CONTENTS

Exercise No.	Date	EXPERIMENTS
1.		Implementation of Find-S algorithm
2.		Implementing candidate elimination algorithm
3.		Ensemble method Bagging
4.		Ensemble method Boosting
5.		Ensemble method stacking

1. Study of Machine Learning libraries in python.

* The libraries used in python for machine learning are

* Numpy

Numpy short for Numerical python is a fundamental package for numerical computing in python. It provides support for large multi-dimensional arrays and matrices, along with a collection of mathematical functions to operate on these arrays efficiently. Numpy is widely used in scientific computing, data analysis, and machine learning due to it's speed and versatility.

* Pandas

pandas is a powerful open-source data analysis and manipulation library built on top of Numpy. It provides easy to use data structures and functions for working with structured data, such as time series tabular data and heterogeneous data. pandas is widely used for data cleaning, transformation, exploration, and analysis in data science and machine learning projects

* Scikit-learn :

Scikit-learn, often abbreviated as sklearn, is a popular python library for machine learning tasks. It provides a wide array of tools for building and applying various machine learning algorithms.

Some of the key libraries and modules within Scikit-learn.

* dataset : This module includes utilities for loading datasets, both standard datasets included with Scikit-learn and functions to load external datasets. It provides easy access to sample datasets for practice and experimentation

* Preprocessing : This module contains functions for preprocessing data before feeding it into Machine learning models. It includes tools for scaling centering, normalizing, binarizing and imputing missing values in datasets.

* Feature_extraction : This module is for feature extraction from raw data. It includes methods such as Bag of words, TF-IDF, and feature hashing, which are commonly used in natural language processing (NLP) tasks.

* Feature_selection : This module provides utilities for selecting relevant features or reducing the dimensionality of the dataset. Techniques include univariate feature selection, recursive feature elimination and feature importances.

* model_selection : This module include tools for model selection and evaluation. It provides

functions for splitting datasets into training and test sets cross validation, hyperparameter tuning using grid search and performance metrics evaluation

* metric : This module contains various metrics for evaluating the performance of machine learning models. It includes metrics for classification, regression clustering, and ranking tasks, such as accuracy precision, recall, f1-score, mean squared error and ROC AOC

* classification : This module includes algorithms and tools for classification tasks. It provides implementations for popular classifiers such as Support Vector Machine (SVM), Random Forest, k-Nearest Neighbors (k-NN), and Naive Bayes.

* Regression : Similar to the classification module, this contains algorithms and tools for regression, Lasso regression, and support Vector regression, among others.

* clustering : This module provides algorithms and tools for clustering data into groups based on similarity. It includes implementations of k-means, hierarchical clustering, DBSCAN, and spectral clustering algorithms.

* decomposition: This module includes various techniques for reducing the dimensionality of data while preserving its structure and important features. Examples include principal component analysis (PCA), truncated singular value decomposition (TSVD), and manifold learning techniques.

* ensemble: This module includes ensemble methods for combining multiple base estimators to improve predictive performance. It provides implementations of techniques such as bagging, boosting and random forests.

* neighbors: This module provides functionality for working with nearest neighbors algorithms, including k-nearest neighbors (kNN) for classification and regression tasks.

The sklearn.compose module in scikit-learn provides tools for combining and transforming datasets. It includes classes and functions that allow you to create composite estimators and pipelines, perform feature unions and apply column transformations.

* Few classes or functions within the scikit-learn library:

1. Simple Imputer: This is a class within the sklearn.impute module. It is used for imputing missing

values in datasets. The SimpleImputer class provides several strategies for imputation, such as replacing missing values with the mean, median, most frequent value or a constant value.

2. LabelEncoder :- This is a class within the sklearn.preprocessing module. It is used for encoding categorical labels as numerical values. The LabelEncoder class assigns a unique integer to each unique category in the categorical variable.

3. OneHotEncoder :- Also a class within the sklearn.preprocessing module. It is used for converting categorical variables into one-hot encoded format. Onehot encoding is a process where categorical variables are converted into binary vectors, with each binary vector representing a unique category. This is useful for algorithms that requires numerical input

4. StandardScaler :- This is another class within the sklearn.preprocessing module. It is used for standardizing features by removing the mean and scaling to unit variance. Standardization is important for many machine learning algorithms as it helps in bringing features onto the same scale, which can prevent some features from dominating others in the model training process

5. Column Transformer : key component of the sklearn. compose. This class allows you to apply different transformations to different columns of a dataset. It's particularly useful when dealing with datasets containing a mix of numerical and categorical features. You can specify the transformations to be applied to each column using a list of tuples, where each tuple consists of a transformer and a list of column indices or names.

