**Classification**

* **Types of Classification Model**
  + Binary Classification
  + Multiclass Classification
  + Multilabel Classification

**MNIST Dataset Used**

* Each image is of 28\*28 Pixels i.e., 784 Features
* Total 7000 images

**# fetch\_mldata downloads data in the file structure scikit\_learn\_data/mldata/mnist-original.mat**

from sklearn.datasets import fetch\_mldata

mnist = fetch\_mldata("MNIST original")

X, y = mnist["data"], mnist["target"]

X.shape

**Plot the Image**

import matplotlib

some\_digit = X[36000] **# Selecting the 36,000th image.**

some\_digit\_image = some\_digit.reshape(28, 28) # Reshaping it to get the 28x28 pixels

plt.imshow(some\_digit\_image, cmap = matplotlib.cm.binary, interpolation="nearest")

plt.axis("off")

plt.show()

y[36000]

**Dividing dataset into training and test in python:**

X\_train, X\_test, y\_train, y\_test = X[:60000], X[60000:], y[:60000], y[60000:]

np.random.seed(42)

shuffle\_indices = np.random.premutation(len(X))

X\_train,X\_test = X\_train[shuffle\_indices],X\_test[shuffle\_indices]

**Stochastic Gradient Descent (SGD) Classifier**

* Capable of handling large datasets
* Deals with training instances independently
* Well suited for online training
* Instead of trying all lines, go into the direction yielding better results

**Hinge Loss :** Hinge Loss is a Loss function, which will tell how badly a Classifier is Doing.

**Problem Statement : *‘5’ and ‘Not 5’ classifier (Binary Classifier)***

y\_train\_5 = (y\_train == 5)

y\_test\_5 = (y\_test == 5)

from sklearn.linear\_model import SGDClassifier

sgd\_clf = SGDClassifier(random\_state=42,max\_iter=10)

sgd\_clf.fit(x\_train,y\_train\_5)

sgd\_clf.predict(X[36000])

**Performance Measure :**

1. **Cross Validation – Accuracy :**

from sklearn.model\_selection import cross\_val\_score

cross\_val\_score(sgd\_clf,X\_train,y\_train\_5,cv=10,scoring=’Accuracy’)

**Sometimes Accuracy is not a good performance Measure, especially in case of Skewed Dataset. So we need other Performance Measure also.**

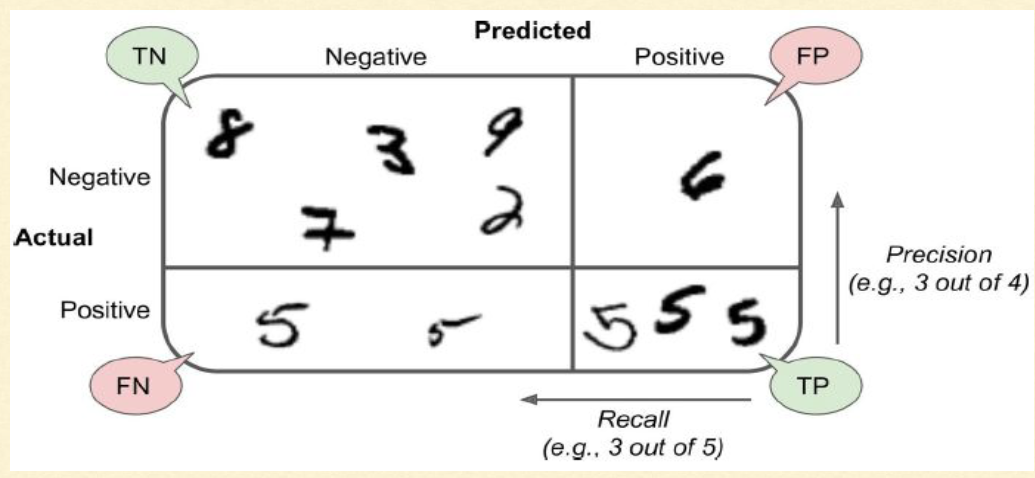
1. **Confusion Matrix :** The general idea is to count the number of times instances of class A are classified as class B.

