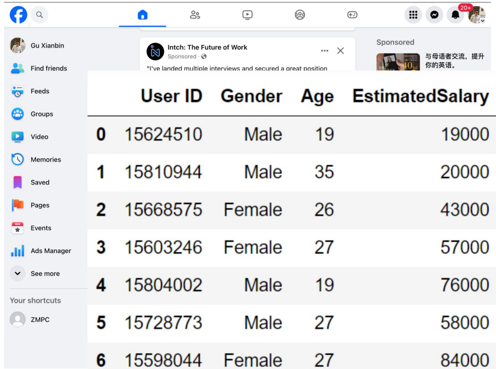
[ICDS Spring 2025]

Data Science: Part 2

Supervised and Unsupervised Learning

Recap: Machine Learning

- Definition: **PET!**
  - A computer program that learns from experience E with respect to some class of task T and performance measure P



	User ID	Gender	Age	EstimatedSalary	Purchased
0	15624510	Male	19	19000	0
1	15810944	Male	35	20000	0
2	15668575	Female	26	43000	0
3	15603246	Female	27	57000	0
4	15804002	Male	19	76000	0
5	15728773	Male	27	58000	0
6	15598044	Female	27	84000	0
7	15694829				
8	15600575				
9	15727311				

```
When k=1, the accuracy is 0.68.
When k=2, the accuracy is 0.68.
When k=3, the accuracy is 0.62.
```

Task: predicting whether a user will make purchase or not

Experience: the observations/dataset on how existing users responding to the ads

Performance: prediction accuracy

Exercise 1: Types of ML Tasks

We used KNN algorithm in the previous example. What type of machine learning task did it perform?

A. Clustering

B. Classification

C. Regression

D. Dimension reduction

Scan to answer!

Exercise 2: KNN Algorithm

Scan to answer!

Which of the problem below can be solved by KNN?

A. Partitioning Genomic Data: Given a dataset containing gene expression profiles, partition genes into clusters based on their expression patterns to uncover underlying biological processes.

B. Movie Recommendation: Given a dataset of user ratings for different movies, recommend movies to a new user based on the ratings of similar users.

C. Segmenting Satellite Images: Given a collection of satellite images, group similar regions together based on pixel intensities to identify different land cover types.

D. Travel Time Estimation: Given a set of records of traffics, weather conditions, and the time spent traveling from your apartment to the campus, predict tomorrow's commuting time.

Recap: Machine Learning Models

- 3 fundamental components of a ML model:
    - Model design (learning/inductive bias)
        - Assumptions to generalize to unseen data samples
        - Principle for choosing the model to use
    - Data representation (features)
        - Feature engineering: deriving task-appropriate features
    - Measurement (distance metrics)
        - Definition for similarity: Euclidean, non-euclidean, etc.

Exercise 3: Feature Engineering

To build a predictor for travel time estimation, what feature can also be helpful besides historical traffics and weathers?

A. Time: time-of-day, day-of-week, whether-holiday

B. Road attributes (e.g., speed limit, #lanes) in the route

C. Historical records of traffic accident occurrences

D. All of above

Scan to answer!

Agenda

- Regression and gradient descent
  - Regression analysis
  - Gradient descent
- <u>Supervised learning</u>
- Clustering and K-Means
- <u>Unsupervised learning</u>
- Appendix: using the sklearn's k-means

Linear Regression
and
Gradient Descent

Predicting the waiting time

The old faithful geyser dataset

- <u>Waiting time</u> between eruptions and the <u>duration of the eruption</u> for the Old Faithful geyser in Yellowstone National Park, Wyoming, USA.
- 272 observations on 2 variables

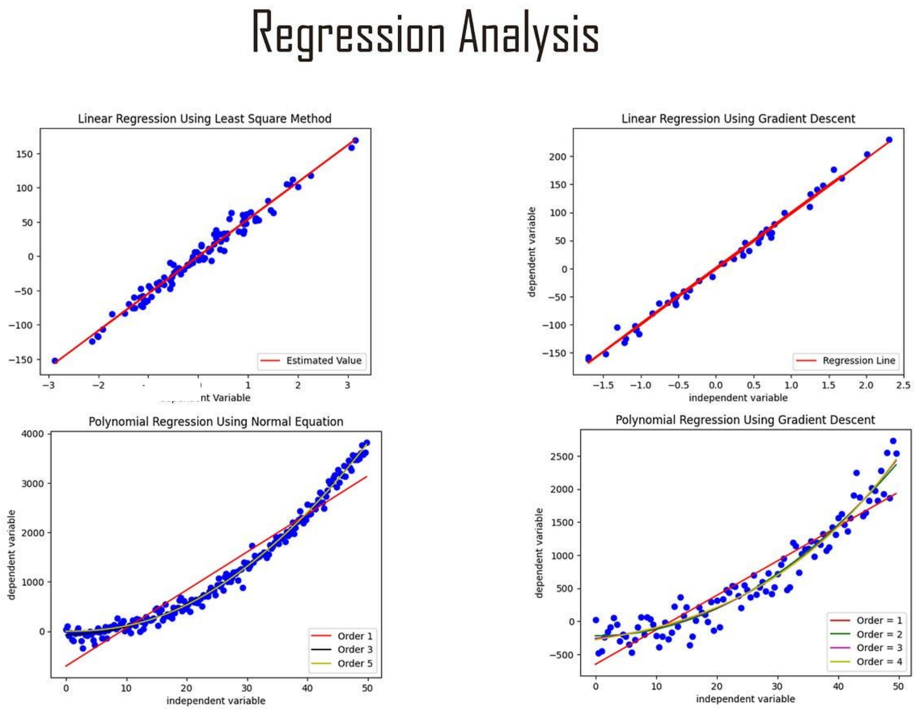
Task: given an eruption duration, predict the waiting time.

faithful

index	eruptions	waiting
1	3.6	79
2	1.8	54
3	3.333	74
4	2.283	62
5	4.533	85
6	2.883	55
7	4.7	88
8	3.6	85
9	1.95	51
10	4.35	85

Regression Analysis

- It is a statistical method for estimating the relationship between a dependent variable and one or more independent variables.
- A tool for making predictions, understanding relationships between variables, and making decisions
- Widely used in economics, finance, engineering, and social science



Regression model

A general form of a regression model,

$$Y_i = f(X_i, \beta) + e_i \qquad (1)$$

where,

- Y_i is the dependent variable;
- X_i is the independent variable;
- β is the model parameter(s);
- e_i is the error terms, representing the observation errors.

- It is a mathematical equation that represents the relationship between Y and X.
- β represents the change of Y for a one-unit change of X.
- e represents the error term (also called, residuals), representing the variability in Y that cannot explained by X.

