**Machine Learning by Andrew Ng, Stanford University on Coursera**

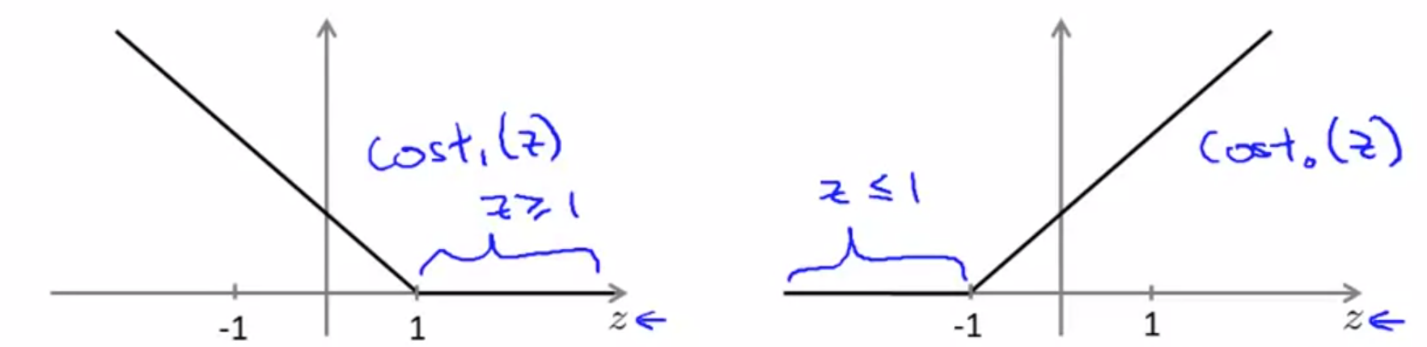
**Notes Part 2, Candice Ooi**

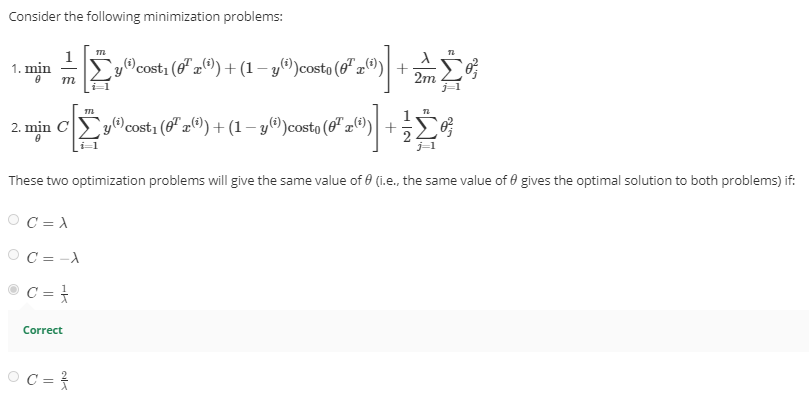
# Week 7 – Support Vector Machines

*Welcome to week 7! This week, you will be learning about the support vector machine (SVM) algorithm. SVMs are considered by many to be the most powerful 'black box' learning algorithm, and by posing a cleverly-chosen optimization objective, one of the most widely used learning algorithms today.*

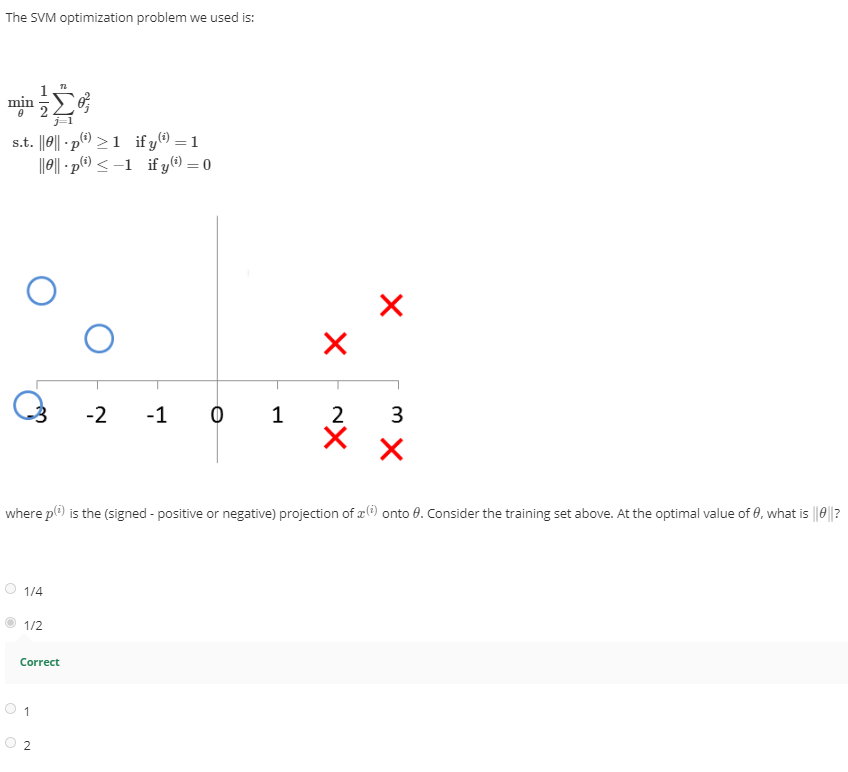
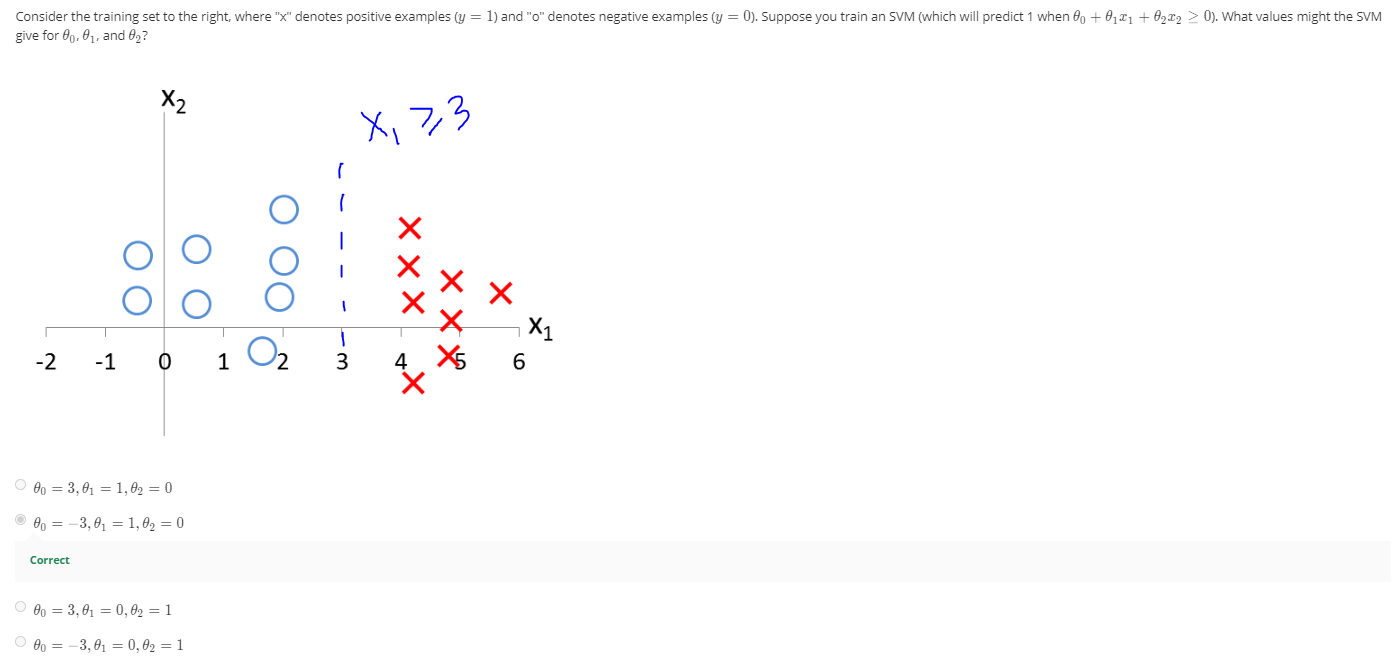
*As always, if you get stuck on the quiz and programming assignment, you should post on the Discussions to ask for help. (And if you finish early, I hope you'll go there to help your fellow classmates as well.)*

## Large Margin Classification – June 17, 2020

Support Vector Machine Cost Function



SVM as Large Margin Classifier. Large margin is the consequence of the optimization.



## Kernels

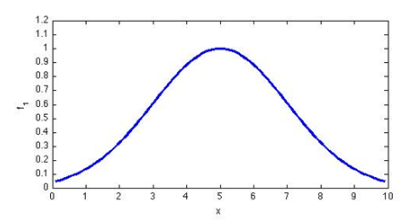
Kernels are similarity functions, e.g. Gaussian kernel

Measures similarity between x and landmark, l, on the chart. X0 = 1

Gaussian kernel as a similarity function that measures the “distance" between a pair of examples, (x(i); x(j)).

The Gaussian kernel is also parameterized by a bandwidth parameter, σ, which determines how fast the similarity metric decreases (to 0) as the examples are further apart.

**If x ~ l1, f1 = 1; If x is far from l1, f1 = 0**. Gaussian kernel parameter, σ2. Affect how steep the slope of the hill is. The larger the parameter, the less steep the hill.



Given a value x and 3 landmarks, l1, l2 and l3, we can compute 3 features of x, f1, f2, f3

Predict y = 1 if Ɵ0 + Ɵ1f1 + Ɵ2f2 + Ɵ3f3 ≥ 0

Set lm = xm

Predict y = 1 if ƟTf ≥ 0

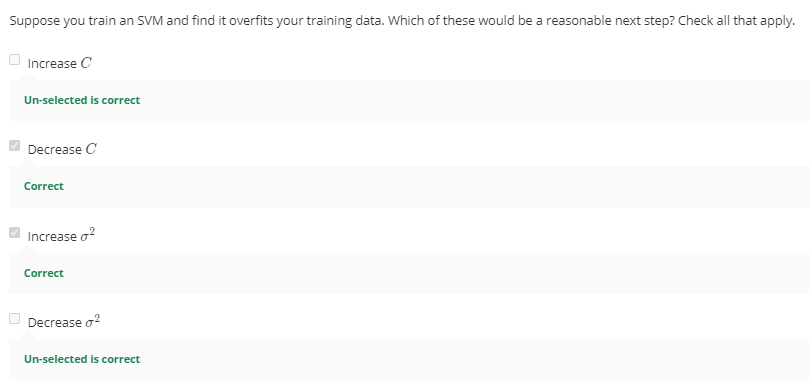
this term is calculated a bit differently: ƟT\*Ɵ (ignoring Ɵ0)or ƟTM\*Ɵ

C (= 1/ λ )

* Large C, small λ, lower bias, higher variance, overfitting
* Small C, large λ, higher bias, lower variance, underfitting

σ2, Gaussian Kernel

* Large σ2, falls off smoothly, higher bias, lower variance, underfitting
* Small σ2 vary less smoothly, lower bias, higher variance, overfitting



## SVMs in Practice

Using an SVM

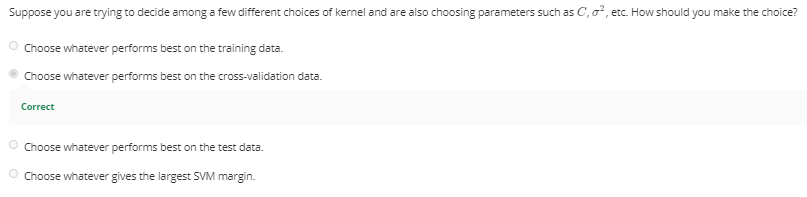
Use SVM software package like liblinear, libsvm, etc to solve Ɵ.

You need to specify C and kernel.

* Linear Kernel means no kernel. Standard Linear Classifier
  + Predict y = 1 if ƟTx ≥ 0, i.e.: Ɵ0 + Ɵ1x1 + Ɵ2x2 + … + Ɵnxn ≥ 0
  + Use it when n (# features/dimension) is large, m (# training data) is small to prevent overfitting
* Gaussian Kernel
  + Need to choose σ2
  + Use it when n is small and/or m is large to fit a more complex non-linear decision boundary.
  + Landmarks are training examples too.
  + **Perform feature scaling before using Gaussian Kernel**
* Polynomial kernel
  + k(x,l) = (xTl + constant)degree , e.g.: (xTl)2 , (xTl)3 , (xTl + 1)2 df
  + performs worst than Gaussian, less used
  + x and l is strictly non-negative
* More esoteric kernels, String kernel (for texts), chi-square kernel, histogram intersection kernel, etc

The kernels need to satisfy Mercer’s Theorem to ensure SVM pacakages’ optimizations run correctly.

Logistic Regression



Many SVMs have built-in multi-class classification.