* Seaborn is a python data visualization library based on matplotlib. It provides a high-level interface for drawing attractive and informative statistical graphics. Seaborn is particularly useful for visualizing statistical relationships and patterns in datasets.

Seaborn will generate a grid of plots showing the pairwise relationships between the variables in the dataset. The diagonal plots will display histograms of kernal density estimates of each variables distribution, while the off-diagonal plots will show scatterplots of the relationships between pairs of variables.

There are commonly used libraries for machine learning workflows. They provide convenient functions and methods to perform various tasks in machine learning.

2. Data exploration and pre-processing in machine learning.

```
import matplotlib.pyplot as plt
import numpy as np
import pandas as pd
df = pd.read_csv('data.csv')
df.head()

df.tail()

df.info()

df.describe()
```

Output :

	country	age	salary	purchased
0	France	NaN	1200	NO
1	Spain	27.0	4800	YES
2	Germany	30.0	5400	YES
3	UK	49.0	98000	NO

	country	age	salary	purchased
0	France	NaN	1200	NO
1	Spain	27.0	4800	YES
2	Germany	30.0	5400	YES
3	UK	49.0	98000	NO

#	column	Non-Null Count	Dtype
0	country	4 non-null	object
1	age	3 non-null	float64
2	salary	4 non-null	int64
3	purchased	4 non-null	object

	age	salary
count	3.000000	4.000000
mean	35.333333	28330.000000
std	11.930353	46111.212441
min	27.000000	1200.000000
25%	28.500000	5250.000000
50%	30.000000	8100.000000
75%	39.500000	29400.000000
max	49.000000	98000.000000

```
Country     0
age         1
Salary      0
purchased   0
dtype: int64
```

```
country     0
age         0
salary      0
purchased   0
dtype: int64
```

```
array [['France', 35.333333336, 72000.0],
       ['Spain', 27.0, 48000.0],
       ['Germany', 30.0, 54000.0],
       ['UK', 49.0, 98000.0]] dtype = object
```

```
array ([['NO'],
        ['yes'],
        ['yes']
        ['NO']], dtype = object
```

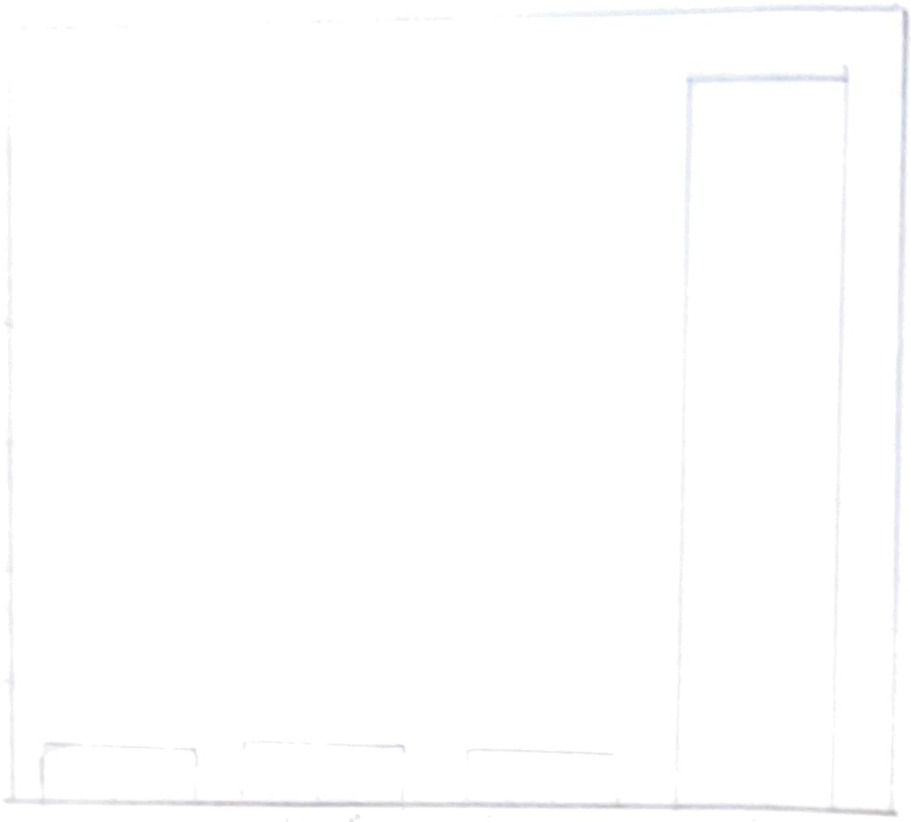
```
df.isnull().sum()
```

```
df['age'].fillna(df['age'].mean(), inplace=True)
df['salary'].fillna(df['salary'].mean(), inplace=True)
df.isnull().sum()
```

```
from sklearn.impute import SimpleImputer
x = df.iloc[:, :-1].values
y = df.iloc[:, 3].values
imp = SimpleImputer(missing_values=np.nan, strategy
x[:, 1:3] = imp.fit_transform(x[:, 1:3])
x
```

```
y
```

```
from sklearn.preprocessing import LabelEncoder
le = LabelEncoder()
h = le.fit_transform(x[:, 0])
y = le.fit_transform(y)
```