Linear regression

If we assume the form of f is linear combination of the parameters β, we have the linear regression. For example, let $f(X_i, \beta) = \beta_0 + \beta_1 X_i$, we have

$$Y_i = \beta_0 + \beta_1 X_i + e_i \tag{1}$$

where

- β_0 is the intercept, the value of Y when $X = 0$;
- β_1 is the slope;
- e_i is the error item; we assume $e_i \sim N(0, 1)$.

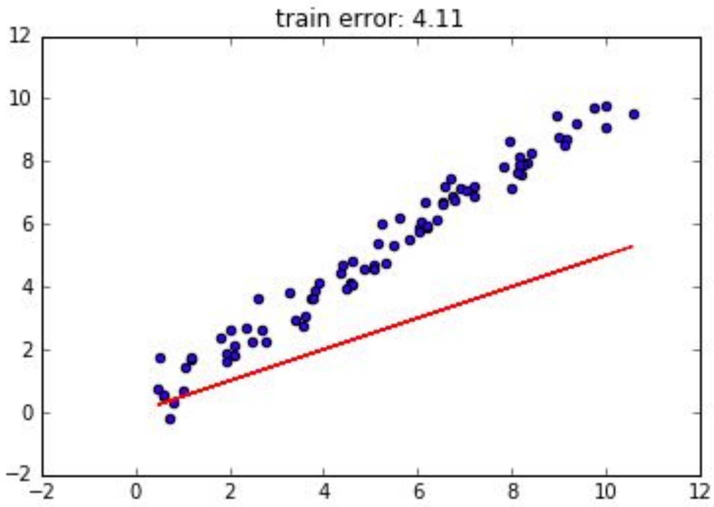
- Linear regression is the most common model used in regression analysis; it assumes the relationship between Y and X can be represented by a straight line.
- In practice, when we have no knowledge about what the relationship between Y and X, we can firstly try the linear regression.

Find the parameters of a regression model

We want the best fit line (making as less error as possible), i.e., the line covers as many points as possible.

In the old faithful geyser example:

- Our model: `y'=wx`
- Goal: find a **w** that minimizes the prediction error, `(y'-y)`.



Loss function for the task

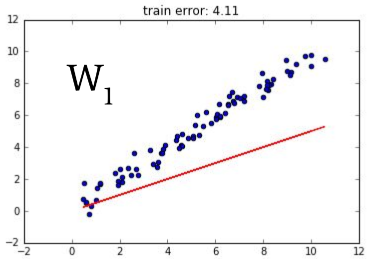
- Model: $y' = wx$
- Dependent variable: y (i.e., the observed true value)

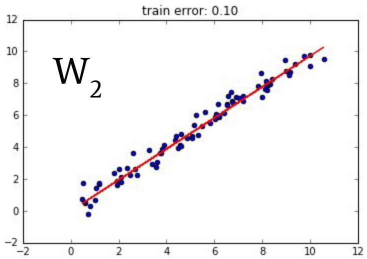
i-th prediction, y_i'

- Loss function: $E(w) = \frac{1}{2n}\sum_{i=1}^{n}(\boxed{wx_i} - y_i)^2$
- Task: find a proper w such that E is minimized

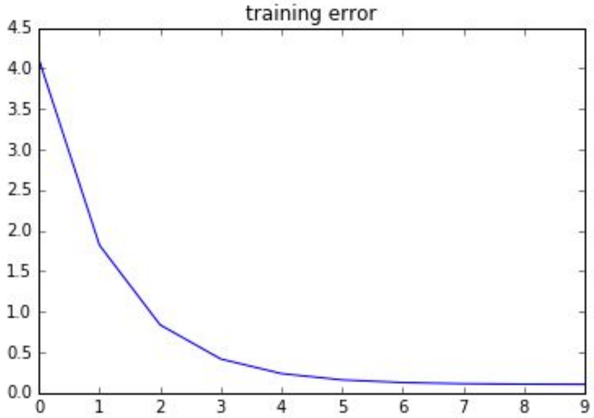
Solving the minimization task

- Adjust w such that E decreases monotonically





What method can achieve this process?



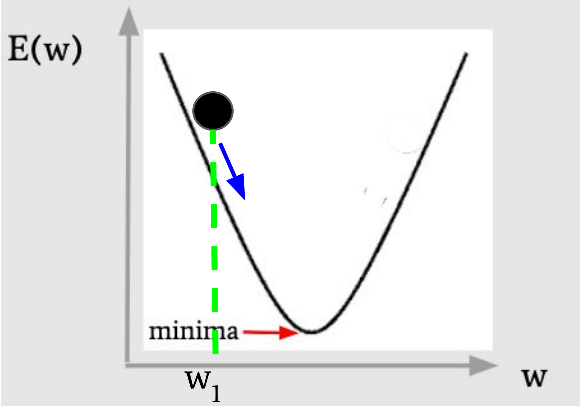
- Gradient descent

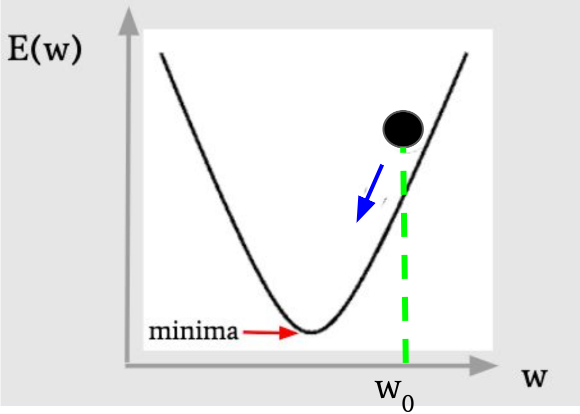
Gradient descent

Gradient: $\frac{dE}{dw}$, is the ratio of changes in E in response to the changes in w.

We have $\boxed{\Delta E = \frac{dE}{dw}\Delta w}$ It represents how much E changes when w changes

Can we find a Δw that always makes ΔE < 0?





Gradient descent

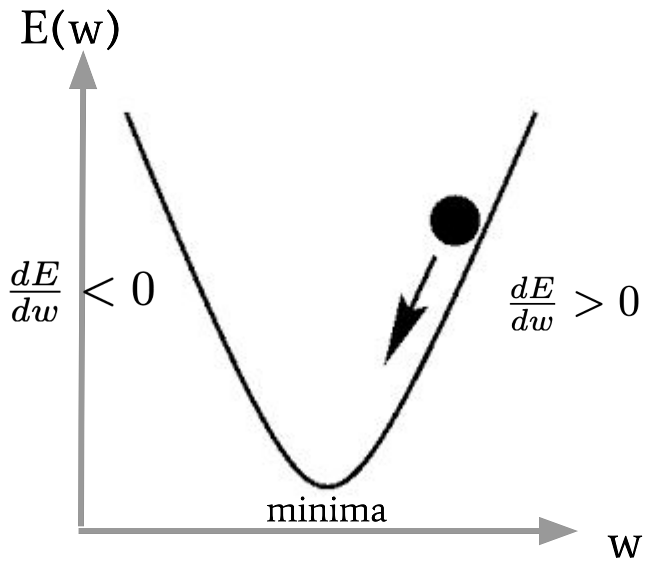
It is always opposite to the gradient so guarantees ΔE<=0

$$\Delta w = \boxed{-\boxed{\lambda}\frac{dE}{dw}}$$

Learning rate, a hyperparameter

$$\Delta E = \frac{dE}{dw}\Delta w$$

ΔE is always non-positive.