Otherwise, use one-vs-all method. Train K SVMs, one to distinguish y = I from the rest for I = 1, 2, … K, find Ɵ1 for y = 1, Ɵ2 for y = 2 , …, ƟK for y = k. Pick class i with largest ƟTx

**Logistic Regression VS SVM**

* Very similar. Use kernel to learn complex non-linear function.
* If n is large relative to m (n = 10,000, m = 10 - 1000), use logistic regression or SVM with linear kernel.
* If n is small, m is intermediate (n = 1 – 1000, m = 10 – 50,000), use SVM with Gaussian kernel.
* If n is small, m is large (n = 1 – 1000, m = > 50,000 - millions), create/add more features, use logistic regression or SVM with linear kernel. Gaussian kernel will be too slow with this large training set.
* Neural network likely to work well for most settings, but can be slow to train.
* Good SVM will always find a global minimum. But local minimum is a problem for NN

# Week 8 – Unsupervised Learning

*Hello all! I hope everyone has been enjoying the course and learning a lot! This week, you will be learning about unsupervised learning. While supervised learning algorithms need labeled examples (x,y), unsupervised learning algorithms need only the input (x). You will learn about clustering—which is used for market segmentation, text summarization, among many other applications.*

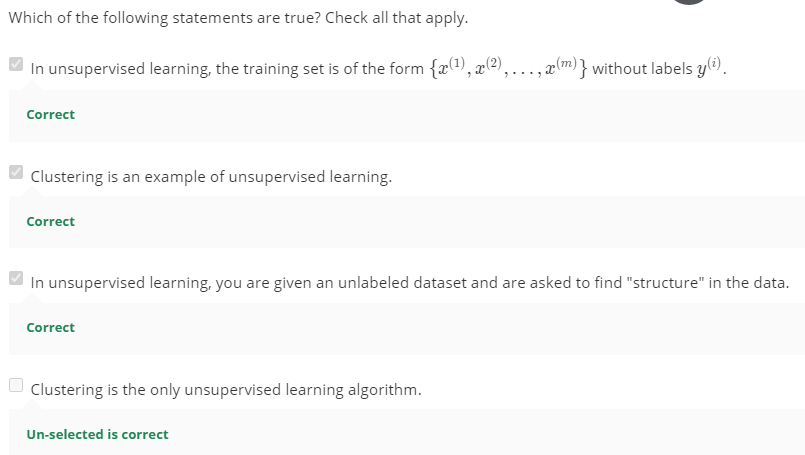
*We will also be introducing Principal Components Analysis, which is used to speed up learning algorithms, and is sometimes incredibly useful for visualizing and helping you to understand your data.*

*As always, if you get stuck on the quiz and programming assignment, you should post on the Discussions to ask for help. (And if you finish early, I hope you'll go there to help your fellow classmates as well.)*

## Clustering – June 21, 2020

**Clustering is an Unsupervised Learning.**

Good for market segmentation (example of non-separated clusters), social network analysis, organize computer clusters, astronomical data analysis



**K-means algorithm**

A type of clustering, most popular and widely used algorithm.

Start with 2 cluster centroids in random space. K = 2

K-means is an iterative algorithm, cluster assignment (based on distance of data point to centroids) and move centroids (mean of all points in the same cluster).

We need to decide K, the # of clusters, and gives the training set X.

Omit x0 = 1 in unsupervised learning. X is in Rn vector.

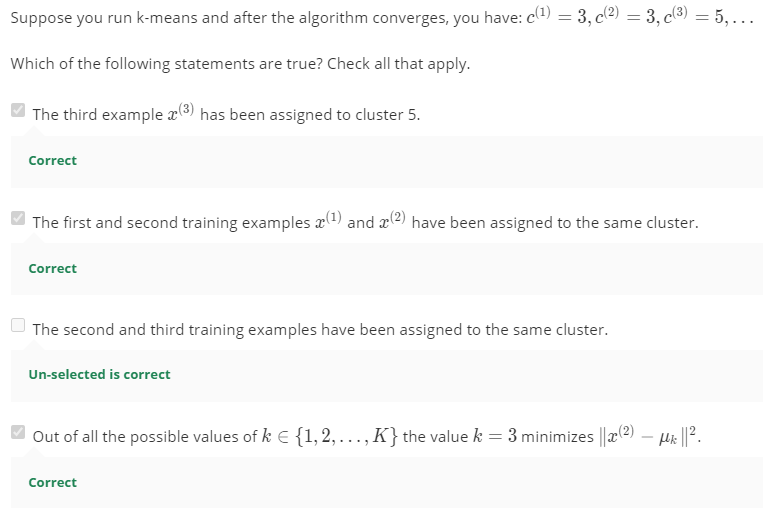
Algorithm:

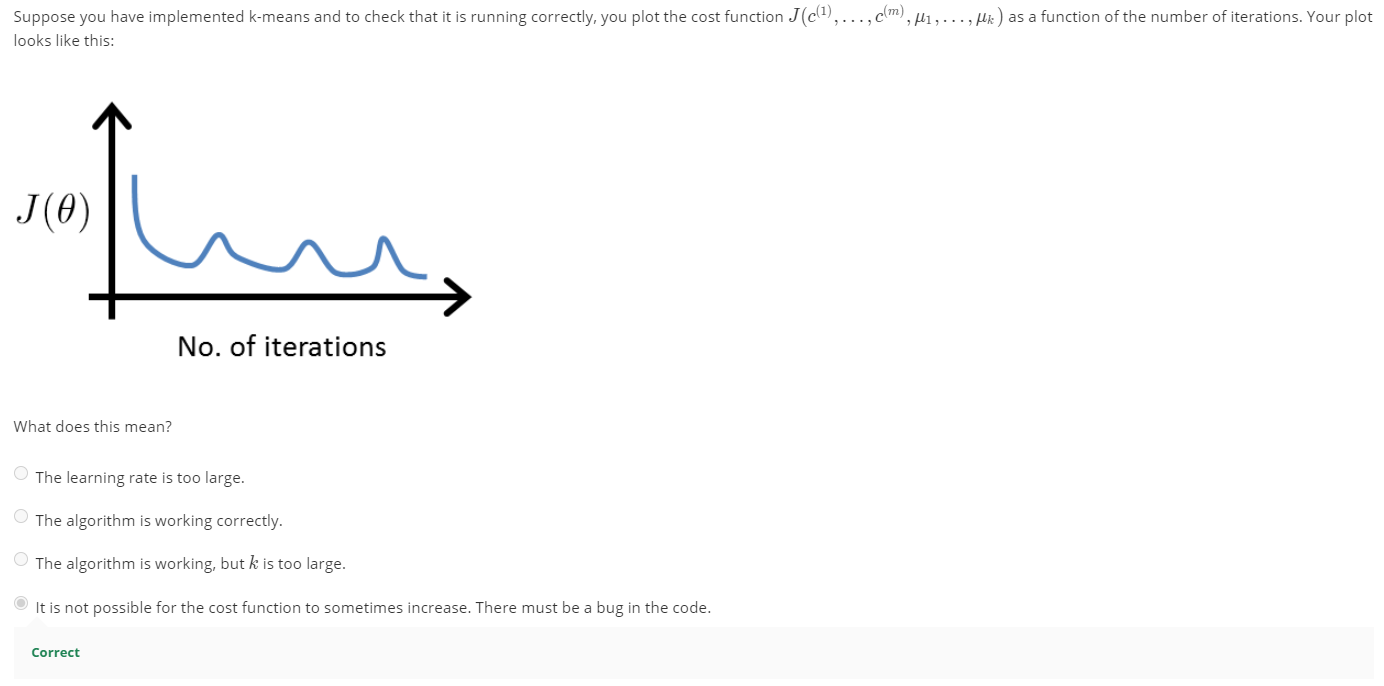
* Randomly initialize K cluster centroids, µ1, µ2, …, µK
* Repeat
  + Cluster assignment step
    - ci := index from 1 to k of cluster centroid closest to xi by measuring distance ||xi- µ1||. Find k that minimizes distance and set as ci.
    - Minimizing J with respect to c1, …, cm, while holding µ1, …, µk fixed.
  + Move centroid
    - µk := average mean of points assigned to cluster k
    - Minimizing J with respect to µ1, …, µk and keeps iterating

Remove cluster centroid with no cluster, so k-1 means algorithm instead.

**Optimization Objective**

For example, a point xi is labeled as cluster 5. So ci = 5, µci = µ5





**Random Initialization**

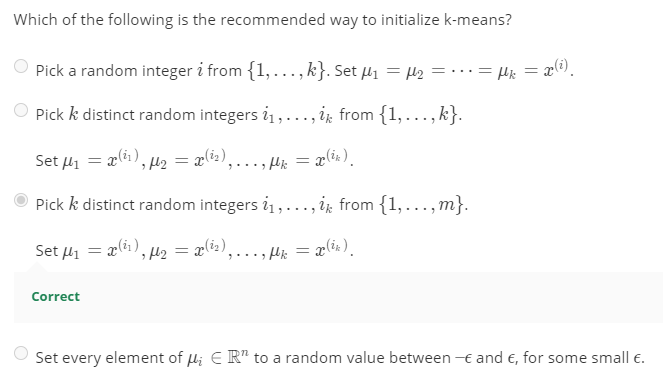
How to initialize K-means

K (# of centroids) should be < m (# of training data).

Pick K training examples and set µ1, …, µk equal to these examples in the first iteration.

How to avoid local optima

Try random initialization multiple times (50 – 1000 times), run k-means and compute cost function and choose the clustering that gives the lowest cost.



**Choosing the # of clusters**

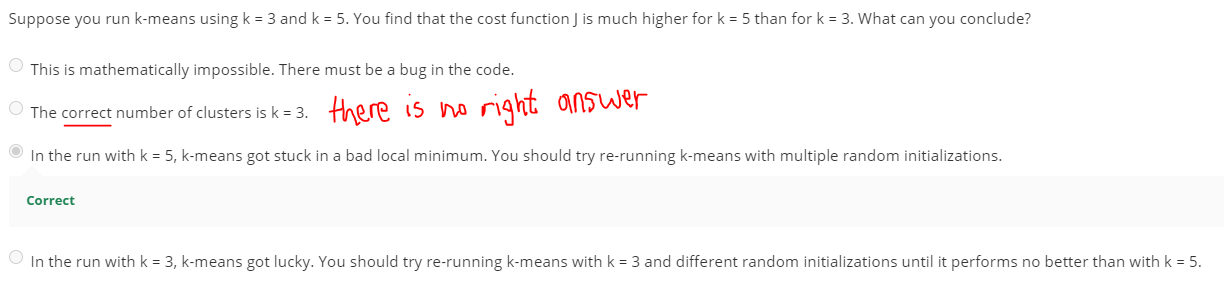
Most commonly choose by hand, visually/human input & insights. There is no right answer to how you cluster.

Some people use **Elbow method**.

Choose the elbow of the chart of J against K. Where the distortion rate starts to peter out drastically.

Pitfall: the chart is too gradual and there is no elbow to choose K from.

Or, evaluate K-means based on a metric for how well it performs for that later purpose, and what purpose the clustering is for.



## Dimensionality Reduction

*In this module, we introduce Principal Components Analysis, and show how it can be used for data compression to speed up learning algorithms as well as for visualizations of complex datasets.*

## Motivation – Dimensionality Reduction

**Data compression**

Dimensionality Reduction, an unsupervised learning algorithm. Two features measuring the same thing in different units (cm vs inch). Reduce redundancy to improve efficiency.

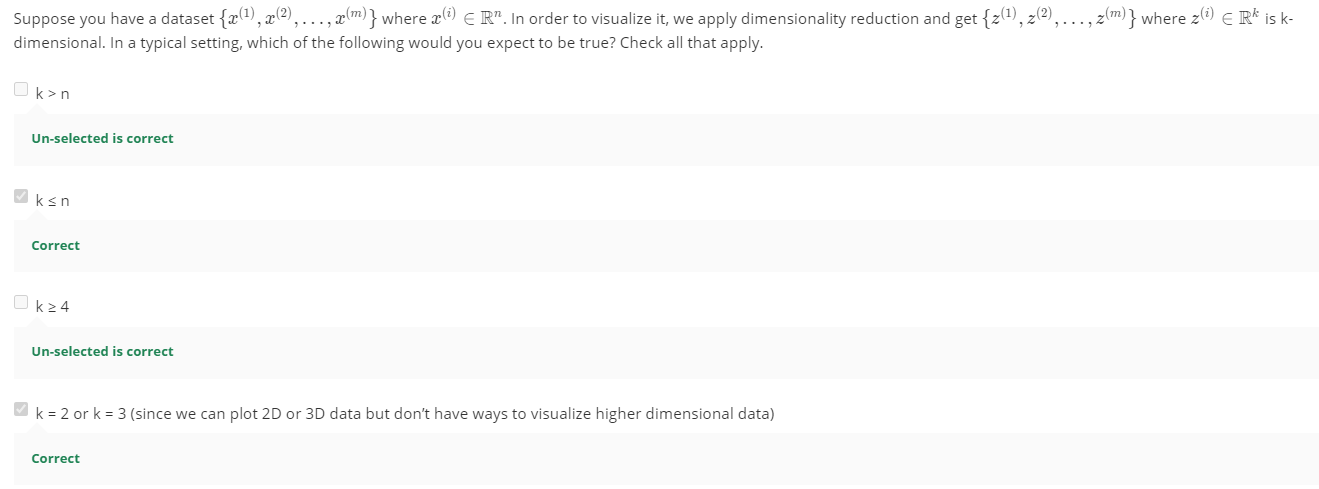
Project data to an imaginary line (a new axis) which reduces data from 2D to 1D.

Dimensionality reduction is good for **data compression and data visualization**.

**Data visualization**

To see data in smaller dimensions 2D – 3D.





## Principal Component Analysis (PCA) – June 28, 2020

**Problem Formulation**

PCA is the most popular dimensionality reduction, unsupervised algorithm.

Must do mean normalization first, mean = 0, before performing PCA.

Project all points into one line where the projection error (distance between the point and the line) is **minimized**.

PCA can give you + projected vector or – projected vector, +ve or –ve doesn’t matter.

In linear algebra terms, is to project the data to the linear subset spanned by this set of k vectors.

