ML - Machine Learning II

Improving a learning algorithm

In general the error of your hypothesis as measured on the data set with which you trained the parameters will be lower than the error on any other data set.
It may happen that you train a prediction model over a data set obtaining a high accuracy, and then moving to a new data set you may find out the accuracy is not so good.
There are a number of possible options you may apply trying to improve the model

- get more training examples (usually fixes high variance problems)
- try a smaller set of feature (usually fixes high variance problems)
- try getting additional features (usually fixes high bias problems)
- try adding polynomial features (usually fixes high bias problems)
- try decreasing λ (usually fixes high bias problems)
- try increasing λ (usually fixes high variance problems)

For a NN you can

- try a smaller architecture (cheaper but more prone to underfitting)
- try a larger architecture (computationally expensive and prone to overfitting if not regularized)

Many people spend months following one of the avenues above without a good rationale.
Fortunately there are some diagnostic tests to guide you

Evaluating the hypothesis

Just because a learning algorithm fits a training set well, that does not mean it is a good hypothesis.
To evaluate a hypothesis given a dataset of training examples, we should split up the dataset into two sets:

- Training set (70% of the examples)
- Test set (30% of the examples)

The new procedure using these two sets is then

1. Train the model and find θ parameters using the Training set
2. Compute the test set error J using the Test set examples

Model selection

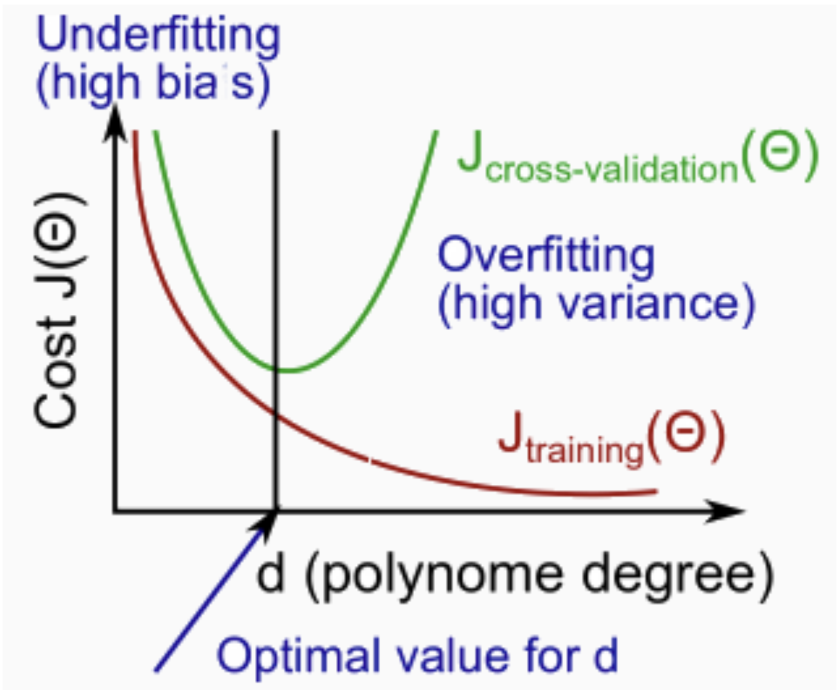
You might want to determine the best polynomial degree (d) or the best λ for the hypothesis. Now d (or λ) represents an additional parameter. Then you have to split the dataset this way

- Training set (60%)
- Cross Validation set (20%)
- Test set (20%)

It is important to understand that the example data over which you select your model, the example data over which you train the selected model and the example data over which you test the generalization accuracy of your model MUST BE DIFFERENT!!!

Diagnose Bias vs Variance problems

An important concept in ML is the bias-variance tradeoff. Models with high bias are not complex enough for the data and tend to underfit, while models with high variance overfit to the training data.
When you plot the errors as a function of d:
- High bias or underfitting: J_cv and J_train are both high and with similar values
- High variance or overfitting: J_cv >> J_train



When you plot the errors as a function of λ:
- High bias or underfitting: J_cv >> J_train
- High variance or overfitting: J_cv and J_train are both high and with similar values

Note that the CV error is computed without regularisation.
Typically the selection process for λ uses the following guess values
0,0.01,0.02,0.04,0.08,0.16,0.32,0.64,1.28,2.56,5.12,10.24

Learning Curves

We plot J_train or J_cv as a function of m (training set size).
For small m J_train are close or identical to 0.
J_train increases with increasing m
J_cv decreases with increasing m

- High bias or underfitting: J_cv and J_train quickly converge to the same (this is because increasing m will not help much)
- High variance or overfitting: J_cv and J_train converge slowly (getting more training data is likely to help)

Exercise 5 (ex5.m)

We want to implement a regularized linear regression model to predict the amount of water flowing out of a dam (y) using the change of water level in a reservoir (x).
We will go through some diagnostics of debugging learning algorithms and examine the effects of bias v.s. variance. Then we will determine the optimal degree of polynomial feature vector and regularisation parameter lambda.
TODO

Hints

- When starting on a new machine learning problem, try quick and dirty implementation of your learning algorithm.
- Don't worry about it being too quick, or don't worry about it being too dirty. But really, implement something as quickly as you can. And once you have the initial implementation, this is then a powerful tool for deciding where to spend your time next.
- Because first you can look at the errors it makes, and do this sort of error analysis to see what other mistakes it makes, and use that to inspire further development.

- Start with a simple algorithm, implement it quickly, and test it early on your cross validation data
- Plot learning curves to decide if more data, more features, etc. are likely to help
- Manually examine the errors on examples in the cross validation set and try to spot a trend where most of the errors were made.
- For example, assume that we have 500 emails and our algorithm misclassifies a 100 of them. We could manually analyze the 100 emails and categorize them based on what type of emails they are. We could then try to come up with new cues and features that would help us classify these 100 emails correctly. Hence, if most of our misclassified emails are those which try to steal passwords, then we could find some features that are

particular to those emails and add them to our model.

Evaluation Metrics for Skewed Classes

- To evaluate the quality of a prediction algorithm only on the base of the accuracy is not a good idea. This is particularly evident we you deal with skewed classes. For example, suppose you have a Logistic Regression classifier h trained to detect the presence (h(x)>=0.5, or y=1) or absence (h(x)<0.5, or y=0) of a cancer from images (x). Suppose it performs with 99% accuracy (1% error) over the test data set. It might even be possible that the real incidence of cancer in the test set was very low (let's say 0.5%), then if I decide to implement an hypothesis function identically equal to 0, I would achieve 0.5% misclassification, which would be better than my classifier!
- We need other metrics to ascertain the goodness of a model.
- We have to set y=1 in presence of rare class that we want to detect. Then we need to differentiate all matching cases of Actual vs Predicted class

– True positive (TP): when Predicted class h(x) >=0.5 and Actual class y = 1
– True negative (TN): when Predicted class h(x) < 0.5 and Actual class y = 0
– False positive (FP): when Predicted class h(x) >=0.5 but Actual class y = 0
– False negative (FN): when Predicted class h(x) < 0.5 but Actual class y = 1

We define:
Accuracy (A) = (# TP + # TN) / Total examples
Precision (P) = # TP / # Predicted pos = # TP / (# TP + # FP)
Recall (R) = # TP / # Actual pos = # TP / (# TP + # FN)

If we obtain a good Precision AND a good Recall then we make sure that our algorithm is really good even with skewed classes.