A little bit calculus

Q: What is the gradient of our loss function?

A: The first-Order derivative.

$$E(w) = \frac{1}{2n}\sum_{i=1}^{n}(wx_i - y_i)^2$$

$$\frac{dE}{dw} = ?$$

The gradient of the loss function

$$E(w) = \frac{1}{2n}\sum_{i=1}^{n}(wx_i - y_i)^2$$

$$\frac{dE}{dw} = \frac{1}{n}\sum_{i=1}^{n}(wx_i - y_i)x_i$$

$$\frac{dE}{dw} = \frac{1}{n}\sum_{i=1}^{n}\underline{(y_i' - y_i)x_i}$$

- Averaging the prediction errors that are weighted by input contribution (i.e., x_i)

Solving the minimization task using gradient descent:

1. Initial w
2. Calculate y' by $y_i' = wx_i$ for $i = 1, 2, 3, ..., n$
3. Calculate gradient $\frac{dE}{dw}$ by $\frac{dE}{dw} = \frac{1}{n}\sum_{i=1}^{n}(y_i' - y_i)x_i;$
4. Increase w by $\Delta w = -\lambda\frac{dE}{dw}$, then, goto step 2.

- Pick a random **w** to start, update **w** until it converges.
- We would like to record the E in each round.
- The algorithm stops when E decreases very little (i.e., |ΔE| < threshold; converge).

Regression with gradient descent

```python
    # training loop
    while True:

        # predict and compute error
        pred_y = [w*x for x in d_x]
        # print(pred_y)
        error = compute_error(pred_y, d_y)
        error_history.append(error)
        #computer the change of error
        delta_error = error - error_last_iter
        error_last_iter = error

        plt.cla()
        plt.scatter(d_x, d_y)
        plt.plot(d_x, pred_y, 'r')
        title_str = "train error: %0.2f" % error
        plt.title(title_str)
        plt.show()

        steps += 1
        if abs(delta_error) <= margin or steps >= num_iter:
            break

        # gradient descent
        grad = compute_grad(pred_y, d_y, d_x)
        w -= learning_rate * grad
```

Computing predictions

Recording errors

Checking the convergence

Computing gradients and updating w

Regression with gradient descent

Complete **compute_error** and **compute_grad**:

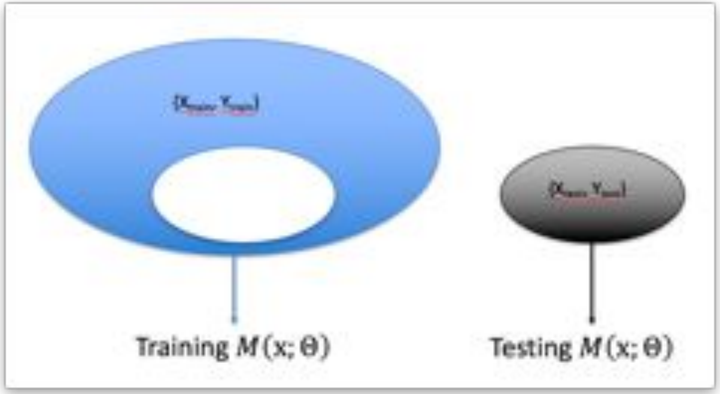
$$E(w) = \frac{1}{2n}\sum_{i=1}^{n}(wx_i - y_i)^2$$

$$\frac{dE}{dw} = \frac{1}{n}\sum_{i=1}^{n}(pred_i - y_i)x_i$$

```python
def compute_error(pred, truth):
    """ mean square error:
        sum of (pred[i] - truth[i])^2, divided by 2*length
    """
    return regression_helper.compute_error(pred, truth)

def compute_grad(pred, truth, x):
    """ gradient is mean of (pred[i] - truth[i]) * x[i]
    """
    return regression_helper.compute_grad(pred, truth, x)
```

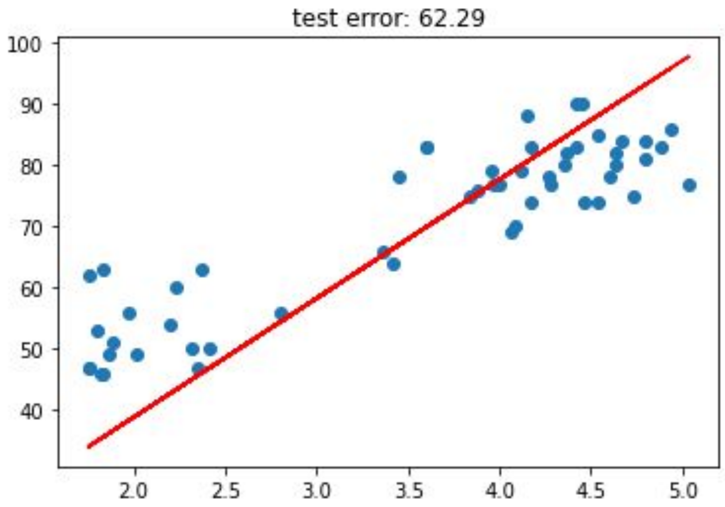
Data processing

```python
if __name__ == "__main__":

    path = './faithful.csv'
    # loading data
    f = open(path, 'r')
    raw_data = f.readlines()
    data = []
    for item in raw_data[1:]:
        item = item.strip().split(',')
        data.append([float(item[1]), float(item[2])])
    print(f"There are {len(data)} data items.")
    # Make a random split of the data
    random.shuffle(data)
    train_data = data[:round(0.8*len(data))]
    test_data = data[round(0.8*len(data)):]

    # Plotting the randomly generated data
    d_x = [d[0] for d in train_data]
    d_y = [d[1] for d in train_data]
#     plt.ion()
    plt.figure(1)
    plt.scatter(d_x, d_y)
    plt.show()
```



