GDP per Capita*: IMF's WebSite will get "GDP per capita USD"

Join the tables and sort by GDP per capita

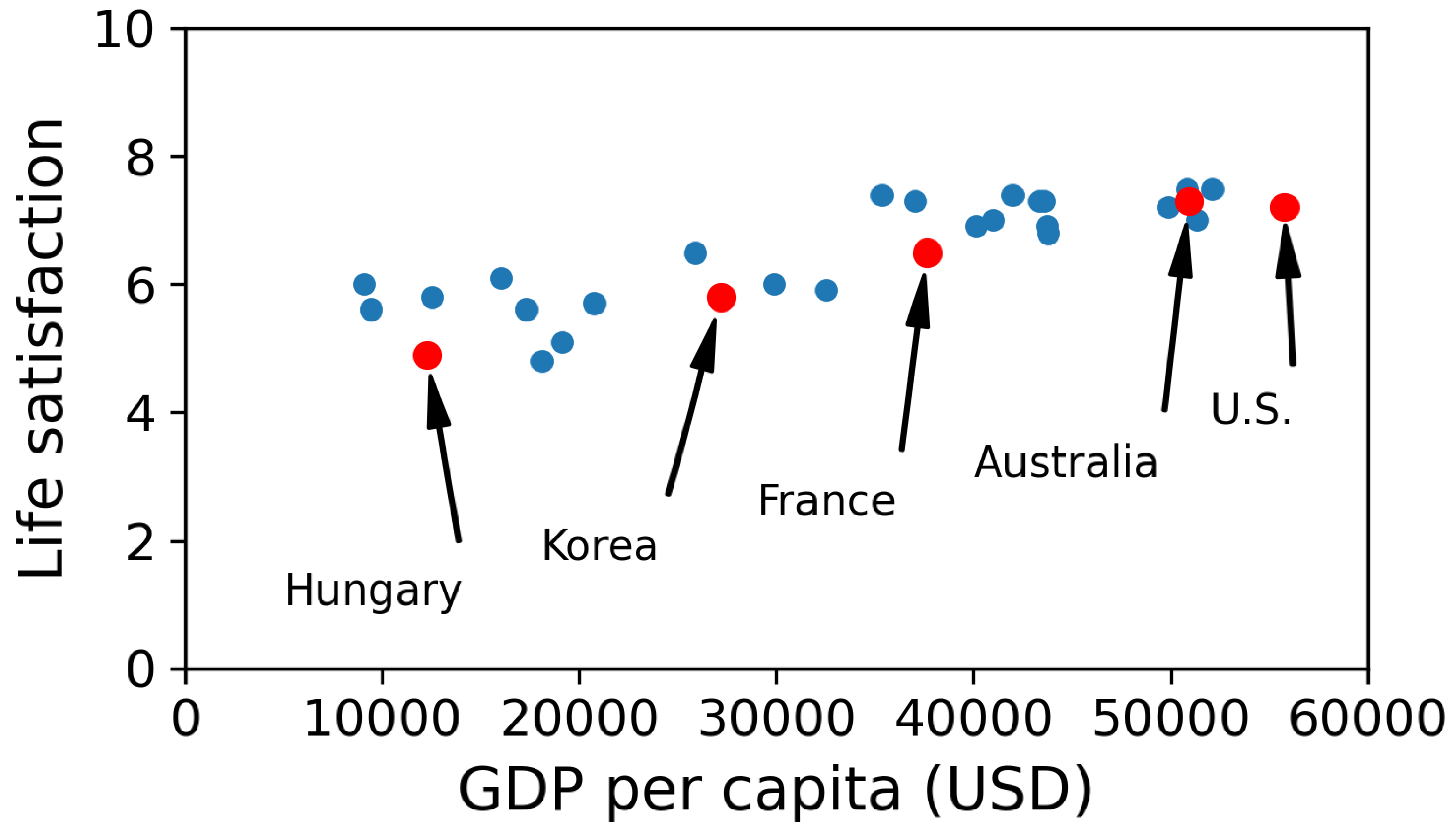


Country	GDP per capita (USD)	Life satisfaction
Hungary	12,240	4.9
Korea	27,195	5.8
France	37,675	6.5
Australia	50,962	7.3
United States	55,805	7.2

2-Model Selection

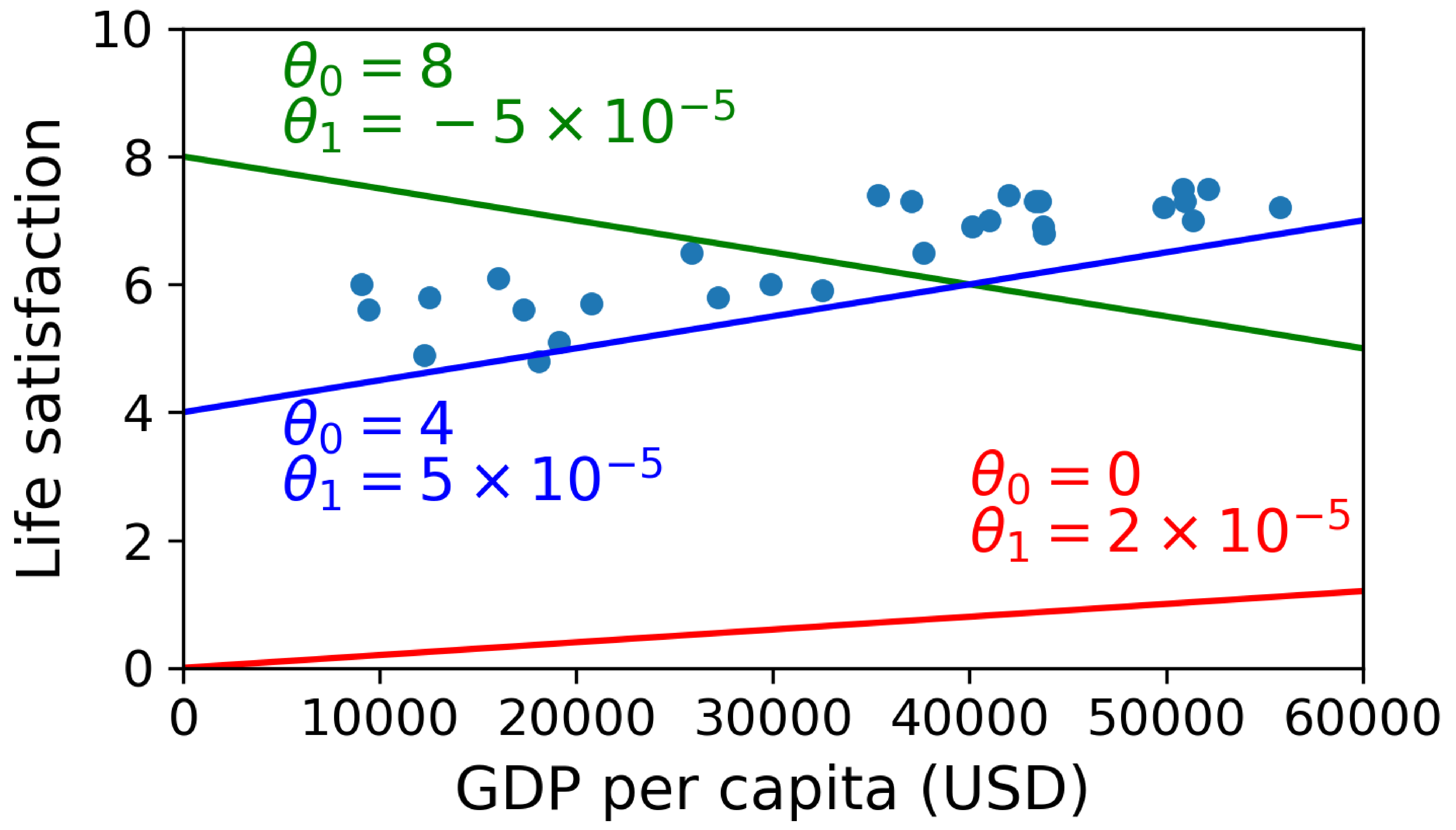
- Do you see the Trend?
  Life satisfaction goes up more or less linearly as the country's GDP per capita increases.
- We can **model** life satisfaction as a **linear function** of GDP per capita
- *Model Selection*: we select **linear model** of life satisfaction with just one attribute (*feature*)

