Assignment 3: Supervised Learning Models

31 March 2024 20:02

MAINAK BISWAS
IISc, PhD student

[30 points – 3hours]

1) Linear Models, Linear Regression [5 points]

(a) Consider a classification problem. $D = \{x_i, y_i\}_{i=1}^{N}$. let $x \in \mathbb{R}^d$ $y = \{0, 1\}$. Let there be n_1 positive datapoints and n_0 negative datapoints.

(i) Now μ_1 is the means of the datapoints s.t $y = 1$. And μ_0 is the mean of features of negative datapoints.

Find μ_0, μ_1. (1)

(ii) Now you make a classifier $h(x)$ that classifies the datapoints based on the proximity to the means μ_0, μ_1, i.e

$$\hat{y}_{pred} = \underset{i=0,1}{\operatorname{argmin}} \|x - \mu_i\|^2$$

find the decision boundary of this model. (2)

(b) Say, you want to predict the number of runs a team will score using 100 features. But you only have 80 training examples. Argue why a linear regression model will not have an well-defined solution. (2)

[Hint: $\underset{80\times 1}{y} = \underset{(80\times 101}{X} \underset{(101\times 1)}{w}$ - Think in terms of the solution of w and the rank of X]

2) Discriminant functions [5 points]

Consider the discriminant-functions for discriminating between 2 classes.

$g_1(x) = P(C_1|x)$, $g_2(x) = P(C_2|x)$; s.t.

Given $x|C_1 \sim N\left(\begin{bmatrix}1\\2\end{bmatrix}, \begin{bmatrix}1 & 0\\0 & 2\end{bmatrix}\right)$ [illegible]

Given $x \mid C_1 \sim N\left(\begin{bmatrix}1\\2\end{bmatrix}, \begin{bmatrix}1 & 0\\0 & 2\end{bmatrix}\right)$ (where μ = $\begin{bmatrix}1\\2\end{bmatrix}$, Σ_1 = $\begin{bmatrix}1 & 0\\0 & 2\end{bmatrix}$) let $P(C_1) = 0.6$, $P(C_2) = 0.4$

$x \mid C_2 \sim N\left(\begin{bmatrix}0\\-1\end{bmatrix}, \begin{bmatrix}2 & 0\\0 & 2\end{bmatrix}\right)$

(i) Find the decision boundary : $g_1(x) = g_2(x)$. (3)

(ii) Hence plot the distribution of data (using contours) and the decision boundary using Python. (2)

[Use the same jupyter notebook for all the subsequent questions].

3) Support Vector Machines ; Sequential Minimal Optimization [7 points]

SVM objective (Recall.)

$$\underset{\alpha}{\text{maximize}}\ L(\alpha) = \sum_{i=1}^{n} \alpha_i - \frac{1}{2} \sum_{i=1}^{n} \sum_{j=1}^{n} \alpha_i \alpha_j y_i y_j K(x_i, x_j)$$

$$\text{s.t.} \sum_{i=1}^{n} \alpha_i y_i = 0$$

$$0 \le \alpha_i \le C$$

$y = \{-1, 1\}$

Now, we know to obtimize this objective we use coordinate ascent with, SMO algorithm; where only 2 variables are optimized in 1 step. Here let us only consider 1 step of the optimization as: (α_1, α_2 is being optimized)

Since we need to optimize only over α_1, α_2 (others: constant)

$$\alpha_1^*, \alpha_2^* = \underset{\alpha_1 \alpha_2}{\text{argmax}}\ w_0 + w_1 \alpha_1 + w_2 \alpha_2 - \{ w_3 \alpha_1 \alpha_2 + w_4 \alpha_1^2 + w_5 \alpha_2^2 \}$$

where $\alpha_1 y_1 + \alpha_2 y_2 = t$

$$0 \leq \alpha_1, \alpha_2 \leq C$$

Note: you don't need this detail for coding - (only for giving a valid input.)

In matrix form

$$\underset{\alpha_1 \alpha_2}{\text{argmax}} \quad w_0 + [w_1 \; w_2]\begin{bmatrix}\alpha_1 \\ \alpha_2\end{bmatrix} - [\alpha_1 \; \alpha_2]\begin{bmatrix} w_4 & w_3/2 \\ w_3/2 & w_5 \end{bmatrix}\begin{bmatrix}\alpha_1 \\ \alpha_2\end{bmatrix}$$

$$\begin{bmatrix} w_4 & w_3/2 \\ w_3/2 & w_5 \end{bmatrix} \overset{(psd)}{\geq} 0 \Leftarrow \text{(According to SVM this should be psd)}$$

(i) find α_1^*, α_2^*. [Hint: Session 19 – full SMO was taught] [5] – [In terms of the coeffs $w_0, w_1, w_2, w_3, y_1, y_2, t, C$]

(ii) Hence implement it using python. [2]

Inputs: $(w_0 \text{ to } w_5)$; y_1, y_2 (± 1), t, C

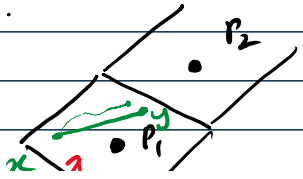
Output: α_1^*, α_2^*

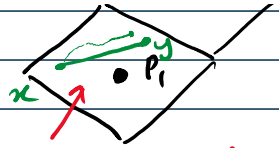
4) Nearest Neighbour Algorithm [3 points]

Consider a 1-nearest neighbour algorithm. Show that it divides the space into Dirichlet/Voronoi tessellation. The partitioning of the space is into 'convex polygons'; s.t. each polygon has exactly 1 generating point:

$\Downarrow$

i.e. There is only one point that points inside the polygon are close to. (Ignore points at the boundary of the polygons).





all pts here belong to P_1.