Output : LogisticRegression(max_iter=10000)

3. Evalute the classifier using various performance measures.

```
From sklearn.datasets import load_iris
From sklearn.model_selection import train_test_split
from sklearn.linear_model import logistic Regression
from sklearn.metrics import accuracy_score, precision
    _score, F1_score, roc_auc, classification_report,
    confusion_matrix, recall_score


iris = load_iris()
X = iris.data
y = iris.target


X_train, X_test, y_train, y_test = train_test_split(X,y,
                                    test_size=0.2, random_state=42)

log_reg = Logistic Regression (max_item = 1000)

log_reg.fit(X_train, y_train)

y_pred = log_reg.predict(X_test)

accuracy = round(accuracy score(y_test, y_pred)*100,2)

print("Accuracy:", accuracy)
```

* Accuracy : 100.0

* Classification Report

	precision	recall	f1-score	support
0	1.00	1.00	1.00	10
1	1.00	1.00	1.00	9
2	1.00	1.00	1.00	11
accuracy			1.00	30
macro avg	1.00	1.00	1.00	30
weighted avg	1.00	1.00	1.00	30

* precision : 1.0

* Recall : 1.0

* f1 score : 1.0

* Roc Auc score : 1.0

* Confusion matrix:

```
[[10  0  0]
 [ 0  9  0]
 [ 0  0 11]]
```

```
result = classification_report(y_test, y_pred)
print("classification Report:",)
print(result)

precision = precision_score(y_test, y_pred, average =
                                        'weighted')
print("precision:", precision)

recall = recall_score(y_test, y_pred, average = 'weighted')
print("Recall:", recall)

f1 = f1_score(y_test, y_pred, average = 'weighted')
  print("F1 score:", f1)

roc_auc = roc_auc_score(y_test, log_reg.predict_
      proba(x_test), multi_class = 'ovr')
print("ROC AUC score:", roc_auc)

conf_matrix = confusion_matrix(y_test, y_pred)
print("Confusion matrix:\n", conf_matrix)
```

Output : Confusion Matrix

[[1 0 0]
 [0 1 0]
 [0 0 1]]

4. Implement k-nearest neighbors classification using python

```
import pandas as pd
import numpy as np
from sklearn.datasets import load_iris

iris = load_iris()
x = iris.data
y = iris.target

from sklearn.model_selection import train_test_split
x_train, x_test, y_train, y_test = train_test_split
                    (x, y, test_size=0.2, random_state=0)

from sklearn.neighbors import KNeighborsClassifier
lr = KNeighborsClassifier(n_neighbors=5)
lr.fit(x_train, y_train)
y_pred = lr.predict(x_test)

from sklearn.metrics import classification_report, confusion
                    matrix, accuracy_score
result = confusion_matrix(y_test, y_pred)
print("Confusion Matrix:")
print(result)
```