- Most of the time we have to reach a tradeoff between Precision and Recall. One way to increase the Precision at the expense of Recall is to increase the threshold over the output probability, i.e. (h(x) >= threshold) => (y=1)
- Suppose you want to predict cancer only if very confident. In this case increase threshold from 0.5 to 0.8. This will increase Precision but will decrease Recall.
- Suppose you want to avoid missing too many cases of cancer (avoid false negatives). In this case decrease threshold from 0.5 to 0.3. This will increase Recall but will decrease Precision.
- When we come to the problem of scoring two or more algorithms with different values of Precision (P) and Recall (R), then we can use the F Score = 2*(PR)/(P+R), that gives you a single real number evaluation metric
- The best way to proceed is to and choose the threshold value which maximises 2*(PR)/(P+R), with P and R evaluated over the cross validation

set (not the test set)

The importance of Data

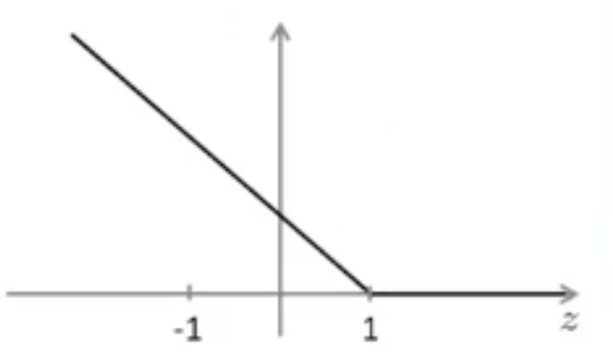
From a classical study we know that the algorithm is not really important when we have a big amount of data to train our model.
We always need to ask: the feature set x has sufficient information to predict y accurately ?
An useful test: given the input x, can a human expert confidently predict y ?

SVM (Support Vector Machine)

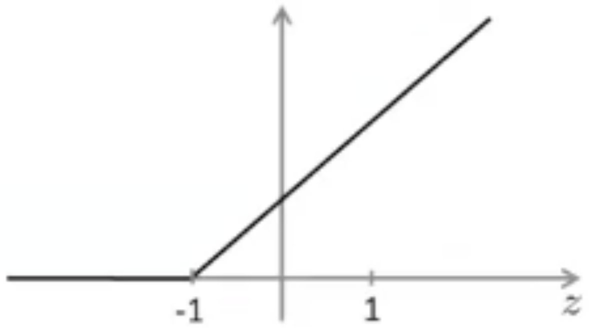
- Besides logistic regression and NN, Supported Vector Machine (SVM) is another powerful supervised learning algorithm for nonlinear function estimation
- To define the SVM we start from the cost function for the logistic regression,

$$\min_\theta \frac{1}{m}\left[\sum_{i=1}^{m} y^{(i)}\left(-\log h_\theta(x^{(i)})\right) + (1-y^{(i)})\left((-\log(1-h_\theta(x^{(i)}))\right)\right] + \frac{\lambda}{2m}\sum_{j=1}^{n}\theta_j^2$$

by replacing a piecewise linear version of the functions -log h(z), which is called cost_1(z),



and -log[1 - h(z)], which is called cost_0(z)

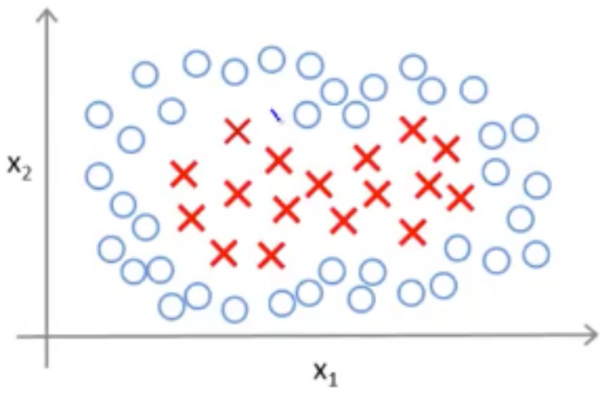


Then, the SVM must minimise

$$\min_\theta C \sum_{i=1}^{m} \left[y^{(i)} cost_1(\theta^T x^{(i)}) + (1 - y^{(i)}) cost_0(\theta^T x^{(i)}) \right] + \frac{1}{2} \sum_{i=1}^{n} \theta_j^2$$

where C = 1/λ takes the role of the regularisation coefficient.
Informally, C is a positive value that controls the penalty for misclassified training examples

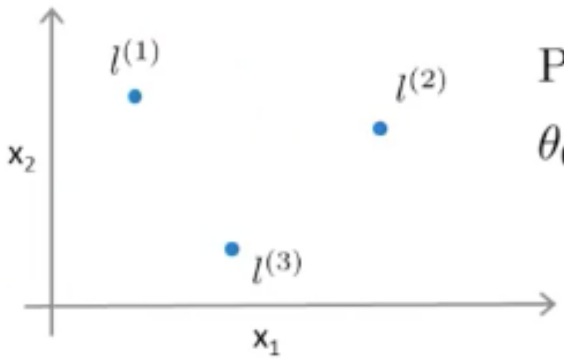
- SVM is also called Large Margin Classifier, because for linearly separable training sets and C>>1, SVM determines a class boundary which is maximally distant from the training examples belonging to the different classes. Intuitively this is due to the fact that minimising J means minimise the projections of x along the θ vector (scalar products) as well as ||θ||.
- Kernels: a concept developed to adapt the SVM to nonlinear decision boundaries. Example



Predict $y = 1$ if

$$\theta_0 + \theta_1 x_1 + \theta_2 x_2 + \theta_3 x_1 x_2 + \theta_4 x_1^2 + \theta_5 x_2^2 + \cdots \geq 0$$

We define a set of point L (or *landmarks*) and a kernel that expresses the proximity (or *similarity*) f of the feature vector x to those landmarks. Then landmarks turns into a new set of features f_1, f_2, .. for the SVM



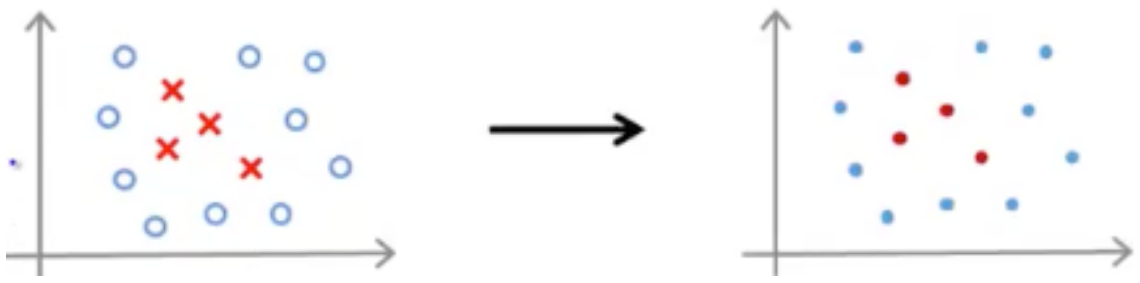
Predict "1" when

$$\theta_0 + \theta_1 f_1 + \theta_2 f_2 + \theta_3 f_3 \geq 0$$

For the Gaussian kernel, the similarity f_1 of x with respect to l_1 is

$$f_1 = \text{similarity}(x, l^{(1)}) = \exp\left(-\frac{\|x - l^{(1)}\|^2}{2\sigma^2}\right)$$

- How to choose the landmarks and the kernel function ? We start choosing the L set equal to the training set, plus f_0=1



The new feature vector f of m+1 elements can be used to solve the prediction problem

$$\min_{\theta} C \sum_{i=1}^{m} y^{(i)} cost_1(\theta^T f^{(i)}) + (1 - y^{(i)}) cost_0(\theta^T f^{(i)}) + \frac{1}{2} \sum_{j=1}^{n} \theta_j^2$$

where n=m.

- In order to solve for the parameter theta use one of the existing SVM library function (e.g. lib linear, libsvm, ..).
- You have to specify C and the kernel (similarity function).
- The no kernel option is called "linear kernel", which

Predict "y = 1" if $\theta^T x \geq 0$

Use this option when n is large and m is small.

