Applied machine learning
Group assignment 4

Team members-Group 8:

- Abdelrhman Gaber Youssef Saad Rezkallah
- Eman Metwally Mohammed Abood
- Basma Reda Shaban Abd-Elsalam Abd-Elwahab

Part 1: Calculation:

Table 1:

Weather (F1)	Temperature (F2)	Humidty (F3)	Wind (F4)	Hiking (Labels)
Cloudy	Cool	Normal	Weak	No
Sunny	Hot	High	Weak	Yes
Rainy	Mild	Normal	Strong	Yes
Cloudy	Mild	High	Strong	No
Sunny	Mild	High	Strong	No
Rainy	Cool	Normal	Strong	No
Cloudy	Mild	High	Weak	Yes
Sunny	Hot	High	Strong	No
Rainy	Cool	Normal	Weak	No
Sunny	Hot	High	Strong	No

Hiking→P(yes) = $\frac{3}{10}$, P(No) = $\frac{7}{10}$

a-

$$\text{Gini} = 1 - \sum_{i=1}^{NC} (p)^2$$

Gini Index For Weather(F1):

$$Gini_{cloudy} = 1 - (\frac{1}{3})^2 - (\frac{2}{3})^2 = \frac{4}{9}$$

$$Gini_{Sunny} = 1 - (\frac{1}{4})^2 - (\frac{3}{4})^2 = \frac{3}{8}$$

$$Gini_{Rainy} = 1 - (\frac{1}{3})^2 - (\frac{2}{3})^2 = \frac{4}{9}$$

Gini Split For Weather = $\frac{3}{10}*\frac{4}{9} + \frac{3}{10}*\frac{4}{9} + \frac{4}{10}*\frac{3}{8} = \frac{5}{12}$

Gini Index For Temperature (F2):

$$Gini_{Cool} = 1 - (\frac{0}{3})^2 - (\frac{3}{3})^2 = 0$$

$$Gini_{Hot} = 1 - (\frac{1}{3})^2 - (\frac{2}{3})^2 = \frac{4}{9}$$

$$Gini_{Mild} = 1 - (\frac{2}{4})^2 - (\frac{2}{4})^2 = \frac{1}{2}$$

Gini Split For Temperature = $\frac{3}{10}*0 + \frac{3}{10}*\frac{4}{9} + \frac{4}{10}*\frac{1}{2} = \frac{1}{3}$

Gini Index For Humidity (F3):

$$Gini_{Normal} = 1 - (\frac{1}{4})^2 - (\frac{3}{4})^2 = \frac{3}{8}$$

$$Gini_{High} = 1 - (\frac{2}{6})^2 - (\frac{4}{6})^2 = \frac{4}{9}$$

Gini Split For Humidity= $\frac{4}{10} * \frac{3}{8} + \frac{6}{10} * \frac{4}{9} = \frac{5}{12}$

Gini Index For Wind(F4):

$$Gini_{Weak} = 1 - (\frac{2}{4})^2 - (\frac{2}{4})^2 = \frac{1}{2}$$

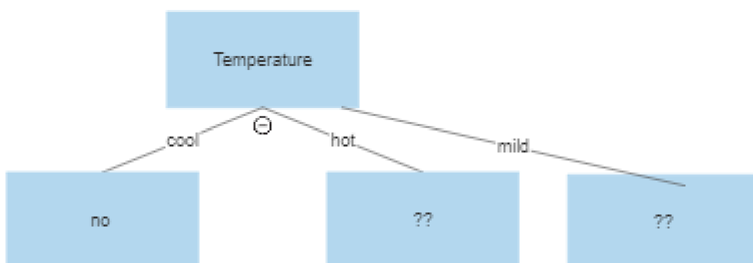
$$Gini_{Strong} = 1 - (\frac{1}{6})^2 - (\frac{5}{6})^2 = \frac{5}{18}$$

Gini Split For Wind= $\frac{4}{10} * \frac{1}{2} + \frac{6}{10} * \frac{5}{18} = \frac{11}{30}$

After computing Gini Split, the minimum value of Gini index is for feature Temperature, So, We Chose Temperature as a root node.

For the "Cool" branch

The Gini index for Weather, Humidity and wind are Equal Zero With Cool, this mean that the "Cool" branch is pure.



For the "Hot" branch

weather	Temperature	Humidity	Wind	Hiking
Sunny	Hot	High	Weak	Yes
Sunny	Hot	High	Strong	No
Sunny	Hot	High	Strong	No

The Gini Index For Weather (F1):

$$Gini_{Sunny} = 1 - (\frac{1}{3})^2 - (\frac{2}{3})^2 = \frac{4}{9}$$

Gini Split For Weather = $\frac{3}{3} * \frac{4}{9} = \frac{4}{9}$

Gini Index For Humidity(F3):

$$Gini_{High} = 1 - (\frac{1}{3})^2 - (\frac{2}{3})^2 = \frac{4}{9}$$

Gini Split For Humidity= $\frac{3}{3}*\frac{4}{9}=\ \frac{4}{9}$

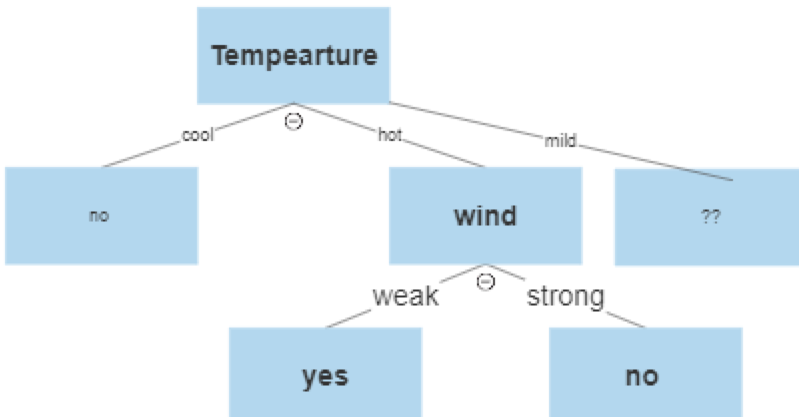
Gini Index For Wind(F4):

$$Gini_{Weak} = 1 - (\frac{1}{1})^2 = 0$$

$$Gini_{Strong} = 1 - (\frac{2}{2})^2 = 0$$

Gini Split For Wind= $\frac{1}{1}*0 + \frac{2}{2}*0 = \ 0$

From the results the minimum value of Gini index is "Wind" so it will be chosen as the child node for "hot" branch.



For the "Mild" branch

weather	Temperature	Humidity	Wind	Hiking
Rainy	Mild	Normal	Strong	Yes
Cloudy	Mild	High	Strong	No
Sunny	Mild	High	Strong	No
Cloudy	Mild	High	Weak	Yes

The Gini Index For Weather (F1):

$$Gini_{Rainy} = 1 - (\frac{1}{1})^2 = 0$$

$$Gini_{Sunny} = 1 - (\frac{1}{1})^2 = 0$$

$$Gini_{cloudy} = 1 - (\frac{1}{2})^2 - (\frac{1}{2})^2 = \frac{1}{2}$$

Gini Split For Weather = $\frac{1}{4}*0 + \frac{2}{4}*\frac{1}{2} + \frac{1}{4}*0 \ = \ \frac{1}{4}$

Gini Index For Humidity(F3)

$$Gini_{Normal} = 1 - (\frac{1}{1})^2 = 0$$

$$Gini_{High} = 1 - (\frac{1}{3})^2 - (\frac{2}{3})^2 = \frac{4}{9}$$

Gini Split For Humidity= $\frac{1}{4}$*0 + $\frac{3}{4}$*$\frac{4}{9}$= $\frac{1}{3}$

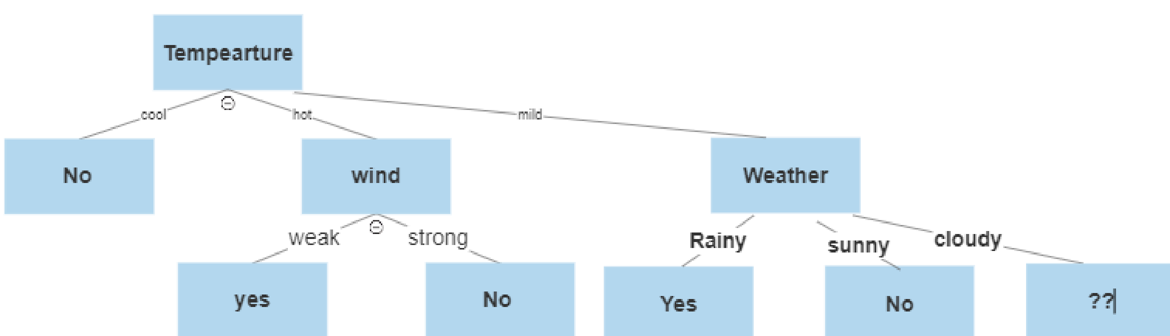
Gini Index For Wind(F4)

$$Gini_{Weak} = 1 - (\frac{1}{1})^2 = 0$$

$$Gini_{Strong} = 1 - (\frac{1}{3})^2 - (\frac{2}{3})^2 = \frac{4}{9}$$

Gini Split For Wind= $\frac{1}{4}$*0 + $\frac{3}{4}$*$\frac{4}{9}$= $\frac{1}{3}$

From the results the minimum value of Gini index is "Weather" so it will be chosen as the child node for "Mild" branch.