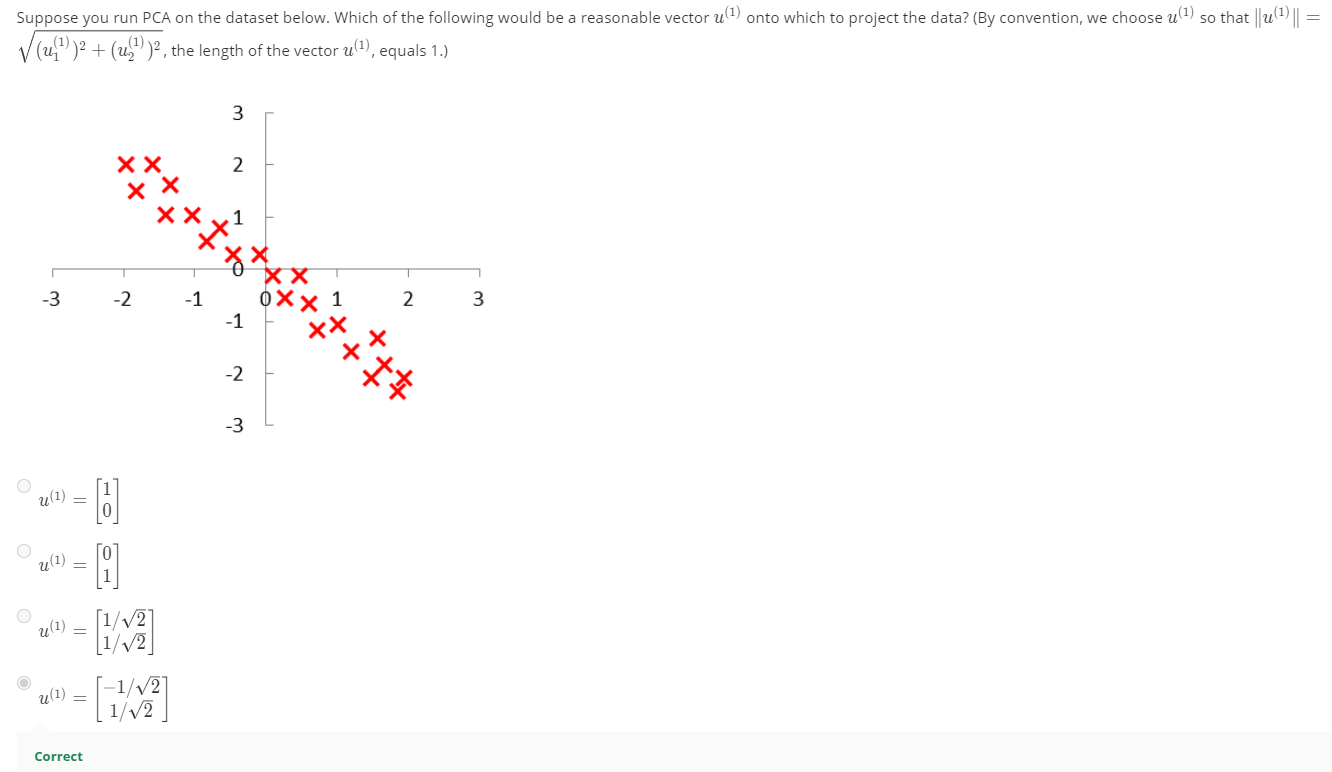
Find k directions to project the data while minimizing projection error.

**PCA is not linear regression.**

In linear regression, we fit a straight line to minimize square error. They are vertical distance to the regression line. We are trying to use x to predict y.

In PCA, we fit the straight line to minimize the orthogonal distance. They are orthogonal distance to the PCA line. There is no special variable that we are trying to predict. All features (x1, x2) are treated equally.





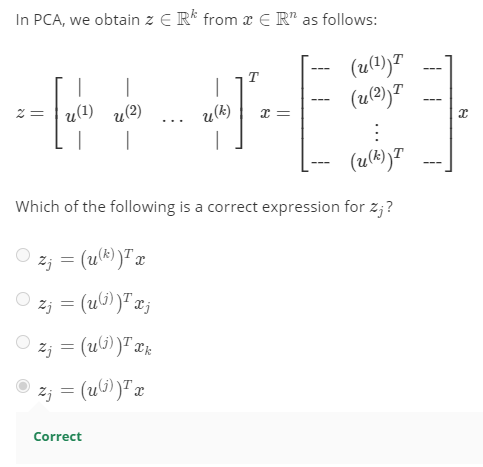
**PCA Algorithm**

1. Pre-processing

Perform mean normalization of training set

Subtract X from the mean to make each feature with zero mean.

Perform feature scaling depending on the dataset, if needed. Should do it if features take on different ranges of value

1. To reduce dimension, compute covariance matrix, upper case sigma
2. Compute eigenvectors of matrix ∑, n x n matrix since x(i) is n x 1 matrix.

*[U, S, V] = svd(Sigma);*

singular value decomposition or eig(Sigma). Svd() is numerically stable for covariance matrix.

Spit out 3 matrices, U, S and V.

We only need U, n x n matrix. Columns of U will be vectors u1, u2, … um.

U is eigenvectors, S is diagonal matrix of eigenvalues.

1. Take the first k vectors in U to reduce dimensions. u1, u2, … uk. Ureduce.

*Ureduce = U(:, 1:k);*

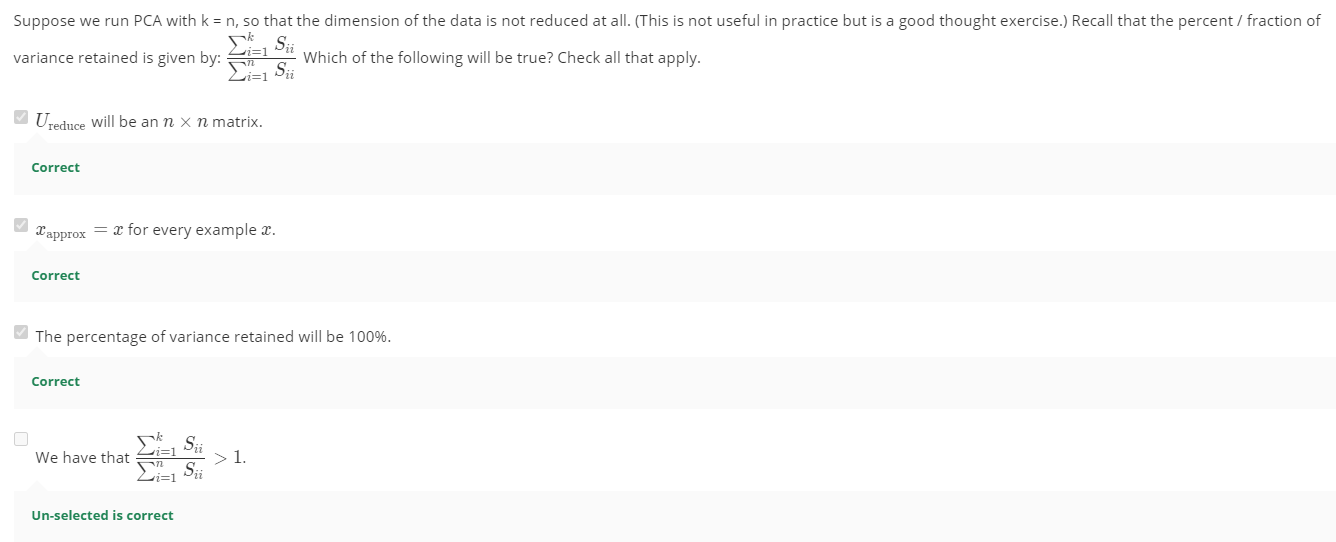
*z = UTreduceX;*

## Applying PCA

**Reconstruction from compressed representation**

Go back to uncompressed dimension

Xapprox = Ureduce\*z; Ureduce is n x k, z is k x 1, X is n x 1



**Choosing the number of principal components, k**

PCA tries to average squared projection error

Total variation is the average of total length from the origin

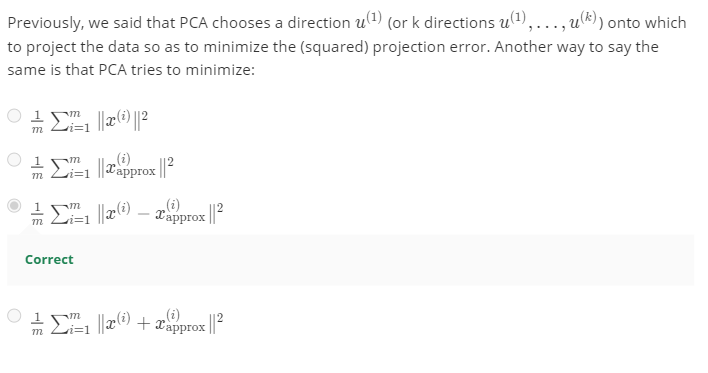
Choose k so that 99% of variance is retained. (85% - 99% commonly used, 95% - 99% is the most popular)

**Choosing k**

When calling *[U, S, V] = svd(Sigma)*;

S gives a diagonal square matrix of eigenvalues.

For given k, keep increasing k until below condition is fulfilled.



**Advice for applying PCA**

PCA can speed up time of a supervised learning algorithm.

1. Extract inputs, x. Leave y aside.
2. Perform PCA and get z.
3. You now have a new training set with z and y.
4. Fit it to a learning algorithm such as neural network, logistic regression.

**Mapping of x to z on training set only. Obtain Ureduce on training set only. Apply the mapping to CV and test set.**

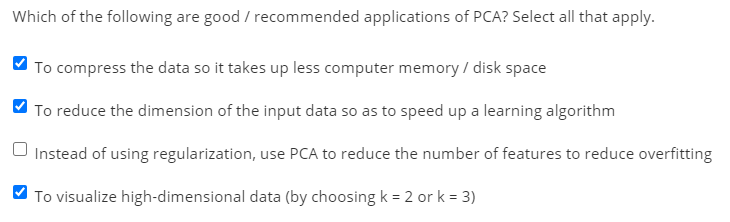
Applications of PCA

* Compression
  + Reduce memory to store data
  + Speed up learning algorithm
  + Choose k by % of variance retained.
* Visualization
  + Plot 2D or 3D data usually k = 2 or k = 3

**Do not use PCA to prevent overfitting!** Reduce the number of features, less likely to overfit is the concept of why people do it. **Use regularization instead**. PCA doesn’t use y, it throws away y to reduce the dimension of data without knowing what y is.

Ask: how about doing it without PCA?

Do it without PCA first. Only consider implementing PCA if it doesn’t work.



# Week 9a – Anomaly Detection – July 1, 2020

*Hello all! I hope everyone has been enjoying the course and learning a lot! This week, we will be covering anomaly detection which is widely used in fraud detection (e.g. ‘has this credit card been stolen?’). Given a large number of data points, we may sometimes want to figure out which ones vary significantly from the average. For example, in manufacturing, we may want to detect defects or anomalies. We show how a dataset can be modeled using a Gaussian distribution, and how the model can be used for anomaly detection.*

*We will also be covering recommender systems, which are used by companies like Amazon, Netflix and Apple to recommend products to their users. Recommender systems look at patterns of activities between different users and different products to produce these recommendations. In these lessons, we introduce recommender algorithms such as the collaborative filtering algorithm and low-rank matrix factorization.*

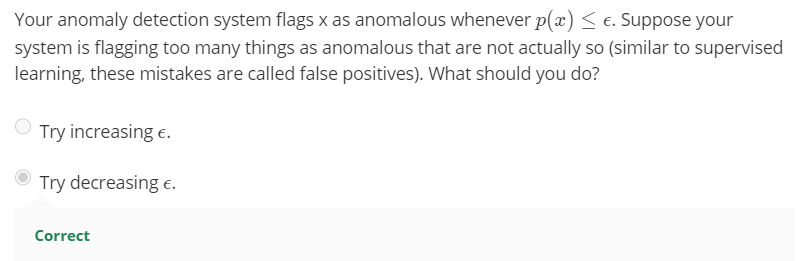
*As always, if you get stuck on the quiz and programming assignment, you should post on the Discussions to ask for help. (And if you finish early, I hope you'll go there to help your fellow classmates as well.)*

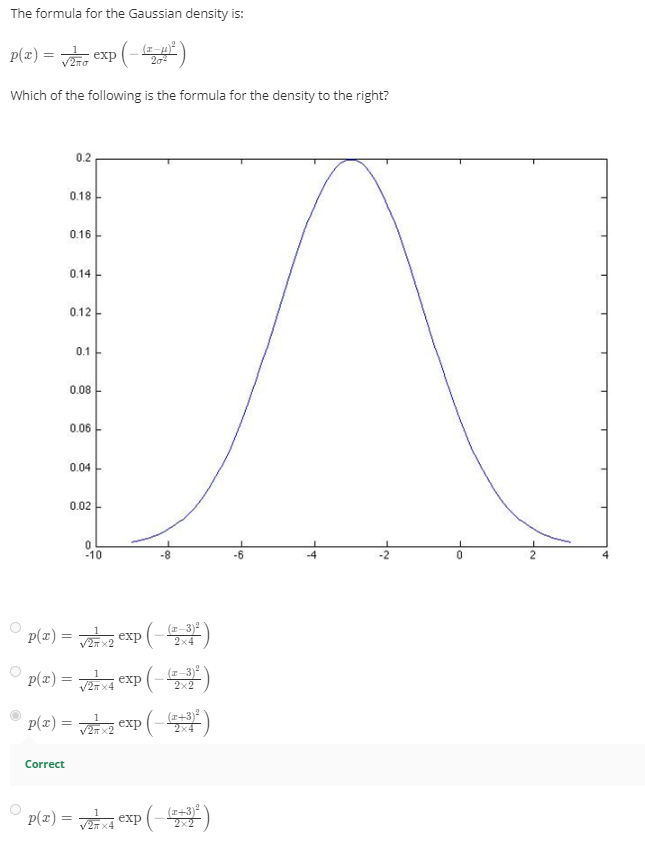
## Density Estimation

**Anomaly detection: Problem Motivation**

Model p(x) probability from data and identify unusual users by checking which data have **p(x) < ɛ,** is suspicious.

