Statistical inference and machine learning
Midterm project

- This project must be solved individually.
- Deadline: Monday 16 November 2020, 23:59 (No late submissions will be accepted)
- Upload a single zip file on Moodle containing your solution and your code implementations. You can use any programming language, but it is recommended to use MATLAB for the part 1 as you are given some functions that can help you.

1 Using SVM for Spam Classification

In the first half of this project, you will be using support vector machines (SVMs) with two examples. Experimenting with these datasets will help you gain an intuition of how SVMs work and how to use a Gaussian kernel with SVMs. In the next half, you will be using support vector machines to build a spam classifier.
For SVM hypothesis we use the following notation (linear kernel):

$$\min_{\theta} C \sum_{i=1}^{m} [y^{(i)} \text{cost}_1(\theta^T x^{(i)}) + (1 - y^{(i)}) \text{cost}_0(\theta^T x^{(i)})] + \frac{1}{2} \sum_{j=1}^{n} \theta_j^2.$$

If we use non-linear kernels then we use this form:

$$\min_{\theta} C \sum_{i=1}^{m} [y^{(i)} \text{cost}_1(\theta^T f^{(i)}) + (1 - y^{(i)}) \text{cost}_0(\theta^T f^{(i)})] + \frac{1}{2} \sum_{j=1}^{m} \theta_j^2$$

where $f^{(i)}$ is the feature vector, and for $1 \leq j \leq m$ we have

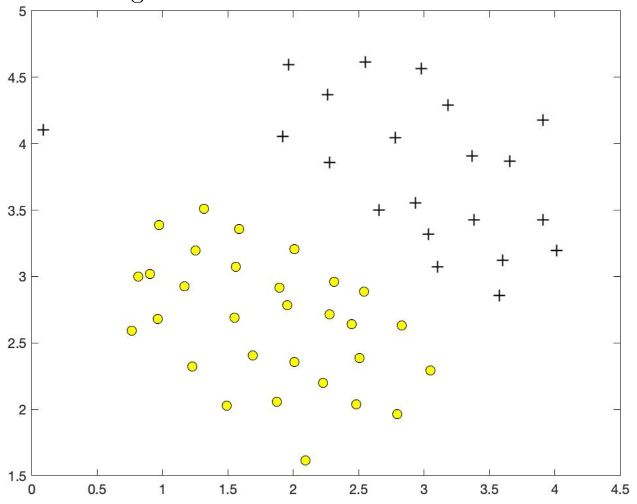
$$f_j^{(i)} = \text{similarity}(x^{(i)}, x^{(j)}).$$

The function "similarity" depends on the choice of kernel.

1.1 SVM with Linear kernels

We begin with a 2D example dataset (data1.mat) which can be separated by a linear boundary. (Figure 1)

Figure 1: Visualization of data1.mat



In this part, you will try using different values of the C parameter with SVMs. For a wide range of C (e.g. 1,10,50,100), use linear kernel for SVM to classify this data.

- Plot the decision boundary for each choice of C.
- Discuss what is the effect of C.
- What value of C seems to be good for this dataset?

You don't need to implement the SVM classifier yourself as it requires some technical details that were not covered in the class. A MATLAB function (svmTrain.m) is given to you. You can also use other functions if you want (for example functions available in the MATLAB Statistics and Machine Learning Toolbox for efficiently training SVM's). If you want to use "svmTrain.m", you can call it like this:
model = svmTrain(X, y, C, @linearKernel, 1e-3, 20);
model is an object with this information:
model.X, model.y, model.kernelFunction, model.b, model.alphas, model.w
To make it even easier for you, you are given two functions named "plotData" for plotting data and "visualizeBoundaryLinear.m" that you can use to plot your learned model when using linear kernel.
Implementation Note: Most SVM software packages (including svmTrain.m) automatically add the extra feature $x_0 = 1$ for you and automatically take care of learning the intercept term θ_0 . So when passing your training data to the SVM software, there is no need to add this extra feature $x_0 = 1$ yourself.

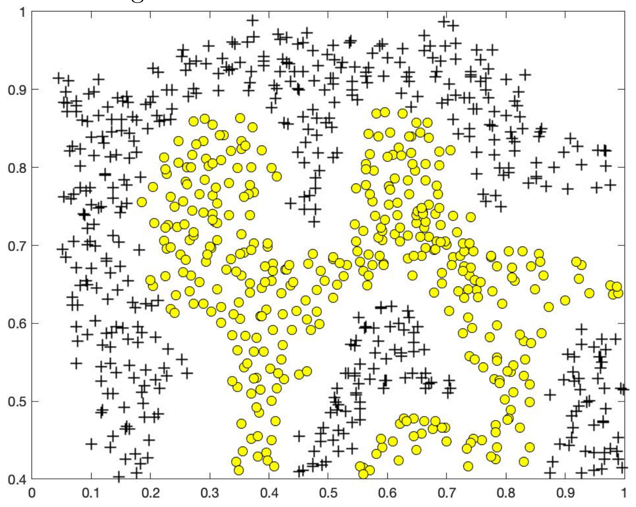
1.2 SVM with Gaussian kernels

In this part of the exercise, you will be using SVMs to do non-linear classification. In particular, you will be using SVMs with Gaussian kernels on datasets that are not linearly separable.