Classification Report

	precision	recall	f1-score	support
0	1.00	1.00	1.00	11
1	1.00	0.92	0.96	13
2	0.26	1.00	0.92	6
accuracy			0.97	30
macro avg	0.95	0.97	0.96	30
weighted avg	0.97	0.97	0.97	30

Accuracy: 96.67

```
result = classification_report(y_test, y_pred)
print("Classification Report:")
print(result)
```

```
result2 = round(accuracy_score(y_test, y_pred)*100,2)
print("Accuracy:", result2)
```

```
result 1 = classification_report(y_test, y_pred)
print("Classification Report:",)
print(result 1)
```

```
result 2 = round(accuracy_score(y_test, y_pred) * 100, 2)
print("Accuracy:", result 2)
```

output:

```
[ [5.1  3.5  1.4  0.2]
 .  .  .  .
 .  .  .  .
 [4.8  3.4  1.6  0.2]]
```

(150, 2)

output:

```
[ [5.1  3.5  1.4  0.2]
 .  .  .  .
 .  .  .  .
 [4.8  3.4  1.6  0.2]]
```

(150, 2)

5. Dimensionality reduction using PCA

```
import numpy as np
import pandas as pd
from sklearn.datasets import load_iris
from sklearn.preprocessing import StandardScaler
from sklearn.decomposition import PCA
import matplotlib.pyplot as plt

iris = load_iris()
x = iris.data
y = iris.target

print(x)

target_names = iris.target_names

scaler = StandardScaler()
x_scaled = scaler.fit_transform(x)
pca = PCA(n_components=2)
x_pca = pca.fit_transform(x_scaled)

print(x_pca.shape)
```

```
plt.scatter(x_pca[:,0], x_pca[:,1], x=y, cmap='
        plasma')
plt.xlabel('First principal component')
plt.ylabel('second principal component')
plt.title('PCA of IRIS dataset')
plt.show()
```

[illegible] - * [illegible]

* [illegible]

* [illegible]

* [illegible]

Output :- * Linear Regression.

* 0.3517090923059894

* 0.9533282488134365

* 0.13682917742580005

6. Regression Analysis using Linear Regression

```
import numpy as np
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_squared_error,
                            mean_absolute_error, r2_score
from sklearn.datasets import load_iris

iris = load_iris()
X = iris.data
y = iris.target

X_train, X_test, y_train, y_test = train_test_split
    (X, y, train_size = 0.7, test_size = 0.3, random_state = 100)

from sklearn.linear_model import LinearRegression
lr = LinearRegression()

lr.fit(X_train, y_train)

y_pred = lr.predict(X_test)

mse = mean_squared_error(y_test, y_pred)
print(mse)

r2 = r2_score(y_test, y_pred)
print(r2)

mae = mean_absolute_error(y_test, y_pred)
print(mae)
```

Output :- 100.0

Classification Report

	precision	recall	F1-score	support
0	1.00	1.00	1.00	11
1	1.00	1.00	1.00	13
2	1.00	1.00	1.00	6
accuracy			1.00	30
macro avg	1.00	1.00	1.00	30
weighted avg	1.00	1.00	1.00	30

7. Classification using Logistic Regression

```
import pandas as pd
import numpy as np
from sklearn.datasets import load_iris

iris = load_iris()
x = iris.data
y = iris.target

from sklearn.model_selection import train_test_split
x_train, x_test, y_train, y_test = train_test_split
    (x, y, test_size = 0.2, random_state 0)
from sklearn.linear_model import logistic Regression
from sklearn.metrics import classification_report,
        confusion_matrix, accuracy score

logreg = LogisticRegression (max_iter = 1000)
logreg.fit (x_train, y_train)
y_pred = logreg.predict (x_test)
acc_logreg = round (accuracy_score (y_pred, y_test)*
        100, 2)
print (acc_logreg)

result 1 = classification_report (y_test, y_pred)
print ("Classification Report:",)
print (result 1)
```

Confusion matrix:

```
[[11  0  0]
 [ 0 13  0]
 [ 0  0  6]]
```

Confusion matrix:

```
[[11  0  0]
 [ 0 13  0]
 [ 0  0  6]]
```

```
result = confusion_matrix(y_test, y_pred)
print("Confusion Matrix:")
print(result)
```

output :- DecisionTreeClassifier
DecisionTreeClassifier()

* array(['setosa', 'versicolor', 'virginica'], dtype='<U10')

