Machine Learning Fundamentals
Lab 1

Shubham Agarwal
Master 2 MSIAM Data Science
shubhamagarwal92@gmail.com

1 Introduction

Definitions and notations
In what follows we shall use the following notations :

- Y denotes the labels. Usually Y = -1,1 (binary classification)
- $x = (x_1, \ldots, x_p) \in X \subset R^P$ are the features
- $D_n = \{(x_i, y_i), i = 1, \ldots, n\}$ is the training set
- We assume that there exists a probabilistic model explaining the generation of our observations :
  $\forall i \in \{i = 1, \ldots, n\}, (x_i, y_i) iid \sim (X, Y)$
- Using the training set D_n we want to construct a prediction function $\hat{f} : X \to \{-1, 1\}$ which predicts an output y for a given new x with a minimum probability of error.

Data generation

We assume in this part that our data are bidimensional

1. Test the functions rand gauss(n,mu,sigma), rand bi gauss, rand clown and rand checkers. Explain what each function is doing

Rand gauss : Sample n points from a Gaussian variable with center mu and std deviation sigma

Rand bi gauss : Sample n1 and n2 points from two Gaussian variables centered in mu1, mu2, with std deviation sigma1, sigma2

Rand clown: Sample a dataset clown with n1 points and noise std deviation epsilon1 for the first class, and n2 points and noise std deviation epsilon2 for the second one

Rand checkers : Sample n1 and n2 points from a noisy checker

2. Save some datasets

We can save dataset using *pylab* package in python. Use the command

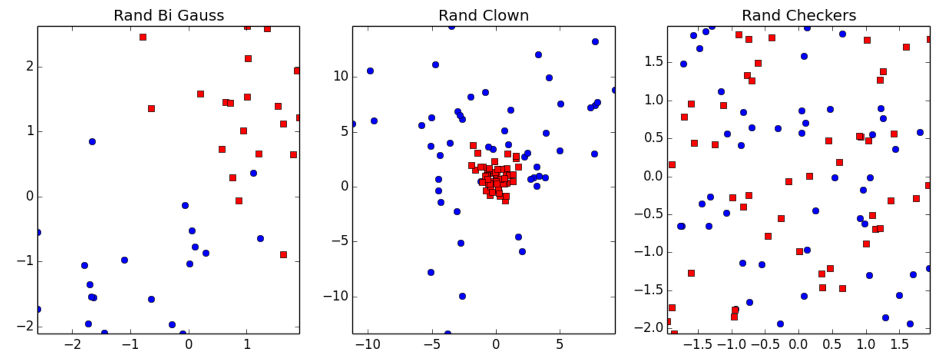
```
pylab.savefig('q1.png')
```

or use matplotlib as

```
plt.savefig('q1.png')
```

3. Plot some dataset using the function plot 2d

Extension to the multi-class setting
In the case where the output variable Y is not binary there is several ways to extend the binary setting



- One against one: one considers all possible pairs of labels (k,l) and fit a classifier $C_{kl}(X)$ for each one. We then predict the output which won most of the fights.
- One against all: for each k, we learn a classifier discriminating between Y = k and $Y \neq k$. Using the a posteriori probabilities, we set the most probable label

2 Logistic regression

Use $np.random.seed(0)$ (Seed for reproducability)

Import the package sklearn.linear model which contains the class LogisticRegression

```
from sklearn import linear_model
```

1. Create a model LogisticRegression :

```
lr = linear_model.LogisticRegression()
```

2. To learn the model from the data dataX and their corresponding labels dataY, one can use fit

```
lr.fit(dataX, dataY)
```

Here data1 = rand_bi_gauss and

```
dataX=data1[:,:2]
dataY=data1[:,2]
```

3. What is the variable coef of the model ? intercep ?

coef: array, shape(n_classes, n_features)Coefficient of the features in the decision function.

intercept_ : array, shape (n_classes,) Intercept (a.k.a. bias) added to the decision function. If fit_intercept is set to False, the intercept is set to zero.

```
Coeff: [[ 1.3469398    1.73858361]]
Intercept: [ 0.08632149]
```

4. What is the output of the function predict ? This of the function score ?

Predict: predict class labels for samples in X.