Typing speed (robot vs human) for fraudulent behavior to find unusual users.

Applications: Fraud detection, manufacturing, monitoring computers in a data center.

**Gaussian Distribution / Normal distribution (Bell curve)**

x is a distributed Gaussian with mean µ and variance σ2.

µ is where the curve is centered, σ is the width of curve. Area under the curve must be 1.

The parameters are maximum likelihood estimation.

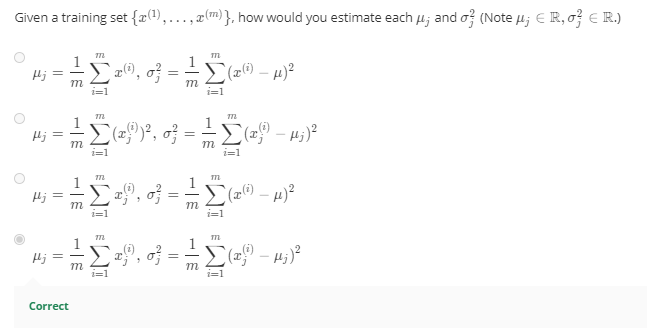
m-1 instead of m for denominator. In ML, we used m. m or m-1 makes very little difference.

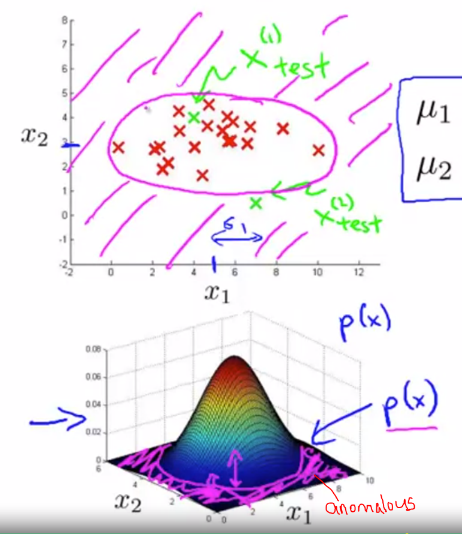
**Algorithm**

Density estimation

Assume each x (x1, x2, …, xn) is distributed (~) on a Gaussian distribution.

P(X) = p(x1; µ1, σ12) \* p(x2; µ2, σ22) \* … \* p(xn; µn, σn2), product notation.



Anomaly detection algorithm

1. Choose features x that might be indicative of anomalous examples (very large or very small values)
2. Fit model p(X) on training set parameters, μ and σ2for j from 1 to n.
3. Compute p(x), a product of probabilities for each feature.
4. Anomaly if p(x) < ɛ

In a 3D surface plot, anomalies lie around the flat base.

## Building an Anomaly Detection System

**Developing and evaluating an anomaly detection system**

Training set: unlabelled (normal data 60%), assumed non-anomalous (y = 0), some anomalies are acceptable.

Cross validation (normal data 20% + anomalous data 50%) & Test set (normal data 20% + anomalous data 50%) should have some anomalies (y = 1)

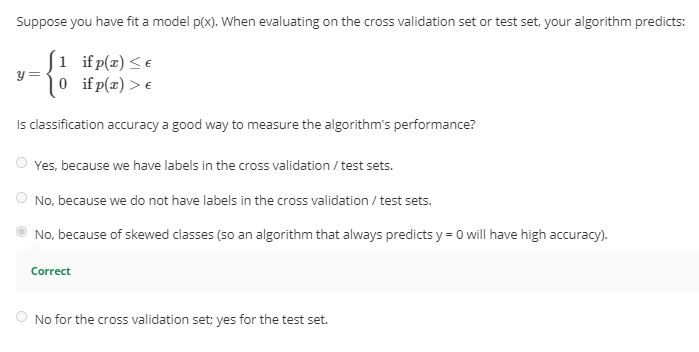
**Predict y on CV/test set (y = 1 if p(x) < ɛ or y = 0 if p(x) ≥ ɛ)**

Classification accuracy will not be accurate because there is a skew of more normal data than anomalous data.

Possible evaluation metrics:

* True/false +ve, true/false -ve
* Precision/Recall
* F1-score

Use CV set to choose ɛ. Trial and error and **pick ɛ that maximizes F1-score**. Then do a final evaluation on the test set using the model with the ɛ selected.



**Anomaly detection VS Supervised Learning**

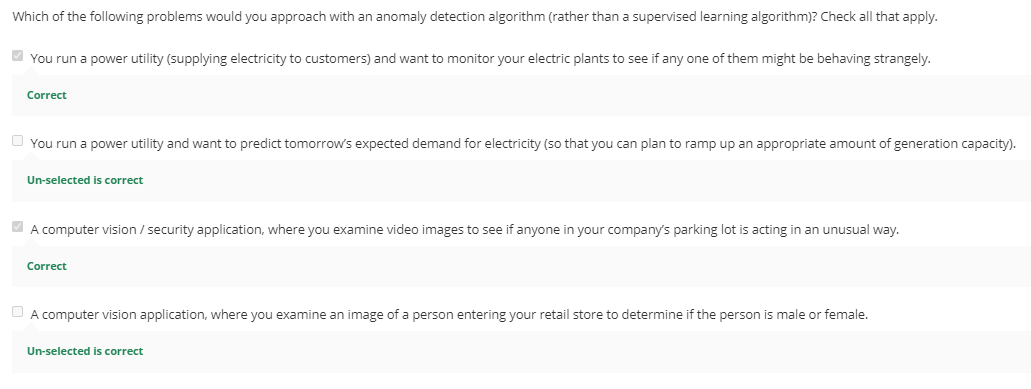
When you should use anomaly detection or supervised learning.

Anomaly detection when:

* Very small number of positive/anomalous examples (y = 1). 0 – 20 examples.
* Large number of negative/normal examples (y = 0)
* Many different kinds of anomalies that’s hard for algorithm to learn from positive examples.
* Possibility of having future anomalies that are never seen before.
* Eg: Manufacturing (Airplane failure), fraud detection, monitoring machines in a data center
* Fraud detection sometimes can be switched to supervised learning if anomalies are too ubiquitous.

Supervised learning (linear regression, etc) when:

* Large number of positive and negative examples.
* Future positive examples likely to be similar to the training set.
* Eg: Email spam classification, weather prediction, cancer classification



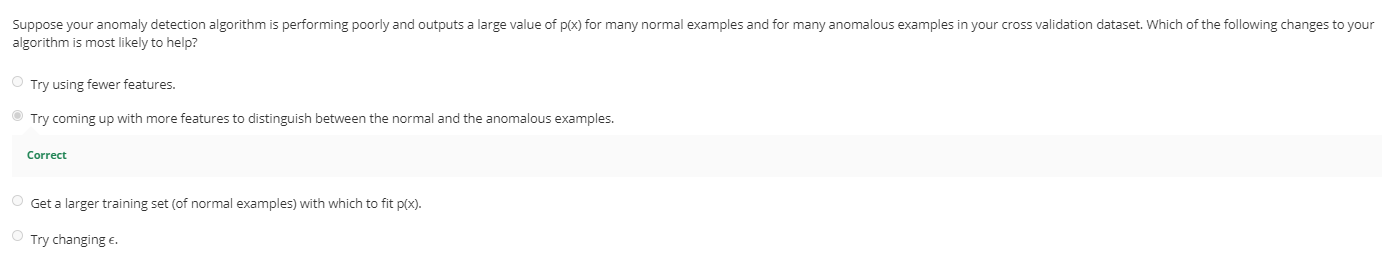
**Choosing What Features to Use**

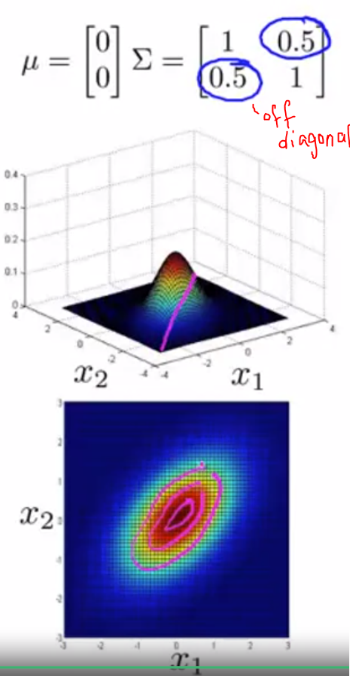
Plot histogram to see it looks Gaussian. If it doesn’t look Gaussian, transform it (log, exponential, etc) to look more Gaussian, it will still work fine if you don’t transform.

Transformation: log(x1) / log(x2 +1)/ log(x2 +c) / x3^1/3

Error Analysis for anomaly detection

Choose features that might take on unusually large or small values. Can create a new feature that derives from existing feature to help better for anomaly detection.





## Multivariate Gaussian distribution

Model p(x) in one go instead of separately. To improve the algorithm in detecting anomalies. Watch video Week9c-1 for more info.

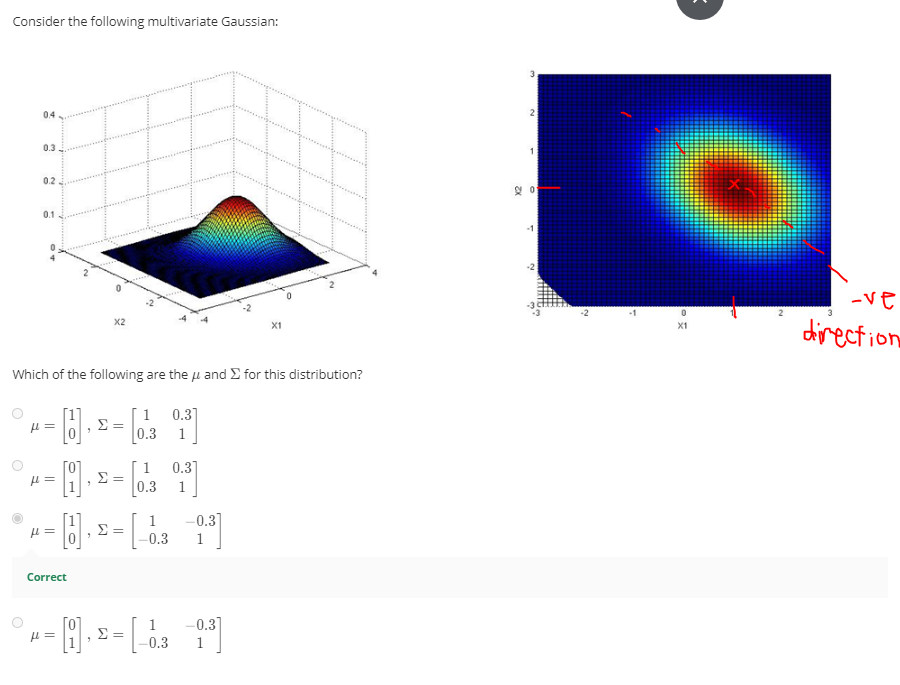
Multivariate Gaussian distribution allows you to capture when you expect 2 different features to be positively and/or negatively correlated.

Parameters: µ a constant, ∑, covariance matrix **n x n** (identity/diagonal matrix) not summation.

Bigger ∑, wider base, take wider range of values, lower top in a 3D plot. The first value in ∑ is for x1, second is for x2, etc. The shape changes depending on the values.

The off diagonal value, is the x1 = x2 line. The bigger these values, the narrower the plot. Negative value x1 = -x2, captures negative correlation.

µ shifts the center of the distribution.



**Anomaly Detection Using Multivariate Gaussian Distribution**

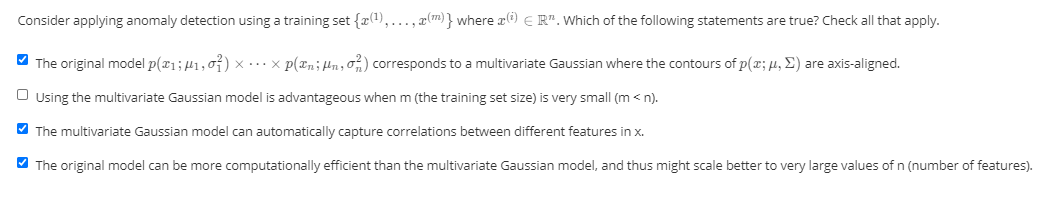
1. Fit model p(x) by setting µ and ∑
2. Given a new example from test set, compute p(x)
3. Flag an anomaly if p(x) < ε

The original model p(x) is a subset of multivariate Gaussian distribution that is axis aligned, be it to x1 or x2. It is the same as the multivariate Gaussian distribution with a constrain, ie: non-diagonal components in ∑ matrix are zeros.

Original model (used more often) VS Multivariate Gaussian

* OG: Need to manually create features to capture anomalies where x1, x2 take unusual combinations of values.
* MG: Automatically captures correlation between features.
* OG is computationally cheaper than MG (need to compute ∑ inverse).
* OG: is fine with small training set.
* MG: compulsory for # of examples >> # of features or ∑ is non-invertible. m >> 10n is a good rule.

If you find out ∑ is non-invertible, you either don’t have m >> n or you have redundant/linearly dependent features.