- What is the Gaussian kernel with parameter σ for a pair of examples $(x^{(i)}, y^{(i)})$?
- Write a function like
  function sim = gaussianKernel(x1, x2, sigma)
  that returns the Gaussian kernel between x1 and x2 and returns the value in sim.

You are given a dataset (data2.mat) that looks like this:

Figure 2: Visualization of data2.mat



You can use the "svmTrain" function this time with a Gaussian kernel to train a SVM classifier. To do so you can call it like this:
model= svmTrain(X, y, C, @(x1, x2) gaussianKernel(x1, x2, sigma));
where gaussianKernel is the function you implemented in the last part.

- For different parameters (C and σ) learn the SVM model.
- In order to see the decision boundary for your learned model you can use the function visualizeBoundary(X, y, model). Plot the decision boundaries of your learned models.
- What is the effect of σ and C?

1.3 Spam Classification

Many email services today provide spam filters that are able to classify emails into spam and non-spam email with high accuracy. In this part of the project, you will use SVMs to build your own spam filter. You will be training a classifier to classify whether a given email, x, is spam ($y = 1$) or non-spam ($y = 0$). In particular, you need to convert each email into a feature vector $x \in \mathcal{R}^n$. The following parts first will walk you through how such a feature vector can be constructed from an email.
The dataset included for this project is based on a subset of the SpamAssassin Public Corpus. You will only be using the body of the email (excluding the email headers).

1.3.1 Preprocessing emails [The code for this part is given to you]

Before starting on a machine learning task, it is usually insightful to take a look at examples from the dataset. Figure 3 shows a sample email ("emailSample1.txt") that contains a URL, an email address (at the end), numbers, and dollar amounts. While many emails would contain similar types of entities (e.g., numbers, other URLs, or other email addresses), the specific entities (e.g., the specic URL or specific dollar amount) will be different in almost every email.

Figure 3: Sample 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
```

Therefore, one method often employed in processing emails is to normalize these values, so that all URLs are treated the same, all numbers are treated the same, etc. For example, we could replace each URL in the email with the unique string "httpaddr" to indicate that a URL was present. This has the effect of letting the spam classifier make a classification decision based on whether any URL was present, rather than whether a specific URL was present. This typically improves the performance of a spam classifier, since spammers often randomize the URLs, and thus the odds of seeing any particular URL again in a new piece of spam is very small.
In processEmail.m, we have implemented the following email preprocessing and normalization steps:

– Lower-casing: The entire email is converted into lower case, so that capitalization is ignored (e.g., IndIcaTE is treated the same as indicate).

– Stripping HTML: All HTML tags are removed from the emails. Many emails often come with HTML formatting; we remove all the HTML tags, so that only the content remains.

– Normalizing URLs: All URLs are replaced with the text "httpaddr".

– Normalizing Numbers: All numbers are replaced with the text "number".

– Normalizing Dollars: All dollar signs ($) are replaced with the text "dollar".

– Word Stemming: Words are reduced to their stemmed form. For example, "discount", "discounts", "discounted" and "discounting" are all replaced with "discount". Sometimes, the Stemmer actually strips off additional characters from the end, so "include", "includes", "included", and "including" are all replaced with "includ".

– Removal of non-words: Non-words and punctuation have been removed. All white spaces (tabs, new-lines, spaces) have all been trimmed to a single space character.

Figure 4: Preprocessed Sample Email