Parameters: X : (array-like, sparse matrix), shape = [n_samples, n_features] Samples.

Returns: C : array, shape = [n_samples] Predicted class label per sample.

We will generally use predict and score function on test set

Score: Returns the mean accuracy on the given test data and labels. In multi-label classification, this is the subset accuracy which is a harsh metric since you require for each sample that each label set be correctly predicted.

Parameters:

X : array-like, shape = (n_samples, n_features) Test samples.

y : array-like, shape = (n_samples) or (n_samples, n_outputs) True labels for X.

sample_weight : array-like, shape = [n_samples], optional Sample weights.

Returns: score : float Mean accuracy of self.predict(X) wrt. y.

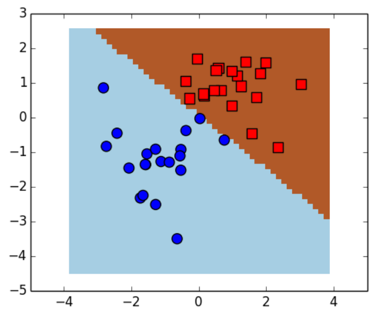
```
n1=10
n2=10
dataTest=rand_bi_gauss(n1,n2,mu1,mu2,sigma1,sigma2)
testX=dataTest[:,:2]
testY=dataTest[:,2]
predictClass = lr.predict(testX)
print "Score: "+ str(lr.score(testX,testY))
```

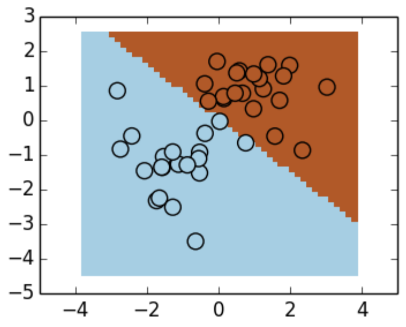
Thus, predict function predicts the class of each sample in the test set. (Here, we generated 20 test samples). Trained on rand gauss dataset and testing on different test samples generated from the same distribution we get mean accuracy as score:

```
Score: 0.9
```

5. Visualise the decision boundary.

We can visualize the decision boundary using three plotting options:

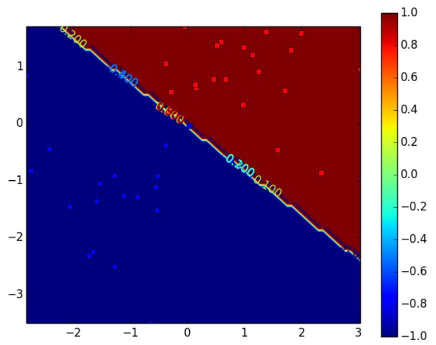




6. Apply the logistic regression of the data from the database zipcode available on the website

We have used zip dataset as stated. The dataset has the first column as class label and the rest features from 1:257 column(0-base indexing of column). We compared OneVsOne and OneVsRest and found OneVsOne to perform better:

```
One vs rest accuracy: 0.911
One vs one accuracy: 0.933
```



3 The perceptron

A perceptron is a linear binary classifier projecting each observation in R. The set of decision boundaries is the set of affine hyperplanes defined from a given vector $w = (w_0, w_1, \ldots w_p) \in R^{p+1}$ as

$H_w = \{x : \widehat{f_w}(x) = w_0 + \sum_i w_i x_i = 0\}$

To classify an observation x, one considers the position of x with respect to the hyperplane H_w. The predicted label is then $sign(\widehat{f_w}(x))$. The aim is to find the best hyperplane which separates the data according to their labels.

Classical perceptron We assume here that our datas are bidimensionnal. Use the synthetic datas to answer the following questions

1. What is the boundary of the perceptron? When is $\widehat{f_w}(x)$ large ? negative ? positive ? What does it mean ?

Given a perceptron, any decision boundary learnt must be a hyperplane

When $\widehat{f_w}(x)$ is large, it is well classified. When using step function, we can say that when $\widehat{f_w}(x)$ is positive, it is on the positive class and when $\widehat{f_w}(x)$ is negative, it is classified as negative class. When we make a decision based on a smooth function: If $f(z) > p$, classify "positive". Else, classify "negative". Where $z = \sum_i w_i x_i$ is the linear combination learned by the perceptron.