$$life\_satisfaction = \theta_0 + \theta_1 \times GDP\_per\_capita$$



2-Model Selection

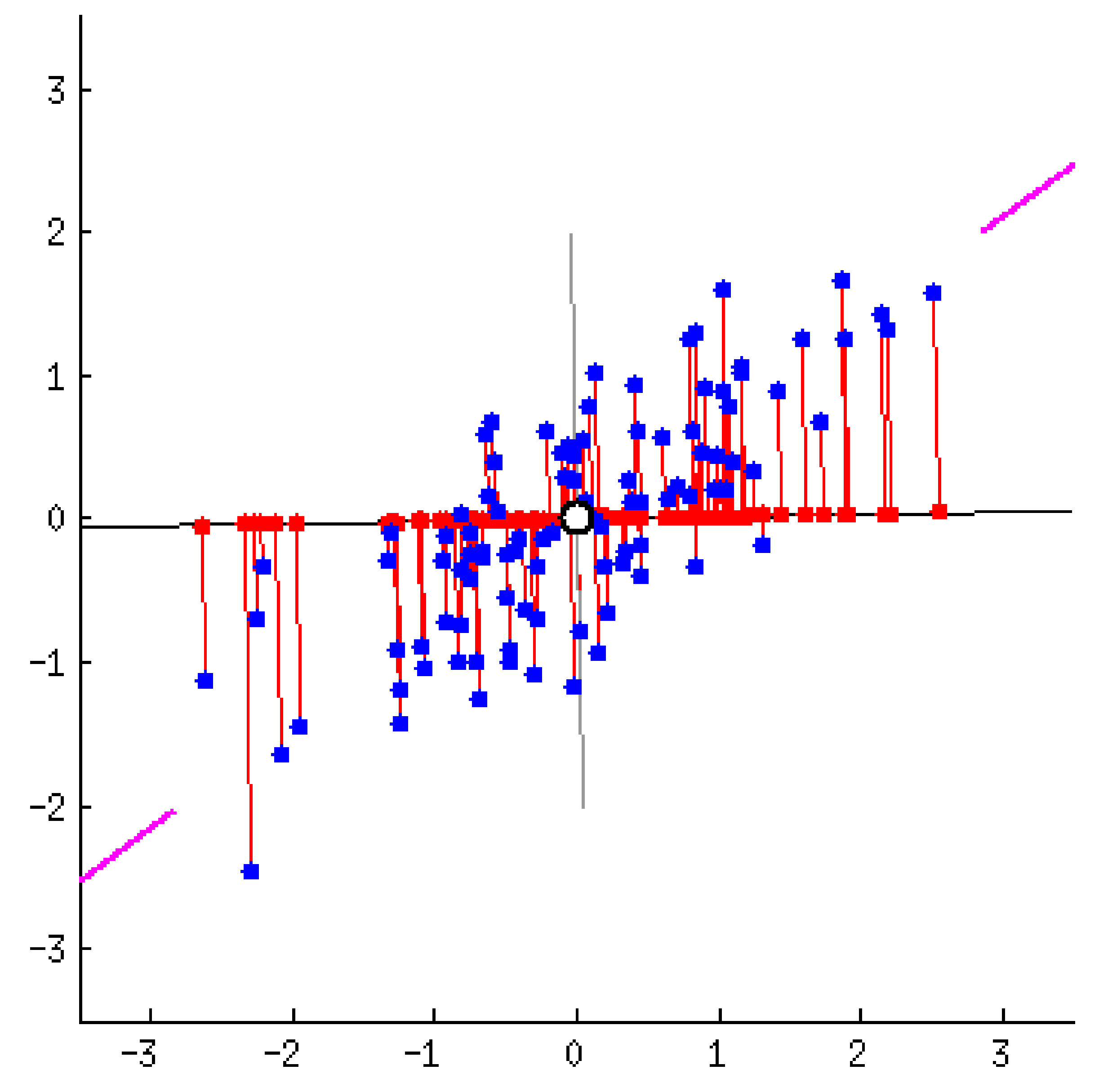
- 2 Model Parameters θ_0 θ_1
- By tweaking these parameters, we can make our model represent any linear function



3-Training

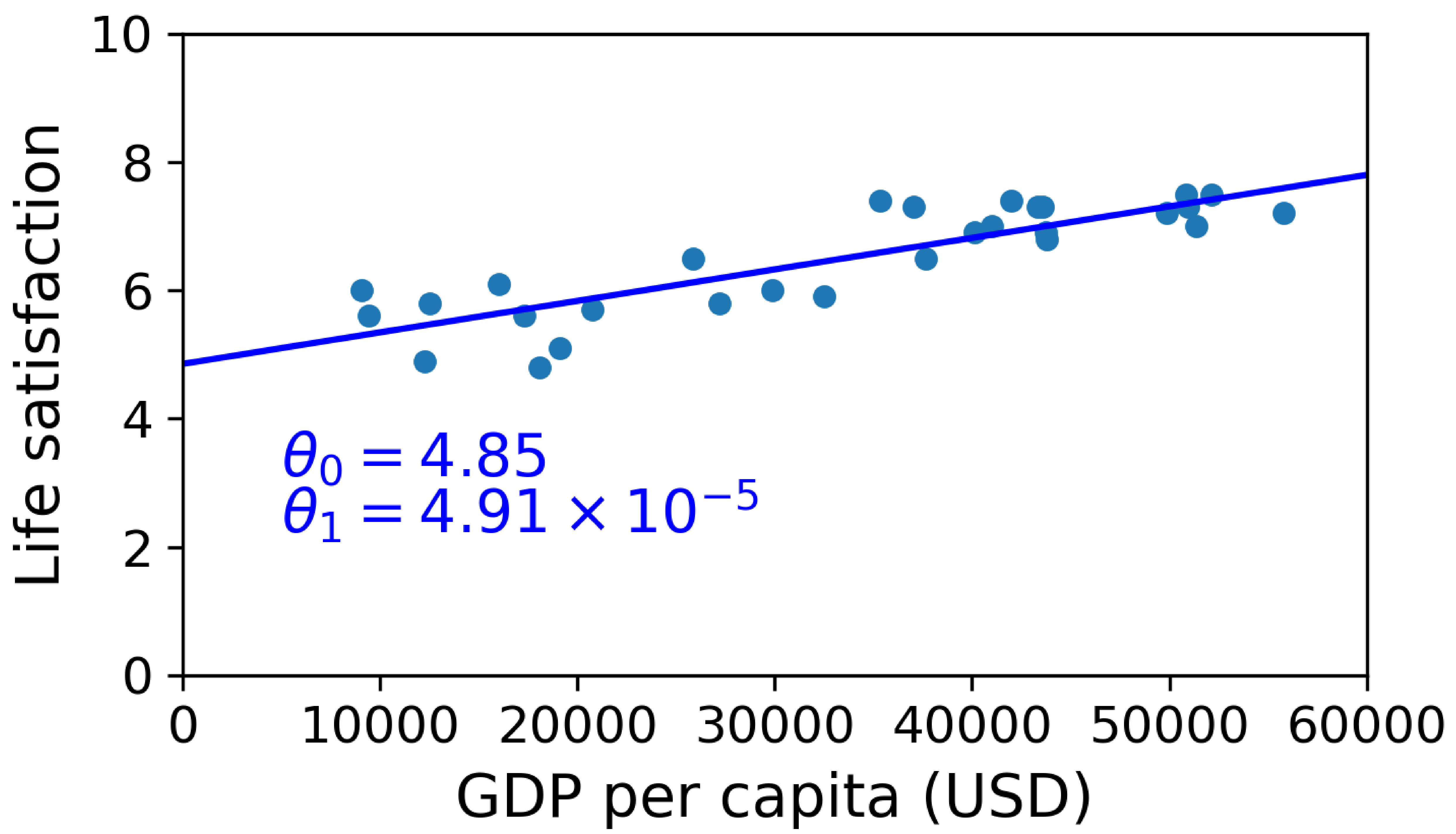
1. Utility (Fitness) Function: measure how *good* your model is
2. Cost Function: measure how *bad* your model is
   - Measure the distance between model prediction and training example