For the "Cloudy" branch

weather	Temperature	Humidity	Wind	Hiking
Cloudy	Mild	High	Strong	No
Cloudy	Mild	High	Weak	Yes

Gini Index For Humidity(F3):

$$Gini_{High} = 1 - (\frac{1}{2})^2 - (\frac{1}{2})^2 = \frac{1}{2}$$

Gini Split For Humidity= $\frac{2}{2}$*$\frac{1}{2}$ = $\frac{1}{2}$

Gini Index For Wind(F4)

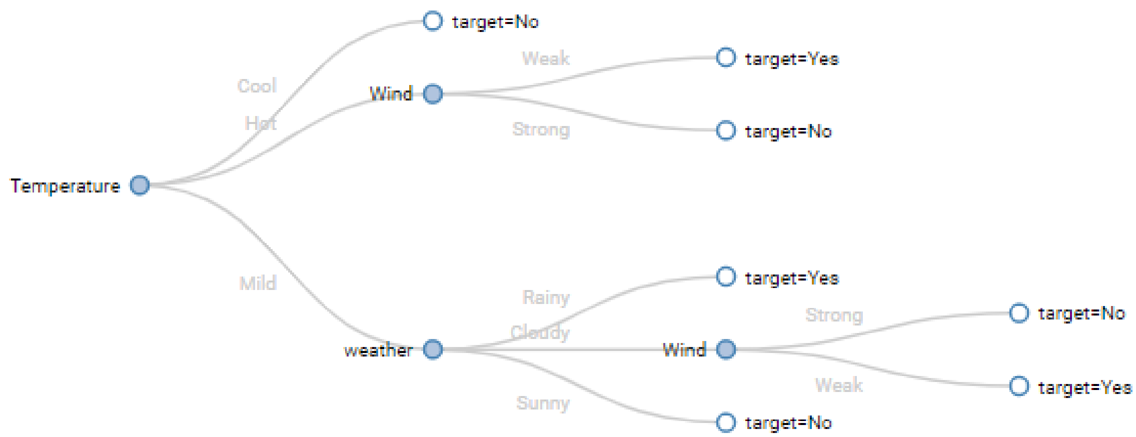
$$Gini_{Weak} = 1 - (\frac{1}{1})^2 = 0$$

$$Gini_{Strong} = 1 - (\frac{1}{1})^2 = 0$$

Gini Split For Wind= $\frac{1}{2} * 0 + \frac{1}{2} * 0 =$ 0

From the results the minimum value of Gini index is “Wind” so it will be chosen as the child node for “Cloudy” branch.

The final decision tree based on the Gini index split:



b-

Entropy(p) = -p Log p

IG(T, a) = Entropy(T)—Entropy(T|a)

The Entropy of "hiking" labels

Entropy $= \frac{-3}{10}\log_2\frac{3}{10} - \frac{7}{10}\log_2\frac{7}{10} = 0.88$

Information gain for weather (f1)

$P(\text{Cloudy and Hiking}) = \frac{-1}{3}\log_2\frac{1}{3} - \frac{2}{3}\log_2\frac{2}{3} = 0.918$

$P(\text{Sunny and Hiking}) = \frac{-1}{4}\log_2\frac{1}{4} - \frac{3}{4}\log_2\frac{3}{4} = 0.81$

$P(\text{Rainy and Hiking}) = \frac{-1}{3}\log_2\frac{1}{3} - \frac{2}{3}\log_2\frac{2}{3} = 0.918$

Gain $(\text{Hiking, Weather (F1)}) = 0.88 - \frac{3}{10}*0.918 - \frac{4}{10}*0.81 - \frac{3}{10}*0.918 = 0.097$

Information gain for Temperature (f2)

$P(\text{Cool and Hiking}) = \frac{0}{3}\log_2\frac{0}{3} - \frac{3}{3}\log_2\frac{3}{3} = 0$

$P(\text{Hot and Hiking}) = \frac{-1}{3}\log_2\frac{1}{3} - \frac{2}{3}\log_2\frac{2}{3} = 0.918$

$P(\text{Mild and Hiking}) = \frac{-2}{4}\log_2\frac{2}{4} - \frac{2}{4}\log_2\frac{2}{4} = 1$

Gain $(\text{Hiking, Temperature(F2)}) = 0.88 - \frac{3}{10}*0 - \frac{4}{10}*0.918 - \frac{4}{10}*1 = 0.2$

Information gain for Humidity(F3)

$P(\text{High and Hiking}) = \frac{-2}{6}\log_2\frac{2}{6} - \frac{4}{6}\log_2\frac{4}{6} = 0.918$

$P(\text{Normal and Hiking}) = \frac{-1}{4}\log_2\frac{1}{4} - \frac{3}{4}\log_2\frac{3}{4} = 0.81$

Gain $(\text{Hiking, Humidity(F3)}) = 0.88 - \frac{4}{10}*0.81 - \frac{6}{10}*0.918 = 0.006$

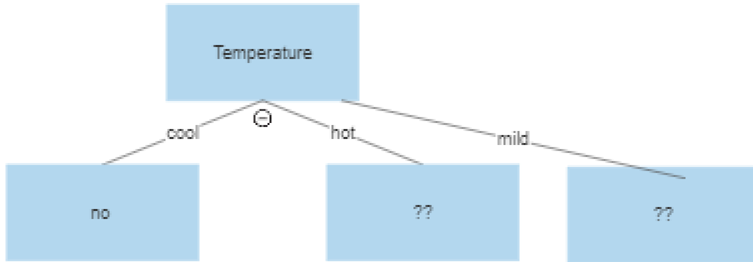
Information gain for Wind(F4)

$P(\text{Strong and Hiking}) = \frac{-1}{6}\log_2\frac{1}{6} - \frac{5}{6}\log_2\frac{5}{6} = 0.65$

$P(\text{Weak and Hiking}) = \frac{-2}{4}\log_2\frac{2}{4} - \frac{2}{4}\log_2\frac{2}{4} = 1$

Gain $(\text{Hiking, Wind(F4)}) = 0.88 - \frac{4}{10}*1 - \frac{6}{10}*0.65 = 0.09$

From the results the maximum value of Information Gain is "Temperature" so it will be chosen as the Root node.



For the "Cool" branch

The Information gain for Weather, Humidity and wind are Equal Zero With Cool , this mean the "Cool " branch was pure.

For the "Hot" branch

weather	Temperature	Humidity	Wind	Hiking
Sunny	Hot	High	Weak	Yes
Sunny	Hot	High	Strong	No
Sunny	Hot	High	Strong	No

The Entropy = $\frac{-1}{3}\log_2\frac{1}{3} - \frac{2}{3}\log_2\frac{2}{3} = 0.918$

Information gain for weather (f1)

P(Sunny and Hiking) $= \frac{-1}{3}\log_2\frac{1}{3} - \frac{2}{3}\log_2\frac{2}{3} = 0.918$

Gain (Hiking, Weather (F1)) $= 0.918 - \frac{3}{3} * 0.918 = 0$

Information gain for Humidity(F3)

P(High and Hiking) $= \frac{-1}{3}\log_2\frac{1}{3} - \frac{2}{3}\log_2\frac{2}{3} = 0.918$

Gain (Hiking, Humidty(F3)) $= 0.918 - \frac{3}{3} * 0.918 = 0$

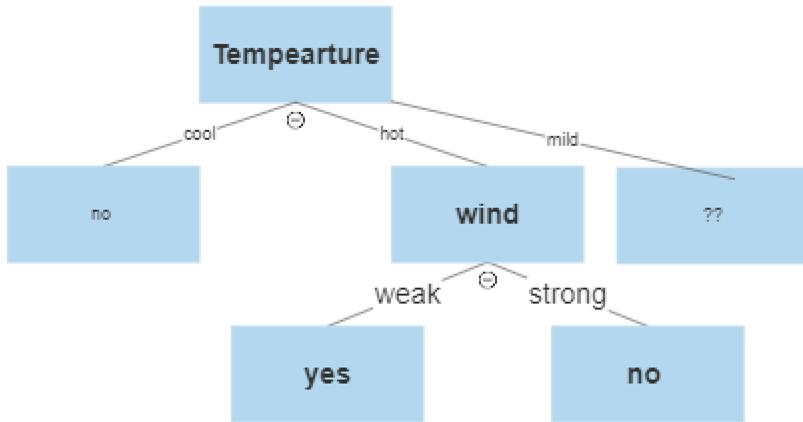
Information gain for Wind(F4)

P(Strong and Hiking) $= -\frac{2}{2}\log_2\frac{2}{2} = 0$

P(Weak and Hiking) $= \frac{-1}{1}\log_2\frac{1}{1} = 0$

Gain (Hiking, Wind(F4)) $= 0.918 - \frac{1}{3} * 0 - \frac{2}{3} * 0 = 0.918$

From the results the maximum value of Information Gain is "Wind" so it will be chosen as the child node for "Hot" branch.



For the "Mild" branch

weather	Temperature	Humidity	Wind	Hiking
Rainy	Mild	Normal	Strong	Yes
Cloudy	Mild	High	Strong	No
Sunny	Mild	High	Strong	No
Cloudy	Mild	High	Weak	Yes

The Entropy = $\frac{-2}{4}\log_2\frac{2}{4} - \frac{2}{4}\log_2\frac{2}{4} = 1$

Information gain for weather (f1)

P(Cloudy and Hiking) = $\frac{-1}{2}\log_2\frac{1}{2} - \frac{1}{2}\log_2\frac{1}{2} = 1$

P(Sunny and Hiking) $= \frac{-1}{1}\log_2\frac{1}{1} = 0$

P(Rainy and Hiking) $= \frac{-1}{1}\log_2\frac{1}{1} = 0$

Gain (Hiking, Weather (F1)) = $1 - \frac{2}{4}*1 - \frac{1}{4}*0 - \frac{1}{4}*0 = 0.5$

Information gain for Humidity(F3)

P(High and Hiking) $= \frac{-1}{3}\log_2\frac{1}{3} - \frac{2}{3}\log_2\frac{2}{3} = 0.918$

P(Normal and Hiking) $= \frac{-1}{1}\log_2\frac{1}{1} = 0$

Gain (Hiking, Humidity(F3)) = $1 - \frac{3}{4}*0.918 - \frac{1}{4}*0 = 0.31$

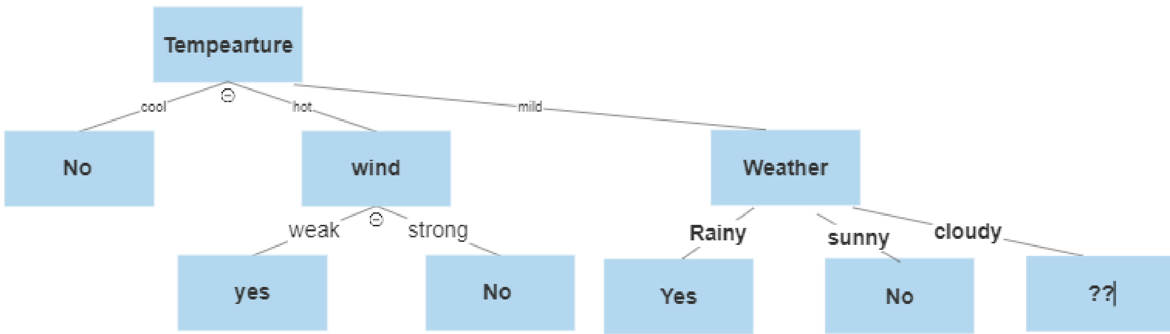
Information gain for Wind(F4)

P(Strong and Hiking) $= \frac{-1}{3}\log_2\frac{1}{3} - \frac{2}{3}\log_2\frac{2}{3} = 0.918$

P(Weak and Hiking) $= \frac{-1}{1}\log_2\frac{1}{1} = 0$

Gain (Hiking, Wind(F4)) = $1 - \frac{3}{4}*0.918 - \frac{1}{4}*0 = 0.31$

From the results the maximum value of Information Gain is “Weather” so it will be chosen as the child node for “Mild” branch.



