Machine Learning - Lecture Notes and Examples

Introduction

Material Basis

In this notebook, we are going to introduce both from a theoretical point of view as well as an experimental point of view (python implementation) some Machine Learning Algorithms. The current material, is heavily based uppon online: MIT - Machine Learning 6.036 Course (Fall 2020), provided by M.I.T. Associate Professor Tamara Broderick .

Lecture notes, implementations and approaches are derived by (at current time) Undergraduate Student Julian Proko, Department of Computer Science, University of Piraeus. - 05/2022.

Introductory Concepts

Machine Learning is used to provide a robust mechanism, in order for humans to aqcuire predictions (hypothesis functions) regarding some given problem under study. In order for us to present the main components used at our disposals, we will assume that we want to study factors which may or may not result for a human being to develop in some point in his/her life, type-2 diabetes:

Data Points:
Observations in our desired problem domain. We represent each i-th data point (or i-th observation) as $x^{(i)}$.

For example, if we have 7 patients that we study, each patient will be a data point ranging from $x^{(1)}$ up to $x^{(7)}$.

Features:
For each data point $x^{(i)}$, we extract numerical vectors, which represent a desired - feature set - of our observation points, as a numerical value. We represent the feature vector of $x^{(i)}$-th observation, as: $x^{(i)} = (x_1^{(i)}, x_2^{(i)}, x_3^{(i)}, \ldots, x_d^{(i)})$, $x^{(i)} \in R^d$.

For example, if for each patient we want to study their (1) avg B.M.I. for the last 5 years, (2) their avg daily calorie - balance for the last 5 years

their (3) avg excersise activity for the last 5 years, as well as (4) their current age, we will have to store a 4 numerical - value vector. Therefore, each $x^{(i)}$ - patient, would be represented by a feature - vector: $x^{(i)} = (x_1^{(i)}, x_2^{(i)}, x_3^{(i)}, x_4^{(i)}), x^{(i)} \in R^4$.

Data Labels:
Data labels, represent a fact for each of our observation, regarding the problem domain under study. Data labels, are represented by $y^{(i)}$, which indicates the label (or the known fact), for our i-th observation.

Following our example, we may want to know whether or not, each one of our patients (each $x^{(i)}$) has currently developed type - 2 diabetes. Given that our problem can be answered by only two possible answers : namely, yes, the patient has developed type - 2 diabetes or no, the patient has not developed type - 2 diabetes, we can represent each label (the known fact) as a binary value: example: $y^{(i)} \in -1, 1$. Therefore, if $y^{(i)} = -1$, we know for a fact that the i-th patient has not developed type - 2 diabetes whereas, if $y^{(i)} = 1$, we know for a fact that i-th patient has developed type 2 diabetes.

Summarization:
Based uppon all the prementioned concepts, for our given type - 2 diabetes problem, we want to be able to have a hypothesis - function returned by our Machine Learning Algorithm, uppon which we may perform the following operations:

$$\forall x_{new}^{(i)}, h(x_{new}^{(i)}) = y_{new}^{(i)}, y_{new}^{(i)} \in \{-1, 1\}$$

to forecast, whether or not any new patient (of whom we do not have observed a decisive fact), will or will not develop type - 2 diabetes.