- The Gaussian kernel is

$$f_i = \exp\left(-\frac{||x - l^{(i)}||^2}{2\sigma^2}\right), \text{ where } l^{(i)} = x^{(i)}$$

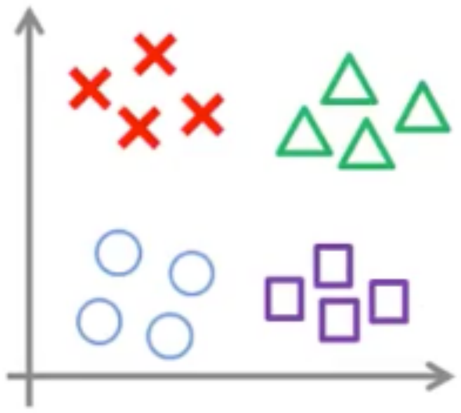
Use this option when n is small and m is (ideally) large.
In Octave you may need to define the kernel function.
Do perform feature scaling before using Gaussian kernel. This is particularly evident with the Gaussian kernel because ||x - l|| is a sum of squares of individual features, and this could easily amplify the effect of features with higher range values.

- The Polynomial kernels look like ..
- There are also more esoteric kernels: String kernel, chi-square kernel, histogram intersection kernel, etc.

String kernel is used when the input data are represented by words and strings, in text classification problems.

- An important aspect is the choice of the C factor as a tradeoff between bias and variance
- large C (small λ): lower bias, high variance, the system is prone to overfitting
- small C (large λ): higher bias, low variance, the system is prone to undercutting

- Another important optimisation coefficient is sigma for the Gaussian kernel
- large sigma: features f vary more smoothly, higher bias, low variance
- small sigma: features f vary more sharply, lower bias, high variance
- To make the best choices for kernel, C, sigma, etc. you have to use whatever performs best on the cross-validation data.
- For multi-class classification problems the one-vs-all strategy is very common.



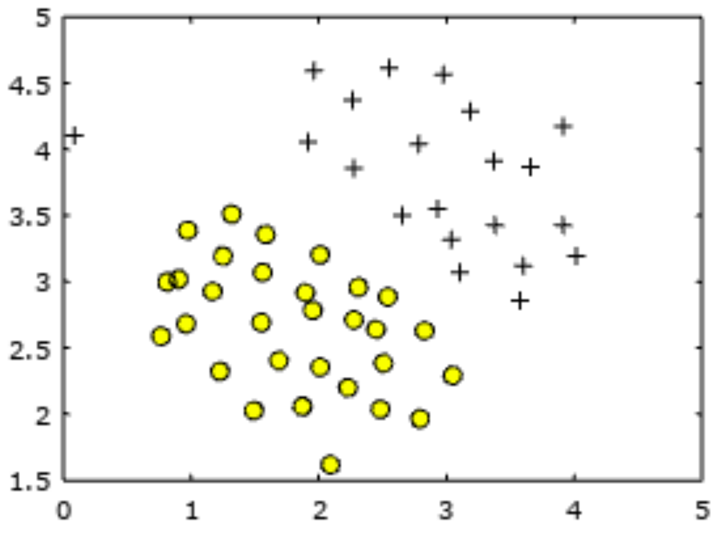
$$y \in \{1, 2, 3, \ldots, K\}$$

Just you need to train K SVMs, obtaining K optimal param vectors θ_1,..θ_K, then pick the class i for with largest θ_i'x.
Many SVM packages already have built-in support for multi-class classification problems.

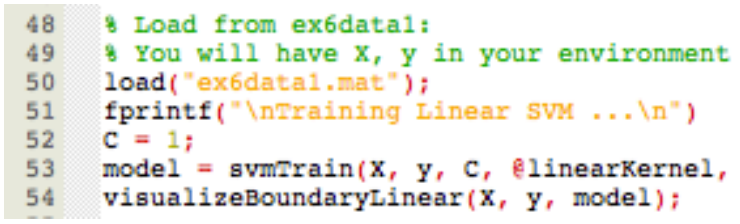
- If n >= 10*m then use the logistic regression or SVM with linear kernel
- If m>= 10*n then use the SVM with Gaussian kernel
- If m >> n then create and add more features, then use logistic regression or SVM with linear kernel
- NNs work well in all cases, but may be slower to train
- SVM have convex optimisation, so do not worry about local minima. There is only one global minimum of the cost function

Exercise 6 (ex6.m)

We want to use SVMs over 2D data sets. In the next half of the exercise, we will be using SVM to build a spam classifier.
Load and plot this simple data set

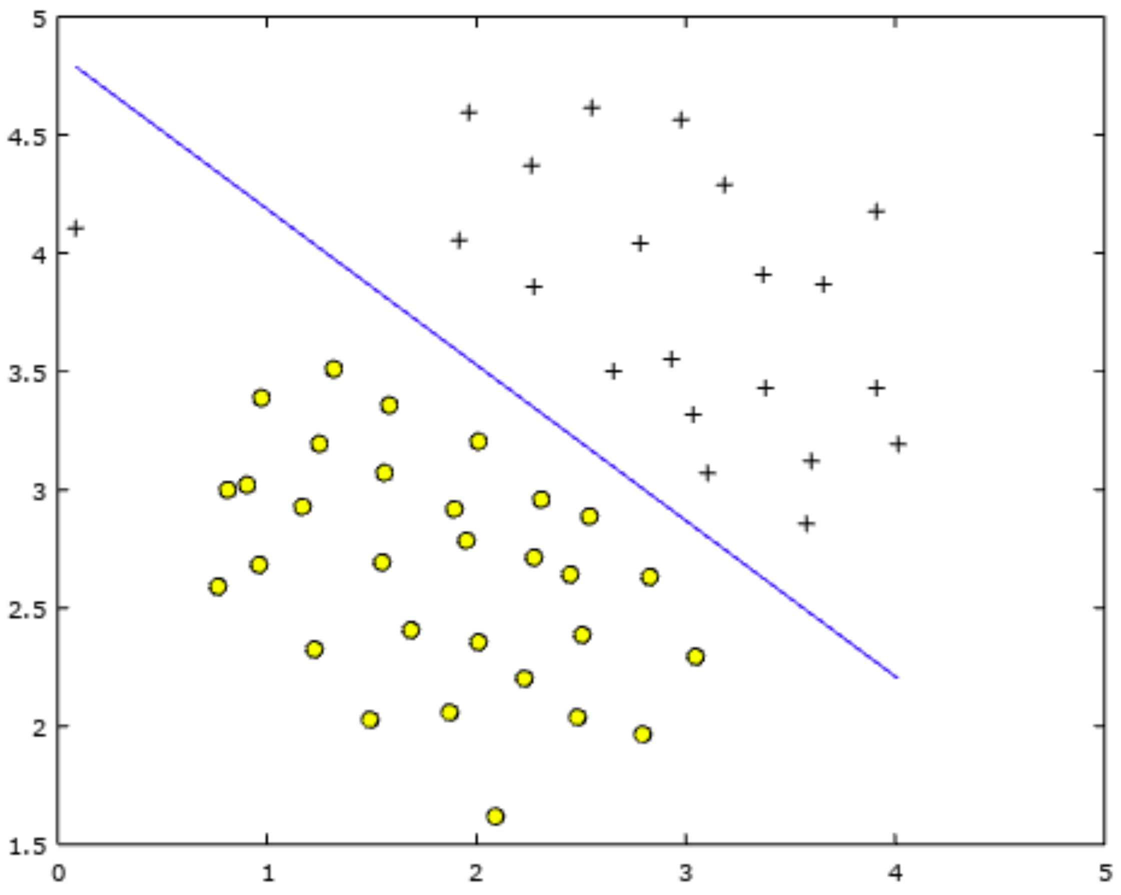


We can use the SVM software packages called svmTrain.m
The kernel can be linear (no kernel) because the data set is is linearly separable



