Pipeline ?

Supervised Learning

Date
Page No.

(2)

Code for a basic Pipeline for supervised learning

→ Multiple classifier can solve the same problem.

We will build up a little more intuition for what it means for an algorithm to learn something from data, because that sounds kinds of magical, but it is not

*

Imagine you are building a spam classifier - that's just a function that labels an incoming email as spam or not spam

Measurement/Dataset/feature

Email	Label
Click here to claim your prize!	Spam
What's new	Not spam
Hang out later?	Not spam
You have won $1000,000	Spam

Q how accurate will it be when you use it to classify emails that were not in ~~the~~ your training data?

→ So 1st we verify our model work well before deploy them by doing an experiment — the approach is to partion our data set into two parts (1) train (2) Test

We use train to train our model & test to see how accurate it is on new data.

Code.

```
from sklearn.datasets import load_iris
iris = load_iris()
```

X = iris.data → features
Y = iris.target → labels.

f(x) = y

At high level classifier is a function where X behave like input & Y behave like output.

spliting the data for training & testing

```
from sklearn.model_selection import train_test_split
X_train, X_test, Y_train, Y_test = train_test_split
(X, Y, test_size = 0.5)
```

↳ Half data used for testing (50/50 split)

features & labels for testing set

features & labels for the training set

Creating classifier

```
from sklearn import tree
my_classifier = tree.DecisionTreeClassifier()
```

There may be different type of classifier at high level having similar interface

using diffent classifer for same task

```
from sklearn.neighbors import KNeighborsClassifier
my_classifier = KNeighborsClassifier()
```

train our classifier using training data

```
my_classifier.fit(X_train, Y_train)
```

Call predict method & use it to classify our test data

```
predictions = my_classifier.predict(X_test)
print predictions
```

↳ output → A list of numbers that correspond to the type of iris the classifier predicts for each testing data.

Now let's see how accurate our classifier has on the testing set.

→ To calculate accuracy, we compare the predicted labels to the true labels & tally up the score.

```
from sklearn.metrics import accuracy_score
print accuracy_score(y_test, predictions)
```

Output → 0.97333

$f(x) = y$

features → labels

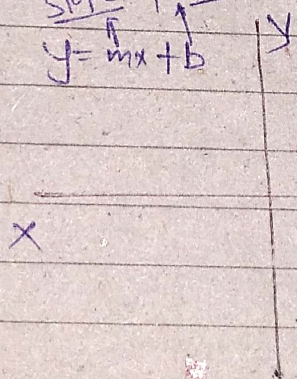
X & y are the input & output of a function

```
def classify(features):
    # logic
    return label
```

As we already know in supervised learning we don't need to write ourselves. We want algorithm to learn it from training data.

function is the mapping from input to output values

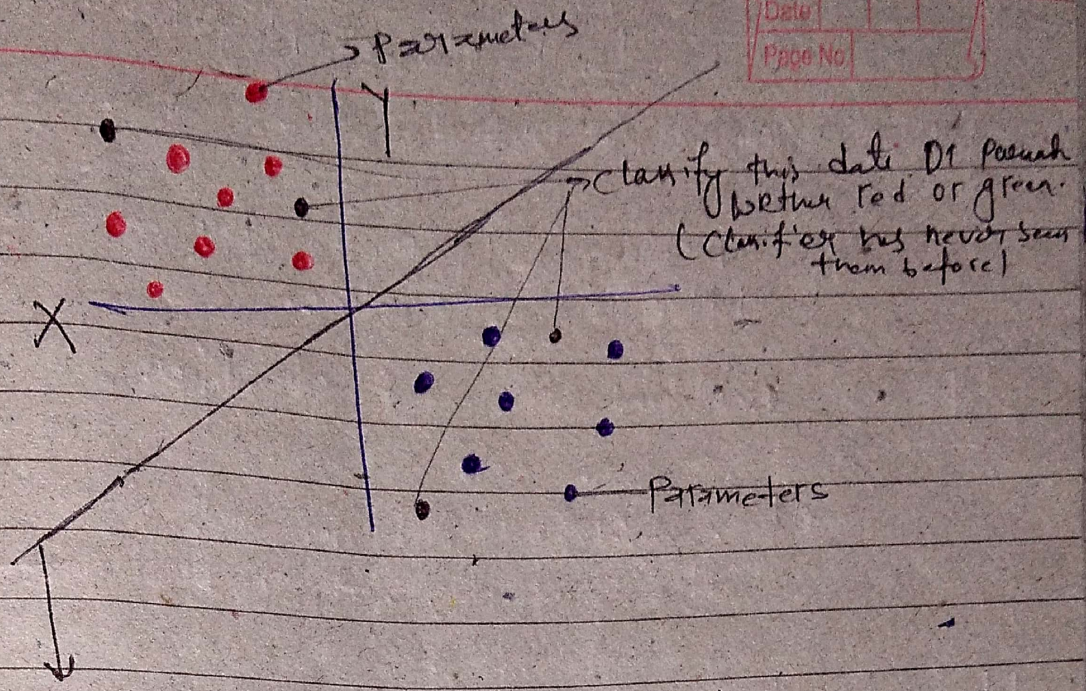
$y = mx + b$ (slope, y intercept)



We start with the model, It is the prototype or the rule that define the body of our function.

typically a model has parameters that we can adjust with our training data.

Neural Network is also a type of classifi



Probably we may be we draw a line & say the left of line is red data & Right is blue data & this line can serve as a classifier

$y = mx + b$

$m =$ So we only need to adjust two

$b =$ Parameter m & b

move the line Accordig to the Parameter

Right = Dont move

Wrong → adjust

Tensorflow Playground