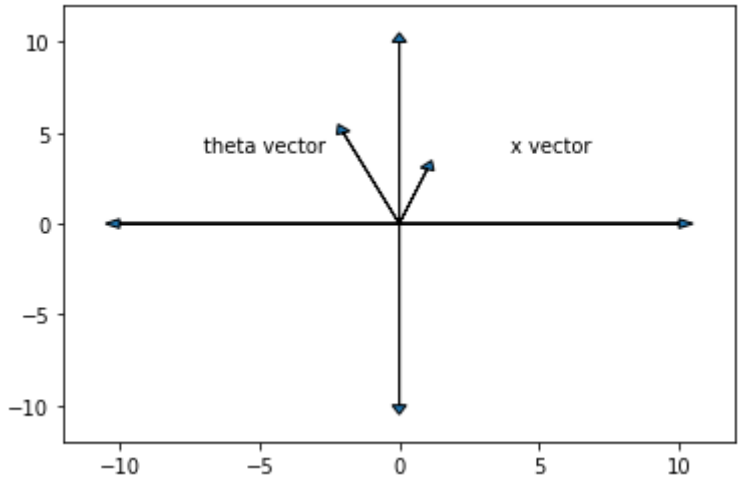
Linear Classifiers

The first family of algorithms that produce hypothesis functions for classification, is going to be the set of Linear Classifiers. In order for us to aqcuire a more in - depth understanding of linear classifiers and the logic used behind them, we are going to set some mathematical groundwork.

Let us assume that we are in the 2d Euclidean Space, and we want to represent 2 vectors. We are going to refer to these vectors as $\overline{x}$ and $\overline{\theta}$. We can represent those vectors in the plane as follows:

In [1]:
```
# Package for scientific computing with Python
# https://numpy.org/
import numpy as np
```

```python
# Matplotlib is a library for creating static, animated, and interactive visualizations in Python.
# https://matplotlib.org/
import matplotlib.pyplot as plt

# We want to depict vertical and horizontal x,y axis as well as our desired x and theta vector

# our horizontal and vertical graph axes
ax = plt.axes()

# x - axis positive :
# from  (0.0, 0.0) to (10.0, 0.0)
ax.arrow(0.0, 0.0, 10.0, 0.0, head_width=0.5, head_length=0.5)

# x - axis negative:
# from  (0.0, 0.0) to (-10.0, 0.0)
ax.arrow(0.0, 0.0, -10.0, 0.0, head_width=0.5, head_length=0.5)

# y-axis positive
# from (0.0, 0.0) to (0.0, 10.0)
ax.arrow(0.0, 0.0, 0.0, 10.0, head_width=0.5, head_length=0.5)

# y-axis negative
# from (0.0, 0.0) to (0.0, -10.0)
ax.arrow(0.0, 0.0, 0.0, -10.0, head_width=0.5, head_length=0.5)

# x vector
# from (0.0, 0.0) to (1.0, 3.0)
ax.arrow(0.0, 0.0, 1.0, 3.0, head_width=0.5, head_length=0.5)
ax.text(4, 4, "x vector")

# theta vector
# from (0.0, 0.0) to (-2.0, 5.0)
ax.arrow(0.0, 0.0, -2.0, 5.0, head_width=0.5, head_length=0.5)
ax.text(-7, 4, "theta vector")

# y axes of figure limits
plt.ylim(-12,12)

# x axes of figure limits
plt.xlim(-12,12)

plt.show()
```



If we want to find the projection α of $\overline{x}$ vector on to $\overline{\theta}$ vector, we can derive that from:

$$\overline{\alpha} = \frac{\overline{\theta^T} \cdot \overline{x}}{\|\theta\|}$$

Let us find use the prementioned formula based uppon our example:

$\overline{\theta} = (-2.0, 5.0)$

$\overline{x} = (1.0, 3.0)$

$\overline{\theta^T} \cdot \overline{x} = (-2.0, 5.0)^T \cdot (1.0, 3.0) = (-2.0 \cdot 1.0) + (5.0 \cdot 3.0) = -2 + 15 = 13$

$\|\theta\| = \sqrt{(-2.0 - 0.0)^2 + (5.0 - 0.0)^2} = \sqrt{4 + 25} = \sqrt{29} = 5.38$