- The objective is to *minimize this distance*
- Where Linear Regression Algorithm Comes
- We feed it (x, y) training set, and it finds θ_0 θ_1 that make linear model fit best to your data
- This process called *Training*



3-Training

- $\theta_0 = 4.85$ $\theta_1 = 4.91 \times 10^{-5}$ Are the best values in our case
- Which make the Linear Model Fit best to our data



4-Prediction

- How happy Cypriots are? (OECD data does not have the answer)
- We can use our model
  1. Look up Cyprus's GDP per capita, find $22,587
  2. Apply your model $4.85 + 22587 \times 4.91 \times 10^{-5} = 5.96$

Demo

Life Satisfaction

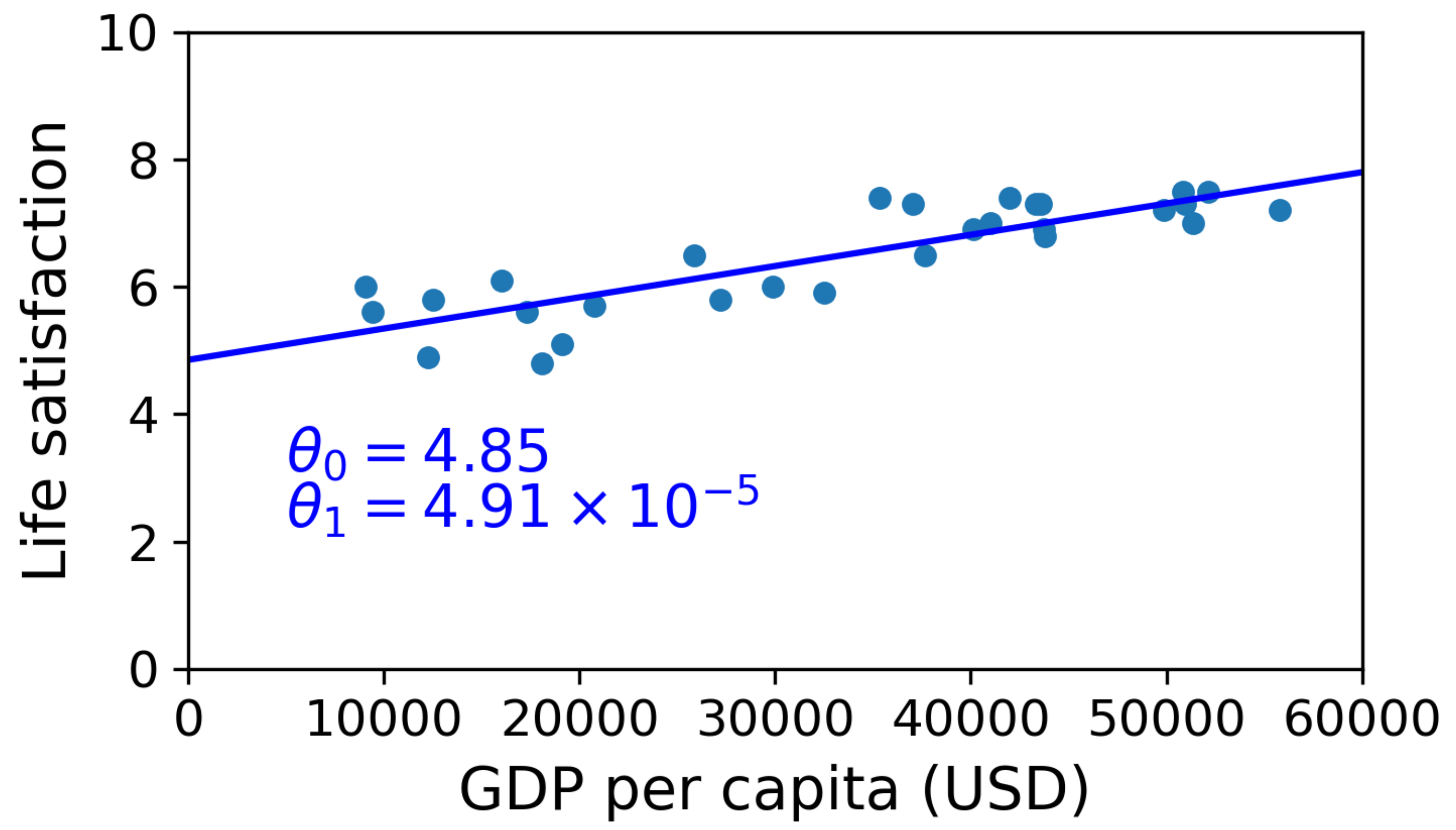
Alternate Algorithm

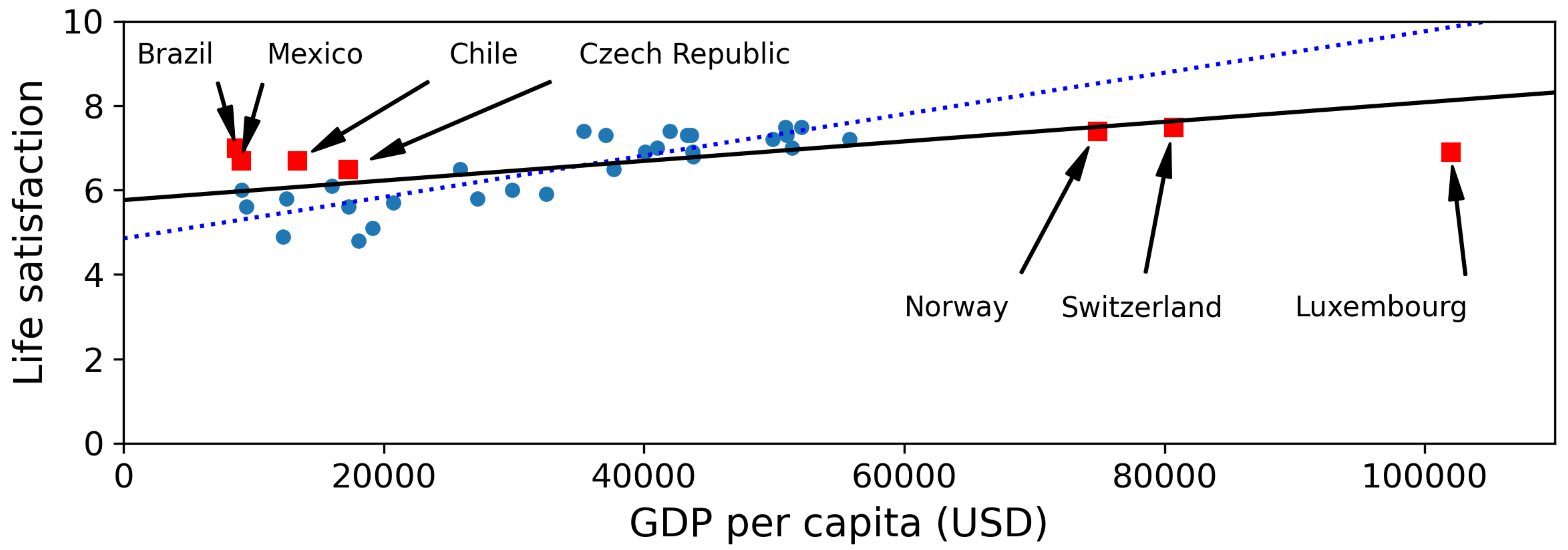