8. Classification using Decision Tree

```
import pandas as pd
from sklearn.tree import DecisionTreeClassifier, plot_tree
from sklearn.model_selection import train_test_split
from sklearn import metrics

from sklearn.datasets import load_iris
iris = load_iris()
x = iris.data
y = iris.target

dtc = DecisionTreeClassifier()

x_train, x_test, y_train, y_test = train_test_split
    (x, y, test_size=0.3, random_state=1)
dtc.fit(x_train, y_train)

iris.target_names
```

```
import matplotlib.pyplot as plt
plt.figure(figsize=(12,8))
plot_tree(dtc, feature_names=iris.feature_names,
    class_names=list(iris.target_name).filled=True)
plt.show()
```

Output : * SVC
SVC ()

Confusion matrix
[[16 0 0]
[0 11 0]
[0 1 17]]

9. Classification using Support Vector machine

```
import numpy as np
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.datasets import load_iris

iris = load_iris()
x = iris.data
y = iris.target

from sklearn import svm
lr = svm.SVC()
lr.fit(x_train, y_train)

y_pred = lr.predict(x_test)
from sklearn.metrics import classification_report,
        confusion_matrix, accuracy_score
result = confusion_matrix(y_test, y_pred)
print("Confusion matrix:")
print(result)
```

Classification report

	precision	recall	f1-score	support
0	1.00	1.00	1.00	16
1	0.92	1.00	0.96	11
2	1.00	0.94	0.97	18
accuracy			0.98	45
macro avg	0.97	0.98	0.98	45
weight avg	0.98	0.98	0.98	45

Allwacy : 97.9733333

DATE

result 1 = classification_report(y_test, y_pred)
print("Classification Report:",)
print(result1)

result2 = accuracy_score(y_test, y_pred)*100.2
print("Accuracy:", result2)

* output :- GaussianNB
GaussianNB()

Accuracy : 96.67

10. Classification Techniques - Naive Bayes

```
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.naive_bayes import GaussianNB
from sklearn.metrics import accuracy_score
from sklearn.datasets import load_iris

iris = load_iris()
x = iris.data
y = iris.target

from sklearn.model_selection import train_test_split
x_train, x_test, y_train, y_test = train_test_split
        (x, y, test_size=0.2, random_state=0)

from sklearn.naive_bayes import GaussianNB
gnb = GaussianNB()
gnb.fit(x_train, y_train)

y_pred = gnb.predict(x_test)

from sklearn.metrics import classification_report, confusion
    _matrix, accuracy_score
result2 = round(accuracy_score(y_test, y_pred)*100)
print("Accuracy:", result2)
```

Classification Report

	precision	recall	f1 score	support
0	1.00	1.00	1.00	11
1	0.93	1.00	0.96	13
2	1.00	0.83	0.91	6
accuracy			0.97	30
macro avg	0.98	0.94	0.96	30
weighted avg	0.97	0.97	0.97	30

Classification Report

	precision	recall	f1-score	support
0	1.00	1.00	1.00	11
1	0.93	1.00	0.96	13
2	1.00	0.83	0.91	6
accuracy			0.97	30
macro avg	0.98	0.94	0.96	30
weighted avg	0.97	0.97	0.97	30

```
result1 = classification_report(y_test, y_pred)
print("Classification Report:",)
print(result1)
```

Output Shape of X : (500, 2)
        Shape of y : (500,)

* KMeans
  KMeans(n_clusters=4, random_state=42)

array([1, 2, 0, 3, 2 - - - - - - - -
- - - - - - - - 1, 3, 2, 2, 0])

11. Clustering using k-means clustering

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.datasets import make_blobs
from sklearn.cluster import kmeans

X, Y = make_blobs(n_samples = 500, centers = 4,
    cluster_std = 8, random_state = 42)

print("shape of X: ", X.shape)

print("shape of Y :", Y.shape)

K-means = kmeans(n_clusters = 4, random_state
        = 42)
k means . fit(X)

labels = kmeans.labels
labels
```

```
plt.figure(figsize=(8,6))
plt.scatter(x[:,0], x[:,1], c=labels, cmap='viridis')
plt.scatter(kmeans.cluster_centers_[:,0], kmeans.
    cluster_centers_[:,1], s=100, c="red", label
                                ='centroids')
plt.title('K means clustering')
plt.xlabel('x')
plt.ylabel('y')
plt.show()
```

output : Random Forest classifier

Random Forest classifier (criterion = 'entropy',
n_estimators = 10)

Confusion matrix