For the “Cloudy” branch

weather	Temperature	Humidity	Wind	Hiking
Cloudy	Mild	High	Strong	No
Cloudy	Mild	High	Weak	Yes

The Entropy = $\frac{-1}{2}\log_2\frac{1}{2} - \frac{1}{2}\log_2\frac{1}{2} = 1$

Information gain for Humidty(F3)

P(High and Hiking) $= \frac{-1}{2}\log_2\frac{1}{2} - \frac{1}{2}\log_2\frac{1}{2} = 1$

Gain (Hiking, Humidty(F3)) = $1 - \frac{1}{2}*1 = 0$

Information gain for Wind(F4)

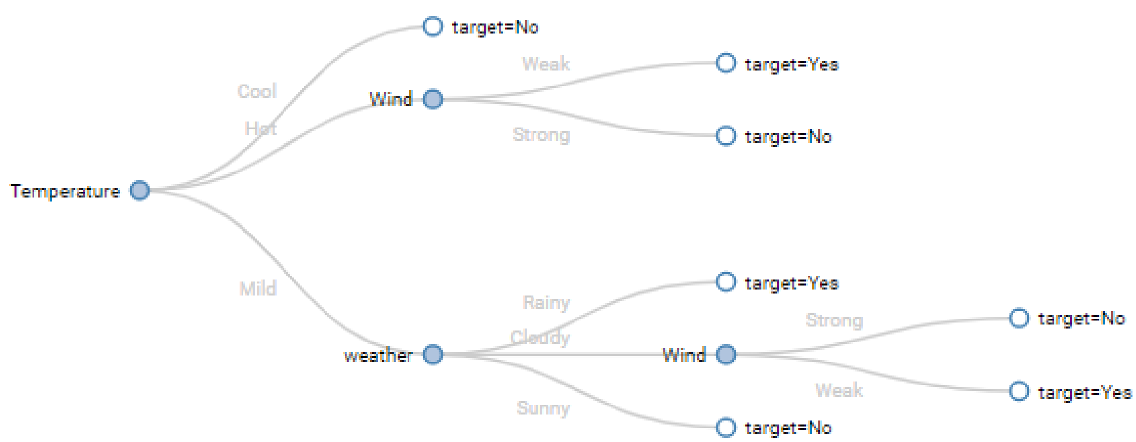
P(Strong and Hiking) $= \frac{-1}{1}\log_2\frac{1}{1} = 0$

P(Weak and Hiking) $= \frac{-1}{1}\log_2\frac{1}{1} = 0$

Gain (Hiking, Wind(F4)) = $1 - \frac{1}{2}*0 - \frac{1}{2}*0 = 1$

From the results the maximum value of Information Gain is “Wind” so it will be chosen as the child node for “Cloudy” branch.

The final decision tree based on the Information gain split:



c-

Advantages of Gini index Vs. Information gain	
Gini index	**Information gain**
1- Used by CART algorithms	1- Used in ID3, C4.5 algorithms
2- computes the degree of probability of a specific variable that is wrongly being classified when chosen randomly and a variation of Gini Coefficient.	2- computes the difference between entropy before and after split and specifies the impurity in class elements.
3- can handle the values that are non-negative because it is measured by subtracting the sum of squared probabilities of each class from one.	3- measures the entropy differences before and after splitting and depicts the impurity in class variables.
4- Facilitates larger distributions that are very easy to implement.	4- Supports smaller distributions with smaller numbers and more specific values
Disadvantages of Gini index Vs. Information gain	
Gini index	**Information gain**
1- prone to systematic and random data errors. Therefore, inaccurate data can distort the validity of the coefficient.	1- supports smaller partitions (distribution) with a variety of different values.
2- operates on the categorical target variables in terms of "success" or "failure" and performs only binary split.	2- can't handle the values that are non-positive.

Part 2: programming

Import Important Libraries and load the dataset

```
import pandas as pd #for working with dataframes
import numpy as np #to deal with arrays
import matplotlib.pyplot as plt #for plotting
import seaborn as sns
%matplotlib inline
```

Load the training dataset:

```
columns = ['pixel ' + str(i) for i in range(17)]
columns[-1] = 'label'
#load the train dataset
train = pd.read_csv('/content/pendigits-tra.csv',names=columns)
train.head()
```

	pixel 0	pixel 1	pixel 2	pixel 3	pixel 4	pixel 5	pixel 6	pixel 7	pixel 8	pixel 9	pixel 10	pixel 11	pixel 12	pixel 13	pixel 14	pixel 15	label
0	47	100	27	81	57	37	26	0	0	23	56	53	100	90	40	98	8
1	0	89	27	100	42	75	29	45	15	15	37	0	69	2	100	6	2
2	0	57	31	68	72	90	100	100	76	75	50	51	28	25	16	0	1
3	0	100	7	92	5	68	19	45	86	34	100	45	74	23	67	0	4
4	0	67	49	83	100	100	81	80	60	60	40	40	33	20	47	0	1

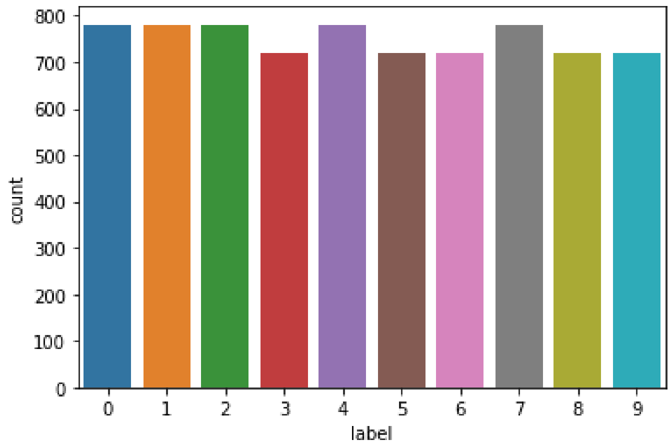
Get some information about the training dataset:

```
#get some information about the train dataset
train.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 7494 entries, 0 to 7493
Data columns (total 17 columns):
 #   Column    Non-Null Count  Dtype
---  ------    --------------  -----
 0   pixel 0   7494 non-null   int64
 1   pixel 1   7494 non-null   int64
 2   pixel 2   7494 non-null   int64
 3   pixel 3   7494 non-null   int64
 4   pixel 4   7494 non-null   int64
 5   pixel 5   7494 non-null   int64
 6   pixel 6   7494 non-null   int64
 7   pixel 7   7494 non-null   int64
 8   pixel 8   7494 non-null   int64
 9   pixel 9   7494 non-null   int64
 10  pixel 10  7494 non-null   int64
 11  pixel 11  7494 non-null   int64
 12  pixel 12  7494 non-null   int64
 13  pixel 13  7494 non-null   int64
 14  pixel 14  7494 non-null   int64
 15  pixel 15  7494 non-null   int64
 16  label     7494 non-null   int64
dtypes: int64(17)
memory usage: 995.4 KB
```

Count the numbers occurrence in the label column:

```python
#count the variables in label column
sns.countplot(train['label'])
```



Split data into x_train and y_train:

```python
#split the data into x_train and y_train
x_train = train.drop('label',axis= 1)
y_train = train['label']
```

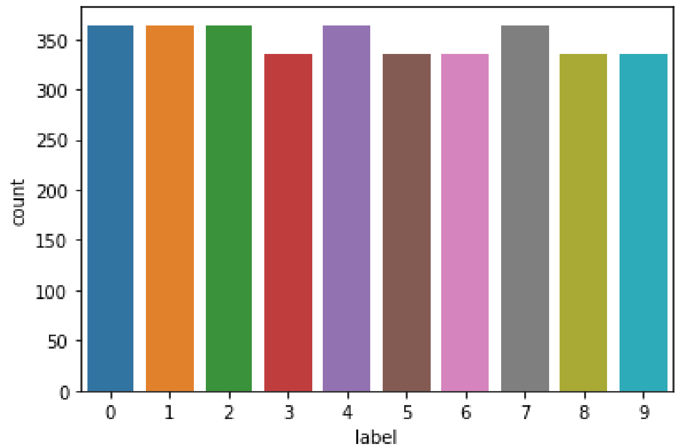
Load the test dataset:

```python
[6] columns = ['pixel ' + str(i) for i in range(17)]
    columns[-1] = 'label'
    #load the test dataset
    test = pd.read_csv('/content/pendigits-tes.csv',names=columns)
    test.head()
```

	pixel 0	pixel 1	pixel 2	pixel 3	pixel 4	pixel 5	pixel 6	pixel 7	pixel 8	pixel 9	pixel 10	pixel 11	pixel 12	pixel 13	pixel 14	pixel 15	label
0	88	92	2	99	16	66	94	37	70	0	0	24	42	65	100	100	8
1	80	100	18	98	60	66	100	29	42	0	0	23	42	61	56	98	8
2	0	94	9	57	20	19	7	0	20	36	70	68	100	100	18	92	8
3	95	82	71	100	27	77	77	73	100	80	93	42	56	13	0	0	9
4	68	100	6	88	47	75	87	82	85	56	100	29	75	6	0	0	9