- Slovenia (20732) has closest GDP per capita to Cyprus (22587)
- Slovenia Life Satisfaction is 5.7 then KNeighborsRegressor will predict the same to Cypurs
    - Portugal = 5.1
    - Spin = 6.5
    - Average = 5.77

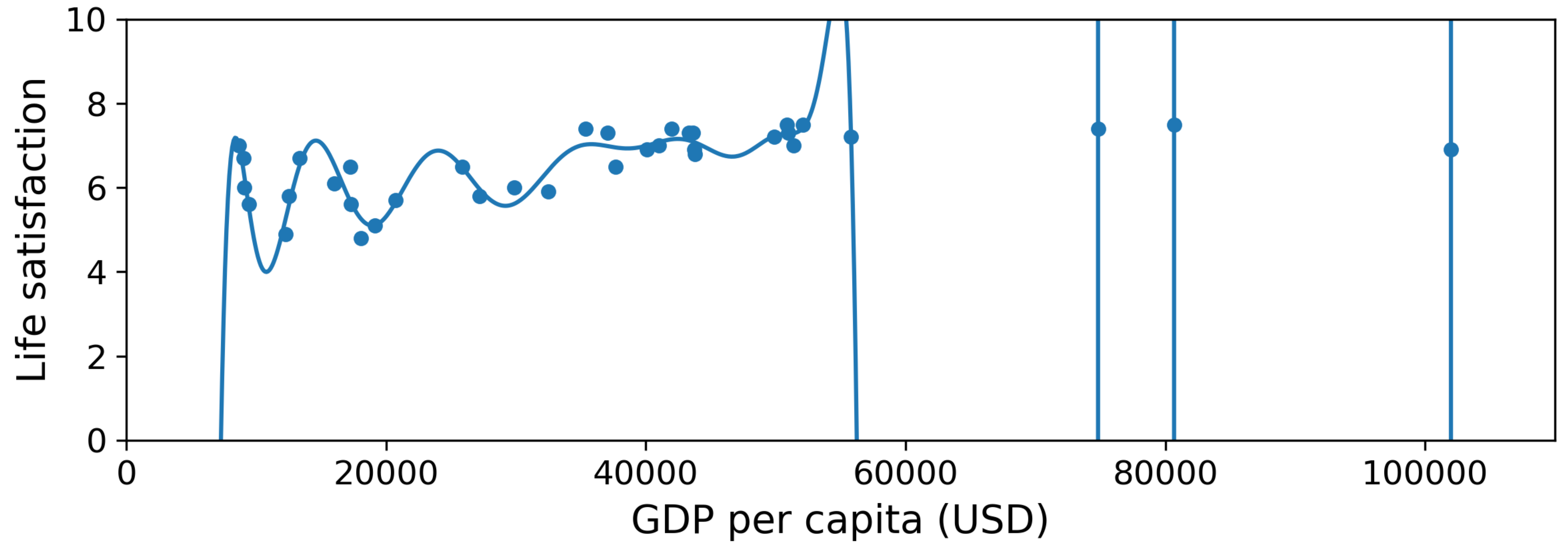
- Model Based Algorithm (`LinearRegression`) Predict 5.96
- Instance Based Algorithm (`KNeighborsRegressor`) Predict 5.77

How to evaluate the performance (accuracy) of our model?

Plot Missing Data

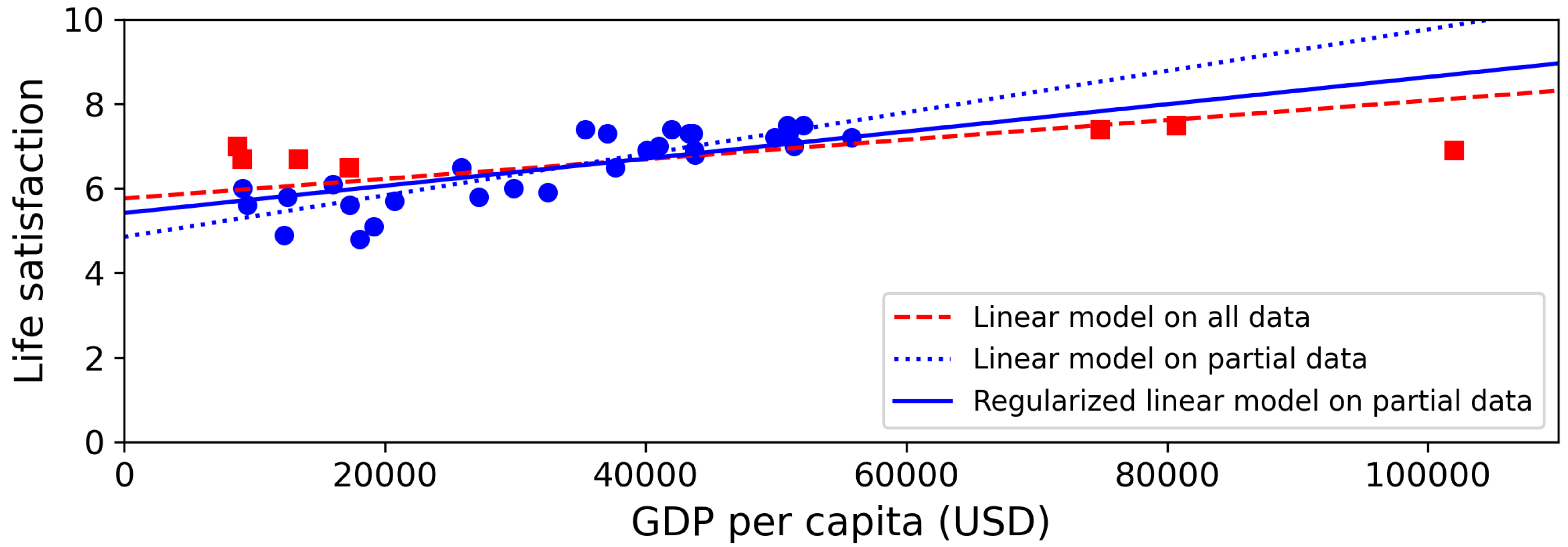






Regularization

- Constraining a model to make it simpler => Reduce the risk of overfitting
- Linear model has 2 degree of freedom θ_0 θ_1
- If we force $\theta_1 = 0$ what will happen?
- If we force $\theta_1 = small\ values$ what will happen?