```
[[19  0  0]
 [ 0 13  0]
 [ 0  0 13]]
```

12. Ensemble Method Bagging

```
import pandas as pd
import numpy as np
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score

iris = load_iris()
x = iris.data
y = iris.target

x_train, x_test, y_train, y_test = train_test_split
    (x, y, test_size=0.3, random_state=42)

from sklearn.ensemble import RandomForestClassifier
dt = RandomForestClassifier(n_estimators=10,
                            criterion="entropy")

dt.fit(x_train, y_train)

y_pred = dt.predict(x_test)

from sklearn.metrics import classification_report,
    confusion_matrix, accuracy_score
result1 = confusion_matrix(y_test, y_pred)
print("Confusion matrix:")
print(result1)
```

Classification Report

	precision	recall	f1-score	support
0	1.00	1.00	1.00	19
1	1.00	1.00	1.00	13
2	1.00	1.00	1.00	13
accuracy			1.00	45
macro avg	1.00	1.00	1.00	45
weighted avg	1.00	1.00	1.00	45

Accuracy = 1.0

result 2 = classification_report(y_test, y_pred)
print("Classification Report:")
print(result 2)

result 3 = accuracy_score(y_test, y_pred)
print("Accuracy:", result 3)

Output : * Grandient Boosting classifier
Grandient Boosting classifier( )

* confusion matrix

```
[[ 19  0  0]
 [  0 13  0]
 [  0  0 13]]
```

* classification Report

	precision	recall	f1-score	support
0	1.00	1.00	1.00	19
1	1.00	1.00	1.00	13
2	1.00	1.00	1.00	13
accuracy			1.00	45
macro avg	1.00	1.00	1.00	45
weighted avg	1.00	1.00	1.00	45

13 Ensemble method - Boosting

```
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score, classification
    _report, confusion_matrix

iris = load_iris()
x = iris.data
y = iris.target
x_train, x_test, y_train, y_test = train_test_split
    (x, y, test_size = 0.3, random_state=42)

from sklearn.ensemble import GradientBoosting
    Classifier
gb = GradientBoostingClassifier()
gb.fit(x_train, y_train)

y_pred = gb.predict(x_test)

result = confusion_matrix(y_test, y_pred)
print("Confusion matrix:")
print(result)

result1 = Classification_report(y_test, y_pred)
print("Classification Report:")
print(result1)
```

14. Ensemble method - stacking

```
from sklearn.datasets import load_iris
from sklearn.ensemble import StackingClassifier
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.svm import SVC
from sklearn.metrics import accuracy_score,
    classification_report, confusion_matrix

iris = load_iris()
X = iris.data
y = iris.target
X_train, X_test, y_train, y_test = train_test_split
    (X, y, test_size = 0.3, random_state = 42)

base_learners = [('decision_tree', DecisionTree
                  classifier(max_depth = 1)),
                 ('lr', LogisticRegression())]

meta_learner = SVC(probability = True, random_
    state = 42)
stack_clf = StackingClassifier(estimators = base_
    learners, final_estimator = meta_learner)
stack_clf.fit(X_train, y_train)
```

```
y_pred = stack_clf.predict(x_test)

print ["stacking model accuracy:", accuracy_score
    (y_test, y_pred)]

result = confusion_matrix(y_test, y_pred)
print("confusion matrix:")
print(result)
```

Output:

Stacking model accuracy : 1.0

Confusion matrix:

```
[[19  0  0]
 [ 0 13  0]
 [ 0  0 13]]
```

Classification Report

	precision	recall	f1-score	support
0	1.00	1.00	1.00	19
1	1.00	1.00	1.00	13
2	1.00	1.00	1.00	13
accuracy			1.00	45
macro avg	1.00	1.00	1.00	45
weighted avg	1.00	1.00	1.00	45

```
y_pred = stack_clf.predict(x_test)

print ["stacking model accuracy:", accuracy_score
    (y_test, y_pred)]

result = confusion_matrix(y_test, y_pred)
print ("confusion matrix:")
print (result)
```