# Week 9b- Recommender Systems

*When you buy a product online, most websites automatically recommend other products that you may like. Recommender systems look at patterns of activities between different users and different products to produce these recommendations. In this module, we introduce recommender algorithms such as the collaborative filtering algorithm and low-rank matrix factorization.*

## Predicting Movie Ratings - July 5, 2020

**Problem Formulation**

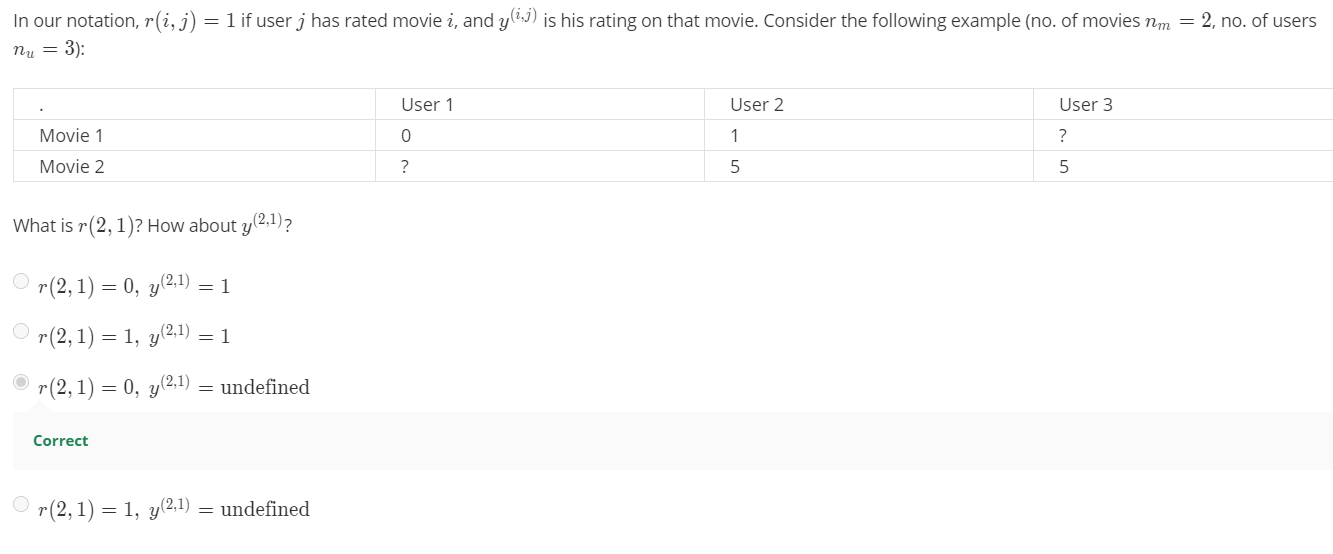
The algorithm learn what features to use.

**nu = # of users, j = jth user**

**nm = # of movies, i = ith movie**

**r(i, j) = 1 if user j rated movie i**

**y(i, j) = rating on that movie i rated by user j**



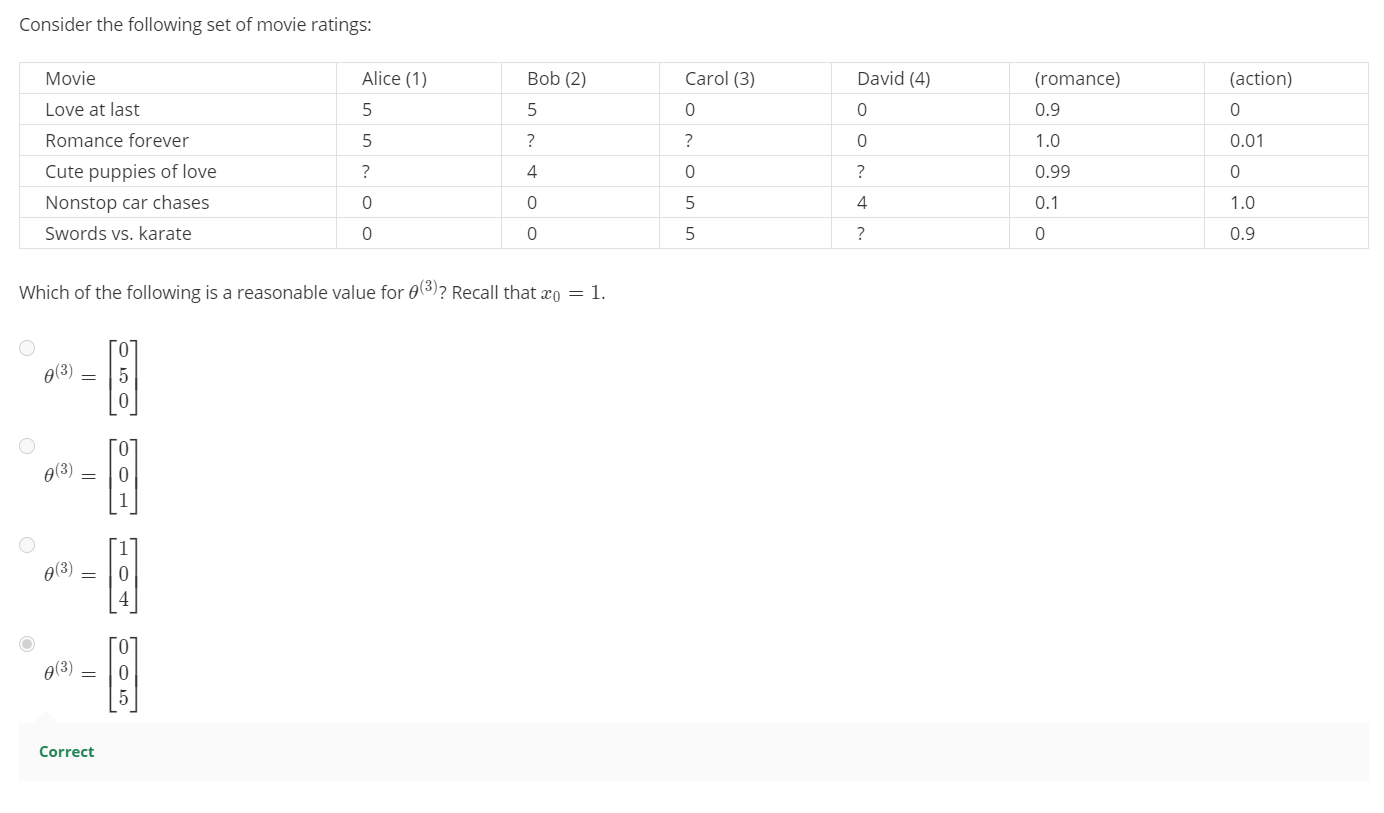
**Content-based recommendation**One approach to recommender systems.

X0 = 1 always. X is a vector for each movie, indicating the genre of the movie e.g.: action, romance, comedy, etc ranging from 0 to 1.

Xi = feature vector for movie i.

n = # of features excluding x0.

Each user has a parameter vector, Ɵj.



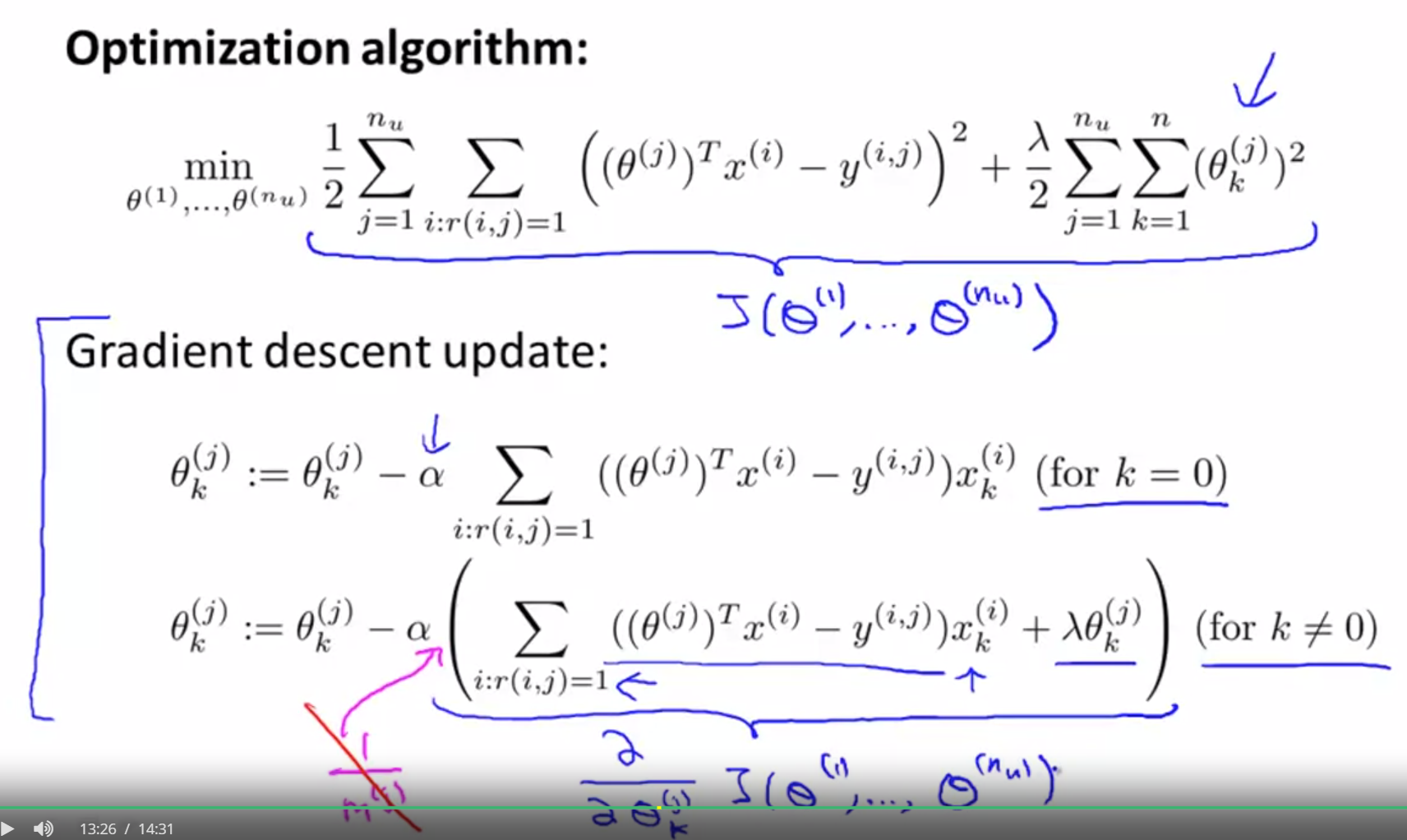
A linear regression problem.

Minimize the squared error term of prediction VS actual Ɵj. Can add a regularization term.

To learn parameter Ɵ for user(s), Optimization algorithm

Gradient Descent

Similar to linear regression, except there is no 1/m term.

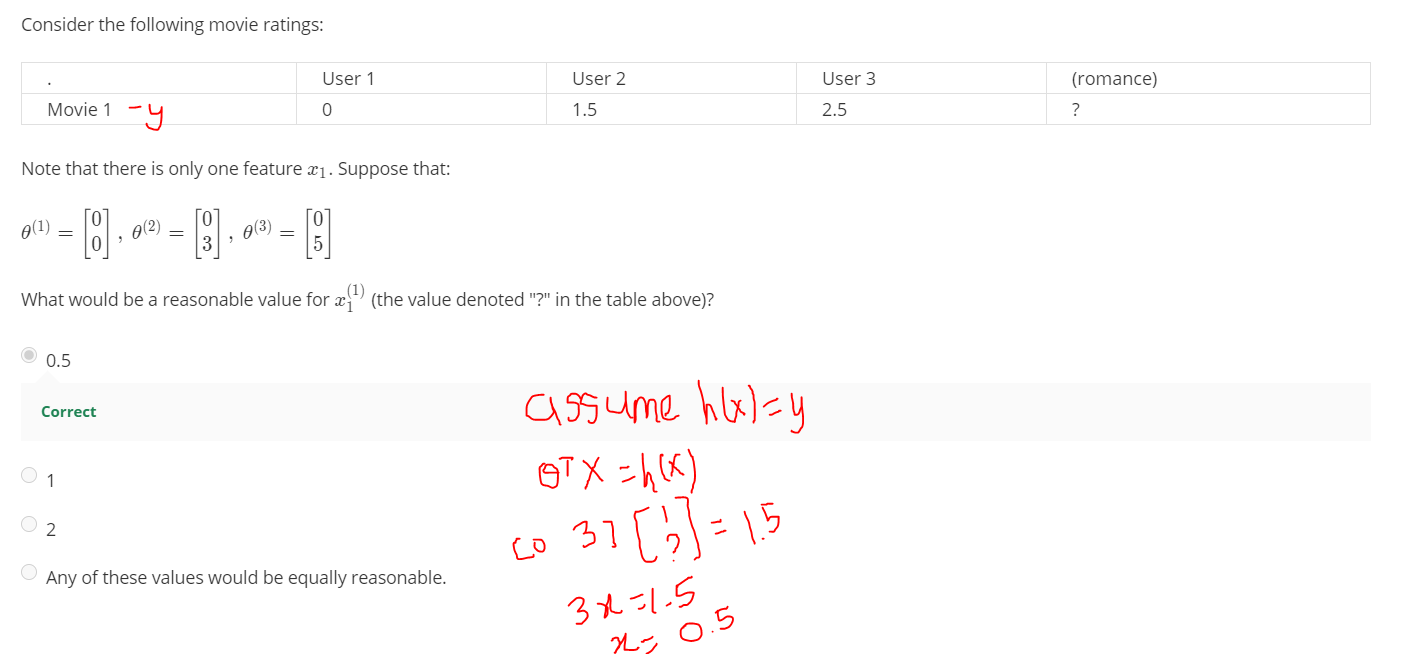


## Collaborative Filtering

Not content based algorithm

**Feature learning.** The algorithm can learn what features x to use.

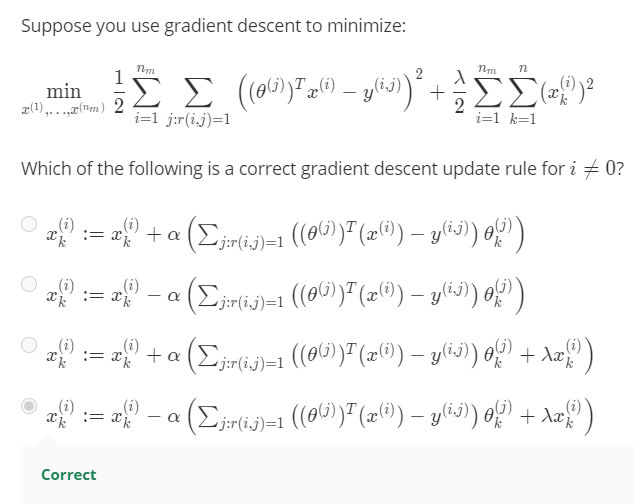
Say you don’t know the genre of the movie. But you know the rating from users. You can determine what the movie genre is based on users’ preference.