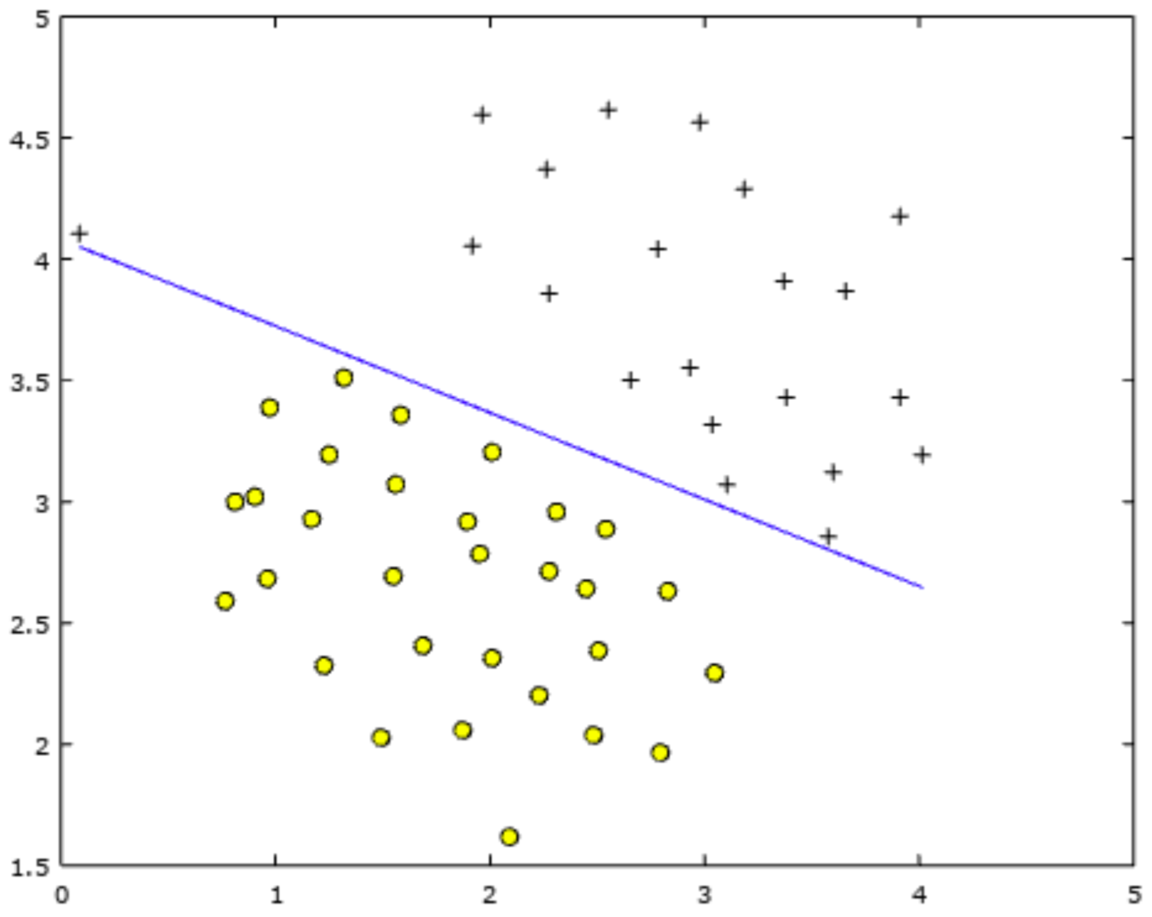
```
% Load from ex6data1:
% You will have X, y in your environment
load("ex6data1.mat");
fprintf("\nTraining Linear SVM ...\n")
C = 1;
model = svmTrain(X, y, C, @linearKernel, 1e-3, 20);
visualizeBoundaryLinear(X, y, model);
```

With C=1 (small value) we have

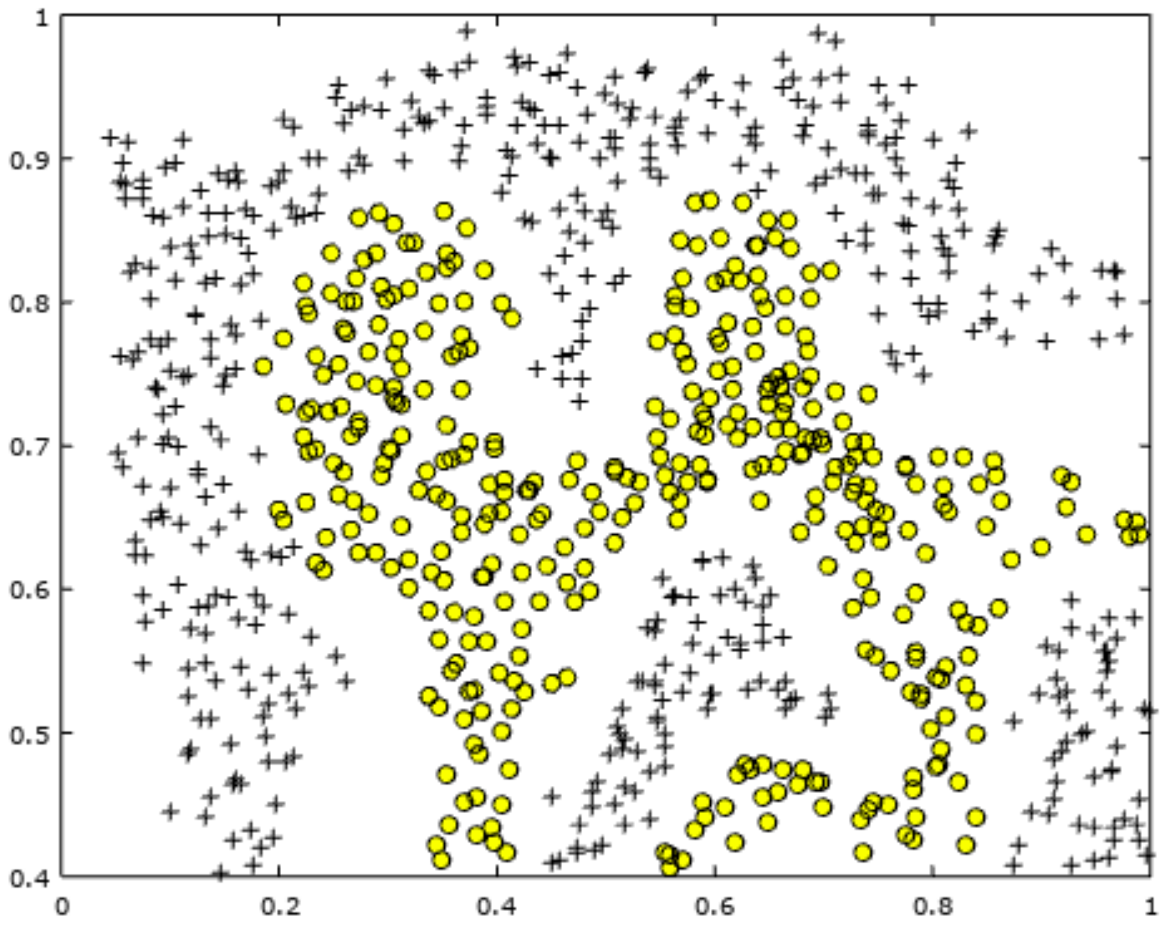


the model has small variance, then the overall separation margin is good, but

the "+" leftmost example is misclassified.
With C=100 (high value) we obtain



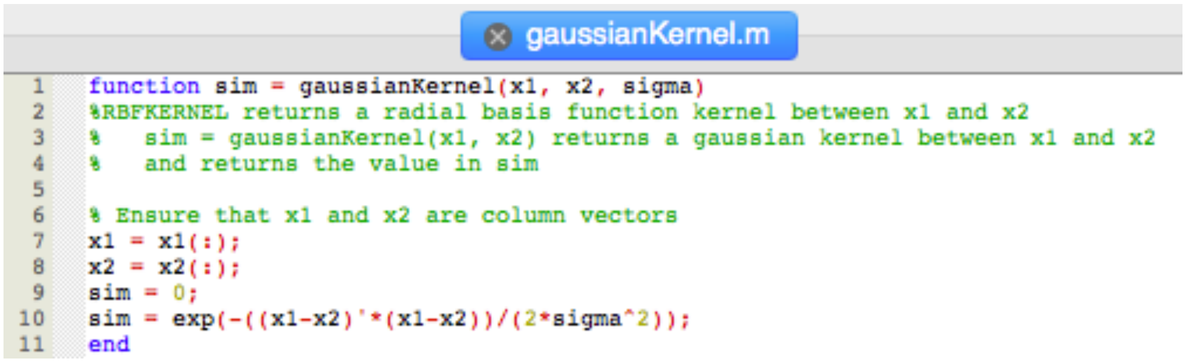
Now consider datasets that are not linearly separable.