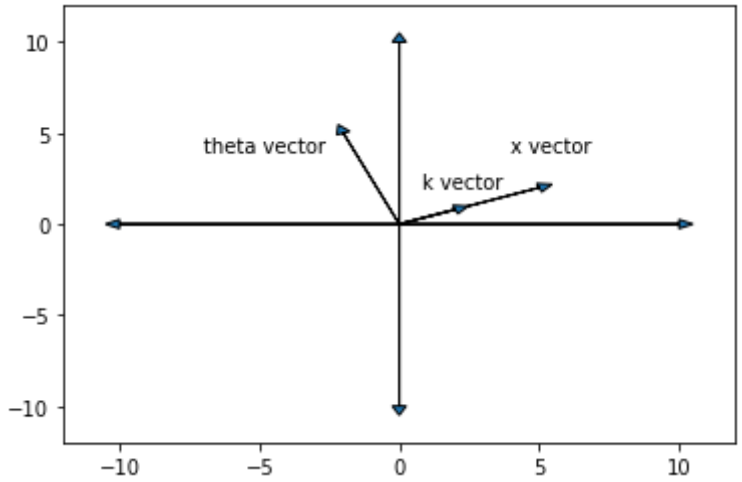
$\alpha = 2.41$

This α actually indicates that there is α quantity, of vector $\overline{x}$ uppon vector $\overline{\theta}$.

If vector $\overline{x}$ is perpendicular to vector $\overline{\theta}$, then we expect $\alpha = 0$, no matter the scale of the vector. That can be easily verified for vectors $\overline{x} = [5, 2]$, or $\overline{k} = [2, 4/5]$

In [2]:
```
# Package for scientific computing with Python
# https://numpy.org/
import numpy as np

# Matplotlib is a library for creating static, animated, and interactive visualizations in Python.
# https://matplotlib.org/
import matplotlib.pyplot as plt

# We want to depict vertical and horizontal x,y axis as well as our desired x and theta vector

# our horizontal and vertical graph axes
ax = plt.axes()

# x - axis positive :
# from  (0.0, 0.0) to (10.0, 0.0)
ax.arrow(0.0, 0.0, 10.0, 0.0, head_width=0.5, head_length=0.5)

# x - axis negative:
# from  (0.0, 0.0) to (-10.0, 0.0)
ax.arrow(0.0, 0.0, -10.0, 0.0, head_width=0.5, head_length=0.5)

# y-axis positive
# from (0.0, 0.0) to (0.0, 10.0)
ax.arrow(0.0, 0.0, 0.0, 10.0, head_width=0.5, head_length=0.5)

# y-axis negative
# from (0.0, 0.0) to (0.0, -10.0)
ax.arrow(0.0, 0.0, 0.0, -10.0, head_width=0.5, head_length=0.5)

# x vector
# from (0.0, 0.0) to (5.0, 2.0)
ax.arrow(0.0, 0.0, 5.0, 2.0, head_width=0.5, head_length=0.5)
ax.text(4, 4, "x vector")

# k vector
# from (0.0, 0.0) to (2.0, 0.8)
```

```
ax.arrow(0.0, 0.0, 2.0, 0.8, head_width=0.5, head_length=0.5)
ax.text(0.8, 2, "k vector")


# theta vector
# from (0.0, 0.0) to (-2.0, 5.0)
ax.arrow(0.0, 0.0, -2.0, 5.0, head_width=0.5, head_length=0.5)
ax.text(-7, 4, "theta vector")

# y axes of figure limits
plt.ylim(-12,12)

# x axes of figure limits
plt.xlim(-12,12)

plt.show()
```



We can generate a more general observation, that every vector $\overline{x}$ that is perpendicular to the vector θ, can be derived from the following formula:

$$\overline{x} : \frac{\overline{\theta^T} \cdot \overline{x}}{\|\overline{\theta}\|} = 0$$

We observe, that the perpendicular line towards vector $\overline{\theta}$ is described by the prementioned formula.

We can generilize even more the previous observation, by suggesting that every vector $\overline{x}$ that has a projection of some constant α uppon the

vector $\overline{\theta}$, can be derived from the formula:

$$\overline{x} : \frac{\overline{\theta^T} \cdot \overline{x}}{\|\overline{\theta}\|} = \alpha$$

for positive projection of length α, or:

$$\overline{x} : \frac{\overline{\theta^T} \cdot \overline{x}}{\|\overline{\theta}\|} = -\beta$$

for negative projection of length β.