Given Ɵ to learn xi for movie i.

Recommender system: you are given xi to estimate Ɵ

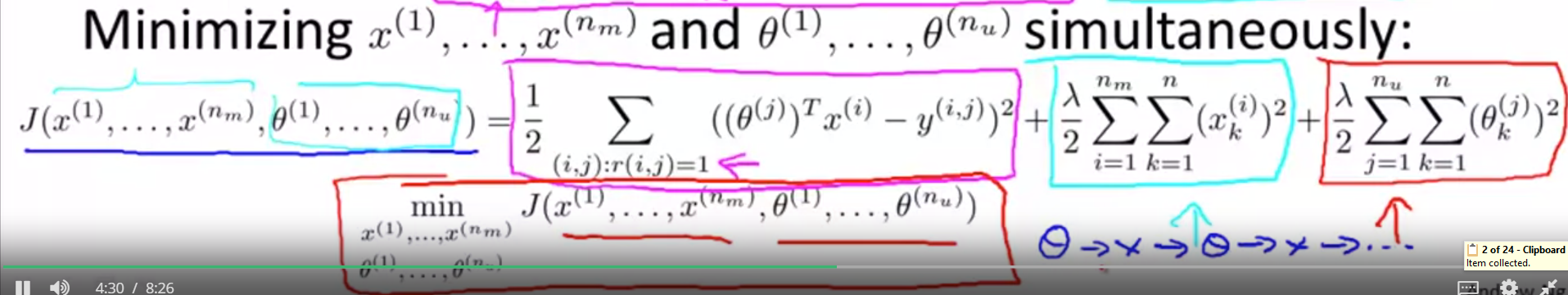
You can guess a random Ɵ to learn xi for movie i,use the x you find to find better Ɵ and keep going.



**Collaborative Filtering Algorithm**

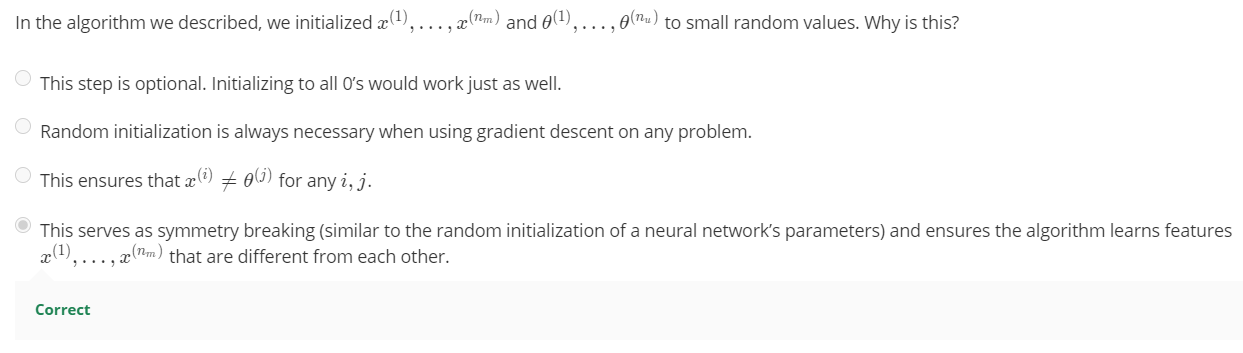
Optimization objective is to minimize x and Ɵ simultaneously.

X0 is omitted in this algorithm because we are learning all features, we don’t need to hard code x0.



Algorithm:

1. Initialize to small random values
2. Minimize using gradient descent or advanced optimization algorithm.
3. For a user with parameters Ɵ and a movie with learned features, x, predict a star rating of ƟTx.



## Vectorization: Low Rank Matrix Factorization

**H(x) = XƟT,** low rank matrix factorization

Use learned features to find related movies.

To find movies j related to movie I, by finding

**Small ||x(i) – x(j) ||,** the smaller the value the more similar they are.

**Mean Normalization**

For users who have not rated any movie

Compute average rating for each movie in a column vector µ.

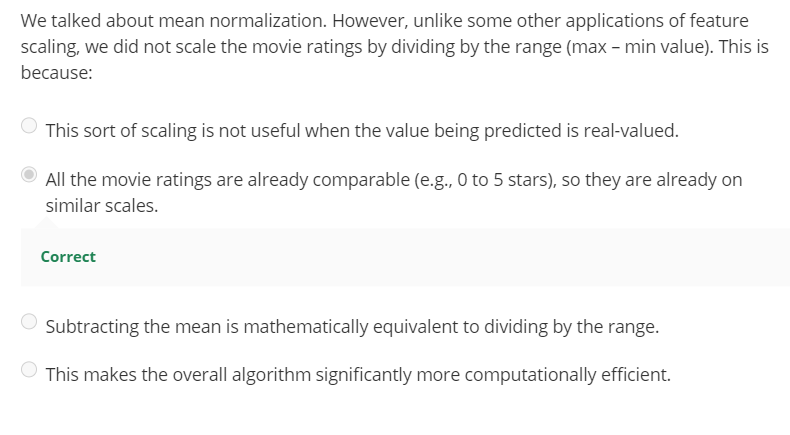
Y is a matrix of ratings (row) by all users (columns)

Transform Y into Y - µ.

This way to normalize it to have 0 for each movie rating.

Use the transformed data to learn Ɵ and x.

**Prediction = ƟjTxi + µi**



# Week 10 – Large Scale Machine Learning

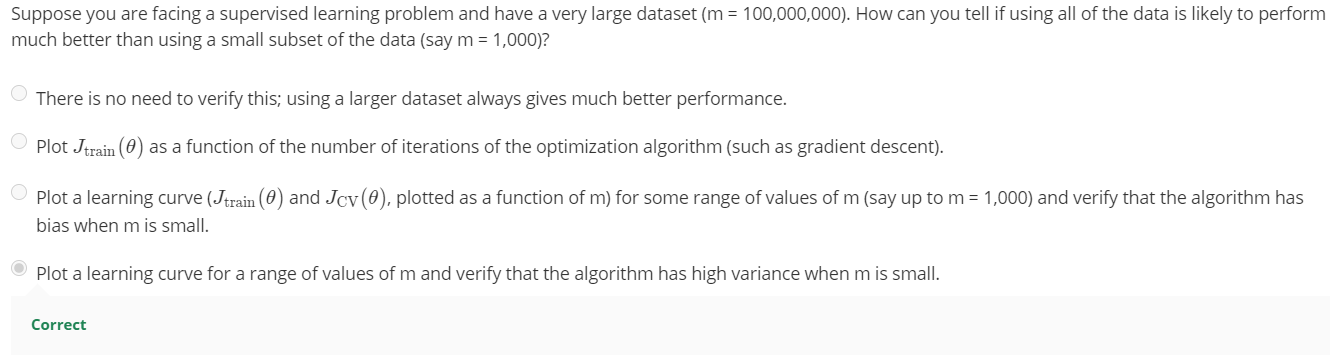
*Welcome to week 10! This week, we will be covering large scale machine learning. Machine learning works best when there is an abundance of data to leverage for training. With the amount data that many websites/companies are gathering today, knowing how to handle ‘big data’ is one of the most sought after skills in Silicon Valley.*

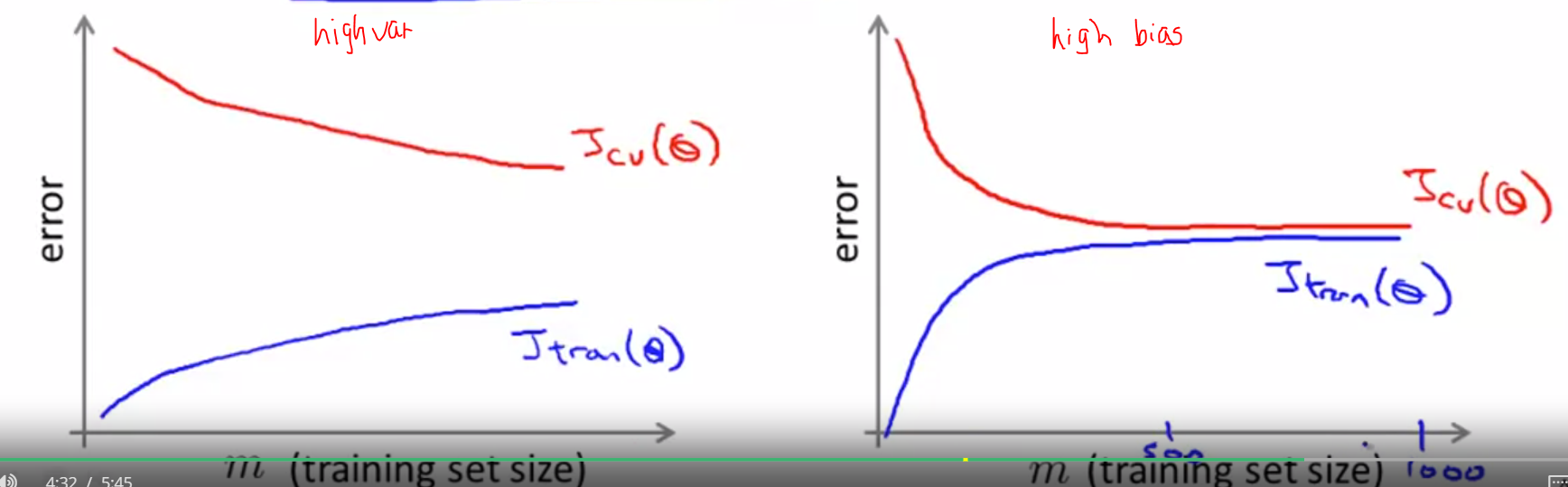
## Gradient Descent with Large Datasets – July 12, 2020

**Learning with Large Datasets**

We've already seen that one of the best ways to get a high performance machine learning system, is if you take a low-bias learning algorithm, and train that on a lot of data.

Plot learning curve to see if the amount of data used matters, whether it’s high variance or high bias.