The model never looks at test data

```
    # testing on test data
    d_x = [d.getFeatures()[0] for d in test_data]
    d_y = [d.getFeatures()[1] for d in test_data]
    pred_y = [w*x for x in d_x]
    error = compute_error(pred_y, d_y)

    plt.scatter(d_x, d_y)
    plt.plot(d_x, pred_y, 'r')
    title_str = "test error: %0.2f" % error
    plt.title(title_str)
    plt.show()

    plt.plot(error_history)
    plt.title('training error')
    plt.show()
```



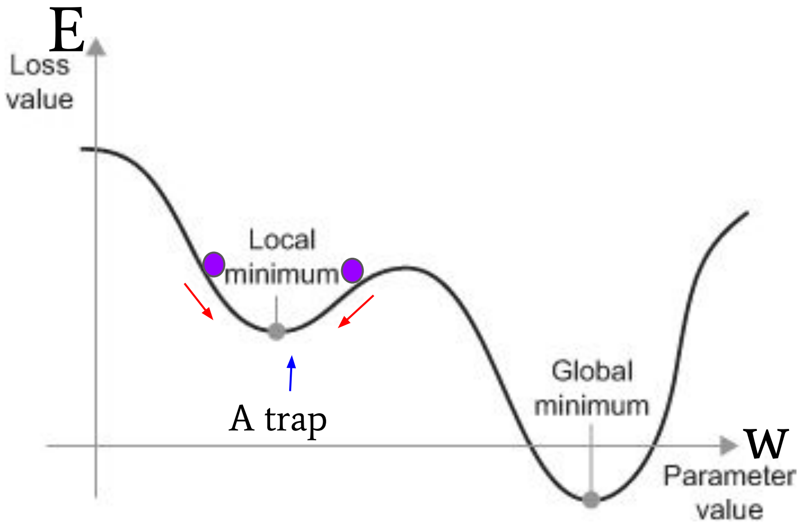
Settings of hyperparameters

```python
num_iter = 100
steps = 0
learning_rate = 0.01
n = len(train_data)
error_history = []
error_last_iter = 0
margin = 0.01

# w is the parameter to be learne
w = 0.5
```

The learning rate is a hyperparameter in gradient descent. It has large influence on the learning process.

- An improper learning rate may ruin the learning process. (You may change it to 1, and run the program to see its impact.)
- In practice, we often normalize the raw data before using them to train models which can help us find a suitable learning rate.

Traps of local minimum



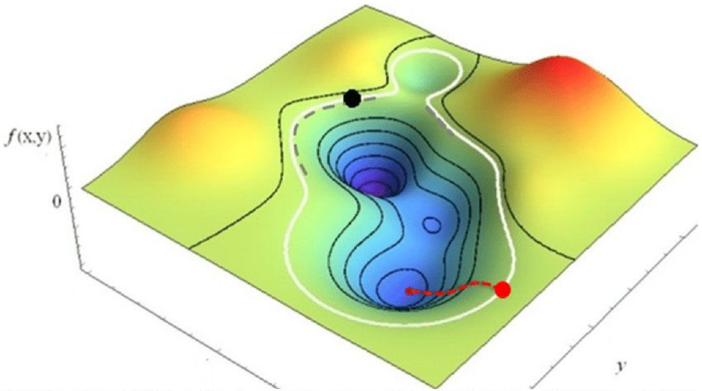
$$\Delta W = -\lambda \frac{dE}{dw}$$

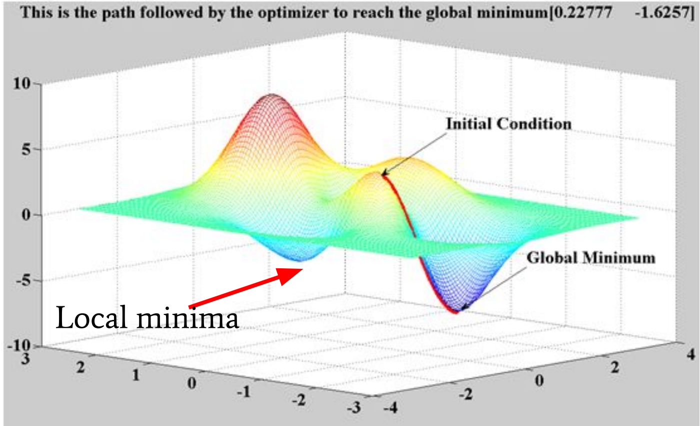
- The art of setting λ
  - increasing λ, roll across the local minima.
  - An oversize λ results in non convergence (i.e., jump over the global minima)

Model's performance is influenced by the hyperparameters. For each model, we need to tune them for fitting the dataset.

- How can we find the best hyperparameters? We may use grid search, but no perfect solution yet.
- This is why some researchers alleged machine learning is alchemy.

Solving optimization problems





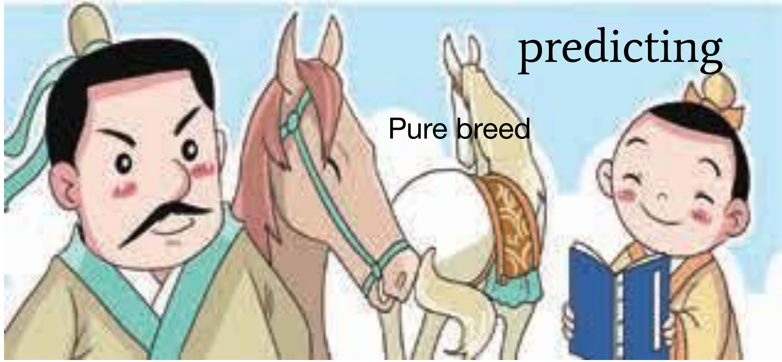
- Gradient descent is widely used in solving optimization problems.
- When there are multiple parameters in the loss function, the surfaces will be intricate.
- Gradient descent cannot promise to find the global optima.

Agenda

- Regression and gradient descent
  - Regression analysis
  - Gradient descent
- **Supervised learning**
- Clustering and K-Means
- Unsupervised learning
- Appendix: using the sklearn’s k-means

Supervised learning

- the most common approach in ML
  - “right answers” are given
  - models learn relations between inputs and answers
  - task: make predictions (label/value)



We always need to test the model (validation)

It is important to know how many correct/incorrect predictions from the model.