By all the predefined notions, we can come up with a formula to classify all given vectors $\overline{x}$ in respective relation to vector of reference θ, which is based uppon the following:

If our vector $\overline{x}$ makes an angle $< 90^o$ in relation with vector $\overline{\theta}$, the projection of $\overline{x}$ up to $\overline{\theta}$ will be positive ($\alpha > 0$).

If the angle of the two vectors is $> 90^o$, the projection will become negative ($\alpha < 0$).

If the angle is exactly $= 90^o$, the projection will equal to 0 ($\alpha == 0$).

We can divide therefore our data-set hyper-space (R^d) into 2 disjoint sub - hyper - spaces ($S_1 \in R^d$, $S_2 \in R^d$), by hyper-plane:

$$\overline{\theta^T} \cdot \overline{x} + \beta \cdot \|\overline{\theta}\| = 0 \Rightarrow$$

$$\overline{\theta^T} \cdot \overline{x} + \theta_0 = 0,$$
$$\theta_0 = \beta \cdot \|\overline{\theta}\|$$

$$\overline{\theta}, \overline{x} \in R^d$$

We can therefore derive a linear classifying hypothesis (meaning a hypothesis that is described by a line or linear hyper-plane), as:

$$h(x^{(i)}) = sign(\overline{\theta^T} \cdot \overline{x^{(i)}} + \theta_0) =$$
$$\{-1 : h(x^i) \leq 0,$$
$$+1 : h(x^i) > 0\}$$

Evaluation of Classifiers

There are some parameters that can determine whether or not a classifier is performing in a desired manner.

1. First of all, our classifier must perform well in future (not yet seen by classifier) data points (observations). In order to evaluate this parameter, we must check how good or how bad our classifier performs for a set of test - data points, from which the label (a certain fact) is already known.

2. To measure the behaviour of the classifier in a numerical term, we introduce the notion of the loss function. This function is responsible for describing whether or not the classifier prediction, matches the observation data - point label. A very simple example of loss function in linear classifiers is the assymetric function. By assymetric loss function, we assign a loss of

$$0, y^{(i)} == h(x^{(i)})$$

,

$$+1, y^{(i)} ! = h(x^{(i)})$$

By taking the average of k testing observation data points, we retrieve the training error of the classifier. More specifically, training error $E(h)$:

$$E(h) = \frac{1}{k} \sum_{i=1}^{k} L(h(x^{(i)}, y^{(i)}))$$

Our ultimate goal is to minimize the training error.

Another way to view this observation is that our ultimate goal, is to extract a hypothesis, which will return the minimum training error.

Perceptron Algorithm

In this section, we are going to introduce the percepton algorithm. Perceptron algorithm, will generate a hypothesis function (linear classifier) for a given data set under consideration. From the explanation of the perceptron algorithm, we may receive a wealth of introductory needed concepts.

Introductory Example

Task: Given 2-d observation data points, return a hypothesis (using perceptron algorithm) that classifies them in the best possible way

In [3]:
```
# Package for scientific computing with Python
# https://numpy.org/
import numpy as np

# Matplotlib is a library for creating static, animated, and interactive visualizations in Python.
# https://matplotlib.org/
import matplotlib.pyplot as plt

# our horizontal and vertical graph axes
ax = plt.axes()

# x - axis positive:
# origin_vector_x, origin_vector_y, dx, dy
ax.arrow(0.0, 0.0, 12, 0.0, head_width=0.8, head_length=0.8)

# x - axis negative:
# origin_vector_x, origin_vector_y, dx, dy
ax.arrow(0.0, 0.0, -5.0, 0.0, head_width=0.8, head_length=0.8)

# y-axis positive
```