Regularization

- Amount of regularization applied during training controlled by *hyperparameter*
- Hyperparameter is learning algorithm parameter (not model parameter)
- Tuning hyperparameter is important part of building Machine Learning System

Testing and Validating

How to know how our model works on new cases that are not seen before?

- The better way is to split data into *training set* and *test set*.
- By Evaluate the mode on test set we got Error rate that called ***generalization error*** or ***out-of-sample error***
- This value tell you how well your model will perform on cases that are never seen before

Hyperparameter tuning

- If you hesitate between two models (Linear and Polynomial)
    - Calculate Generalization Error from test set
- If you choose Linear Model but you want to apply some Regularization?
    - Calculate Generalization Error from test set gives you 5% however on production give you 15%
    - This because Hyperparameter adjusted based on test set

- Holdout Validation: split you training data and keep validation set away

End-to-End Machine Learning Project

Real Estate Company, District house Pricing

Machine Learning Project Main Steps

Pretending to be a recently hired as a Data Scientist, here are main step you will go through

1. Look at the big picture.
2. Get the data.
3. Discover and visualize the data to gain insights.
4. Prepare the data for Machine Learning algorithms.
5. Select a model and train it.
6. Fine-tune your model.
7. Present your solution.
8. Launch, monitor, and maintain your system.

Machine Learning Project Check

Working with Real Data

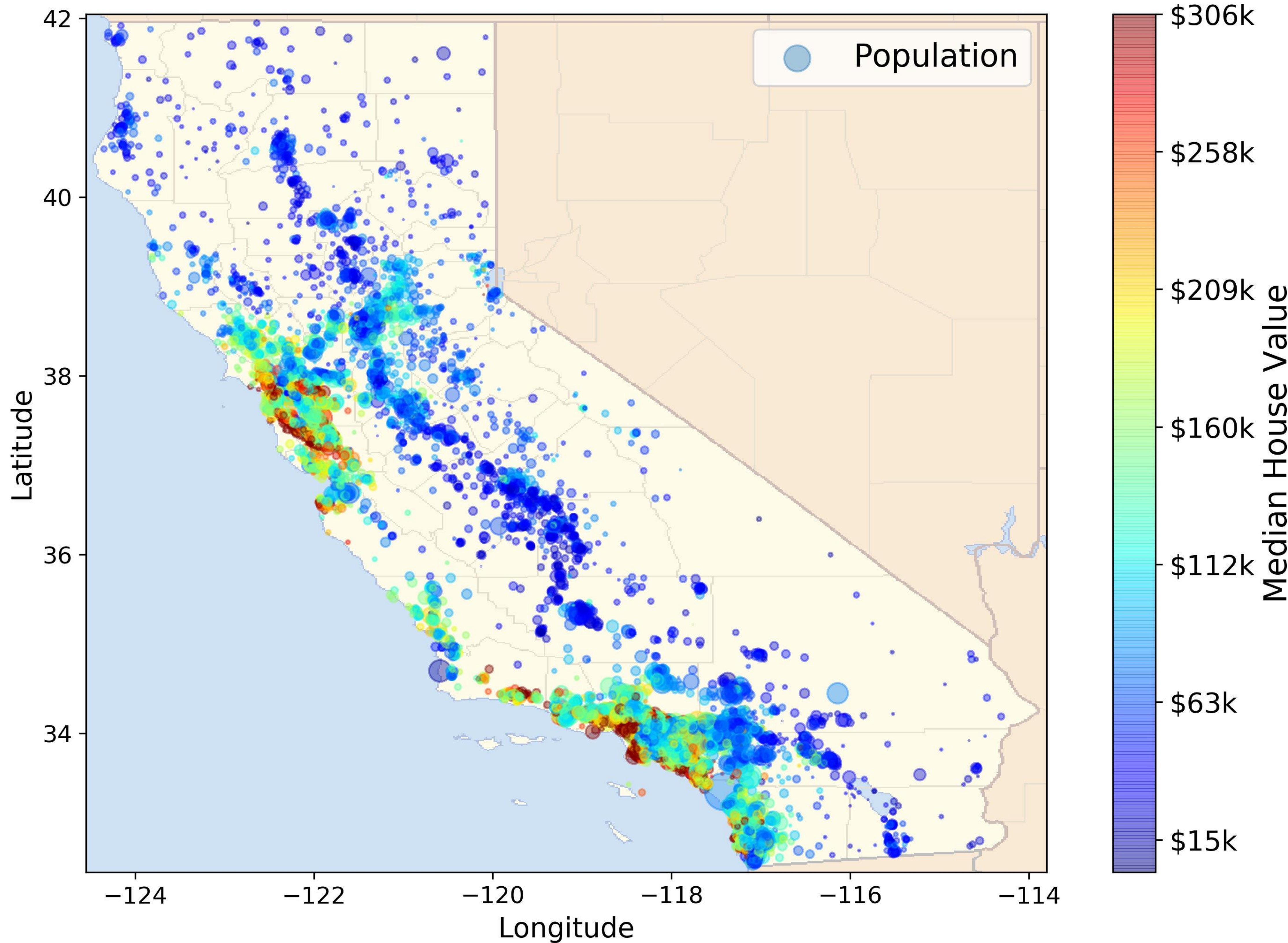
It is best to experiment with real-world data, not just artificial datasets.

- Popular open data repositories:
  - UC Irvine Machine Learning Repository http://archive.ics.uci.edu/ml
  - Kaggle datasets https://www.kaggle.com/datasets
  - Amazon's AWS datasets https://registry.opendata.aws/
- Meta portals (they list open data repositories):
  - http://dataportals.org/
  - https://opendatamonitor.eu/
  - https://data.nasdaq.com/
- Other pages listing many popular open data repositories:
  - Wikipedia's List of ML Dataset https://homl.info/9
  - Quora.com question https://homl.info/10
  - Reddit https://www.reddit.com/r/datasets

Project Dataset

California housing prices

- Based on data from the 1990 California census
- Some attributes are added or removed for teaching purposes
- Has 10 Columns
    1. Longitude and Latitude: geographical location
    2. housingMedianAge: lower number is newer building
    3. totalRooms: within a block
    4. totalBedrooms
    5. population
    6. households
    7. medianIncome: tens of thousands of USD for households
    8. medianHouseValue: in USD
    9. oceanProximity: Location of the house w.r.t ocean/sea



1. Look at the Big Picture

Frame the Problem

Select a Performance Measure

Check the Assumptions

Look at the Big Picture

- You are asked to build a model of housing prices in California using the California census data
- This data has metrics such as the population, median income, median housing price, etc. for each block group in California.
- Block group is smallest geographical unit, typically has a population of 600 to 3,000 people called **district**

Your model should learn from this data and be able to predict the median housing price in any district, given all the other metrics.