In this case we use SVM with Gaussian kernel, or RBF.
Let's define the similarity function, which is defined as

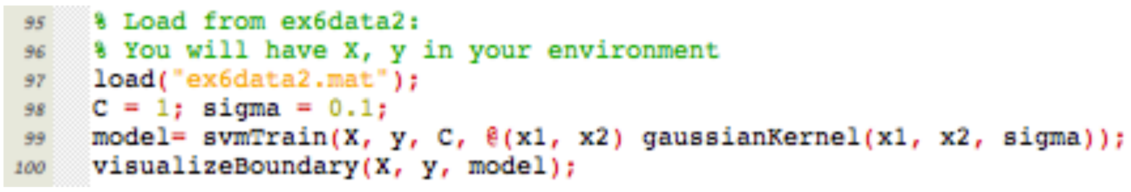
$$K_{gaussian}(x^{(i)}, x^{(j)}) = \exp\left(-\frac{\|x^{(i)} - x^{(j)}\|^2}{2\sigma^2}\right) = \exp\left(-\frac{\sum_{k=1}^{n}(x_k^{(i)} - x_k^{(j)})^2}{2\sigma^2}\right)$$

In Octave,



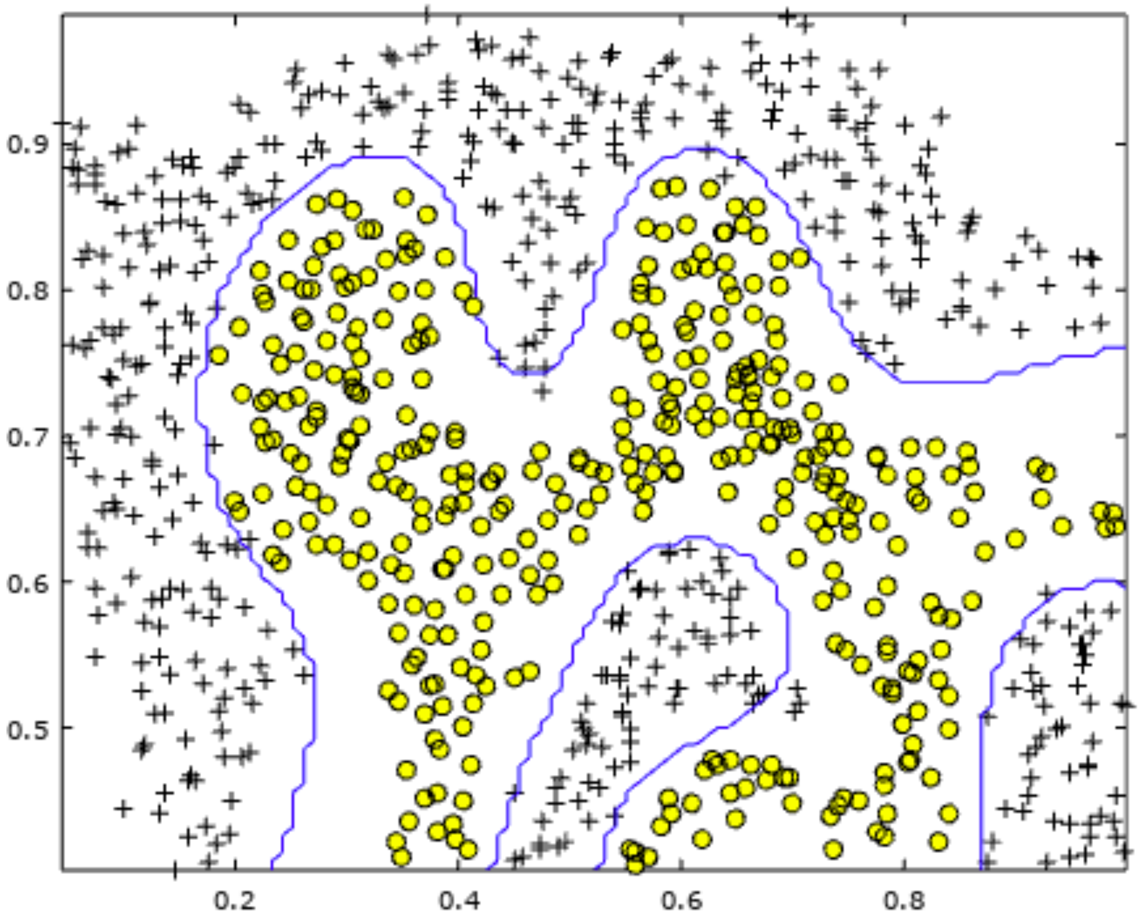
```
function sim = gaussianKernel(x1, x2, sigma)
%RBFKERNEL returns a radial basis function kernel between x1 and x2
%   sim = gaussianKernel(x1, x2) returns a gaussian kernel between x1 and x2
%   and returns the value in sim

% Ensure that x1 and x2 are column vectors
x1 = x1(:);
x2 = x2(:);
sim = 0;
sim = exp(-((x1-x2)'*(x1-x2))/(2*sigma^2));
end
```

Now, let's train the SVM

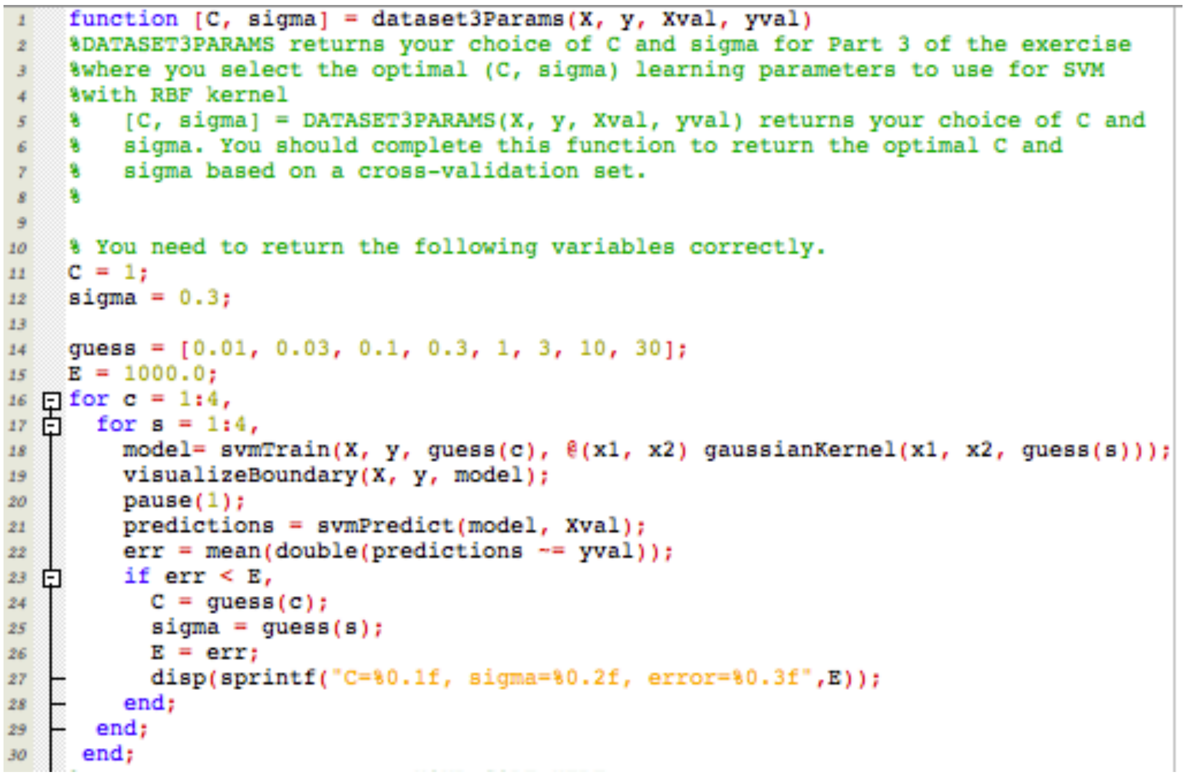


```
% Load from ex6data2:
% You will have X, y in your environment
load("ex6data2.mat");
C = 1; sigma = 0.1;
model= svmTrain(X, y, C, @(x1, x2) gaussianKernel(x1, x2, sigma));
visualizeBoundary(X, y, model);
```

and finally let's plot the decision boundary, which is able to separate most of the positive and negative examples correctly and follows the contours of the dataset well

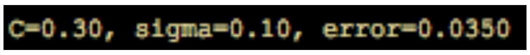


In the following section we want to select the best C and sigma of a Gaussian kernel SVM trained over a cross-validation set.