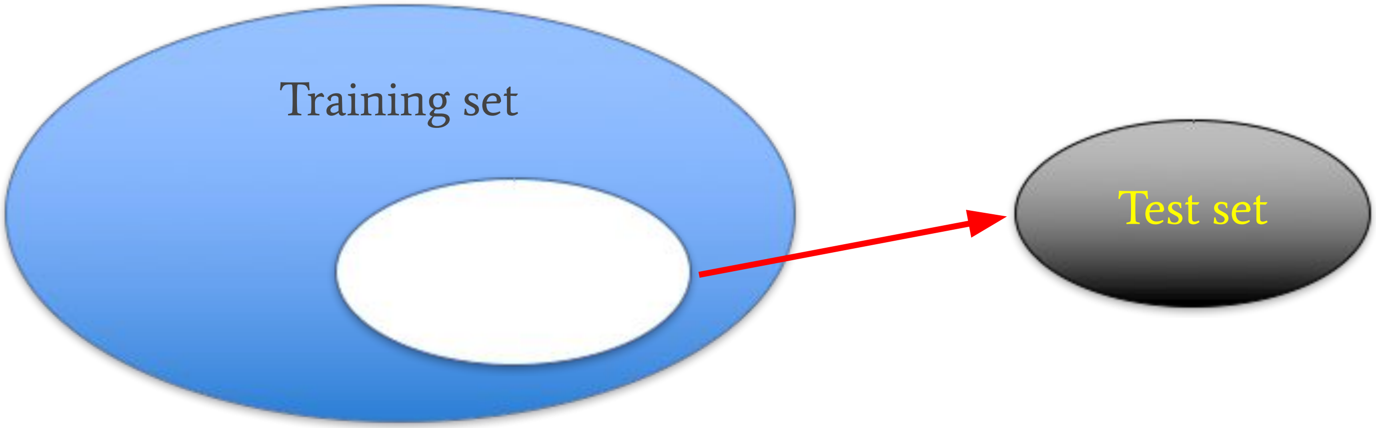
Good learner makes correct predictions on unseen cases

Bad learner only works well on training data. (overfitting)

- Generalizability: a good model should work well not only on the train data but also the unseen data (i.e., testing data)

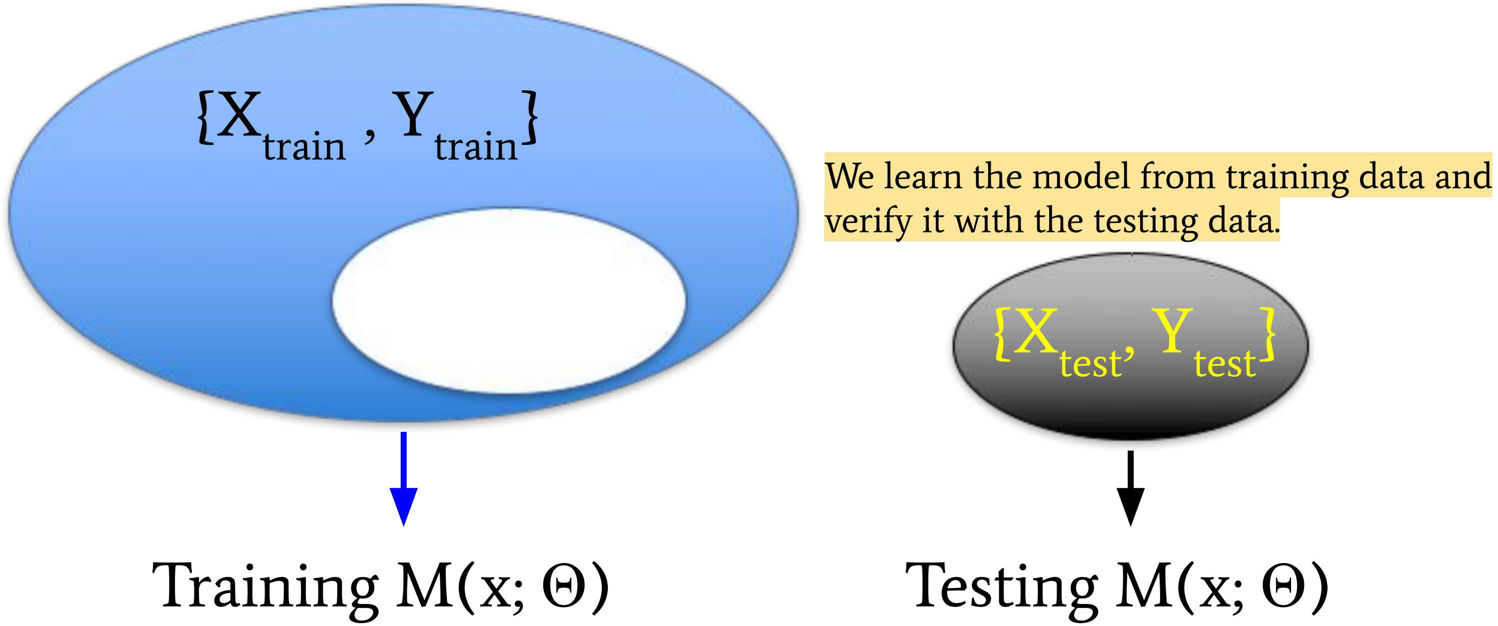
How to test?

- Splitting the samples into two parts:
  - One for training (inferring model parameters)
  - One for testing (verifying)



Why this makes sense?

Inductive bias: “the unseen samples are similar to the given samples”.



Quantify the Performance

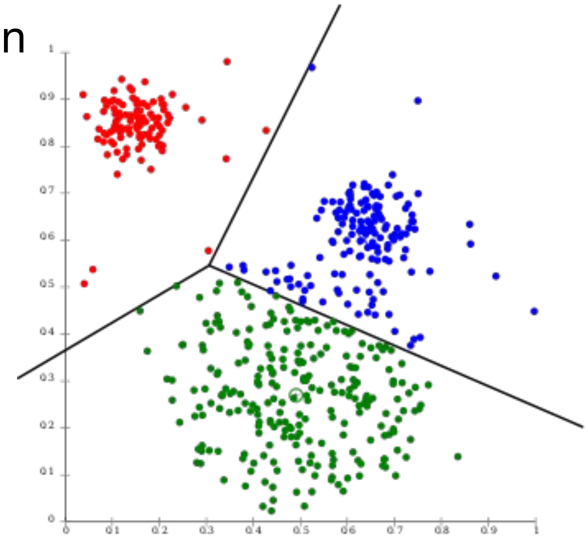
- Main idea: counting the correct predictions on the test dataset
- Statistics for performance measurement:
  - Confusion matrix/precision-recall
    - Confusion matrix - Wikipedia
    - Precision and recall
    - Accuracy and precision

Clustering analysis

Clustering analysis

- It is a type of unsupervised learning technique for grouping similar data points together based on certain features.

- **Task**: to divide a dataset into clusters, where data points within the same cluster are more similar to each other than those in other clusters.
    - It helps in discovering hidden structures within datasets.
    - Applications: data mining, pattern recognition, image analysis, market segmentation, and etc.