Count the numbers occurrence in label column:

```python
#count the numbers occurrence in label column
sns.countplot(test['label'])
```



Split the test data into x_test and y_test:

```
[8] #split the test dataset into x_test and y_test
    x_test = test.drop('label',axis= 1)
    y_test = test['label']
```

Train the Decision Tree classification Model:

Decision Tree is a Supervised learning technique that can be used for both classification and Regression problems, but mostly it is preferred for solving Classification problems. It is a tree-structured classifier, where internal nodes represent the features of a dataset, branches represent the decision rules and each leaf node represents the outcome.

```
[10] dt = DecisionTreeClassifier(random_state=0)
     dt.fit(x_train,y_train)
     y_pred = dt.predict(x_test)
```

Accuracy of Decision Tree model: `Accuracy  0.9208118925100057`

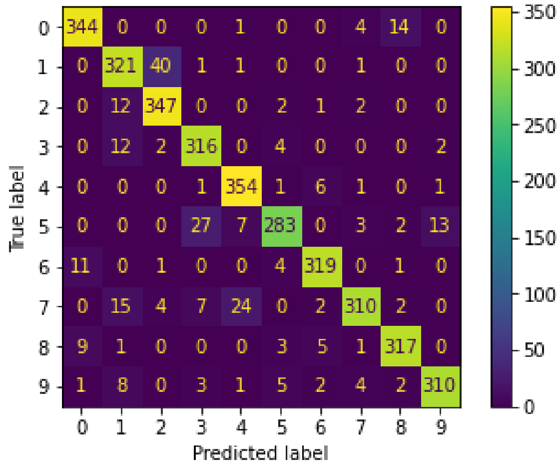
Classification report of Decision Tree model:

```
              precision    recall  f1-score   support

           0       0.94      0.95      0.95       363
           1       0.87      0.88      0.88       364
           2       0.88      0.95      0.92       364
           3       0.89      0.94      0.91       336
           4       0.91      0.97      0.94       364
           5       0.94      0.84      0.89       335
           6       0.95      0.95      0.95       336
           7       0.95      0.85      0.90       364
           8       0.94      0.94      0.94       336
           9       0.95      0.92      0.94       336

    accuracy                           0.92      3498
   macro avg       0.92      0.92      0.92      3498
weighted avg       0.92      0.92      0.92      3498
```

Confusion matrix of Decision Tree model:



Train SVM model:

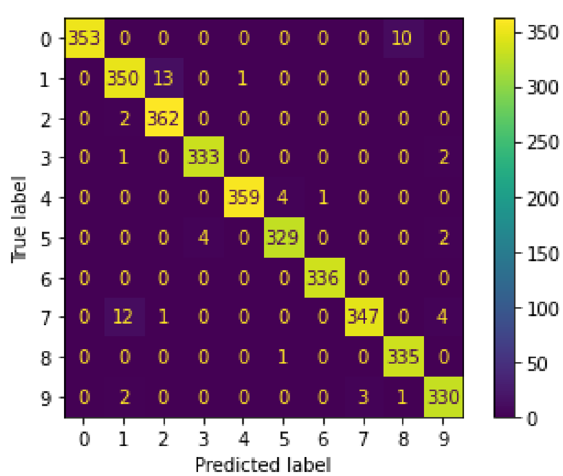
SVM offers very high accuracy compared to other classifiers such as logistic regression, and decision trees. It is known for its kernel trick to handle nonlinear input spaces. It is used in a variety of applications such as face detection, intrusion detection, classification of emails, news articles and web pages, classification of genes, and handwriting recognition.

```
from sklearn.svm import SVC
#train SVM model
svm = SVC(random_state=0)
svm.fit(x_train,y_train)
y_pred = svm.predict(x_test)
```

Accuracy score of SVM model: Accuracy 0.9817038307604345

Classification report and Confusion matrix of SVM model:

	precision	recall	f1-score	support
0	1.00	0.97	0.99	363
1	0.95	0.96	0.96	364
2	0.96	0.99	0.98	364
3	0.99	0.99	0.99	336
4	1.00	0.99	0.99	364
5	0.99	0.98	0.98	335
6	1.00	1.00	1.00	336
7	0.99	0.95	0.97	364
8	0.97	1.00	0.98	336
9	0.98	0.98	0.98	336
accuracy			0.98	3498
macro avg	0.98	0.98	0.98	3498
weighted avg	0.98	0.98	0.98	3498



Apply Hard and soft Voting Classifiers:

Hard voting entails picking the prediction with the highest number of votes.

soft voting entails combining the probabilities of each prediction in each model and picking the prediction with the highest total probability.

```
from sklearn.ensemble import VotingClassifier
#train the hard voting classifier
hard_votting = VotingClassifier(estimators=[('dt', dt), ('svm', svm)], voting='hard')
hard_votting.fit(x_train, y_train)
y_pred = hard_votting.predict(x_test)
```

```
[22] #train the soft voting classifier
svm = SVC(probability=True)
soft_votting = VotingClassifier(estimators=[('dt', dt), ('svm', svm)], voting='soft')
soft_votting.fit(x_train, y_train)
y_pred = soft_votting.predict(x_test)
```

Accuracy of hard voting	Accuracy of soft voting
0.9405374499714122	0.9208118925100057

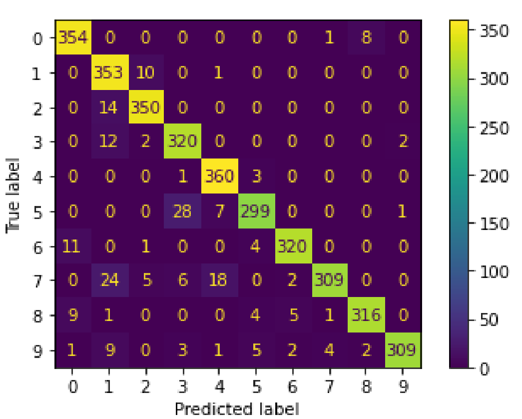
Classification report of hard voting

	precision	recall	f1-score	support
0	0.94	0.98	0.96	363
1	0.85	0.97	0.91	364
2	0.95	0.96	0.96	364
3	0.89	0.95	0.92	336
4	0.93	0.99	0.96	364
5	0.95	0.89	0.92	335
6	0.97	0.95	0.96	336
7	0.98	0.85	0.91	364
8	0.97	0.94	0.95	336
9	0.99	0.92	0.95	336
accuracy			0.94	3498
macro avg	0.94	0.94	0.94	3498
weighted avg	0.94	0.94	0.94	3498

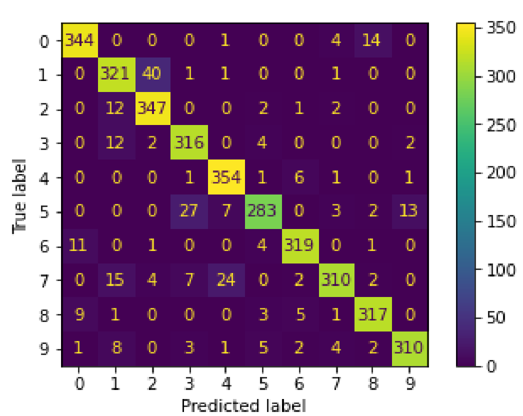
Classification report of soft voting

	precision	recall	f1-score	support
0	0.94	0.95	0.95	363
1	0.87	0.88	0.88	364
2	0.88	0.95	0.92	364
3	0.89	0.94	0.91	336
4	0.91	0.97	0.94	364
5	0.94	0.84	0.89	335
6	0.95	0.95	0.95	336
7	0.95	0.85	0.90	364
8	0.94	0.94	0.94	336
9	0.95	0.92	0.94	336
accuracy			0.92	3498
macro avg	0.92	0.92	0.92	3498
weighted avg	0.92	0.92	0.92	3498

Confusion matrix of hard voting



Confusion matrix of soft voting



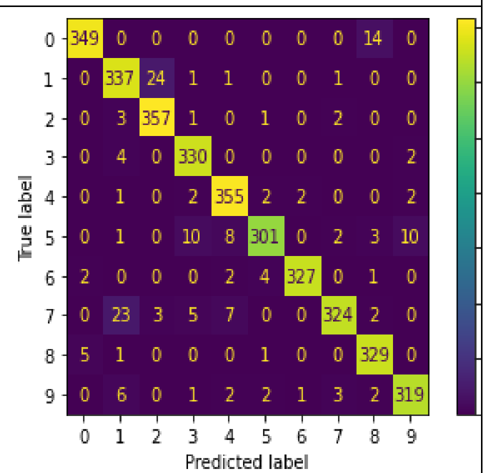
Apply Bagging Classifier in range[10:200]:

A Bagging classifier is an ensemble meta-estimator that fits base classifiers each on random subsets of the original dataset and then aggregate their individual predictions (either by voting or by averaging) to form a final prediction.