2. Implement a function predict(x,w) which gives $\widehat{f_w}(x)$ from x. Implement also the function predict_class(x,w) which gives as output the label $sign(\widehat{f_w}(x))$

```
def predict(x,w):
    return np.dot(x,w[1:])+w[0]

def predict_class(x,w):
    return np.sign(predict(x,w))
```

Cost function We want to measure the error on a dataset D_n. We then need to precise the loss function l that we consider. The cost $C_w = E[l(y, \widehat{f_w}(x))]$ is the expectation of the loss function on the whole dataset. Three loss functions are classically used

- The zero-one loss: $ZeroOneloss(y, \widehat{f_w}(x)) = |y - sign(\widehat{f_w}(x))|/2$
- The quadratic error: $MSEloss(y, \widehat{f_w}(x)) = (y - \widehat{f_w}(x))^2$
- The hinge loss: $Hingeloss(y, \widehat{f_w}(x)) = max(0, 1 - y\widehat{f_w}(x))$

We now study these different loss functions

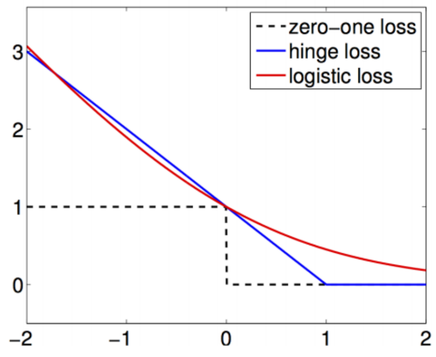
1. Implement these functions.

```
def zero_one_loss(x,y,w):
    return abs(y-np.sign(predict(x,w)))/2

def hinge_loss(x,y,w):
    return np.maximum(0,1-y*predict(x,w))

def mse_loss(x,y,w):
    return (y-predict(x,w))*(y-predict(x,w))
```

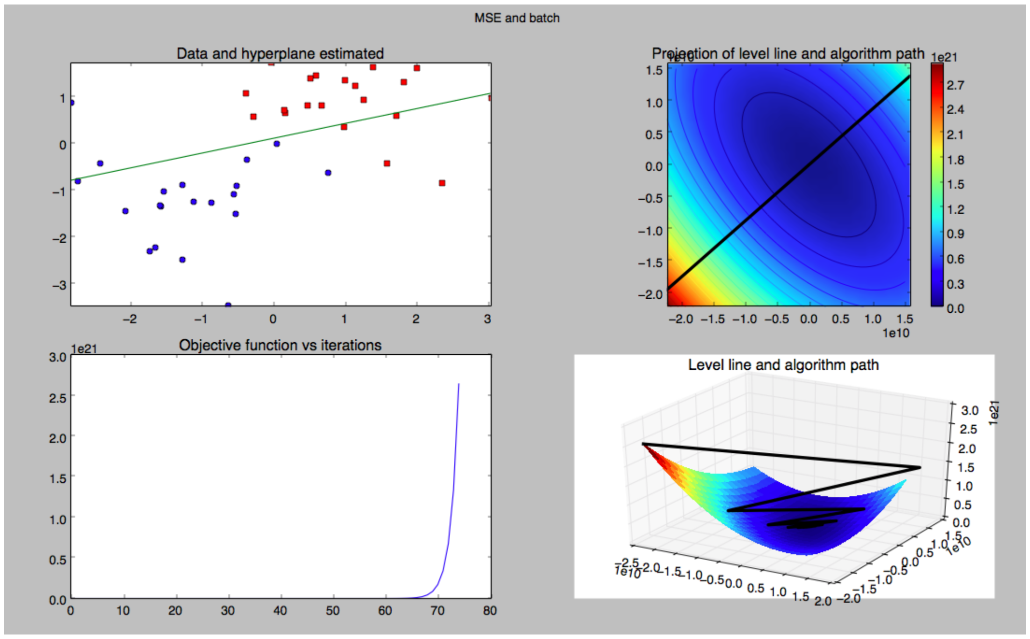
2. Fix w on a bidimensional example. How vary these function with respect to $\widehat{f_w}(x)$? With respect to x ? What is the interpretation?