Goals of Clustering analysis

The approach:

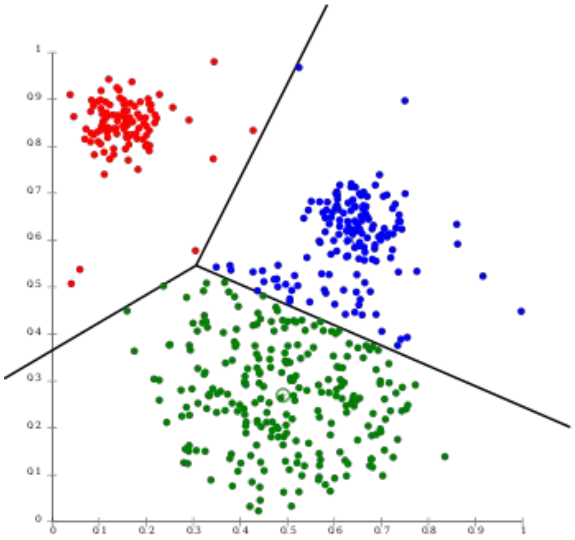
- “Similar” members go into the same group

Key metrics:

- Intra-cluster variance (difference within a group)
- Inter-cluster variance (difference across groups)

Common objectives:

- Minimize intra-cluster variance
- Maximize inter-cluster variance



The k-means algorithm

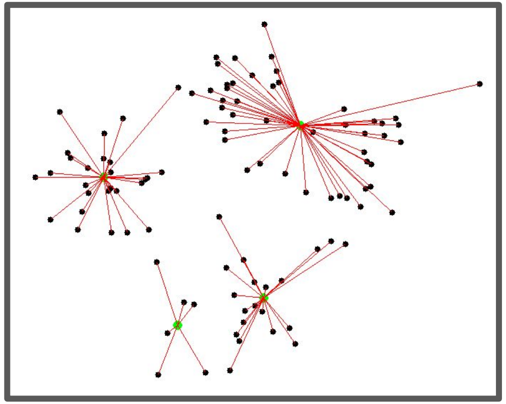
How to achieve the following objectives?

- Minimize intra-cluster variance
- Maximize inter-cluster variance

Assume that a dataset has k clusters; each cluster has a cluster center.

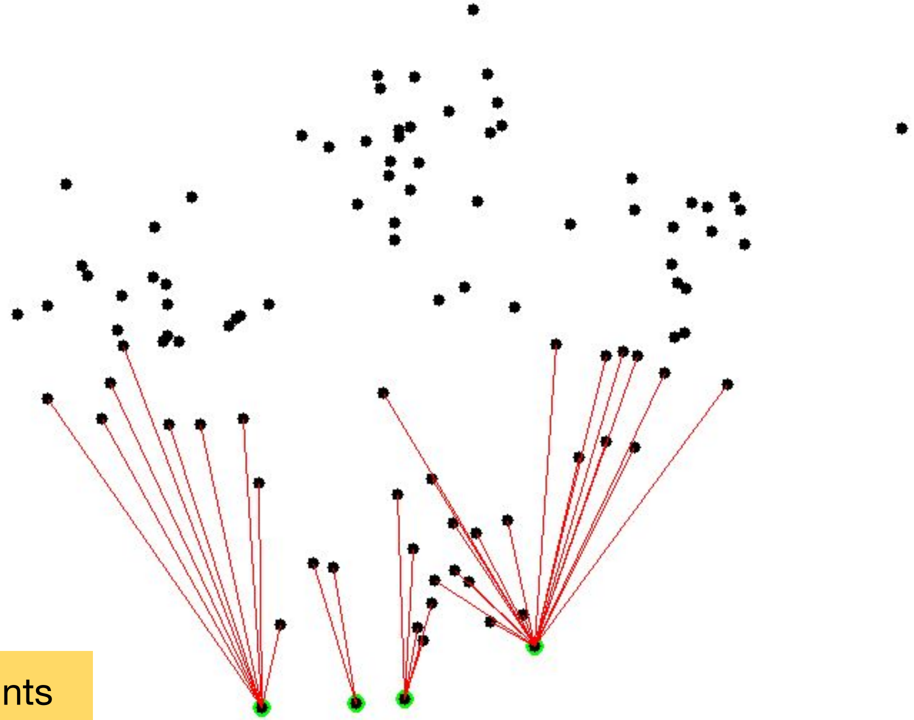
- If we put every data item to a cluster whose centroid is the closest one to it, we will obtain a clustering that meets the objectives.
- k-means is the algorithm to realize the above operations.