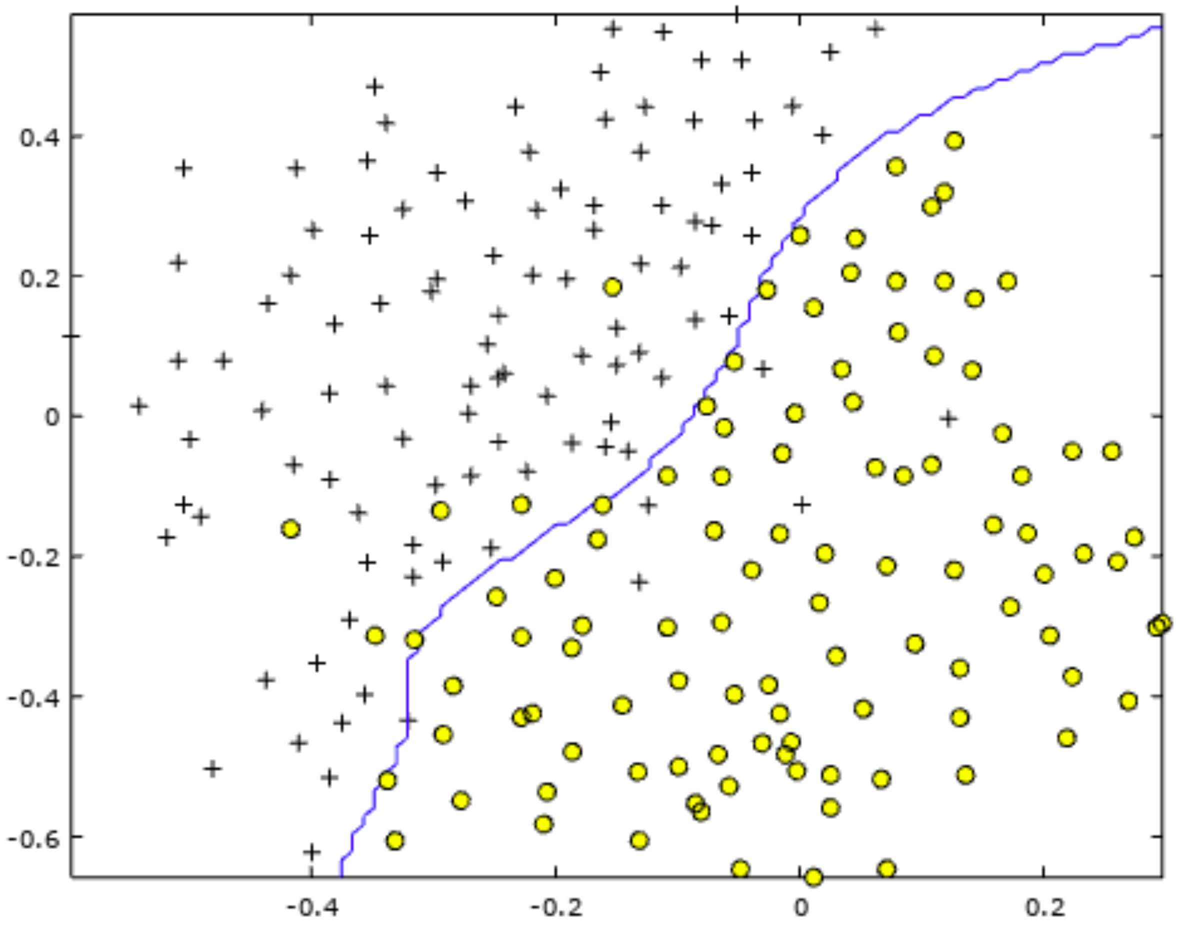


```matlab
function [C, sigma] = dataset3Params(X, y, Xval, yval)
%DATASET3PARAMS returns your choice of C and sigma for Part 3 of the exercise
%where you select the optimal (C, sigma) learning parameters to use for SVM
%with RBF kernel
%   [C, sigma] = DATASET3PARAMS(X, y, Xval, yval) returns your choice of C and
%   sigma. You should complete this function to return the optimal C and
%   sigma based on a cross-validation set.
%

% You need to return the following variables correctly.
C = 1;
sigma = 0.3;

guess = [0.01, 0.03, 0.1, 0.3, 1, 3, 10, 30];
E = 1000.0;
for c = 1:4,
  for s = 1:4,
    model= svmTrain(X, y, guess(c), @(x1, x2) gaussianKernel(x1, x2, guess(s)));
    visualizeBoundary(X, y, model);
    pause(1);
    predictions = svmPredict(model, Xval);
    err = mean(double(predictions ~= yval));
    if err < E,
      C = guess(c);
      sigma = guess(s);
      E = err;
      disp(sprintf("C=%0.1f, sigma=%0.2f, error=%0.3f",E));
    end;
  end;
 end;
```

The best C and sigma are



```
C=0.30, sigma=0.10, error=0.0350
```



Spam Classification

This exercise is a SVM implementation based on the *SpamAssassin Public Corpus* (https://spamassassin.apache.org/old/publiccorpus/).
For the purpose of the exercise, you will only be using the body of the email (excluding the email headers).

- The first preprocessing step is to "normalize" URLs, dollar amounts and email addresses, which are objects frequently found in spammed email.

The following preprocessing steps are applied

- lower-casing: to ignore capitalisation
- stripping html: removing the html tags. HTML tags are the hidden keywords within a web page that define how your web browser must format and display the content. Most tags must have two parts, an opening and a closing part. For example, <html> is the opening tag and </html> is the closing tag. Note that the closing tag has the same text as the opening tag, but has an additional forward-slash (/) character. I tend to interperet this as the "end" or "close" character.
- URLs normalisation: they are replaced by "httpaddr"
- Email address normalisation: they are replaced by "emailaddr"
- Number normalisation: they are replaced by "number"
- Dollars normalisation: "$" are replaced by "dollar"
- Word stemming: replaces words in they stemmed form. For example "discount", "discounted", "discounting" all are replaced by "discount"
- Non-words removal: non-words and punctuation are removed. All white spaces have all been trimmed

```
email_contents = lower(email_contents);
email_contents = regexprep(email_contents, '&lt;[^&lt;&gt;]+&gt;', ' ');
email_contents = regexprep(email_contents, '[0-9]+', 'number');
email_contents = regexprep(email_contents, '(http|https)://[^\s]*', 'httpaddr');
email_contents = regexprep(email_contents, '[^\s]+@[^\s]+', 'emailaddr');
email_contents = regexprep(email_contents, '[$]+', 'dollar');
```

- Example:

Original ample email:
> Anyone knows how much it costs to host a web portal ?
>
Well, it depends on how many visitors youre expecting. This can be anywhere from less than 10 bucks a month to a couple of $100. You should checkout http://www.rackspace.com/ or perhaps Amazon EC2 if youre running something big..
To unsubscribe yourself from this mailing list, send an email to:
groupname-unsubscribe@egroups.com

Preprocessed version:
anyon know how much it cost to host a web portal well it depend on how mani visitor your expect thi can be anywher from less than number buck a month to a coupl of dollarnumb you should checkout httpaddr or perhap amazon ecnumb if your run someth big to unsubscrib yourself from thi mail list send an email to emailaddr

Word indices version:
86 916 794 1077 883 370 1699 790 1822 1831 883 431 1171 794 1002 1893 1364
592 1676 238 162 89 688 945 1663 1120 1062 1699 375 1162 479 1893 1510 799
1182 1237 810 1895 1440 1547 181 1699 1758 1896 688 1676 992 961 1477 71 530
1699 531

- Our vocabulary list was selected by choosing all words which occur at least a 100 times in the spam corpus, resulting in a list of 1899 words. In practice, a vocabulary list with about 10,000 to 50,000 words is often used.