From this graph, we can see zero-one loss as a step function which is not differentiable at 0. Logistic loss acts as convex upper bound on the minimization of the zero-one loss.

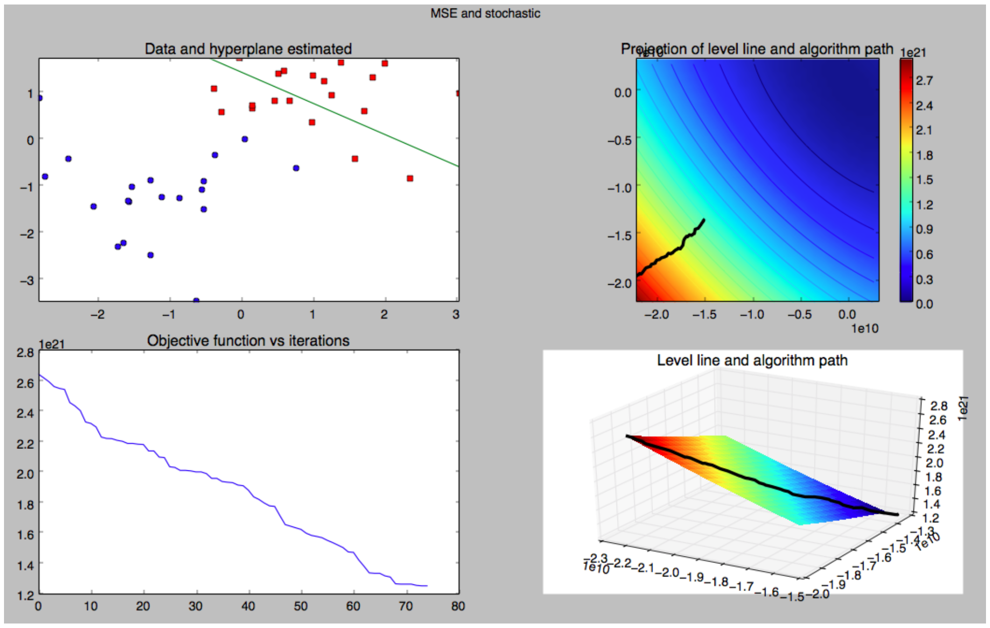
3. How can we observe the evolution of these two functions with respect to w for a given data set? Where is the vector

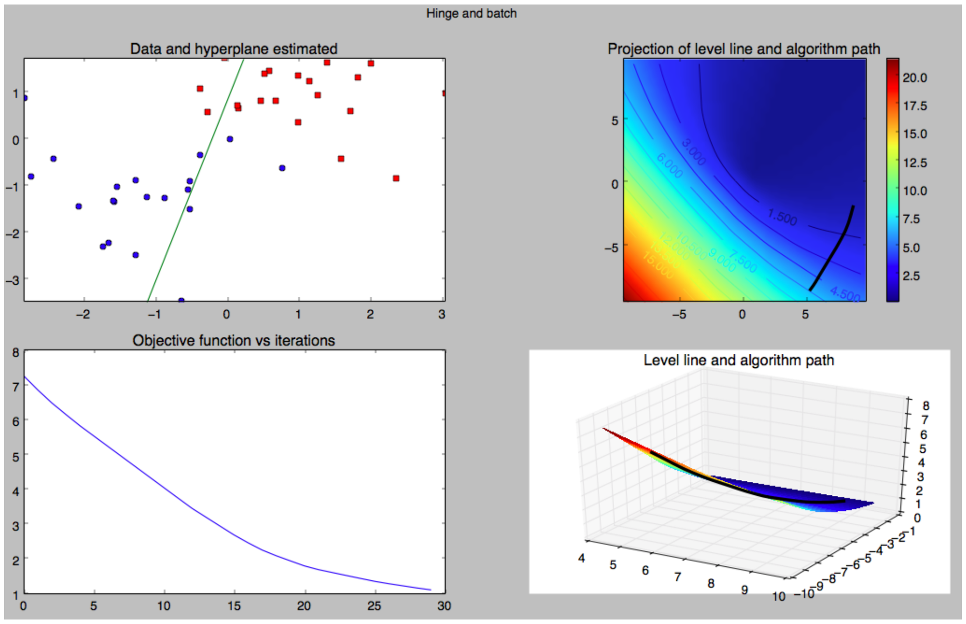
$w \in argmin_{w \in R^{p+1}} \frac{1}{n} \sum_{i=1}^{n} l(y, \widehat{f_w}(x_i))$