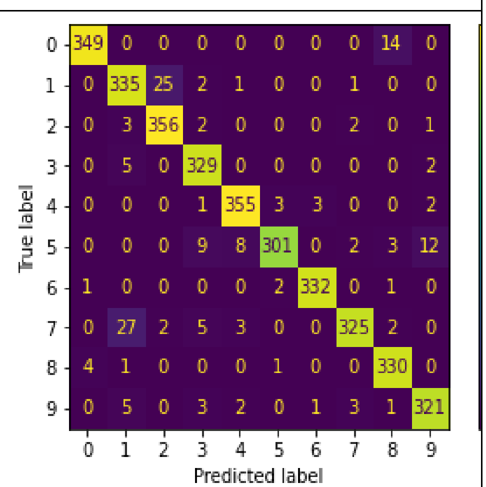
```
from sklearn.ensemble import BaggingClassifier
scores =[]
for i in [30,70,100,150,200]:
  ba = BaggingClassifier(n_estimators=i,random_state=0)
  ba.fit(x_train,y_train)
  y_pred = ba.predict(x_test)
  print('Accuracy ', accuracy_score(y_test,y_pred))
  print(classification_report(y_test,y_pred))
  plot_confusion_matrix(ba,x_test,y_test)
  scores.append(accuracy_score(y_test,y_pred))
```

<table>
<tr><th>Accuracies</th><th>Classification reports</th><th>Confusion matrices</th></tr>
<tr><td>0.9514008004574043</td><td>precision recall f1-score support

0 0.98 0.96 0.97 363
1 0.90 0.93 0.91 364
2 0.93 0.98 0.95 364
3 0.94 0.98 0.96 336
4 0.95 0.98 0.96 364
5 0.97 0.90 0.93 335
6 0.99 0.97 0.98 336
7 0.98 0.89 0.93 364
8 0.94 0.98 0.96 336
9 0.96 0.95 0.95 336

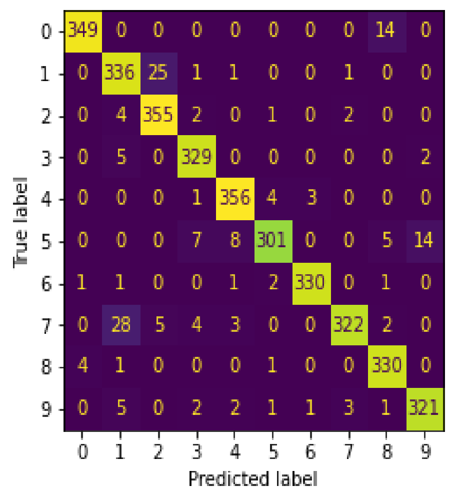
accuracy 0.95 3498
macro avg 0.95 0.95 0.95 3498
weighted avg 0.95 0.95 0.95 3498</td><td></td></tr>
<tr><td>0.9528301886792453</td><td>precision recall f1-score support

0 0.99 0.96 0.97 363
1 0.89 0.92 0.91 364
2 0.93 0.98 0.95 364
3 0.94 0.98 0.96 336
4 0.96 0.98 0.97 364
5 0.98 0.90 0.94 335
6 0.99 0.99 0.99 336
7 0.98 0.89 0.93 364
8 0.94 0.98 0.96 336
9 0.95 0.96 0.95 336

accuracy 0.95 3498
macro avg 0.95 0.95 0.95 3498
weighted avg 0.95 0.95 0.95 3498</td><td></td></tr>
</table>

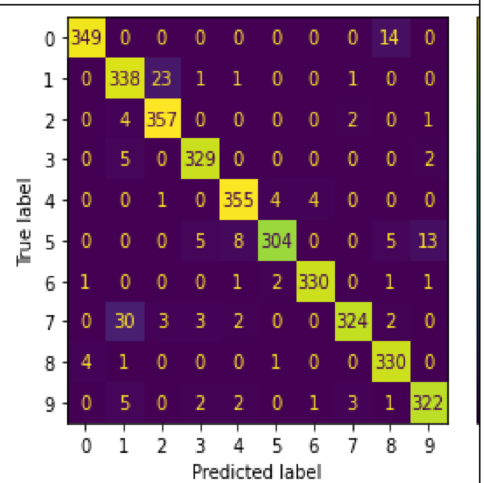
0.9516866781017724

	precision	recall	f1-score	support
0	0.99	0.96	0.97	363
1	0.88	0.92	0.90	364
2	0.92	0.98	0.95	364
3	0.95	0.98	0.96	336
4	0.96	0.98	0.97	364
5	0.97	0.90	0.93	335
6	0.99	0.98	0.99	336
7	0.98	0.88	0.93	364
8	0.93	0.98	0.96	336
9	0.95	0.96	0.95	336
accuracy			0.95	3498
macro avg	0.95	0.95	0.95	3498
weighted avg	0.95	0.95	0.95	3498



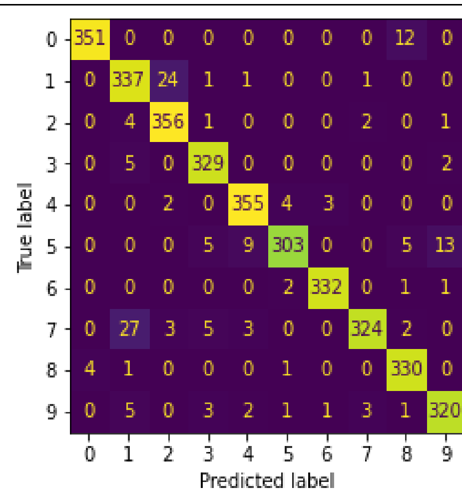
0.9542595769010863

	precision	recall	f1-score	support
0	0.99	0.96	0.97	363
1	0.88	0.93	0.90	364
2	0.93	0.98	0.95	364
3	0.97	0.98	0.97	336
4	0.96	0.98	0.97	364
5	0.98	0.91	0.94	335
6	0.99	0.98	0.98	336
7	0.98	0.89	0.93	364
8	0.93	0.98	0.96	336
9	0.95	0.96	0.95	336
accuracy			0.95	3498
macro avg	0.96	0.95	0.95	3498
weighted avg	0.96	0.95	0.95	3498

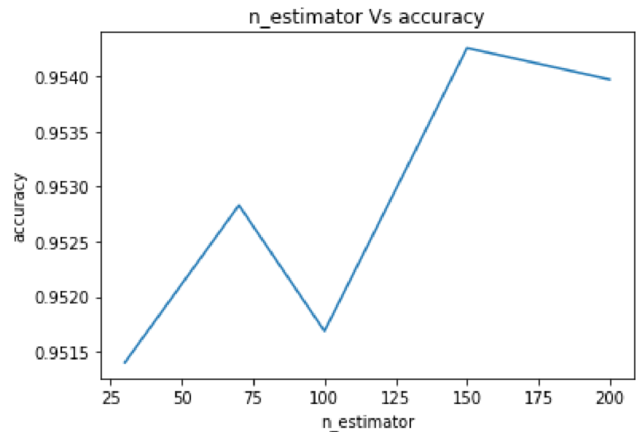


0.9539736992567182

	precision	recall	f1-score
0	0.99	0.97	0.98
1	0.89	0.93	0.91
2	0.92	0.98	0.95
3	0.96	0.98	0.97
4	0.96	0.98	0.97
5	0.97	0.90	0.94
6	0.99	0.99	0.99
7	0.98	0.89	0.93
8	0.94	0.98	0.96
9	0.95	0.95	0.95
accuracy			0.95
macro avg	0.96	0.95	0.95
weighted avg	0.95	0.95	0.95



Plot the numbers of estimators vs. accuracy score:



So, the highest accuracy at n = 150

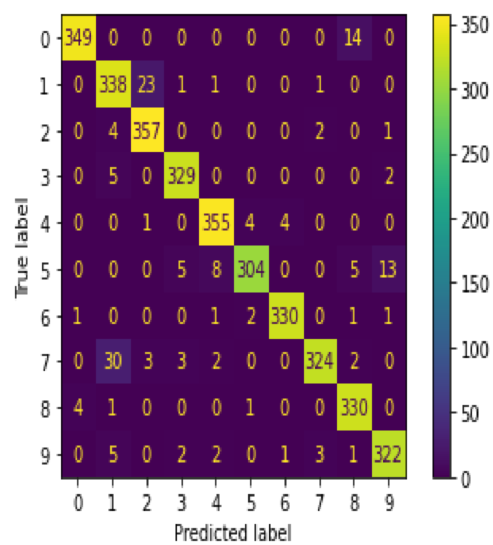
Train bagging classifier with the best number of estimators:

```
ba = BaggingClassifier(n_estimators=150,random_state=0)
ba.fit(x_train,y_train)
y_pred = ba.predict(x_test)
```

Accuracy is: 0.9542595769010863

Classification report and confusion of bagging with estimator number = 150:

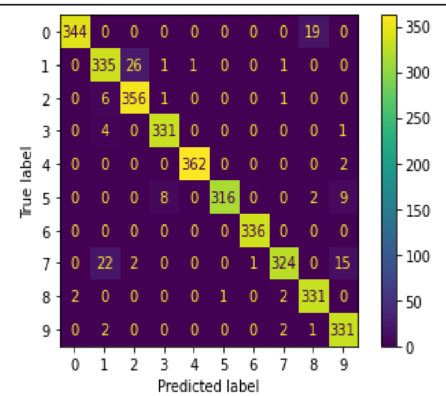
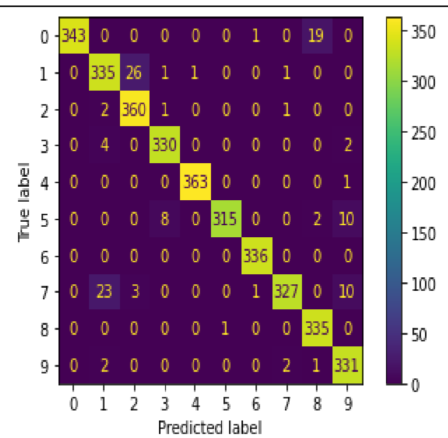
	precision	recall	f1-score	support
0	0.99	0.96	0.97	363
1	0.88	0.93	0.90	364
2	0.93	0.98	0.95	364
3	0.97	0.98	0.97	336
4	0.96	0.98	0.97	364
5	0.98	0.91	0.94	335
6	0.99	0.98	0.98	336
7	0.98	0.89	0.93	364
8	0.93	0.98	0.96	336
9	0.95	0.96	0.95	336
accuracy			0.95	3498
macro avg	0.96	0.95	0.95	3498
weighted avg	0.96	0.95	0.95	3498