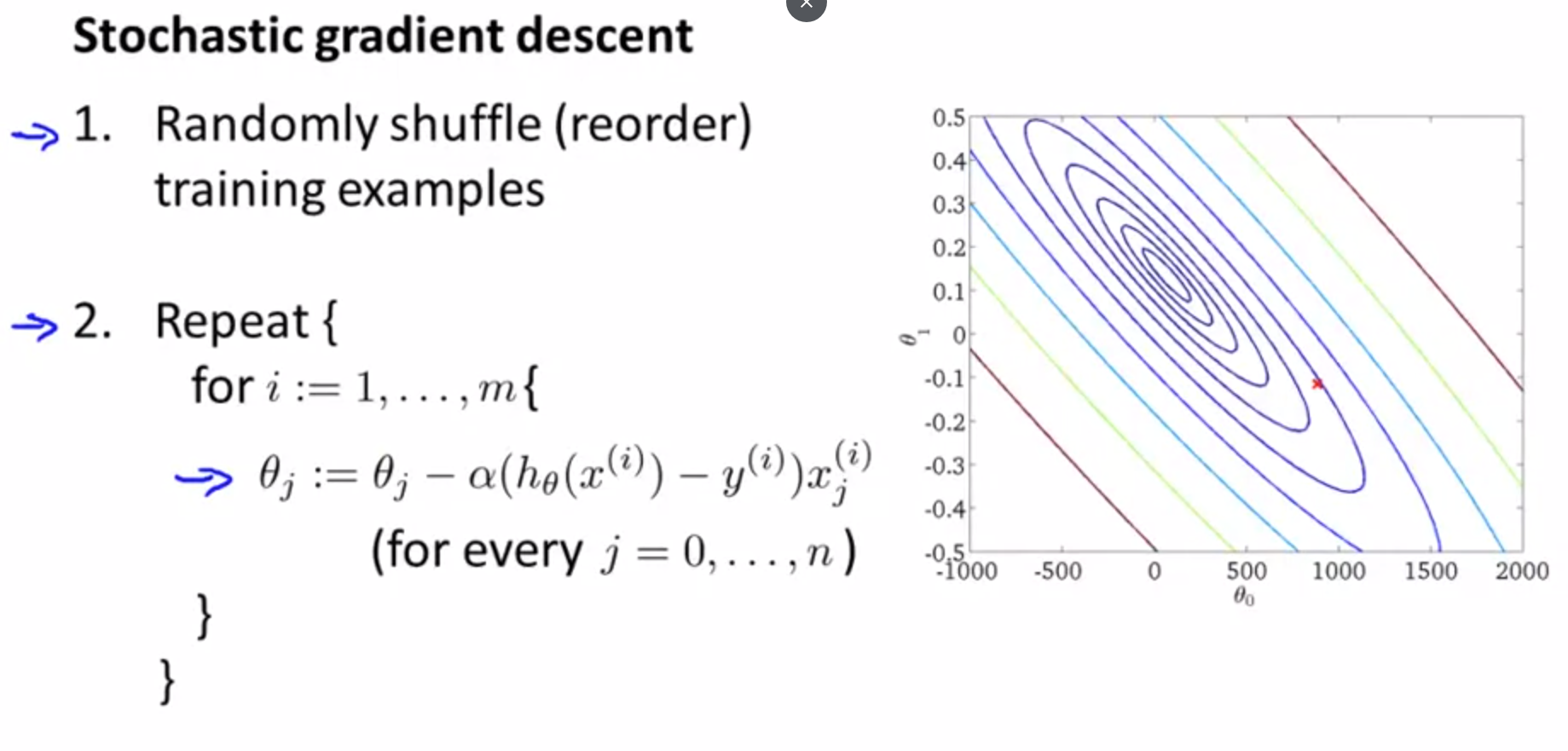




**Stochastic Gradient Descent – July 18, 2020**

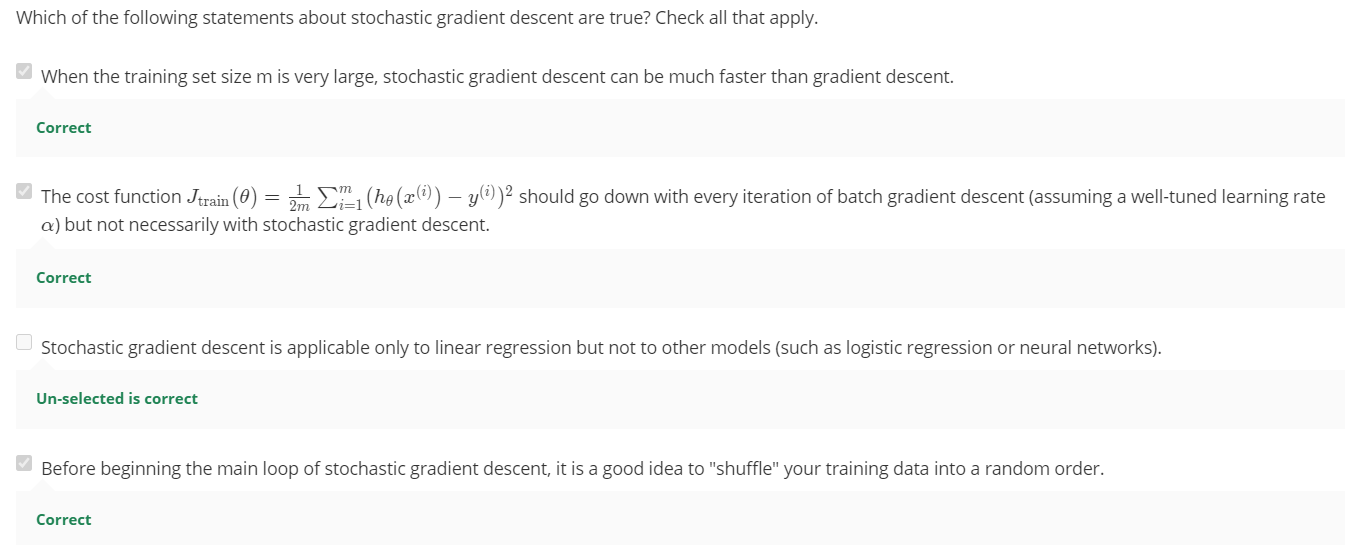
For large training set

**Batch Gradient Descent**, what we learnt in the previous classes. Looking at all training examples, m, at a time. You need all training data into your computer to compute the derivative. Computationally expensive. Think 300 million records, not very feasible, because take 300 M to calculate gradient descent and that is only one step of descent.



Instead of taking the sum, just using one training example to make the descent. Every iteration is faster, trying to fit a training example better. Batch is trying to fit all better.

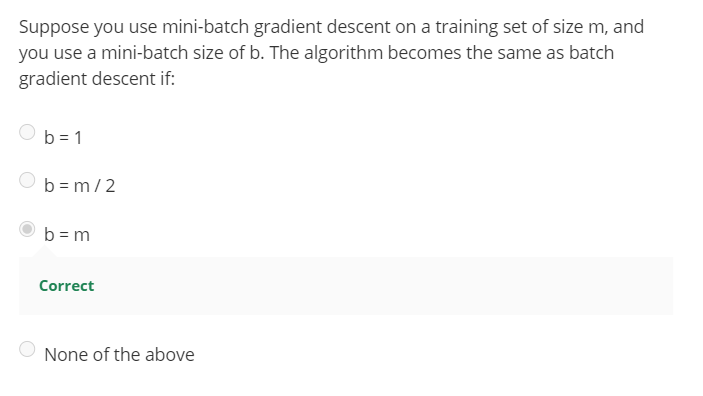
Stochastic has an outer loop, repeat 1 – 10 times. If m is very large 1 can be enough.



**Mini-Batch Gradient Descent**

Can sometimes be even faster than stochastic gradient descent. Use b, mini-batch size, examples in each iteration. b = ranging from 2 to 100, usually 10. Looking at 10 examples at a time, for instance. Instead of summing m in batch, look at the first b, in mini-batch. So, faster.

We want to look at b examples at a time rather than one example at a time in stochastic because of a good vectorized implementation will speed up the process. Downside is having another parameter, b, to fiddle with.



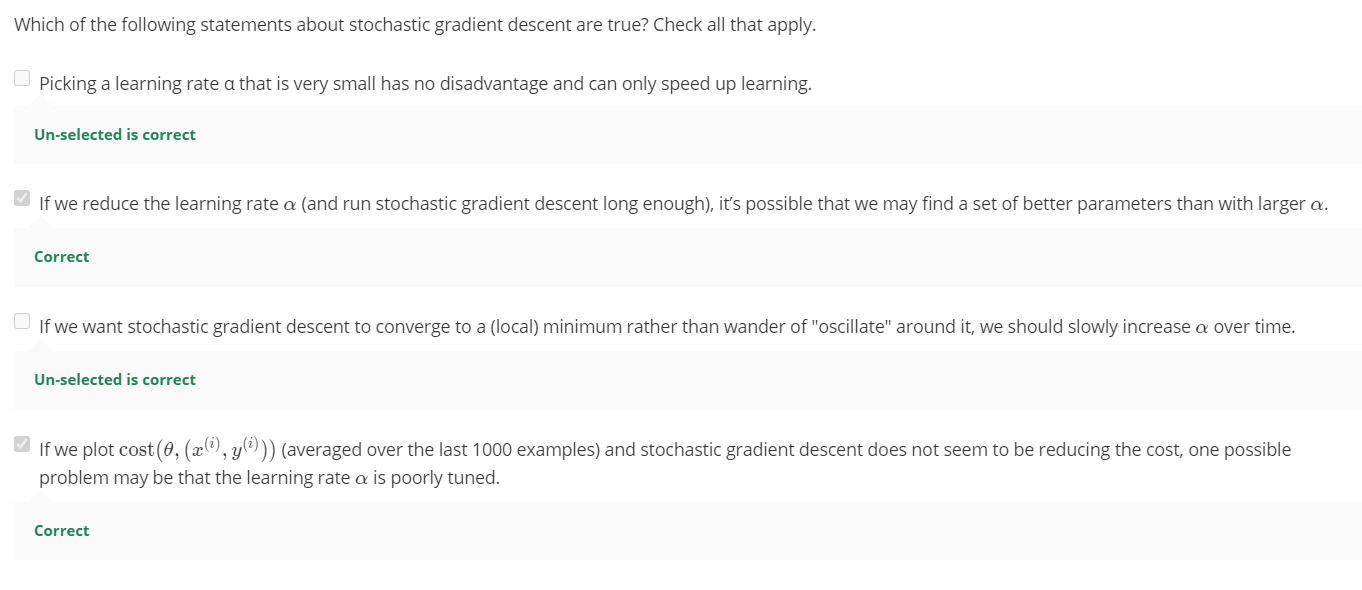
|  |  |  |  |
| --- | --- | --- | --- |
| Gradient Descent | Batch | Stochastic | Mini-Batch |
| Examples Used to descend | Use all m examples in each iteration | Use 1 example in each iteration | Use b examples in each iteration. |
| Speed | Slowest | Faster | Fastest (sometimes) |

**Stochastic Gradient Descent Convergence**

Plot averaged cost function of every 1000 examples processed by algorithm, to see how well the algorithm is doing to check for convergence and if the learning rate alpha is good. The more examples averaged, the smoother the curve gets. You should see a decreasing curve for an algorithm to work properly, i.e.: Converge.

If diverging, use smaller learning rate, α.

Alpha can be held constant. Can slowly decrease alpha over time for theta to converge. Alpha = constant 1/ (iteration # + constant 2). People tend not to do this because it’s more time consuming to fiddle with more parameters.

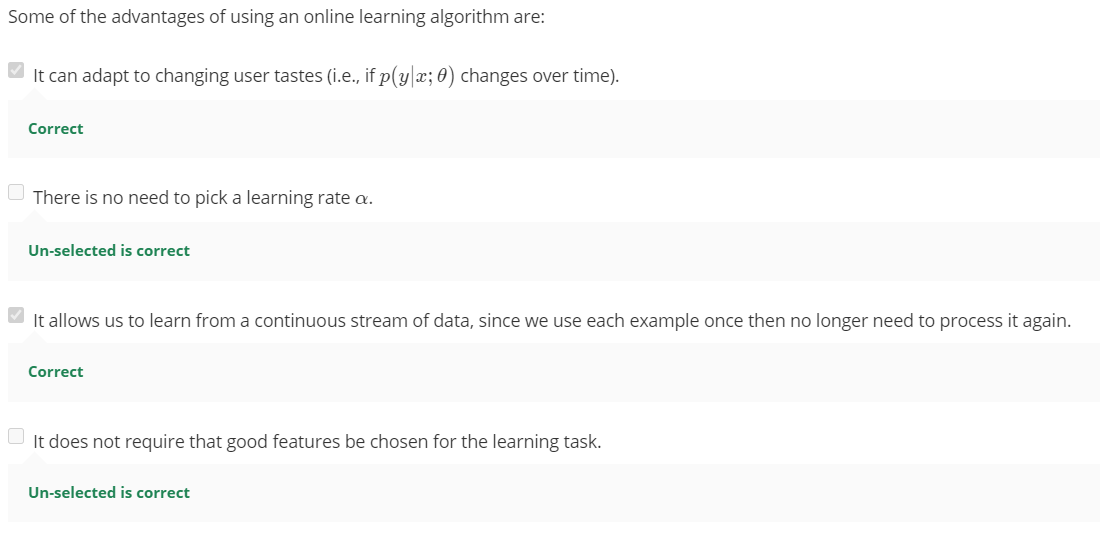


## Advanced Topics

**Online Learning**

CTR: click through rate

Repeat the algorithm forever. Update theta (based on users’ preferences) using the current example x and y. After training, discard the example and never use it again. So no need to index xi and yi.

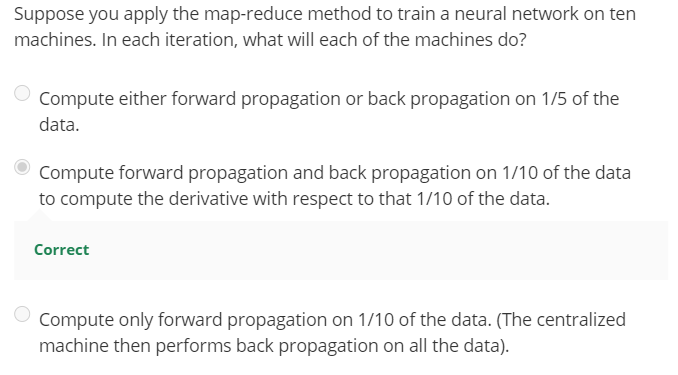


**Map-reduce and data parallelism**

Map-reduce is equally important to stochastic gradient descent.

Split data equally to compute in four separate machines. When complete, combine in a master server and compute/update theta. Able to process more data in a shorter time. Compute sums of functions over the training set.

Map-reduce can be applied on multiple cores in the same machine. Don’t need to worry about network latency.



# Week 11 – Application Example: Photo OCR

*Congratulations on making it to the eleventh and final week! This week, we will walk you through a complex, end-to-end application of machine learning, to the application of Photo OCR. Identifying and recognizing objects, words, and digits in an image is a challenging task. We discuss how a pipeline can be built to tackle this problem and how to analyze and improve the performance of such a system.*

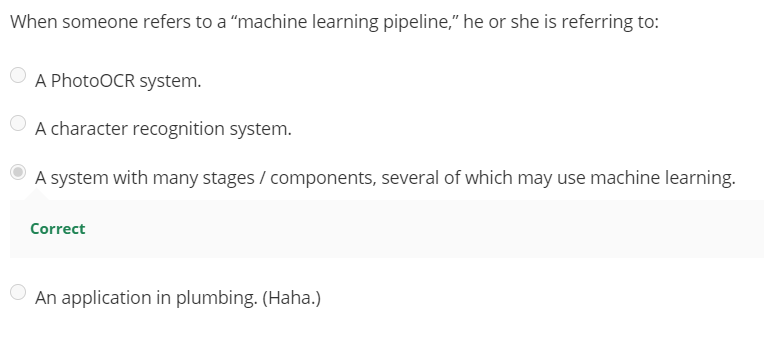
## Photo Optical Character Recognition (OCR) – July 19, 2020

Detect text in pictures, read text in these areas. Scanned documents are easier, photographs are harder.