Demo: K-means clustering, starting with 4 points



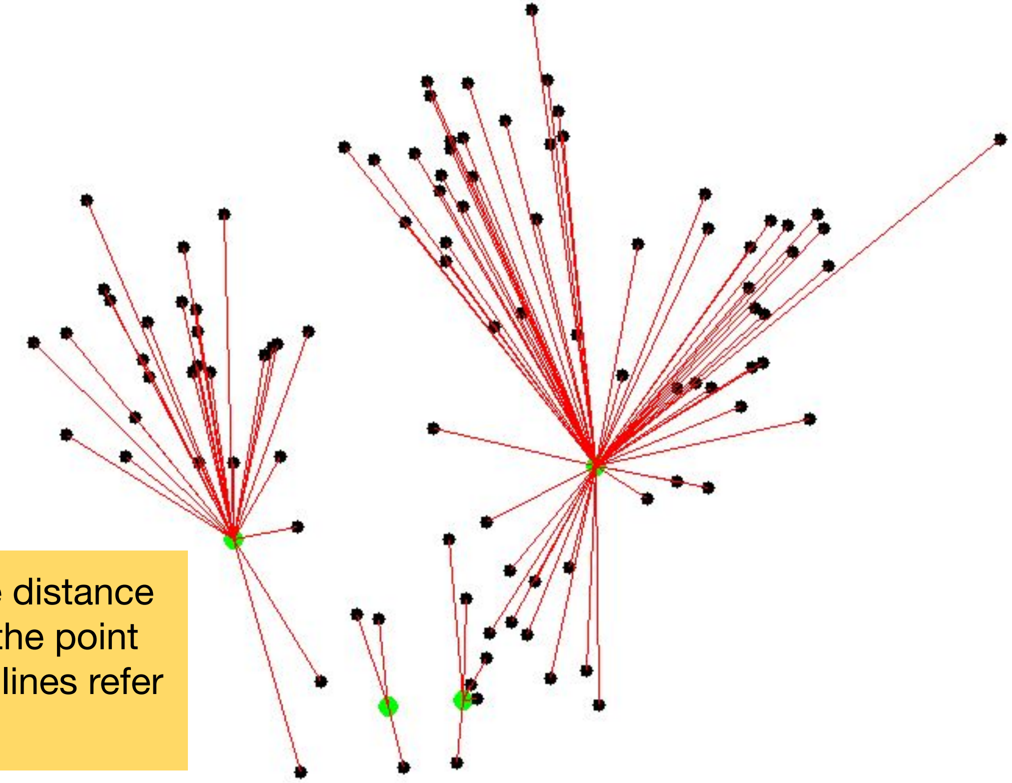
k-means: given a dataset

k-means: generating k centroids



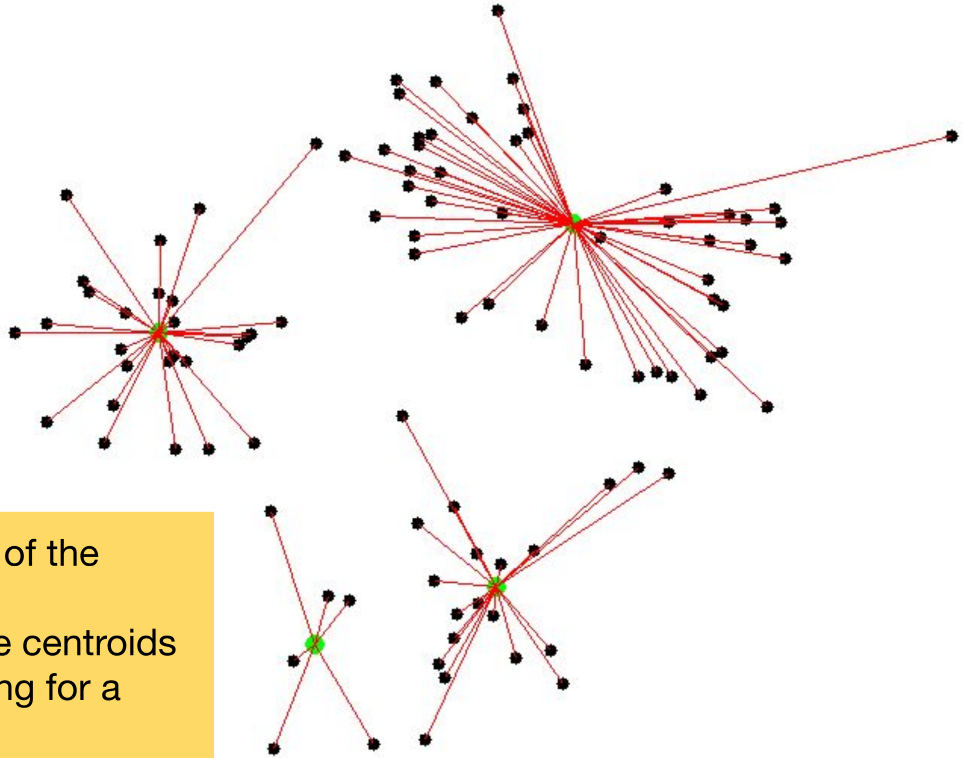
We can randomly pick **k** points as the cluster centroids.

k-means: assigning points to a centroid



For each point, compute the distance to each centroid; associate the point to the nearest centroid. (red lines refer to the distances)

k-means: updating centroids



Update the centers to the average of the points belonging it.

- We stop updating until **all** the centroids don't move (or, doing updating for a number of times)

k-means: the algorithm (Lloyd's algorithm)

Initialization:

- k empty clusters, each centroid is at one of k randomly picked members

Iterate until converge (i.e. no cluster moves its centroid).

1. Each sample finds its nearest cluster, add itself to the cluster
2. Each cluster computes its new centroid
3. **If** no cluster changes its centroid – done! (condition)
4. Otherwise, all clusters clear their samples, **go to** 1. (repetition)

Let’s play!

The Iris Dataset: [Iris flower data set - Wikipedia]

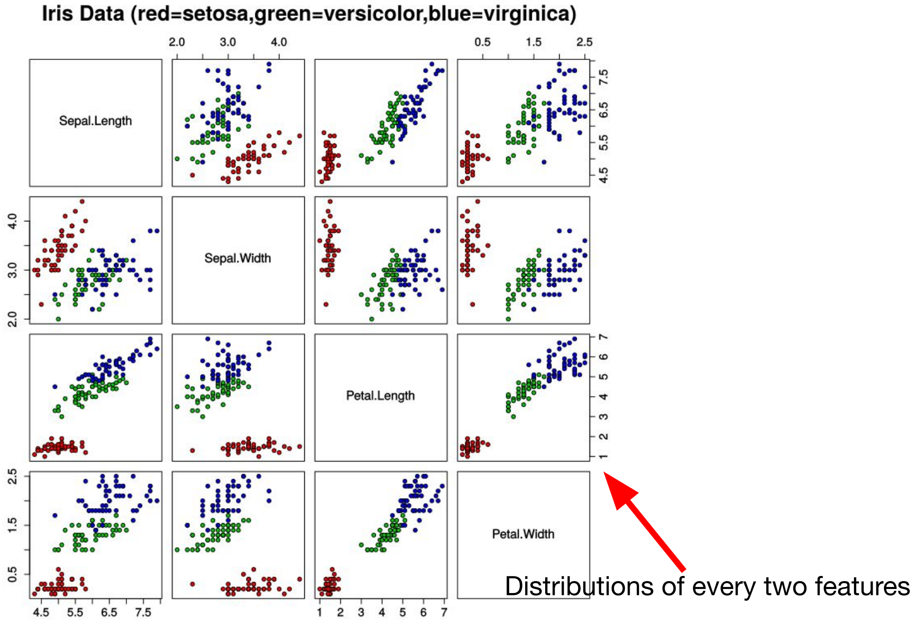
- 150 samples of three species of Iris (setosa, virginica, and versicolor)
- Each sample has four features: length and width of the spedals and petals respectively.

setosa

versicolor

virginica

Dataset: https://docs.google.com/spreadsheets/d/1jpUmkBLe_W_U76f_Efozp9YItER5MkxCRLnUsVOI8xg/edit?usp=sharing



iris.csv: the raw dataset

We can load it by `open('iris.csv', 'r')`

sepal.length	sepal.width	petal.length	petal.width	variety
5.1	3.5	1.4	0.2	Setosa
4.9	3	1.4	0.2	Setosa
4.7	3.2	1.3	0.2	Setosa
4.6	3.1	1.5	0.2	Setosa
5	3.6	1.4	0.2	Setosa
5.4	3.9	1.7	0.4	Setosa
4.6	3.4	1.4	0.3	Setosa
5	3.4	1.5	0.2	Setosa
4.4	2.9	1.4	0.2	Setosa

```
f = open('iris.csv', 'r')
raw_data = f.readlines()
```

```
"sepal.length","sepal.width","petal.length","petal.width","variety"\n
 '5.1,3.5,1.4,.2,"Setosa"\n', '4.9,3,1.4,.2,"Setosa"\n',
.7,3.2,1.3,.2,"Setosa"\n', '4.6,3.1,1.5,.2,"Setosa"\n',
```

Note: we can build a class to represent the data.