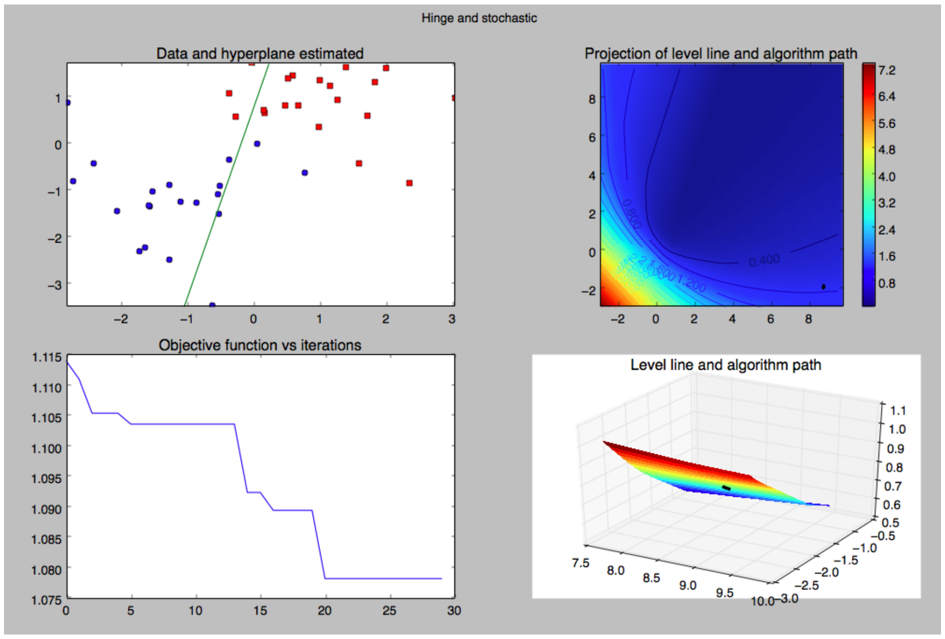


Learning the perceptron in practice

In the general case, to minimize the cost function one use a gradient descent.


MSE and stochastic
Data and hyperplane estimated
Projection of level line and algorithm path
Objective function vs iterations
Level line and algorithm path
1e21
1e10


Hinge and batch
Data and hyperplane estimated
Projection of level line and algorithm path
Objective function vs iterations
Level line and algorithm path



1. Recall the general perceptron algorithm

2. Experiment it on several datasets. Use either the functions given in the file either the sklearn package. One can also use the function SGD (Stochastic Gradient Descent)

```
from sklearn import linear_model
clf = linear_model.SGDClassifier()
clf.fit(dataX, dataY)
```

3. Study the numerical performances of the algorithm

Algorithm 1 The algorithm of perceptron

```
1: Training set $S = \{(x_i, y_i) \mid i \in \{1, \dots, m\}\}$
2: Initialize the weights $w^{(0)} \leftarrow 0$
3: $t \leftarrow 0$
4: Learning rate $\eta > 0$
5: repeat
6:     Choose randomly an example $(x^{(t)}, y^{(t)}) \in S$
7:     if $y \left\langle w^{(t)}, x^{(t)} \right\rangle < 0$ then
8:         $w_0^{(t+1)} \leftarrow w_0^{(t)} + \eta \times y^{(t)}$
9:         $w^{(t+1)} \leftarrow w^{(t)} + \eta \times y^{(t)} \times x^{(t)}$
10:    end if
11:    $t \leftarrow t + 1$
12: until $t > T$
```