Photo OCR/Machine Learning Pipeline

1. Input Image
2. Text detection
3. Character segmentation
4. Character classification/recognition
5. Spelling correction (more advanced)

Machine Learning pipeline – allocate resources



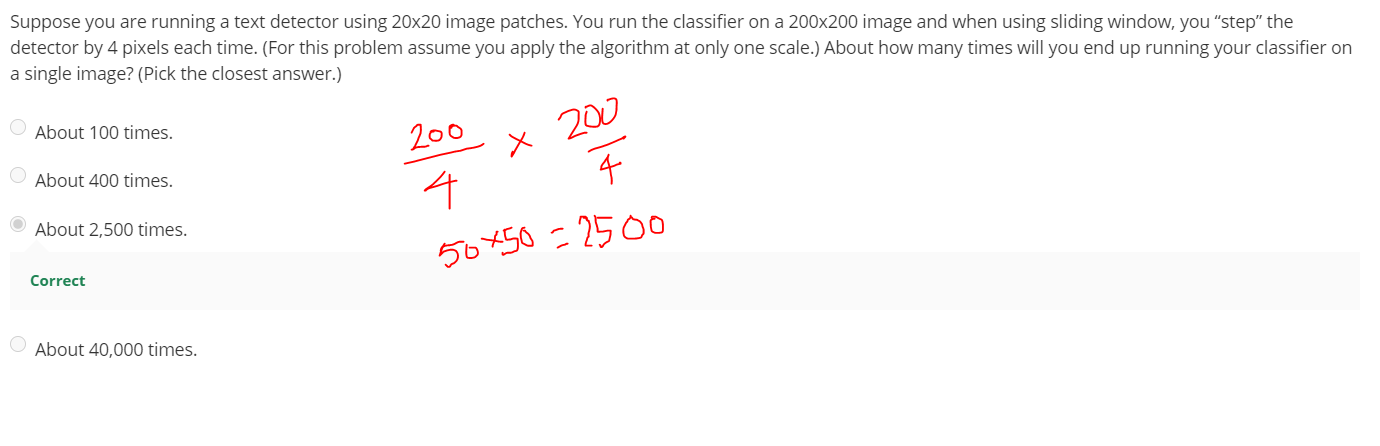
**Sliding Windows**

First stage of the pipeline.

Aspect ratio of objects dictate the difficulty of recognition. Text detection – heights and sizes are different. Pedestrian – aspect ratio is the same, although distance can be different. Need positive and negative examples.

Step size/stride parameter along the picture, x pixels at a time, row by row.

After classifying, apply expansion parameter to expand pixels that are most likely to be text.



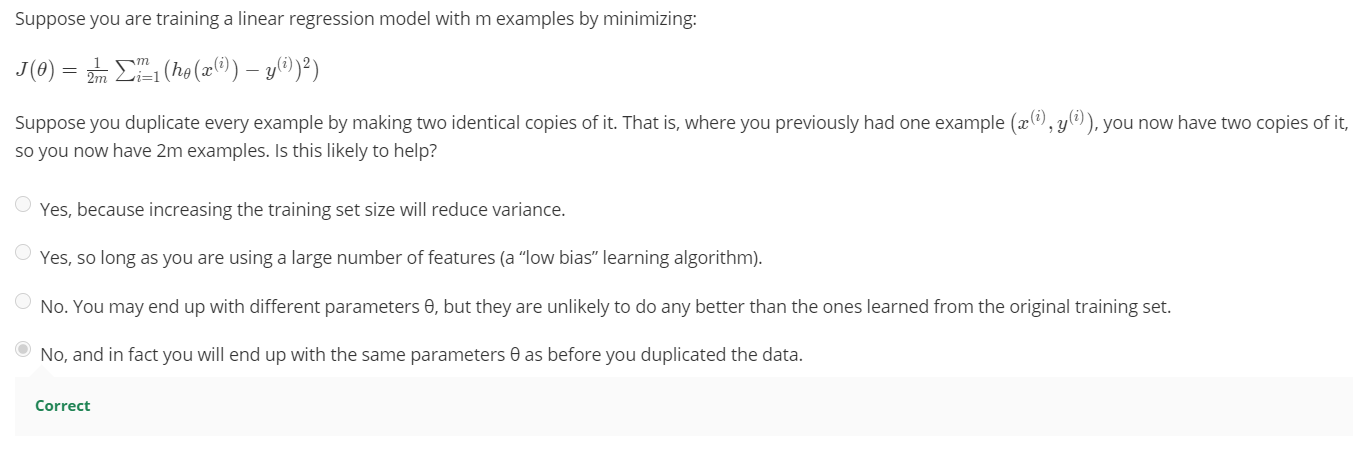
**Getting Lots of Data and Artificial Data**

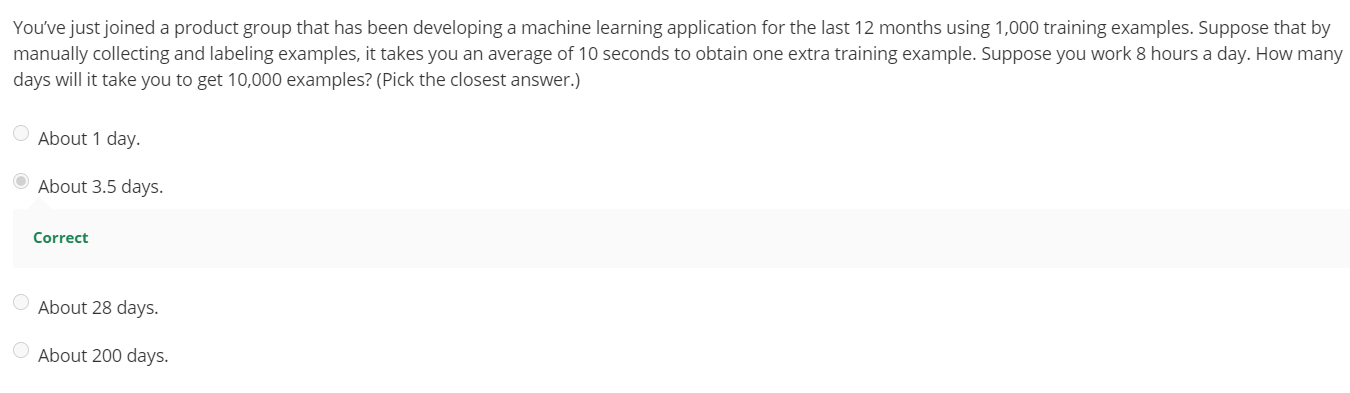
**Artificial Data Synthesis,** only applied to some problems. Use them as training set to train learning algorithm. Plot learning curve to make sure you have a low bias classifier before artificially synthesizing data.

Question to ask: “How much work would it be to get x10 as much data as we currently have?”

Ways to do Artificial Data Synthesis:

1. Create data from scratch, collect/ label it yourself.
2. Amplify small training set into a larger training set. Synthesize data by introducing distortions (text, background noise for speech).
   1. Distortion should be representative and reasonable of the type of noise/distortions we could see in the test set.
   2. Doesn’t help to add purely random/meaningless noise to your data (brightness, etc on pixels)
3. Crowd sourcing, e.g. Amazon Mechanical Turk

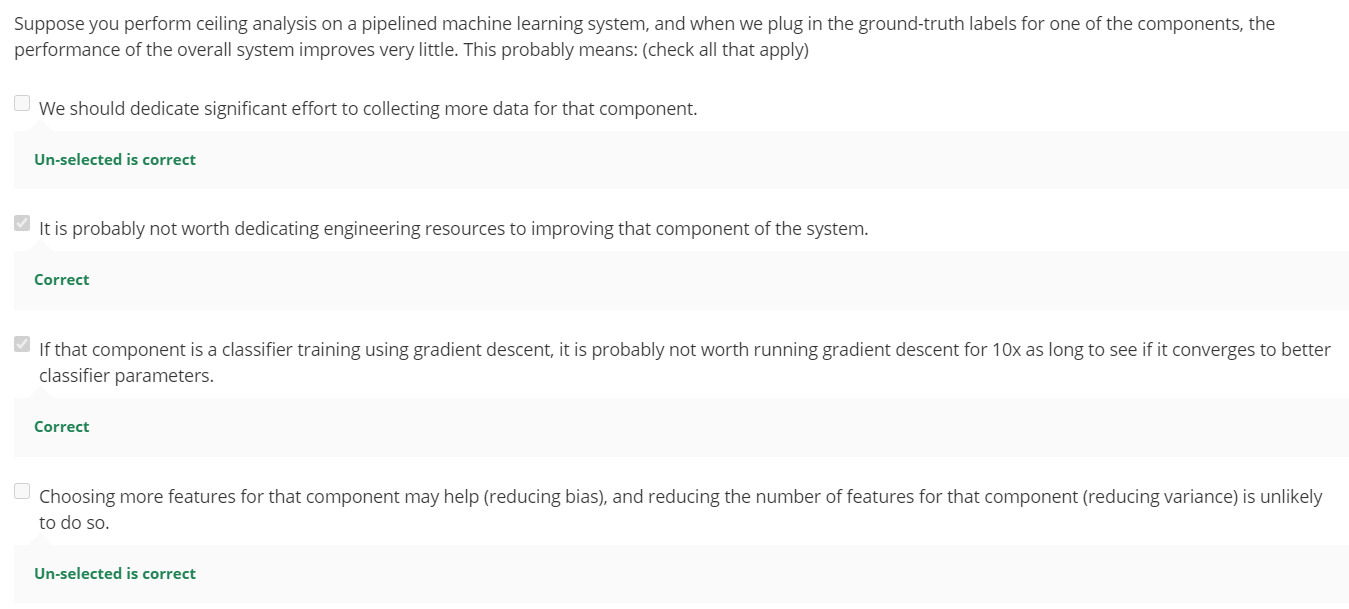




**Ceiling Analysis**

Errors due to each component, to determine which pipeline you should spend the most time on. To save time and effort!

Measure accuracy of each component of manually labelling the test cases. The ones with the highest upside (% change) in between pipelines are the ones you should be focusing on the most.



# Handy Equations

**SVM with Kernels**

* Cost function
* Similarity function

**K-means Unsupervised Algorithm**

* Distortion cost function

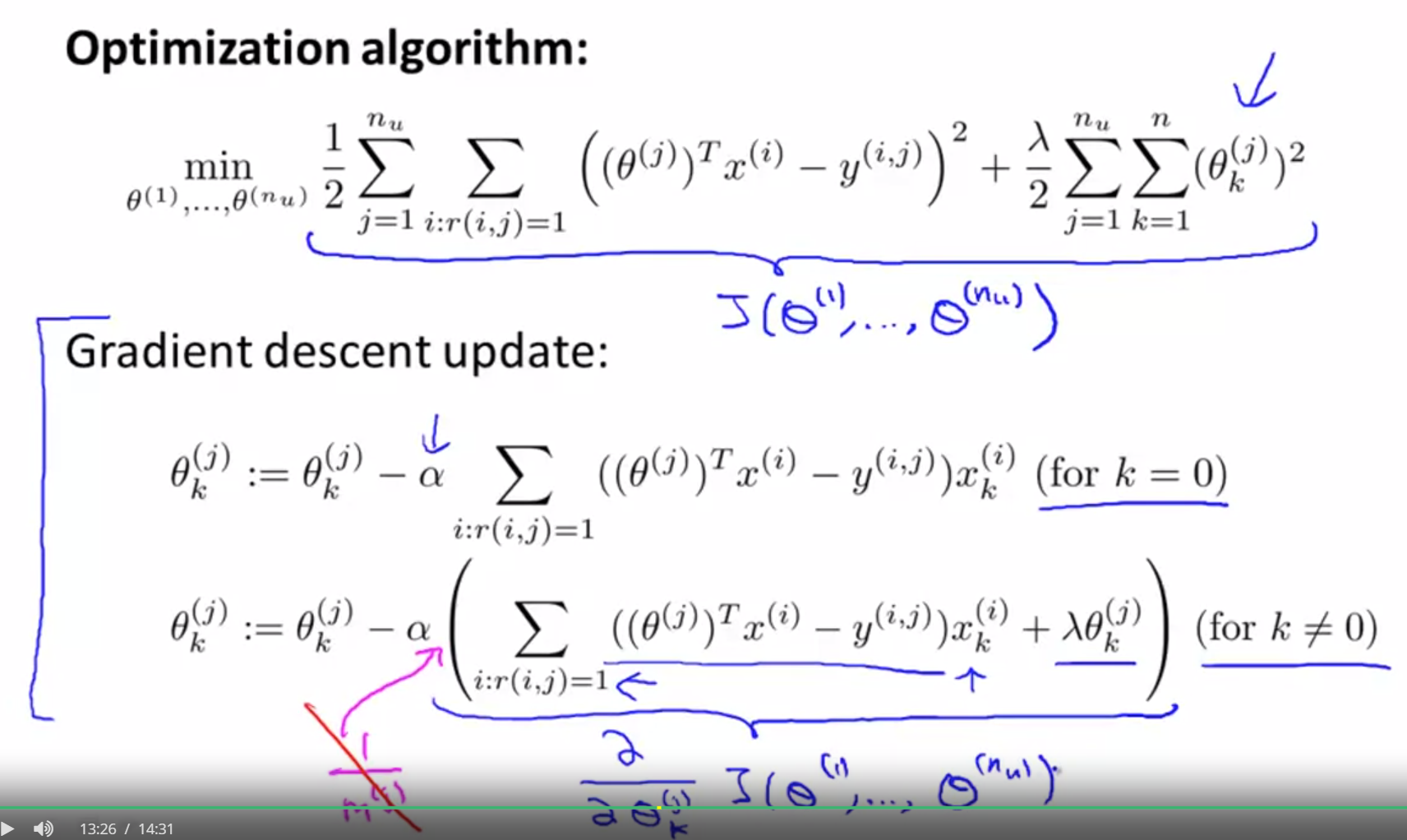
**PCA**

* Covariance matrix
* Reduced matrix

*z = UTreduceX;*

* [U, S, V] = svd(Sigma);

**Recommender Systems**



**Collaborative Filtering**