```
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
```

The result of these preprocessing steps is shown in Figure 4. While preprocessing has left word fragments and non-words, this form turns out to be much easier to work with for performing feature extraction.

1.3.2 Extracting features from emails

After preprocessing the emails, we have a list of words for each email. The next step is to choose which words we would like to use in our classifier and which we would want to leave out. For this exercise, we have chosen only the most frequently occurring words as our set of words considered (the vocabulary list). Since words that occur rarely in the training set are only in a few emails, they might cause the model to overfit

our training set. The complete vocabulary list is in the file “vocab.txt”. 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.
Given the vocabulary list, we can now map each word in the preprocessed emails into a list of word indices that contains the index of the word in the vocabulary list.

- Complete the code in “processEmail.m” to perform this mapping. In the code, you are given a string str which is a single word from the processed email. You should look up the word in the vocabulary list vocabList and find if the word exists in the vocabulary list. If the word exists, you should add the index of the word into the word indices variable. If the word does not exist, and is therefore not in the vocabulary, you can skip the word (you are given all the required functions that this function requires). you can use the function “readFile.m” to open “emailSample1.txt”. So you can check your code like this:
  fileContents = readFile(“emailSample1.txt”);
  wordIndices = processEmail(fileContents);

- Write a function “x = emailFeatures(wordIndices)” to generate a feature vector for an email, given the word indices. Note that $x \in \mathcal{R}^n$ where $n = 1899$ and $x_i \in \{0, 1\}$. Check your function with the sample emails.

1.3.3 Training SVM for spam classification

We have prepared two datasets achieved by using the “processEmail” and “emailFeatures” functions and converted into a vector $x^{(i)} \in \mathcal{R}^{1899}$.

- Choose C properly and train a SVM (with linear kernel) to classify between spam ($y = 1$) and non-spam ($y = 0$) emails using “spamTrain.mat” dataset.
- What is the accuracy of the learned model on the same dataset.
- What is the accuracy of the learned model on “spamTest.mat” dataset?

2 Generative models with model mismatch

In this exercise, we will compare both generative and discriminative approaches for solving a simple 2-dimensional binary classification problem. In particular, we will observe that a generative learning algorithm using the proper model can perform better than a discriminative algorithm such as logistic regression. However, if the generative model does not describe the data properly, its accuracy can degrade significantly.
let $\{(x^i, y^i)\}_{i=1,\dots,N}$, $x^i \in \mathbb{R}^2$, $y^i \in \{0, 1\}$ be a training set produced by the following generative model:

$$\begin{aligned} y &\sim \text{Bernoulli}(\phi) \\ x|y = 0 &\sim \mathcal{N}(\mu^0, \Sigma^0) \\ x|y = 1 &\sim \mathcal{N}(\mu^1, \Sigma^1) \end{aligned} \tag{1}$$

where

$$\begin{aligned} \phi &= 0.3, \\ \mu^0 &= [-3, 0]^T, \qquad \mu^1 = [3, 0]^T \\ \Sigma_0 &= \begin{pmatrix} 11 & -9 \\ -9 & 11 \end{pmatrix} \qquad \Sigma_1 = \begin{pmatrix} 11 & 9 \\ 9 & 11 \end{pmatrix} \end{aligned}$$

The goal is to train an algorithm that properly classifies a previously unseen given sample x. In order to solve this binary classification problem, we will compare various approaches:

1. Generative model with Gaussian prior assuming different means and covariance matrices. This is assuming the same generative model as in (1).

2. Generative model with Gaussian prior assuming different means but same covariance matrix for both labels, i.e.,

$$\begin{aligned} y &\sim \text{Bernoulli}(\phi) \\ x|y=0 &\sim \mathcal{N}(\mu^0\Sigma) \\ x|y=1 &\sim \mathcal{N}(\mu^1,\Sigma) \end{aligned}$$

   This is the same generative model as seen in class.

3. Generative model assuming independent Laplace distribution for each variable, i.e.,

$$\begin{aligned} y &\sim \text{Bernoulli}(\phi) \\ x_i|y=0 &\sim \text{Laplace}(\mu_i^0, b_i^0) \quad i=1,2 \\ x_i|y=0 &\sim \text{Laplace}(\mu_i^1, b_i^1) \quad i=1,2 \end{aligned}$$

   where $\mu^0, \mu^1, b^0, b^1 \in \mathbb{R}^2$. Recall that the Laplace distribution is defined as:

$$x \sim \text{Laplace}(\mu, b) \Leftrightarrow p(x) = \frac{1}{2b}\exp\left(-\frac{|x-\mu|}{b}\right).$$

4. Logistic regression: Assume that the posterior probability distribution is of the form

$$p(y=1|x;\theta) = \frac{1}{1+\exp(-\theta^T\tilde{x})},$$

   where $\tilde{x} = [1x]$ and $\theta \in \mathbb{R}^3$. Then, optimize θ so as to maximize the likelihood of the data under this posterior distribution.

You are given a dataset of $N_{train} = 1000$ i.i.d. points $\{(x^i, y^i)\}_{i=1,\ldots,N_{train}}$ generated according to the generative model (1) (train.mat), as well as a test set containing $N_{test} = 1000$ points generated in the same way (test.mat).

- For each approach, train the model using the training data, and compute the classification error on both the train and the test set.
  - For each generative model, you will need to compute the maximum likelihood of all parameters of the model to obtain an approximate generative model $\tilde{p}(y), \tilde{p}(x|y=0), \tilde{p}(x|y=1)$, and predict y by maximizing the approximated posterior distribution, i.e.

$$\hat{y} = \text{argmax}_y \tilde{p}(y)\tilde{p}(x|y)$$

  - For logistic regression, maximize the log-likelihood using Gradient Ascent for $n = 1000$ steps with step size $\alpha = 0.001$.
- Compare the different approaches and explain why some perform better than others.