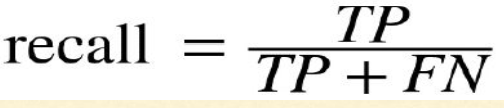
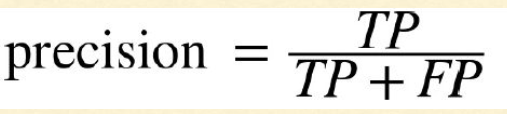
from sklearn.model\_selection import cross\_val\_predict

y\_pred = cross\_val\_predict (sgd\_clf,X\_train,y\_train\_5,cv=3)

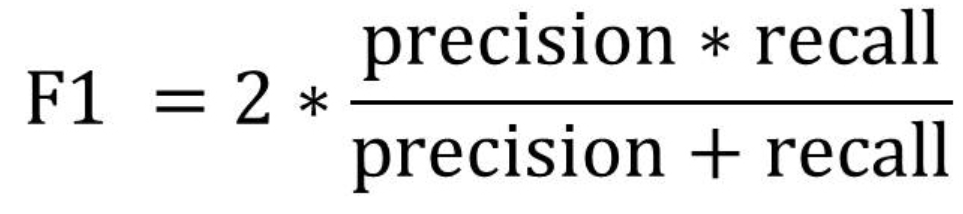
from sklearn.metrics import confusion\_matrix

confusion\_matrix(X\_train,y\_pred)





**F1 Score :** combine precision and recall into a single metric called the F1 score



from sklearn.metrics import precision\_score,recall\_score,f1\_score

precision\_score(y\_train\_5,y\_pred)

recall\_score(y\_train\_5,y\_pred)

f1\_score(y\_tain\_5,y\_pred)

* **Increasing precision/threshold reduces recall, and vice versa.**
* Classification is done based on a score as calculated by the decision function in SGD Classifier
  + Score above a certain threshold is classified as positive class
  + Score below a certain threshold is classified as negative class

y\_score = sgd\_clf.decision\_function([some\_digit])

threshold = 0

y\_some\_dig\_pred = (y\_score>threshold)

threshold = 20000

y\_some\_dig\_pred = (y\_score>threshold)

* **How to decide the best threshold?**
  + **Get the scores of all the training dataset using cross\_val\_predict with decision\_function as function**
  + **Compute the precision and recall for all possible thresholds using precision\_recall\_curve()**
  + **Plot both precision and recall for the thresholds using matplotlib**
  + **Select the threshold value that gives the best precision/ recall tradeoff**

y\_scores = cross\_val\_predict(sgd\_clf,X\_train,y\_train\_5,cv=3,method =’decision\_function’)

from sklearn.metrices import precision\_recall\_curve

precisions,recalls,thresholds = precision\_recall\_curve(y\_train\_5,y\_scores)

***# Plotting our results***

**def** plot\_precision\_recall\_vs\_threshold(precisions, recalls, thresholds):

plt.figure(figsize=(16,5))

*# Removing last value to avoid divide by zero in precision computation*

plt.plot(thresholds, precisions[:-1], "b--", label="Precision")

plt.plot(thresholds, recalls[:-1], "g-", label="Recall")

plt.xlabel("Threshold")

plt.legend(loc="upper left")

plt.ylim([0, 1])

plot\_precision\_recall\_vs\_threshold(precisions, recalls, thresholds)

plt.show()

* **Another way to select a good precision/recall tradeoff is to plot precision directly against recall directly.**

**def** plot\_precision\_vs\_recall(precisions, recalls):

plt.figure(figsize=(18,7))

plt.plot(recalls[:-1], precisions[:-1], "b-", label="Precision")

plt.xlabel("Recall")

plt.ylabel("Precision")

plt.legend(loc="upper left")

plt.ylim([0, 1])

plot\_precision\_vs\_recall(precisions, recalls)

plt.show()

* **ROC Curve : The receiver operating characteristic (ROC) curve is another common tool used with binary classifiers.Instead of plotting precision versus recall, the ROC curve plots the true positive rate (another name for recall) against the false positive rate**

from sklearn.metrices import roc\_curve

fpr,tpr,thresholds = roc\_curve(y\_train\_5,y\_scores)

**def** plot\_roc\_curve(fpr, tpr, label=**None**):

plt.figure(figsize=(18,5))

plt.plot(fpr, tpr, linewidth=2, label=label)

plt.plot([0, 1], [0, 1], 'k--')

plt.axis([0, 1, 0, 1])

plt.xlabel('False Positive Rate')

plt.ylabel('True Positive Rate')

plot\_roc\_curve(fpr, tpr)

plt.show()

* **A perfect classifier shall have a ROC Area Under the Curve (AUC) equal to 1 whereas a purely random classifier shall have ROC AUC = 0.5.**

from sklearn.metrics import roc\_auc\_score

roc\_auc\_score = (y\_train\_5,y\_scores)

* **Comparision of SGDClassifier and RandomForestClassifier on the basis of ROC-AUC**
* **RandomForestClassifier class does not have a decision\_function() method. Instead it has a predict\_proba() method**

from sklearn.ensamble import RandomForestClassifier

forest\_clf = RandomForestClassifier(random\_state=42)

y\_probas\_forest = cross\_val\_predict(forest\_clf,X\_train,y\_train\_5,cv=3,method=’predict\_proba’)

**# ROC auc score of SGDClassifier**

from sklearn.metrices import roc\_auc\_score

roc\_auc\_score(y\_train\_5,y\_probas\_forest)