In other words: s.t. the polygon forms a convex set.

[Hint: $\theta x + (1-\theta) y \in$ Polygon $\forall\ x, y \in$ Polygon.

$\theta \in [0, 1]$] – all pts linearly connecting x and y has to be contained in the polygon.

Note: you can assume l2-distance.

[Hint: the cosine rule may be handy!]

5) Softmax Regression and MLP [10 points]

Dataset

Use the MNIST dataset [handwritten digits → (0-9)];
Use sklearn to import it as:

```
import numpy as np
from sklearn.datasets import fetch_openml

X,y=fetch_openml("mnist_784",version=1,return_X_y=True,as_frame=False)
```

MLP

(i) Divide the dataset into 40K (Train), 10K (CV set), 20K (test)

[s.t they have equal number of 0's, 1's, 2's ..... 9's]

– this is called a stratified split. [1]

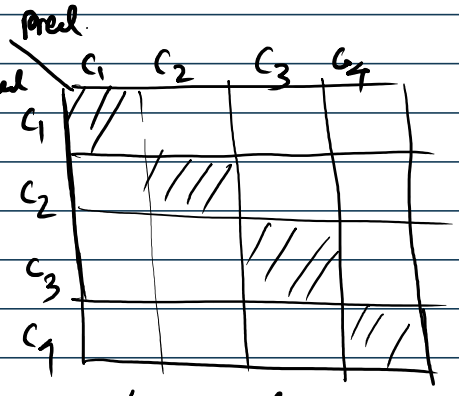
(ii) Now you can use the backprop code I shared. Modify it to make it predict a multiclass classification [0-9]. (10 way)

The last layer has to have 10 nodes.

Test it with 1, 2, and 3 hidden layers. Select the best model using the CV accuracy. Report the Accuracy, and confusion table on test set. (10×10 table)

Confusion table. pred

Contingency table



C_{ij} : the number of points in class i predicted as C_j

[6 marks]

(iii) Logistic Regression

Hence; divide the dataset ST you have to perform binary classification - between odd and even numbers.

Train set size : 60k

Test set size : 10k

O/P → 0 (even number)
1 (odd number)

[Hence implement a linear regression model]

Report. (i) Accuracy
(ii) Confusion table (2x2)

(iii) Precision (P) := $\frac{(\#\text{ True +ve})}{(\#\text{True +ve}) + (\#\text{ False +ve})}$

[% - of your +ve prediction that were correct]

(iv) Recall (R) := $\frac{(\#\text{ True +ve})}{(\#\text{True +ve}) + (\#\text{ False -ve})}$ (# you said you, actually no)

[% - +ve class data pts you missed]

(v) F1 Score := $\frac{2PR}{P+R}$

[3 points]

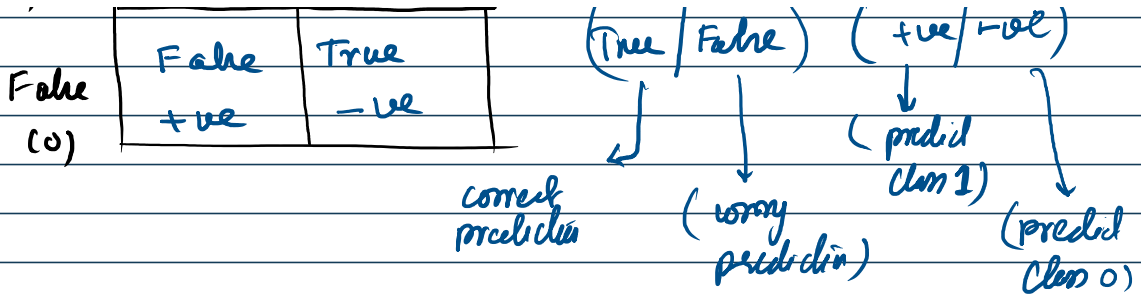
(you said no, actually yes)

: (traditional Confusion table)

Predicted / Actual	True (1)	False (0)
True (1)	True +ve	False -ve
False	False	True

meaning:

(True/False) (+ve/-ve)



Note: Packages allowed (Python)

(i) numpy
(ii) pandas
(iii) matplotlib
(iv) sklearn (only to import data - no predefined functions must be used)

Implementation Tips

1) Images $(N \times N) \longrightarrow$ flatten it to make $N^2 \times 1$ before using a MLP. {i.e. convert the matrix to a vector}

2) Don't forget to divide the image matrix by 255 - so that the image is scaled between 0, 1. It helps in stable training.