Apply Random Forest classifier: (additional model code)

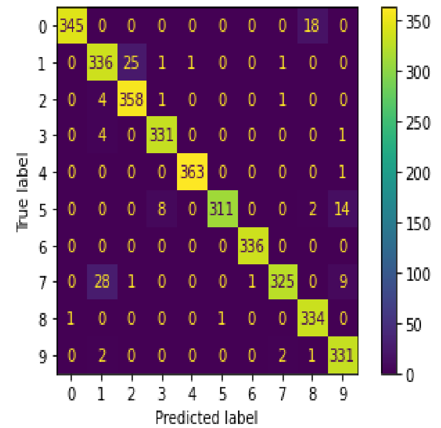
A random forest is a meta estimator that fits a number of decision tree classifiers on various sub-samples of the dataset and uses averaging to improve the predictive accuracy and control over-fitting.

```
[35] from sklearn.ensemble import RandomForestClassifier
     scores =[]
     for i in [30,70,100,150,200]:
       rf = RandomForestClassifier(n_estimators=i,random_state=0)
       rf.fit(x_train,y_train)
       y_pred = rf.predict(x_test)
       #accuracy score for random forest
       print('Accuracy ', accuracy_score(y_test,y_pred))
        #classification report for random forest
       print(classification_report(y_test,y_pred))
        #confusion matrix for random forest
       plot_confusion_matrix(rf,x_test,y_test)
       scores.append(accuracy_score(y_test,y_pred))
```

<table>
<tr><th>Accuracies</th><th>Classification reports</th><th>Confusion matrices</th></tr>
<tr><td>0.9622641509433962</td><td>
<table>
<tr><th></th><th>precision</th><th>recall</th><th>f1-score</th><th>support</th></tr>
<tr><td>0</td><td>0.99</td><td>0.95</td><td>0.97</td><td>363</td></tr>
<tr><td>1</td><td>0.91</td><td>0.92</td><td>0.91</td><td>364</td></tr>
<tr><td>2</td><td>0.93</td><td>0.98</td><td>0.95</td><td>364</td></tr>
<tr><td>3</td><td>0.97</td><td>0.99</td><td>0.98</td><td>336</td></tr>
<tr><td>4</td><td>1.00</td><td>0.99</td><td>1.00</td><td>364</td></tr>
<tr><td>5</td><td>1.00</td><td>0.94</td><td>0.97</td><td>335</td></tr>
<tr><td>6</td><td>1.00</td><td>1.00</td><td>1.00</td><td>336</td></tr>
<tr><td>7</td><td>0.98</td><td>0.89</td><td>0.93</td><td>364</td></tr>
<tr><td>8</td><td>0.94</td><td>0.99</td><td>0.96</td><td>336</td></tr>
<tr><td>9</td><td>0.92</td><td>0.99</td><td>0.95</td><td>336</td></tr>
<tr><td>accuracy</td><td></td><td></td><td>0.96</td><td>3498</td></tr>
<tr><td>macro avg</td><td>0.96</td><td>0.96</td><td>0.96</td><td>3498</td></tr>
<tr><td>weighted avg</td><td>0.96</td><td>0.96</td><td>0.96</td><td>3498</td></tr>
</table>
</td><td>
</td></tr>
<tr><td>0.9648370497427101</td><td>
<table>
<tr><th></th><th>precision</th><th>recall</th><th>f1-score</th><th>support</th></tr>
<tr><td>0</td><td>1.00</td><td>0.94</td><td>0.97</td><td>363</td></tr>
<tr><td>1</td><td>0.92</td><td>0.92</td><td>0.92</td><td>364</td></tr>
<tr><td>2</td><td>0.93</td><td>0.99</td><td>0.96</td><td>364</td></tr>
<tr><td>3</td><td>0.97</td><td>0.98</td><td>0.98</td><td>336</td></tr>
<tr><td>4</td><td>1.00</td><td>1.00</td><td>1.00</td><td>364</td></tr>
<tr><td>5</td><td>1.00</td><td>0.94</td><td>0.97</td><td>335</td></tr>
<tr><td>6</td><td>0.99</td><td>1.00</td><td>1.00</td><td>336</td></tr>
<tr><td>7</td><td>0.99</td><td>0.90</td><td>0.94</td><td>364</td></tr>
<tr><td>8</td><td>0.94</td><td>1.00</td><td>0.97</td><td>336</td></tr>
<tr><td>9</td><td>0.94</td><td>0.99</td><td>0.96</td><td>336</td></tr>
<tr><td>accuracy</td><td></td><td></td><td>0.96</td><td>3498</td></tr>
<tr><td>macro avg</td><td>0.97</td><td>0.97</td><td>0.97</td><td>3498</td></tr>
<tr><td>weighted avg</td><td>0.97</td><td>0.96</td><td>0.96</td><td>3498</td></tr>
</table>
</td><td>
</td></tr>
</table>

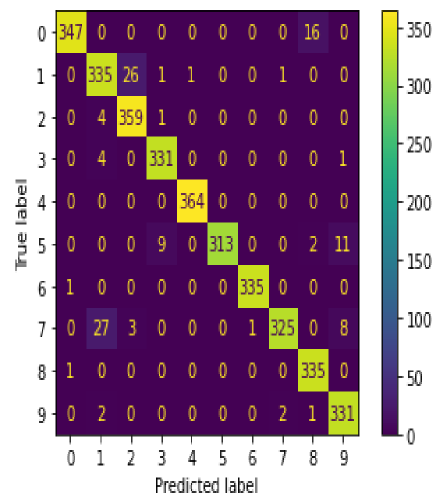
0.9634076615208691

	precision	recall	f1-score	support
0	1.00	0.95	0.97	363
1	0.90	0.92	0.91	364
2	0.93	0.98	0.96	364
3	0.97	0.99	0.98	336
4	1.00	1.00	1.00	364
5	1.00	0.93	0.96	335
6	1.00	1.00	1.00	336
7	0.99	0.89	0.94	364
8	0.94	0.99	0.97	336
9	0.93	0.99	0.96	336
accuracy			0.96	3498
macro avg	0.96	0.96	0.96	3498
weighted avg	0.96	0.96	0.96	3498



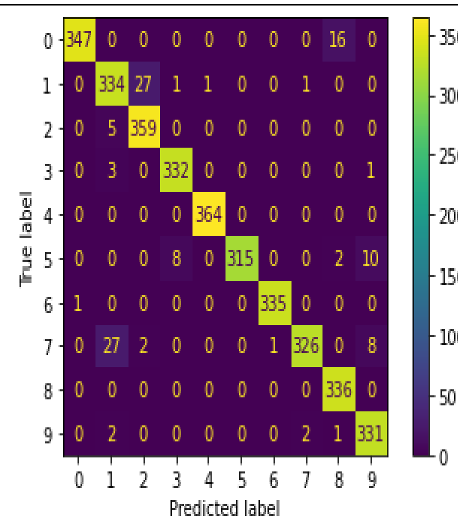
0.9648370497427101

	precision	recall	f1-score	support
0	0.99	0.96	0.97	363
1	0.90	0.92	0.91	364
2	0.93	0.99	0.95	364
3	0.97	0.99	0.98	336
4	1.00	1.00	1.00	364
5	1.00	0.93	0.97	335
6	1.00	1.00	1.00	336
7	0.99	0.89	0.94	364
8	0.95	1.00	0.97	336
9	0.94	0.99	0.96	336
accuracy			0.96	3498
macro avg	0.97	0.97	0.97	3498
weighted avg	0.97	0.96	0.96	3498

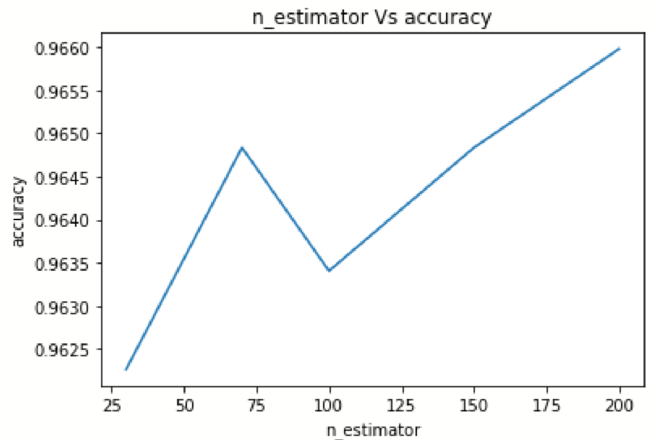


0.965980560320183

	precision	recall	f1-score	support
0	1.00	0.96	0.98	363
1	0.90	0.92	0.91	364
2	0.93	0.99	0.95	364
3	0.97	0.99	0.98	336
4	1.00	1.00	1.00	364
5	1.00	0.94	0.97	335
6	1.00	1.00	1.00	336
7	0.99	0.90	0.94	364
8	0.95	1.00	0.97	336
9	0.95	0.99	0.97	336
accuracy			0.97	3498
macro avg	0.97	0.97	0.97	3498
weighted avg	0.97	0.97	0.97	3498



plot numbers of estimators Vs. accuracy scores:



So, the highest accuracy is at n = 200

Train random forest with estimator = 200:

```
[36] rf = RandomForestClassifier(n_estimators=200,random_state=0)
     rf.fit(x_train,y_train)
     y_pred = rf.predict(x_test)
     print('Accuracy ', accuracy_score(y_test,y_pred))
     print(classification_report(y_test,y_pred))
     plot_confusion_matrix(rf,x_test,y_test)
```