Abstract Data Types (ADT) for the iris dataset

run_kmeans(samples, k)

Run k-means on the the samples:
+ for each sample, find the nearest centroid; add it to the corresponding cluster;
+ for each cluster, update its centroid; check if the new centroid is different from the old one
+ return the clusters if no centroids change

Sample

- attr: features of the data item
- label: data label (if it has one)
- cluster: the cluster index it belongs to

+ get_attr():
+ set_attr():

Cluster

- samples: list
- label: the index of the cluster
- centroid: a sample instance

+ add_sample(sample)
+ get_centroid():
+ reset_sample():

K-means code

Initialization: k empty clusters, each centroid is at one of k randomly picked members

Iterate until converge:

1. Remove the samples of each cluster to be an empty list (reset);
2. Each sample is assigned to its nearest cluster;
3. Each cluster computes its new centroid;
4. **If** all centroids don't change, done! (converged)
5. Otherwise, **go to** 1. (repetition)

```python
def kmeans(samples:list, k=2):

    clusters = []
    for i in range(k):
        cluster = Cluster([], label=i)
        clusters.append(cluster)

    centroids = []
    for c in clusters:
        init_centroid = random.choice(samples)
        centroids.append(init_centroid)

    centroid_changed = True
    max_iteration = 100
    iterations = 0

    while centroid_changed or iterations < max_iteration:
        for c in clusters:
            c.reset_samples()

        for s in samples:
            cluster_idx = find_nearest_centroid(s, centroids)
            clusters[cluster_idx].add_sample(s)

        new_centroids = []
        for c in clusters:
            new_centroid = centroid_update(c)
            new_centroids.append(new_centroid)

        for i in range(k):
            dist = new_centroids[i].distance(centroids[i])
            if dist > 0.001:
                centroid_changed = True
                centroids = new_centroids
                break
            else:
                centroid_changed = False
        iterations += 1
    return clusters
```

Assigning samples to the nearest cluster

Updating centroids

Checking convergence

Sample class

```python
class Sample:

    def __init__(self):

        self.attr = []
        self.label = None
        self.cluster = None

    def get_attr(self):
        return self.attr

    def set_attr(self, attr):
        self.attr = attr

    def set_label(self, newLabel:str):
        self.label = newLabel

    def get_label(self):
        return self.label

    def set_cluster(self, newCluster):
        self.cluster = newCluster

    def get_cluster(self):
        return self.cluster
```

```python
    def distance(self, otherSample):
        '''compute the Eculidian distatnce between
           this sample to the other.
        '''
        d = 0

        for i in range(len(self.attr)):
            d += (self.attr[i] - otherSample.get_attr()[i])**2

        return d**0.5


    def __add__(self, sample):
        new_s = Sample()
        # print(self.data)
        attr = []
        for i in range(len(self.attr)):
            attr.append(self.attr[i]+sample.get_attr()[i])
        new_s.set_attr(attr)
        return new_s


    def __truediv__(self, n):

        new_s = Sample()
        attr = []
        for i in range(len(self.attr)):
            attr.append(self.attr[i]/n)

        new_s.set_attr(attr)
        return new_s
```

Operator overloading of "+"

Operator overloading of "/"

Cluster class

```python
class Cluster:

    def __init__(self, index:int):

        self.index = index
        self.samples = []
        self.centroid = None


    def add_sample(self, sample):
        self.samples.append(sample)


    def get_centroid(self):

        '''compute the centroid of the cluster'''

        return None

    def reset_samples(self):
        self.sample = []
```

To get the centroid, we need to sum all samples in the cluster and divide by the number of samples. [Hint: you should use the + and / of the samples.]

Complexity of k-Means

n: number of points

k: number of clusters

d: feature dimension of each point

i: number of total iterations

The time complexity of k-mean is O(nkdi); can you explain why?

Unsupervised learning

- It is a a type of machine learning where the model learns patterns or structures from unlabeled data without explicit supervision or guidance.
- When the task lacks of labeled data, one may use unsupervised learning methods to discover the intrinsic structures of the dataset
- Applications:
  - clustering similar documents
  - segmenting customers based on their purchasing behavior
  - detecting anomalies in data
  - reducing the dimensionality of high-dimensional data
  - ……

Homeworks

[ML] Hands-on - Finish in-class exercises:

- Regression with Gradient Descent: page 22
- Clustering with K-means: page 48

[ML] Reading - Chapter 22-24: ML, Clustering, Classification

- Brightspace/Reference Books/*Introduction to Computation and Programming Using Python, by John V. Guttag*

Appendix: k-means using sklearn

sklearn: Scikit-learn

- Sklearn is a free machine learning library, which provide you with a quick way to implement data science tools in your practices.
- It has various built-in functions for classification, regression, and clustering
  - knn, regression, and k-mean are all available
  - official documents are under good maintenance, with a variety of examples showing how to use them.

We will use the K-Means algorithm in sklearn.cluster: sklearn.cluster.KMeans — scikit-learn 1.1.1 documentation . Without too many modifications, we can replace our logic of k-means with the sklearn k-means.

Using the K-Means in Sklearn

Examples

```
>>> from sklearn.cluster import KMeans
>>> import numpy as np
>>> X = np.array([[1, 2], [1, 4], [1, 0],
...               [10, 2], [10, 4], [10, 0]])
>>> kmeans = KMeans(n_clusters=2, random_state=0, n_init="auto").fit(X)
>>> kmeans.labels_
array([1, 1, 1, 0, 0, 0], dtype=int32)
>>> kmeans.predict([[0, 0], [12, 3]])
array([1, 0], dtype=int32)
>>> kmeans.cluster_centers_
array([[10.,  2.],
       [ 1.,  2.]])
```

- Need to specify n_clusters (i.e., k);
- Call .fit(X) to run k-means on X;
- X is a list (or numpy array); each element is a list;
- Call .labels_ to access the labels of elements in X;

run_kmeans

```python
def run_kmeans(samples, attributes, k=2):
    data = []
    for s in samples:
        attrs = []
        for attr in attributes:
            attrs.append(s.get(attr))
        data.append(attrs)

    kmeans = KMeans(k).fit(data)
    labels = kmeans.labels_
    clusters = []
    '''
    define a Cluster class; it is a group of samples
    create k clusters; each cluster should have a label
    assign each sample to the cluster according to its labe
    '''
    return clusters
```

K-Means: a class; please use help to view the manual

- .fit(x) method will run k-means on X
- X is an array like dataset; each row is a list of features of a sample
- .labels_: the labels of each row of X

We need a Cluster class to represent the group of samples!

Reference: sklearn.cluster.KMeans — scikit-learn 1.1.1 documentation