4. The stochastic version of the algorithm consists in taking into account only the error on a randomly drawn example at each iteration. Modify the code and test it

This is called Stochastic Gradient Descent which is different from Batch Gradient Descent and have showed significant training reduction time.

```
def gradient(x,y,epsilon ,niter ,w_ini ,lfun ,gr_lfun ,stoch=True):
    w=w_ini
    loss=np.zeros(niter)
    for i in range(niter):
        if stoch:
            idx=[np.random.randint(x.shape[0])]
        else:
            idx=range(x.shape[0])
        w[i,:]=w[i-1,:]-epsilon*gr_lfun(x[idx,:],y[idx],w[i-1,:])
        loss[i]=lfun(x,y,w[i,:]).mean()
    return [w,loss]
```

5. Propose some variants on the stopping conditions of the algorithm

For the algorithm, we generally can take the bound on the number of iterations. Sometimes another stopping criteria is used, whereby if the update is not significant than a tolerance value, the algorithm is thought to have converge.

6. What is the main problem of perceptron ?

Although the perceptron learning algorithm converges to a solution, the number of iterations can be very large if the input vectors are not normalized and are arranged in an unfavorable way. The perceptron can be used to classify input vectors that can be separated by a linear boundary. We call such vectors linearly separable. Unfortunately, many problems are not linearly separable. The classic example is the XOR gate. It was the inability of the basic perceptron to solve such simple problems that led, in part, to a reduction in interest in neural network research during the 1970s.

Is Perceptron linear?

As explained in the previous question. Yes

1. What is the analytic formula of an ellipsis, an hyperbol and a parabol ?

For ellipse:

$\frac{x^2}{a^2} + \frac{y^2}{b^2} = 1$

For hyperbola:

$\frac{x^2}{a^2} - \frac{y^2}{b^2} = 1$

For parabola:

$y^2 = 4ax$

2. Propose a method to classify the clown dataset. Can we generalize? One can use the function poly2 of the source file associated to the lab. How can we use it to learn a perceptron?

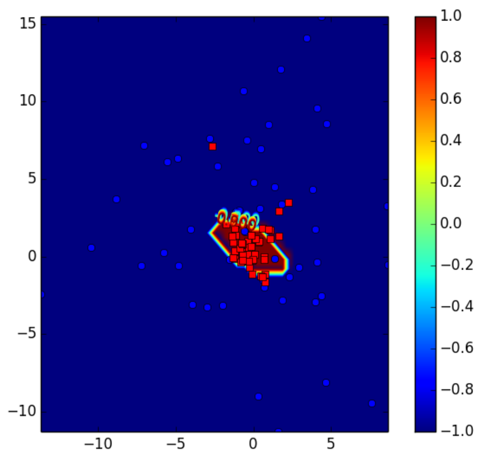
When our data is not linearly separable, we can use the kernel trick to project it on a higher dimensional space where it is linearly separable. poly2 function might act as kernel mapping for the perceptron to transform it into linearly separable data where perceptron can classify it very powerfully.

3. Give the stochastic version of the perceptron algorithm

4. On the clown dataset, perform some numerical experiments and plot the decision boundaries

Algorithm 1 The algorithm of perceptron

```
1:  Training set $S = \{(x_i, y_i) \mid i \in \{1, \dots, m\}\}$
2:  Initialize the weights $w^{(0)} \leftarrow 0$
3:  $t \leftarrow 0$
4:  Learning rate $\eta > 0$
5:  repeat
6:      Choose randomly an example $(x^{(t)}, y^{(t)}) \in S$
7:      if $y \left\langle w^{(t)}, x^{(t)} \right\rangle < 0$ then
8:          $w_0^{(t+1)} \leftarrow w_0^{(t)} + \eta \times y^{(t)}$
9:          $w^{(t+1)} \leftarrow w^{(t)} + \eta \times y^{(t)} \times x^{(t)}$
10:     end if
11:     $t \leftarrow t + 1$
12: until $t > T$
```


15
10
5
0
−5
−10
−10
−5
0
5
1.0
0.8
0.6
0.4
0.2
0.0
−0.2
−0.4
−0.6
−0.8
−1.0