Unsupervised Models

We do not get any labels y^(i) in the training set, and the algorithm has to find out some structure in the data.
- Clustering is one example of unsupervised learning

Applications:
- Market segmentation
- Computing cluster organisation
- Astronomical data analysis
- Social network analysis

- K-means is an iterative clustering algorithm. At each iteration until convergence, points are assigned to clusters based on closeness to current cluster centroids. Centroids are randomly selected at start, and are updated based on the new cluster points. K-means is most effective when clusters are of similar shape and the number of clusters K is known beforehand.
- For K-means we assume

$$x^{(i)} \in \mathbb{R}^n \text{ (drop } x_0 = 1 \text{ convention)}$$

The algorithm is

```
Randomly initialize $K$ cluster centroids $\mu_1, \mu_2, \dots, \mu_K \in \mathbb{R}^n$
Repeat {
    for $i$ = 1 to $m$
        $c^{(i)}$ := index (from 1 to $K$) of cluster centroid
                closest to $x^{(i)}$
    for $k$ = 1 to $K$
        $\mu_k$ := average (mean) of points assigned to cluster $k$


    }
```

Once the process has finished one might find a an empty cluster. In that case you have to eliminate it.

- In order to randomly initialise the K cluster centroids we can pick K training examples and use them as initial centroids.

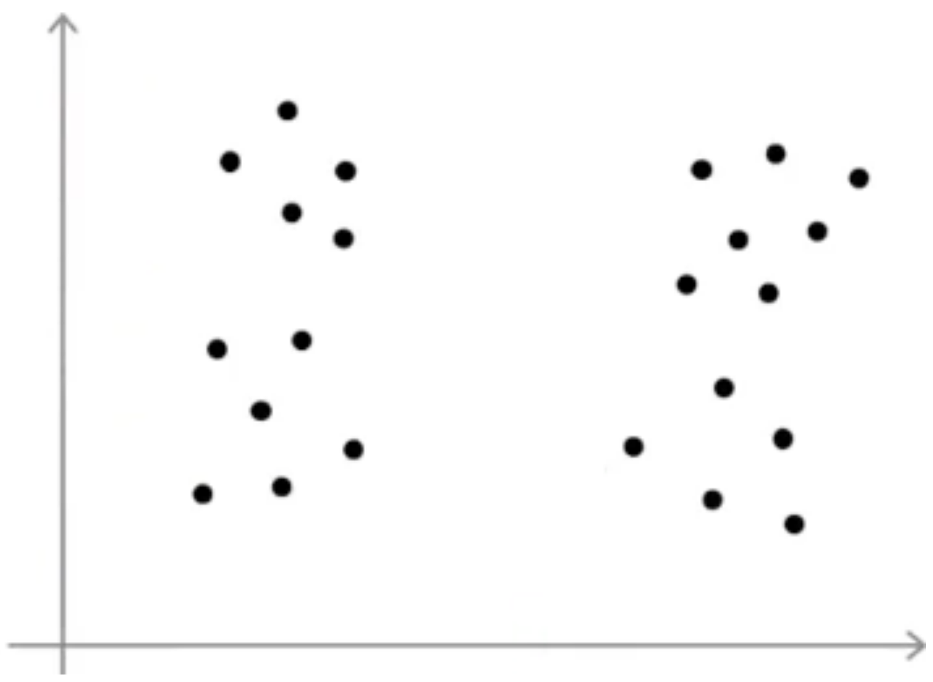
Pick k distinct random integers $i_1, \dots, i_k$ from $\{1, \dots, m\}$

Set $\mu_1 = x^{(i_1)}, \mu_2 = x^{(i_2)}, \dots, \mu_k = x^{(i_k)}$.

- The problem with this approach is that we can came up to local optima. Then we can find the minimal distortion with multiple random initialization, and the pick the clustering with the lowest distorsion

```
For i = 1 to 100 {

    Randomly initialize K-means.
    Run K-means. Get $c^{(1)}, \dots, c^{(m)}, \mu_1, \dots, \mu_K$.
    Compute cost function (distortion)
        $J(c^{(1)}, \dots, c^{(m)}, \mu_1, \dots, \mu_K)$
    }
```

- To choose the number of clusters K is not always simple. For example the following data set might be separated evenly well either in two or four clusters



The elbow method can help. It consist in plotting the distortion for some K values, for example 1 to 8. Then choose the minimum K above which distortion decreases more slowly. The cost function has to be a decreasing function of K

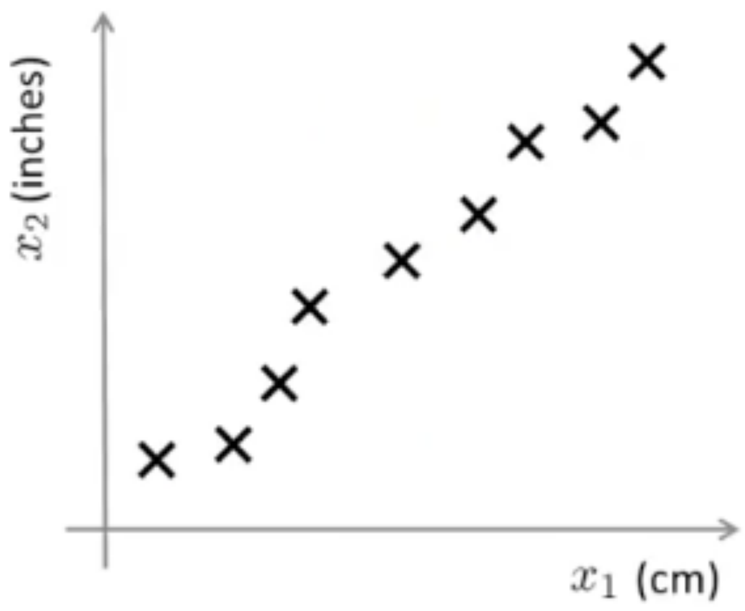
- The cost function that has to be minimised for the K-maen clustering is

$$J(c^{(1)}, \dots, c^{(m)}, \mu_1, \dots, \mu_K) = \frac{1}{m} \sum_{i=1}^{m} ||x^{(i)} - \mu_{c^{(i)}}||^2$$

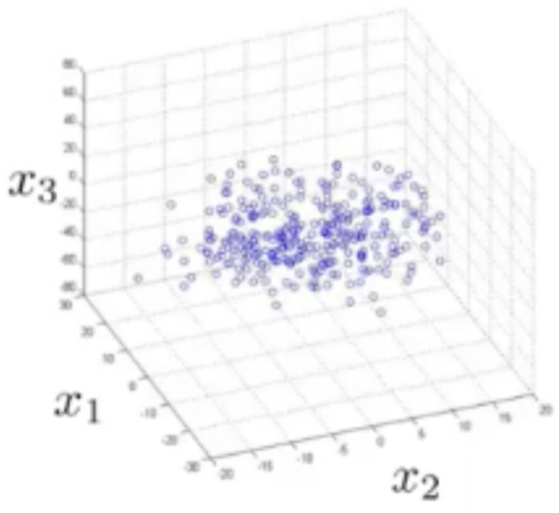
i.e. the sum of the distance of all example points from the centroid of the cluster they belong to. J is also known as Distorsion