Look at the Big Picture

Frame the Problem

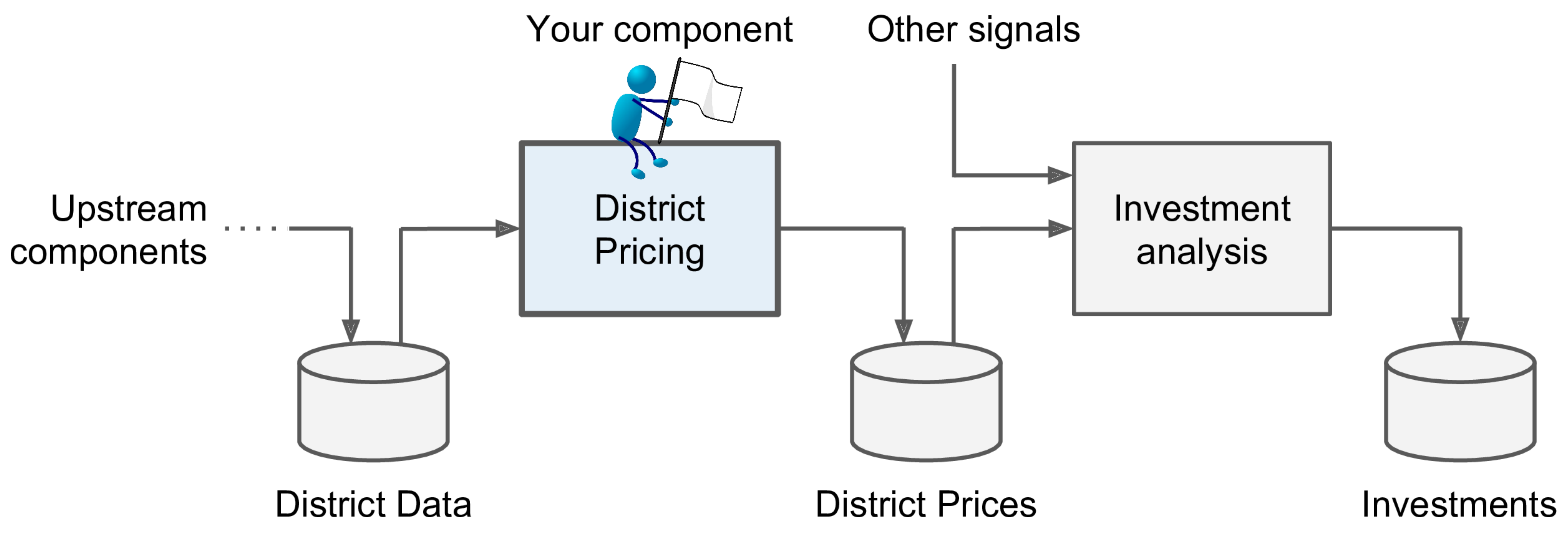
You need to ask investigative question
1. What is business objective?
2. How the company expect to use and benefit from this model?
3. What current solution look like (if any)?

These questions are import Because
1. It will determine how you frame the problem
2. What algorithm you will select
3. What performance measure your will use to evaluate your model
4. How much effort you should spend tweaking it
5. Give performance measure, and insight to solve the problem

Look at the Big Picture

Frame the Problem

1. Your model will be fed to another machine learning system along with many other signals
2. The downstream system will determine whether it is worth investing in a given area or no. It is directly affecting revenue



Look at the Big Picture

Frame the Problem

1. What kind of machine learning algorithm type?
    1. Supervised, Unsupervised, or Reinforcement?
    2. Classification, Regression, or?
    3. Batch learning, Online Learning

- Data is labeled (Supervised)
- Prediction is a value (Regression)
- Multi-regression Problem (more than one feature as input X)
- Univariate regression (only one output Y)

Look at the Big Picture

Select a Performance Measure

1. How much error the system typically makes in its predictions (Cost Function)
   1. Higher weight for large errors
   2. Lower weight for small errors
2. Error: $h\left(x^{(i)}\right) - y^{(i)}$
3. Square Error: $(h(x^{(i)}) - y^{(i)})^2$
4. Mean Square Error: $\frac{1}{m}\sum_{m}^{1}(h\left(x^{(i)}\right) - y^{(i)})^2$
5. Root Mean Square Error: $RMSE(X,h) = \sqrt{\frac{1}{m}\sum_{m}^{1}(h(x^{(i)}) - y^{(i)})^2}$

m: number of records
$x^{(i)}$: vector of all features values without label
X: matrix of all features of all records without label
h: your system's prediction function hypothesis

Look at the Big Picture

Check assumptions

1. List assumptions you made so far
2. Verify assumptions if possible

Example

1. What if downstream system want the output as (Cheap, Medium, Expensive)

2. Get the Data

Create the Workspace

Download the Data

Take a Quick Look at the Data Structure

Create a Test Set

Get the data

Create Isolated Environment

- `conda create --name learning`
- `conda info --envs`
- `conda activate learning`
- `conda list`
- `conda activate base`
- `conda env remove -n learning`
- `conda install jupyter matplotlib numpy pandas scipy scikit-learn`

Coding

1. Function of download, extract data
2. Create DataFrame for the CSV

Housing Dataset URL

Get the Data

Take a quick Look at the Data Structure

1. Take a look at the top 5 rows `housing.head()`
2. Get quick description of the data `housing.info()`
3. Explore Categorical attributes
   `housing["ocean_proximity"].value_counts()`

Get the Data

Take a quick Look at the Data Structure

4. Explore Numerical attributes `housing.describe()`

- std: measures how dispersed the values are
  - it is root of the variance, which is the average of the squared deviation from the mean
  - when feature has bell-shaped "68, 95, 99.7" rule applies
  - mean += std, mean += 2std, mean += 3std
- percentiles: indicates the value below a given percentage of observations

5. Plot histogram for numerical attributes

- x: given value range
- y: number of instances
- Plot histogram for specific attribute housing['median_house_value'].hist(bins=50, figsize=(20,15))
- Plot histogram for each numerical attribute housing.hist(bins=50, figsize=(20,15))

Get the Data

Finding notes

1. median_income attribute is
    - scaled
        - at 10,000 of USD
    - capped
        - 15 for higher median incomes (15.0001)
        - 0.49 for lower median incomes (0.49999)
2. housing_median_age, and median_house_value are also capped
    - Machine Learning algorithm may learn that prices never go beyond that limit
    - If this is a problem, you have 2 options
        - Collect proper labels for districts whose labels were capped
        - Remove those districts from the training, and testing sets

Get the Data

Finding notes

3. Attributes have very different scales; we will explore feature scaling later.
4. Many histograms are tail heavy, we will try transforming these attributes later to have more bell-shaped distributions as this shape make it harder for machine learning algorithm to detect patterns.

Hopefully, you now have a better understanding of data, **and that enough for now**,

We will **create a test set**, **put it aside** and **never look** at it.