# **Multiclass Classification**

* Classification done between multiple Classes.
* Two Strategies : **OVA** & **OVO**
* **OVA:** Classify the image into K-Classes, And, select the Class having maximum score.
* **OVO :** Train every digit in pair. Total Classifier = **N\*(N-1)/2**
* By default, **SGDClassifier** uses **OVA.**

from sklearn.multiclass import OneVSOneClassifier

ovo\_clf = OneVSOneClassifier(SGDClassifier(random\_state=42,max\_iter=20))

ovo\_clf.fit(X\_train,y\_train)

ovo\_clf.predict([Some\_digit])

🡪 In SGD Classifier, better to use Standard Scaler method

from sklearn.preprocessing import StandardScaler

scaler = StandardScaler()

X\_train\_scaled = scaler.fit\_tranform(X\_train.astype(np.float64))

cross\_val\_score(sgd\_clf,X\_train\_scaled,y\_train,cv=3,scoring=’accuracy’)

* **Error Analysis :**
  + Observe by plotting the Confusion Matrix

Y\_train\_pred = cross\_val\_predict(sgd\_clf,X\_train\_scaled,y\_train,cv=3)

con\_mat = confusion\_matrix(y\_train,Y\_train\_pred)

plt.matshow(con\_mat,cmap=plt.cm.gray)

**#Remove the Correct Prediction**

row\_sums = conf\_mx.sum(axis=1, keepdims=**True**)

norm\_conf\_mx = conf\_mx / row\_sums

np.fill\_diagonal(norm\_conf\_mx, 0)

plt.matshow(norm\_conf\_mx, cmap=plt.cm.gray)

plt.show()

*# Analyze individual Error*

**import** **os**

**def** plot\_digits(instances, images\_per\_row=10, \*\*options):

size = 28

images\_per\_row = min(len(instances), images\_per\_row)

images = [instance.reshape(size,size) **for** instance **in** instances]

n\_rows = (len(instances) - 1) // images\_per\_row + 1

row\_images = []

n\_empty = n\_rows \* images\_per\_row - len(instances)

images.append(np.zeros((size, size \* n\_empty)))

**for** row **in** range(n\_rows):

rimages = images[row \* images\_per\_row : (row + 1) \* images\_per\_row]

row\_images.append(np.concatenate(rimages, axis=1))

image = np.concatenate(row\_images, axis=0)

plt.imshow(image, cmap = matplotlib.cm.binary, \*\*options)

plt.axis("off")

**Observations :**

SGDClassifier assigns a weight per class to each pixel and calculate the total score for a class. Since 3s and 5s differ by a few pixels, the classifier is easily confused - Main difference between 3 and 5 is the position of the straight line.

**Solution:** Preprocess the image to ensure that they are well-centered and not rotated.

# **Multilabel Classification :**

Ex : Identify the Digit & whether it is Even or Odd.

y\_train\_large = (y\_train>=7)

y\_train\_odd = (y\_train%2==1)

y\_multilabel = np.c\_[y\_train\_large,y\_train\_odd]

# **KNeighborsClassifier supports multilabel classification but not all classifiers do.**

from sklearn.neighbors import KNeighborsClassifier

knn\_clf = KNeighborsClassifier()

knn\_clf.fit(X\_train,y\_multilabel)

knn\_clf.predict([some\_digit])

y\_knn\_predict = cross\_val\_predict(knn\_clf,X\_train,y\_train,cv=3)

f1\_score(y\_train,y\_knn\_predict,method=’macro’)

f1\_score(y\_train,y\_knn\_predict,method=’weighted’)

# **Multi-Output Classification**

generalization of multilabel classification where each label can be **multiclass. Ex : Removing noise from the Images.**

**#Adding Noise to the Image**

import numpy.random as rnd

noise\_train = rnd.randint(0,100,(len(X\_train)),784)

X\_train\_mod = X\_train + noise\_train

noise\_test = rnd.randit(0,100,(len(X\_test)),784)

X\_test\_mod = X\_test + noise\_test

Y\_train\_mod = X\_train

Y\_test\_mod = X\_test

**#Plotting the Noised Image**

**def** plot\_digit(array):

array\_image = array.reshape(28, 28)

plt.imshow(array\_image, cmap = matplotlib.cm.binary, interpolation="nearest")

plt.axis("off")

plt.show()

plot\_digit(X\_test\_mod[4000])

**#Clean the Image**

from sklearn.neighbors import KNeighborsClassifier

knn\_clf = KNeighborsClassifier()

knn\_clf.fit(X\_train\_mod,y\_train\_mod)

clean\_digit = knn\_clf.predict(X\_test\_mod)

knn\_clf.kneighbors\_graph()