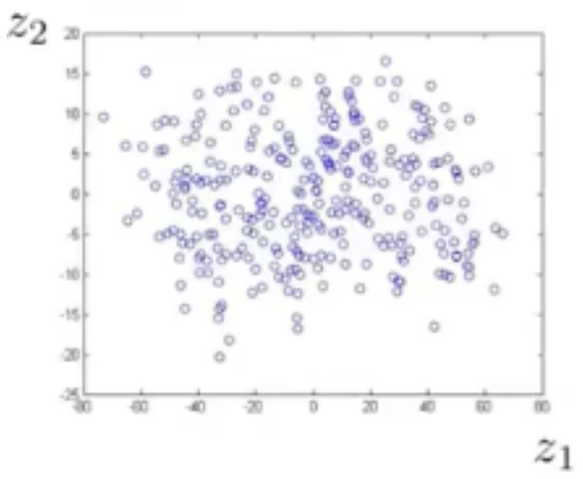
Dimensionality Reduction

Often many features are highly correlated. Thus makes sense to reduce the dimensionality of the features space.
For example to reduce data from 2D to 1D we can simply take the abscissa of the projection of the data point over a straight line z



To reduce data from 3D to 2D we can simply take the coordinate of the projection of the data point over a plane (z_1,z_2)





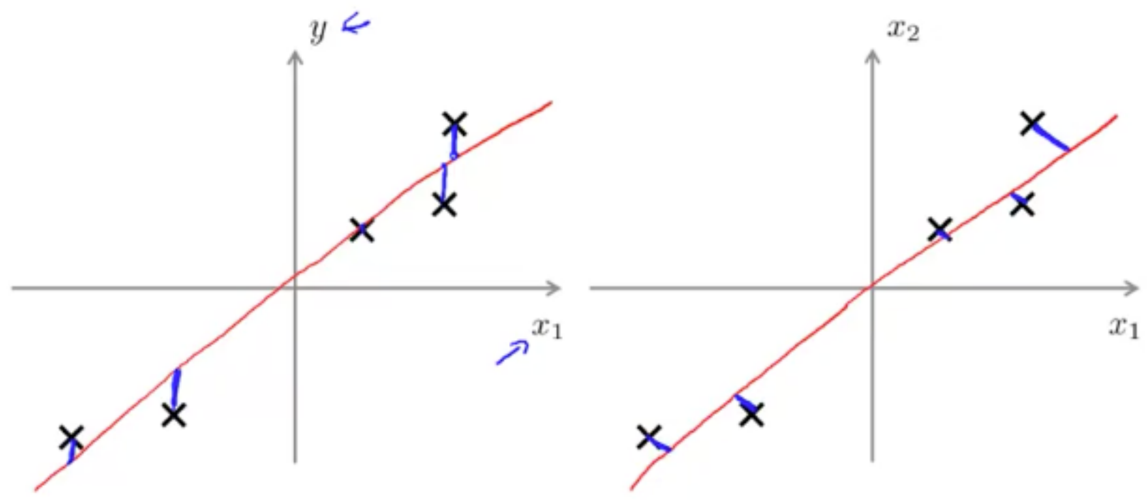
PCA

- Principal Component Analysis (PCA) allows to obtain feature space reduction of dimensionality. We have to find the k-dimensional subspace (k basis vectors) onto which to project data so as to minimise the projection error

$$\frac{1}{m}\sum_{i=1}^{m}\|x^{(i)} - x_{\text{approx}}^{(i)}\|^2$$

- Warning: PCA is not linear regression. The following picture explains why. The minimisation interests different segments, vertical segments for linear regression, and projections for PCA. Moreover in the case of PCA there is no special variable y

PCA is not linear regression



- In order to perform PCA may be important to apply feature mean normalisation (and optionally feature scaling) preprocessing.

Then you have to calculate the covariance matrix

$$\Sigma = \frac{1}{m}\sum_{i=1}^{n}(x^{(i)})(x^{(i)})^T$$

which is always a symmetric and positive definite matrix.
In Octave this is
Sigma = (1/m)*X'*X;
Then you need to find the eigenvectors of Sigma through the Singular Value Decomposition. The command in Octave is

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

The first k columns of U matrix are the k subspace basis vectors

$$U = \begin{bmatrix} | & | & & | \\ u^{(1)} & u^{(2)} & \dots & u^{(n)} \\ | & | & & | \end{bmatrix} \in \mathbb{R}^{n \times n}$$

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

Finally you get the projected data set as

```
z = Ureduce'*x;
```

- PCA can be used for different reasons:
- – to compress data
- – to speed up ML learning algorithm
- – to better visualise data in 2D or 3D
- Sometimes PCA is used improperly, for example to prevent overfitting. It is bad because you might discard features which carry useful informations (even you retain 99% of the variance). Instead use regularisation
- How to choose the number of principal components k ?

Typically, choose k to be smallest value so that

$$\frac{\frac{1}{m}\sum_{i=1}^{m}\|x^{(i)} - x_{approx}^{(i)}\|^2}{\frac{1}{m}\sum_{i=1}^{m}\|x^{(i)}\|^2} \leq 0.01 \qquad (1\%)$$

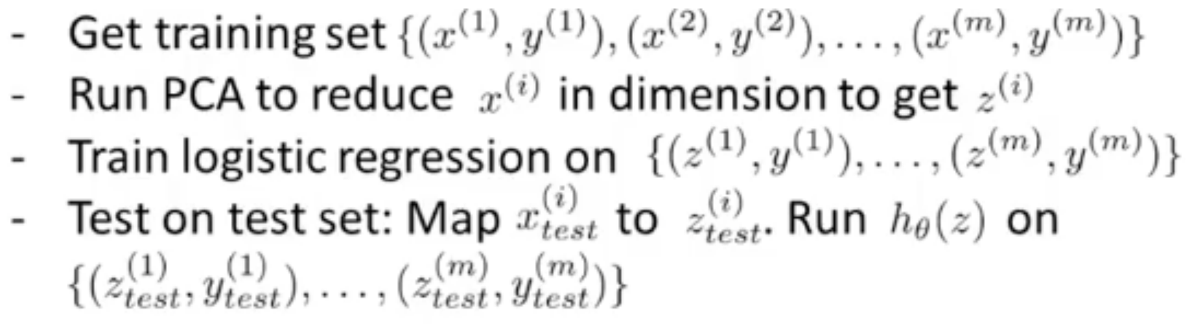
"99% of variance is retained"

One way to find k is to compute PCA for k = 1,2,3 .. until the relation above gets satisfied. There is a more quick way to solve this problem.
The S matrix is diagonal, and the elements in the diagonal are the eigenvalues. For given value of k, the expression above is equal to 1 minus the ratio of the sum of the first k eigenvalues over the sum of all eigenvalues.
Then the criterium is

$$\frac{\sum_{i=1}^{k} S_{ii}}{\sum_{i=1}^{m} S_{ii}} \geq 0.99$$

- Note that mapping x to z should be defined by running PCA only over the training set. Then the mapping can be applied (via U_reduce) as well to the example of the cross-validation set and over the test set.

- The general procedure can be summarised in the following steps



- Get training set $\{(x^{(1)}, y^{(1)}), (x^{(2)}, y^{(2)}), \ldots, (x^{(m)}, y^{(m)})\}$
- Run PCA to reduce $x^{(i)}$ in dimension to get $z^{(i)}$
- Train logistic regression on $\{(z^{(1)}, y^{(1)}), \ldots, (z^{(m)}, y^{(m)})\}$
- Test on test set: Map $x^{(i)}_{test}$ to $z^{(i)}_{test}$. Run $h_\theta(z)$ on $\{(z^{(1)}_{test}, y^{(1)}_{test}), \ldots, (z^{(m)}_{test}, y^{(m)}_{test})\}$

however you should first try to train the ML without OCA. Only if that doesn't do what you want, then use PCA!

Exercise 7 (ex7.m)

In this exercise, we will implement the K-means clustering algorithm and apply it to compress an image. In the second part, we will use principal component analysis to find a low-dimensional representation of face images.