Get the Data

Create a Test Set

Data Snooping Bias

- Your brain is an amazing pattern detection system, it is highly prone to overfitting: if you look at the test set
- You may see interesting pattern in the test data that leads you to select specific model
- Then Generalization Error of Test set will be too optimistic

That's why we need to separate test set aside even before have deep look into our data

Get the Data

Create a Test Set

- Theoretically quite simple, just pick some instances randomly, typically 20% and set them aside

Coding

1. Create random indices
2. Calculate test set percentage
3. Return training, and testing data based on indices

Get the Data

Create a Test Set

- Work! But it is not perfect
  - Because when run the program again, it will generate a different test set! then you get to see the whole dataset, which is what you want to avoid
- Solutions
  1. Save the test set on the first run then load it in subsequent runs
  2. Set the random number generator's seed np.random.seed(42)

Get the Data

Create a Test Set

- Work! But it is not perfect
  - Because both will break next time you fetch an updated dataset.
- Solutions
  1. Use Instance's identifier to decide whether or not it should go in the test set
  2. Compute a hash of each instance and put that instance in test set if the hash is lower or equal to 20% of the maximum hash value

Get the Data

Create a Test Set

- Work! But we don't have record identifier
- Solutions
  1. use the row index as the ID
     1. But make sure that any new record always appended
     2. No old row is deleted
  2. Use the most stable features to build a unique identifier

Get the Data

Create a Test Set

- Scikit-Learn has few function to split data
  1. **from sklearn.model_selection import train_test_split**
     1. random_state parameter that allows you to set the random generator seed
     2. multiple datasets with an identical number of rows

- Sampling Types
  - Purely Random Sampling

Get the Data

Create a Test Set

- Stratified Sampling
  - Take the nature of population into account (51.3% Female, 48.7 Male)
  - Divide Data into homogeneous subgroups called strata
- Suppose median income is a very important attribute to predict median housing prices.
  - median income is a continuous numerical attribute
  - create an income category attribute

3. Discover and Visualize the Data to Gain Insights

Visualize Geographical Data

Looking for Correlations

Experimenting with Attribute Combinations

Discover and Visualize the Data to Gain Insights

Visualizing Geographical Data

- Draw Geographical Information
  - Create Scatterplot of all districts to visualize the data
  - Play with alpha to make it easier to visualize places with high density
  - Let make radius of circle represents the district's population
  - Color represent the price ranges from blue (low values) to red (high prices)
- Findings
  1. Housing Price are very much related to the location and population density
  2. We could use clustering algorithm to detect main clusters and use it as new feature for input, but North prices are not too high
  3. We could use ocean_proximity attribute as well

Discover and Visualize the Data to Gain Insights

Looking for Correlation

Findings

1. Correlation is indeed very strong
2. Upward trend is clear, and points are not too dispersed
3. Price cap at 500,000$ that appear in histogram
4. There are other lines on 450,000$, 350,000$, 280,000$

Discover and Visualize the Data to Gain Insights

Experimenting with Attribute Combinations

- You need to combine attributes with each other and restudy the correlation searching for another relations
  - total_rooms/households
  - total_bedrooms/total_rooms
  - population/household
- These proposed combination could be helpful Why?
- **Findings**
  - lower bedroom/room houses tend to be more expensive.
  - rooms/household is more informative than total_rooms/district
  - larger houses more expensive

4. Prepare the Data for Machine Learning Algorithm

Data Cleaning
Handling Text and Categorical Attributes
Custom Transformation
Feature Scaling
Transformation Pipelines

Prepare the Data for Machine Learning Algorithm

Separate the predictors and the labels as it is not necessary to apply the same transformation to both

```
housing = strat_train_set.drop('median_house_value', axis=1)
housing_labels = strat_train_set['median_house_value'].copy()
```

Prepare the Data for Machine Learning Algorithm

Data Cleaning

Take care of missing values of total_bedrooms as most ML Algorithm cannot work with missing features, we have 3 options:

1. Remove Corresponding Districts
2. Remove whole attributes
3. Set the values to some value (zero, mean, median, etc.)

Prepare the Data for Machine Learning Algorithm

Data Cleaning

- You need to calculate mean on the training set and use it to fill up missing values
- Save mean to use on test set when you evaluate the model, and on live system when new data is coming
- Scikit-Lean provide handy class to fill up missing value with whatever strategy you choose
- Since median computed only on numerical attribute, we need to drop ocean_proximity as it is categorical
- `SimpleImputer` compute median of each attribute and store the result in `statistics_`
- It is safer to apply imputer to all numerical attributes
- Call `fit` method to calculate mean value for all numerical attributes
- Call `transform` method to apply mean to missing values

Prepare the Data for Machine Learning Algorithm

Handling Text and Categorical Attributes

- Most machine learning algorithm prefer to work with number
- Use `fit_transform` from `OrdinalEncoder` to convert texts to numbers
- Check `categories_` attribute to list all categories

Finding

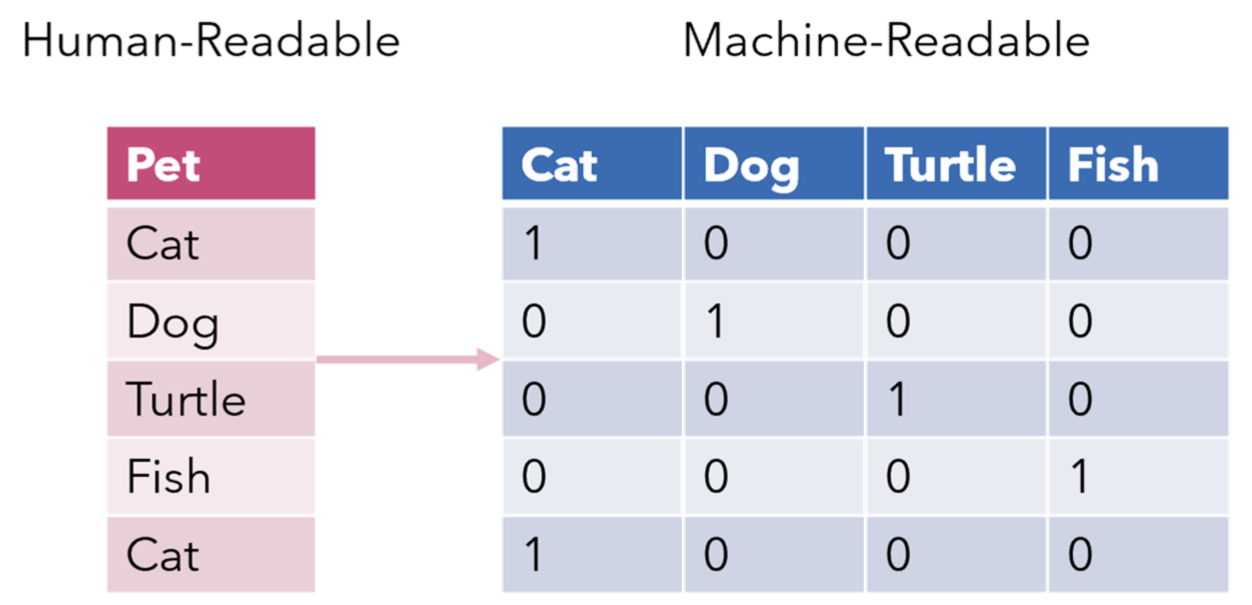
- Machine Learning will learn that nearby values are more similar than distant values
- Which may be good in some cases like (bad, average, good, excellent)
- But it is not the case with ocean_proximity