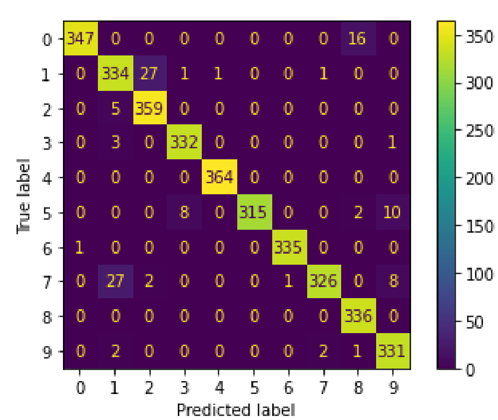
Accuracy score: 0.965980560320183

Classification report and confusion matrix:

```
              precision    recall  f1-score   support

           0       1.00      0.96      0.98       363
           1       0.90      0.92      0.91       364
           2       0.93      0.99      0.95       364
           3       0.97      0.99      0.98       336
           4       1.00      1.00      1.00       364
           5       1.00      0.94      0.97       335
           6       1.00      1.00      1.00       336
           7       0.99      0.90      0.94       364
           8       0.95      1.00      0.97       336
           9       0.95      0.99      0.97       336

    accuracy                           0.97      3498
   macro avg       0.97      0.97      0.97      3498
weighted avg       0.97      0.97      0.97      3498
```



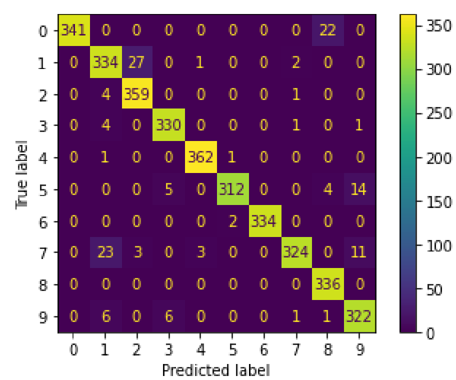
Train Boosting Classifier in range[10:200]:

Boosting is an ensemble technique that attempts to create a strong classifier from a number of weak classifiers.

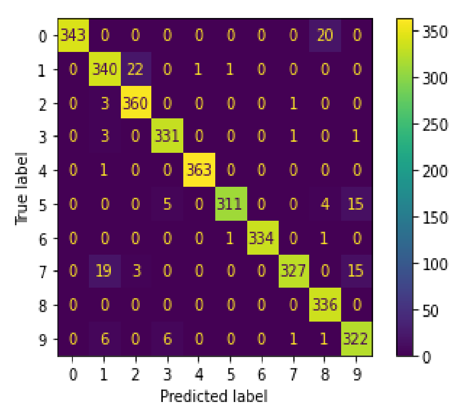
```
from sklearn.ensemble import GradientBoostingClassifier
scores_1 =[]
for i in [70,100,150,200]:
  gb = GradientBoostingClassifier(n_estimators=i,random_state=0)
  gb.fit(x_train,y_train)
  y_pred = gb.predict(x_test)
  print('Accuracy ', accuracy_score(y_test,y_pred))
  print(classification_report(y_test,y_pred))
  plot_confusion_matrix(gb,x_test,y_test)
  scores_1.append(accuracy_score(y_test,y_pred))
```

<table>
<tr><th>Accuracies</th><th>Classification reports</th><th>Confusion matrices</th></tr>
<tr><td>0.9588336192109777</td><td>precision recall f1-score support

0 1.00 0.94 0.97 363
1 0.90 0.92 0.91 364
2 0.92 0.99 0.95 364
3 0.97 0.98 0.97 336
4 0.99 0.99 0.99 364
5 0.99 0.93 0.96 335
6 1.00 0.99 1.00 336
7 0.98 0.89 0.94 364
8 0.93 1.00 0.96 336
9 0.93 0.96 0.94 336

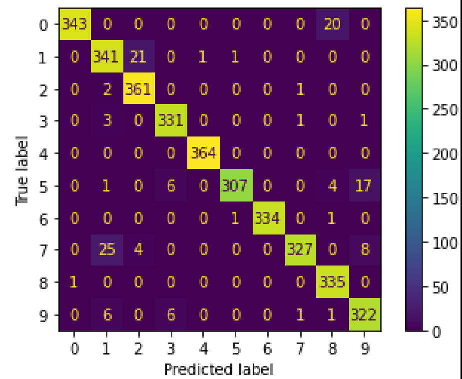
accuracy 0.96 3498
macro avg 0.96 0.96 0.96 3498
weighted avg 0.96 0.96 0.96 3498</td><td>
</td></tr>
<tr><td>0.9625500285877644</td><td>precision recall f1-score support

0 1.00 0.94 0.97 363
1 0.91 0.93 0.92 364
2 0.94 0.99 0.96 364
3 0.97 0.99 0.98 336
4 1.00 1.00 1.00 364
5 0.99 0.93 0.96 335
6 1.00 0.99 1.00 336
7 0.99 0.90 0.94 364
8 0.93 1.00 0.96 336
9 0.91 0.96 0.93 336

accuracy 0.96 3498
macro avg 0.96 0.96 0.96 3498
weighted avg 0.96 0.96 0.96 3498</td><td>
</td></tr>
</table>

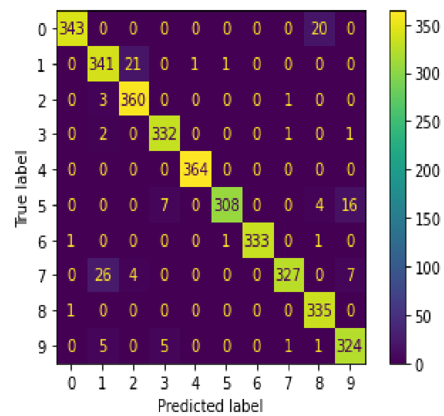
0.961978273299028

	precision	recall	f1-score	support
0	1.00	0.94	0.97	363
1	0.90	0.94	0.92	364
2	0.94	0.99	0.96	364
3	0.97	0.99	0.97	336
4	1.00	1.00	1.00	364
5	0.99	0.92	0.95	335
6	1.00	0.99	1.00	336
7	0.99	0.90	0.94	364
8	0.93	1.00	0.96	336
9	0.93	0.96	0.94	336
accuracy			0.96	3498
macro avg	0.96	0.96	0.96	3498
weighted avg	0.96	0.96	0.96	3498

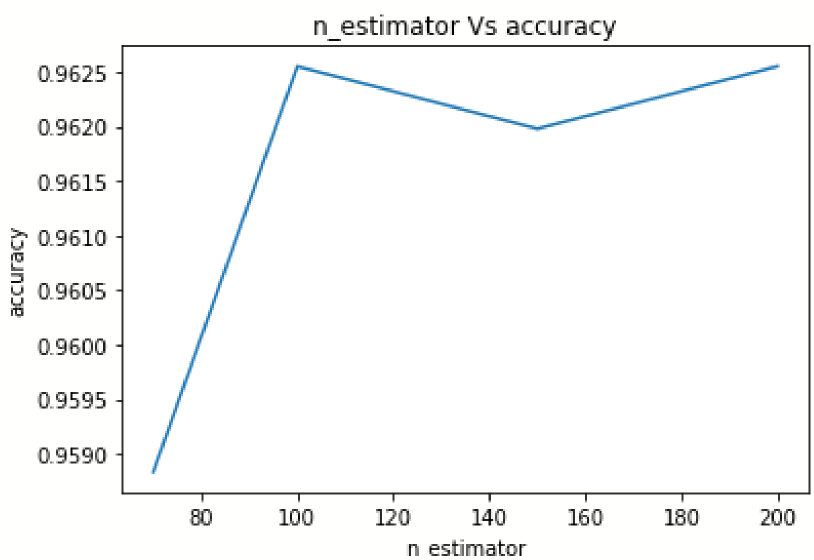


0.9625500285877644

	precision	recall	f1-score	support
0	0.99	0.94	0.97	363
1	0.90	0.94	0.92	364
2	0.94	0.99	0.96	364
3	0.97	0.99	0.98	336
4	1.00	1.00	1.00	364
5	0.99	0.92	0.96	335
6	1.00	0.99	1.00	336
7	0.99	0.90	0.94	364
8	0.93	1.00	0.96	336
9	0.93	0.96	0.95	336
accuracy			0.96	3498
macro avg	0.96	0.96	0.96	3498
weighted avg	0.96	0.96	0.96	3498



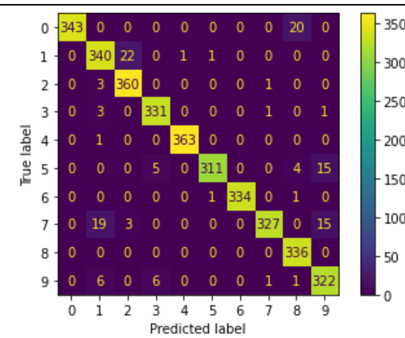
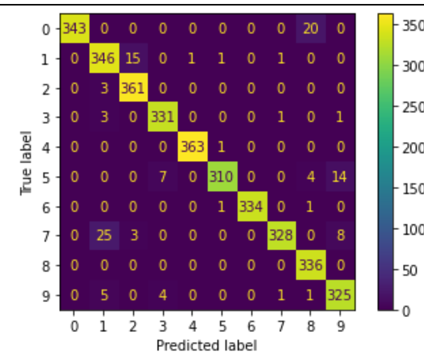
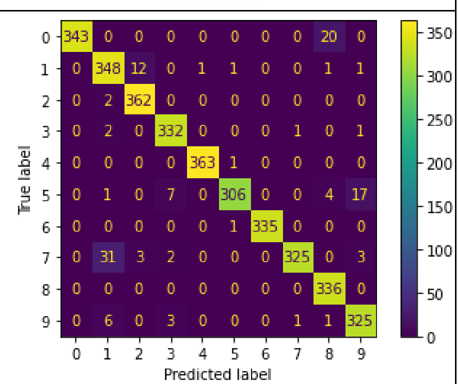
plot numbers of estimators Vs. accuracy scores:



So, the highest accuracy score is at n estimator = 100

Train the boosting classifier at n estimator = 100 and with range [0.1:0.9]:

```
scores_2 =[]
for i in [0.1,0.3,0.5,0.7]:
  gb = GradientBoostingClassifier(n_estimators=100,learning_rate=i,random_state=0)
  gb.fit(x_train,y_train)
  y_pred = gb.predict(x_test)
  print('Accuracy ', accuracy_score(y_test,y_pred))
  print(classification_report(y_test,y_pred))
  plot_confusion_matrix(gb,x_test,y_test)
  scores_2.append(accuracy_score(y_test,y_pred))
```

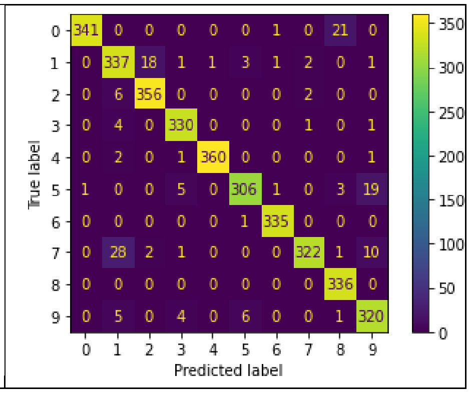
Accuracies	Classification reports	Confusion matrices
0.9625500285877644	precision recall f1-score support 0 1.00 0.94 0.97 363 1 0.91 0.93 0.92 364 2 0.94 0.99 0.96 364 3 0.97 0.99 0.98 336 4 1.00 1.00 1.00 364 5 0.99 0.93 0.96 335 6 1.00 0.99 1.00 336 7 0.99 0.90 0.94 364 8 0.93 1.00 0.96 336 9 0.91 0.96 0.93 336 accuracy 0.96 3498 macro avg 0.96 0.96 0.96 3498 weighted avg 0.96 0.96 0.96 3498	
0.9654088050314465	precision recall f1-score support 0 1.00 0.94 0.97 363 1 0.91 0.95 0.93 364 2 0.95 0.99 0.97 364 3 0.97 0.99 0.98 336 4 1.00 1.00 1.00 364 5 0.99 0.93 0.96 335 6 1.00 0.99 1.00 336 7 0.99 0.90 0.94 364 8 0.93 1.00 0.96 336 9 0.93 0.97 0.95 336 accuracy 0.97 3498 macro avg 0.97 0.97 0.97 3498 weighted avg 0.97 0.97 0.97 3498	
0.9648370497427101	precision recall f1-score support 0 1.00 0.94 0.97 363 1 0.89 0.96 0.92 364 2 0.96 0.99 0.98 364 3 0.97 0.99 0.98 336 4 1.00 1.00 1.00 364 5 0.99 0.91 0.95 335 6 1.00 1.00 1.00 336 7 0.99 0.89 0.94 364 8 0.93 1.00 0.96 336 9 0.94 0.97 0.95 336 accuracy 0.96 3498 macro avg 0.97 0.97 0.96 3498 weighted avg 0.97 0.96 0.96 3498	

0.9556889651229273

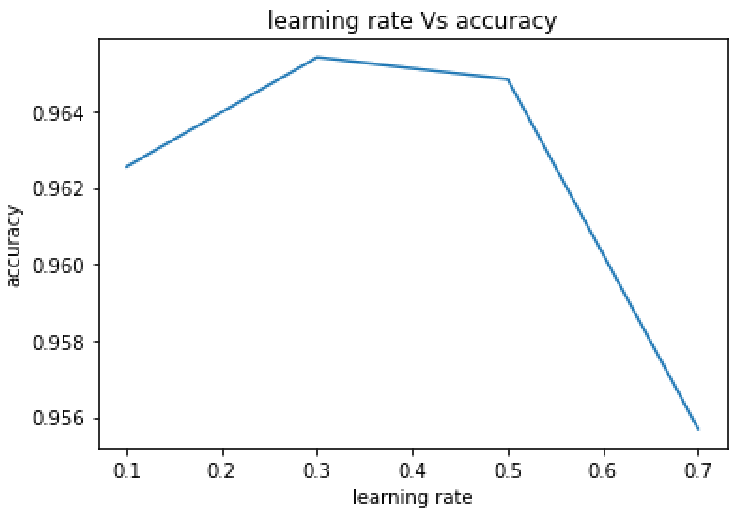
```
              precision    recall  f1-score   support

           0       1.00      0.94      0.97       363
           1       0.88      0.93      0.90       364
           2       0.95      0.98      0.96       364
           3       0.96      0.98      0.97       336
           4       1.00      0.99      0.99       364
           5       0.97      0.91      0.94       335
           6       0.99      1.00      0.99       336
           7       0.98      0.88      0.93       364
           8       0.93      1.00      0.96       336
           9       0.91      0.95      0.93       336

    accuracy                           0.96      3498
   macro avg       0.96      0.96      0.96      3498
weighted avg       0.96      0.96      0.96      3498
```



Plot the learning rate Vs. accuracy scores:



So, the best accuracy is when number of estimator = 100 and with learning rate = 0.3

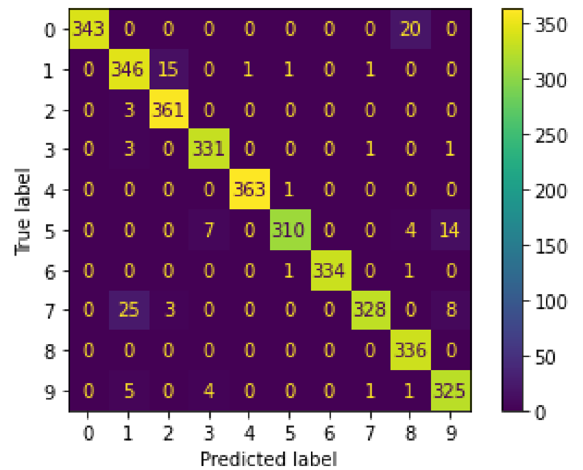
Train Boosting classifier with n_estimators=100, learning_rate=0.3:

```
gb = GradientBoostingClassifier(n_estimators=100,learning_rate=0.3,random_state=0)
gb.fit(x_train,y_train)
y_pred = gb.predict(x_test)
print('Accuracy ', accuracy_score(y_test,y_pred))
print(classification_report(y_test,y_pred))
plot_confusion_matrix(gb,x_test,y_test)
```

Accuracy score: 0.9654088050314465

Classification report and confusion matrix:

	precision	recall	f1-score	support
0	1.00	0.94	0.97	363
1	0.91	0.95	0.93	364
2	0.95	0.99	0.97	364
3	0.97	0.99	0.98	336
4	1.00	1.00	1.00	364
5	0.99	0.93	0.96	335
6	1.00	0.99	1.00	336
7	0.99	0.90	0.94	364
8	0.93	1.00	0.96	336
9	0.93	0.97	0.95	336
accuracy			0.97	3498
macro avg	0.97	0.97	0.97	3498
weighted avg	0.97	0.97	0.97	3498



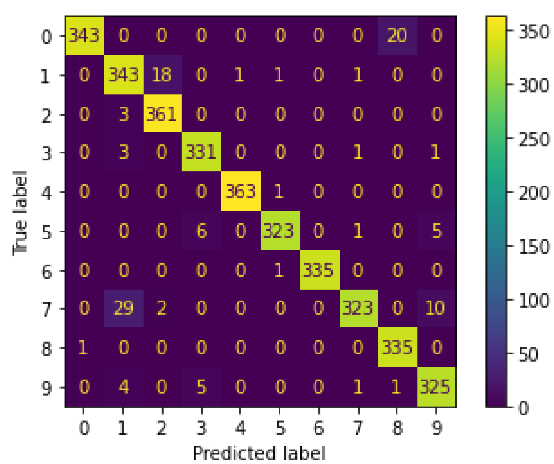
Apply XGBOOST classification model with n_estimators=100, learning_rate=0.3:

XGBoost is an optimized distributed gradient boosting library designed to be highly efficient, flexible and portable.

Accuracy score: `0.9668381932532876`

Classification report and confusion matrix:

	precision	recall	f1-score	support
0	1.00	0.94	0.97	363
1	0.90	0.94	0.92	364
2	0.95	0.99	0.97	364
3	0.97	0.99	0.98	336
4	1.00	1.00	1.00	364
5	0.99	0.96	0.98	335
6	1.00	1.00	1.00	336
7	0.99	0.89	0.93	364
8	0.94	1.00	0.97	336
9	0.95	0.97	0.96	336
accuracy			0.97	3498
macro avg	0.97	0.97	0.97	3498
weighted avg	0.97	0.97	0.97	3498



Which metric is the best to compare performance, accuracy or the confusion matrix?

Accuracy performance metrics can be critical when dealing with imbalanced data and overfitting.
The confusion matrix, recall, precision, and F1 score gives good obviousness of prediction results comparing with the accuracy.
The F1-score can tell us if there is overfitting or not.

After seeing the F1-score, we notice that there is no overfitting in the data, so the accuracy is good here than the confusion matrix.

Conclusion:

During this assignment we learned how to apply different classification techniques, such as decision tree classifier, and apply bagging and boosting with the best estimator and learning rate.

After comparing the models with each other, we noticed that the **decision tree** model is the **lowest accuracy**, which is **0.92**, and the **SVM** model is the **highest accuracy**, which is **0.98**

After applying soft voting and hard voting, the **hard voting is the best**, which have **accuracy = 0.94,** and the **soft voting is the worst**, which have **accuracy = 0.92.**

After comparing bagging and boosting, the **best accuracy is boosting classifier**, which have **accuracy = 0.965,**

When comparing the **XGBOOST model(**with the **best number of estimator and hyperparameter** like the boosting classifier) with **Gradient Boosting classifier,** the **accuracy of XGBOOST is the higher which ism = 0.9668**

References:

https://scikit-learn.org/stable/modules/generated/sklearn.tree.DecisionTreeClassifier.html

https://xgboost.readthedocs.io/en/stable/python/python_api.html

https://stackabuse.com/gradient-boosting-classifiers-in-python-with-scikit-learn/#:~:text=The%20idea%20behind%20%22gradient%20boosting,Approximately%20Correct%20Learning%20(PAC).

https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.BaggingClassifier.html

Google Colab link for the code assignment:
https://colab.research.google.com/drive/1CgJFBf0r9Hc9js783LEYCEZ8dj0iTfpW?usp=sharing