Prepare the Data for Machine Learning Algorithm

Handling Text and Categorical Attributes

One-Hot Encoding

- Create vector with length equal to number of category
- Set 1 (hot) corresponding to category index
- Set 0 (cold) elsewhere
- Scikit-Learn provide `OneHotEncoder` class to convert categorical values into one-hot vectors
- Using `fit_transform` from `OneHotEncoder` return SciPy sparse matrix
- Much efficient for memory usage
- It store location of non-zero element, Instead of store wasteful zeros



Human-Readable

Pet
Cat
Dog
Turtle
Fish
Cat

Machine-Readable

Cat	Dog	Turtle	Fish
1	0	0	0
0	1	0	0
0	0	1	0
0	0	0	1
1	0	0	0

Prepare the Data for Machine Learning Algorithm

Custom Transformation

- Custom cleanup operations or combining specific attributes need custom transformers
- To make custom transformers work seamlessly with Scikit-Learn functionalities (such as pipelines) you need to implement `fit()` (returning self), `transform()`, and `fit_transform()`.
- You can get `fit_transform()` free by add `TransformerMixin` as base class
- You can achieve automatic hyperparameter tuning by add `BaseEstimator` as another base class which provide `get_params()` and `set_params()`

Prepare the Data for Machine Learning Algorithm

Custom Transformation

- We create one hyperparameter, add_bedrooms_per_room set to True by default
- It is good to provide sensible defaults
- Hyperparameters are parameters that are part of Algorithm not part of the Learning
- We can add hyperparameter that gate any data preparation step that you are not 100% sure about.
- More automation more likely you find great combination and Saving you a lot of time

Prepare the Data for Machine Learning Algorithm

Feature Scaling

- Mostly Machine Learning algorithms don't perform well with numerical input attributes that have very different scales.
- Feature Scaling need to be done on input to fix such these differences total_rooms range 6 to 39,320, median_income range 0 to 15
- Generally scaling not required for output or target values
- We have two common ways
  - min-max scaling
  - Standardization

Prepare the Data for Machine Learning Algorithm

Feature Scaling

Min-Max Scaling (Normalization)

- Simply values are shifted and rescaled so that they end up ranging from 0 to 1.
- We do that by (x - min(X)) / max(X) - min(X)
- Scikit-Learn provide MinMaxScaler transformer which has feature_range hyperparameter that allow you to change the range of values

Prepare the Data for Machine Learning Algorithm

Feature Scaling

Standardization

- We do that by (x - mean(X)) / std(X)
  - Values always have zero mean
  - Values resulting distribution has unit variance (std = 1)
  - Much less affected by outliers
  - Does not bound values to specific range (sometimes this is problem e.g. neural networks often expect input range 0 to 1)
- Scikit-Learn provide StandardScaler transformer

Prepare the Data for Machine Learning Algorithm

Transformation Pipeline

- There are many data transformation steps that need to be executed in the right order (Imputer, Attributes Adder, Std Scaler)
- Scikit-Learn Provide Pipeline class for this kind of transformations
- Pipeline constructor takes list of tuple (name, transformer) defining a sequence of steps
- All steps but the last must be transformers (they must have `fit_transform()` method)
- When `fit()` method called all transformers `fit_transform()` methods are called sequentially passing output of each call as parameter to the next call
- Final estimator just calls the `fit()` method.
- Pipeline exposes same method as final estimator

Prepare the Data for Machine Learning Algorithm

Transformation Pipeline

- Scikit-Learn provide ColumnTransformer to handle numerical columns separately from categorical columns
  - Constructor require list of tuples (name, transformer or drop or passthrough, list of columns (names or indices)) that the transformer should apply to
  - Transformer return same number of rows, but may different number of columns
  - When such mix of sparse and dense matrices are exists ColumnTransform estimates the density of the final matrix (ratio of nonzero cells)
  - Return sparse matrix if sparse_threshold=03 is exceeded

5. Select and Train Model

Training and Evaluating on the Training Set

Better Evaluation Using Cross Validation

Select and Train a Model

Training and Evaluating on Training Set

- Thanks to all previous steps that make things much simpler than we might think
- Train first Linear Regression model is just like 3 lines of code

```
from sklearn.linear_model import LinearRegression


lin_reg = LinearRegression()

lin_reg.fit(housing_prepared, housing_labels)
```

- Done! Let's try it out

Select and Train a Model

Training and Evaluating on Training Set

- Works, but predictions are not exactly accurate
- Let's measure RMSE on the whole training set
- Scikit-Learn provide mean_squared_error

Select and Train a Model

Training and Evaluating on Training Set

- Most districts' median_housing_values range between $120,000 and $265,000
- Prediction error of $68,628 is not very satisfying
- But! What exactly the problem is?

- Underfitting
  - Features do not provide enough information to make good predictions
  - model is not powerful enough.
- But first let's try a more complex model

Select and Train a Model

Training and Evaluating on Training Set

- DecisionTreeRegressor. is a powerful model, capable of finding complex nonlinear relationships in the data
- Let's train the model, and evaluate it on the training set

- No error at all? Could this model really be absolutely perfect?
- It is much more likely that the model has badly overfit the data
- How can we make sure? Without touching the test set?
  - Use part of training set for training
  - And part for model Validation

Select and Train a Model

Better evaluation using Cross Validation

- One way to evaluate the Decision Tree is using train_test_split
- Split the training set into a smaller training and validation set
- Training models against the smaller training set and evaluate them against the validation set.
- It's bit of work, but nothing too difficult and it would work fairly well

- Alternatively, is using Scikit-Learn’s K-fold cross-validation feature.
- Randomly splits training set into 10 distinct subsets called folds
- Train the model on 9 folds, Validate on 1 and rotate
- Results array of 10 evaluation scores
- Note: cross-validation expect utility function

Select and Train a Model

Better evaluation using Cross Validation

Findings

- Decision Tree look worse than the Linear Regression model!
- Cross-validation allow to get estimate of the performance and measure of how precise this estimate (i.e., its standard deviation).
- Decision Tree score of approximately 71,407, generally ±2,439
- Cross-validation comes at the cost of training the model several times, so it is not always possible.
- Comparing mean of error tell us that Decision Tree overfit so badly

Select and Train a Model

Better evaluation using Cross Validation

- Let’s try one last model RandomForestRegressor
- Random Forests work by training many Decision Trees on random subsets of features, and Averaging their predictions
- **Ensemble Learning**: Building a model on top of many other models

Select and Train a Model

Better evaluation using Cross Validation

Findings

- Random Forests look very promising.
- Score on training set is much lower than on validation sets. Model is still overfitting the training set

Overfitting

- Simplify model, constrain it (i.e., regularize it)
- Get more training data

Select and Train a Model

Better evaluation using Cross Validation

- Try out other models from various categories, Without spending much time tweaking the hyperparameters
- The goal is to shortlist the promising models

Select and Train a Model

Better evaluation using Cross Validation

- You need to save the model you experiment
- Saving both hyperparameter and trained parameters

Thank you

Eng. Mohammad R. KATBY

morikapt@gmail.com

@morikapt