Python for Neuroscientists

Lecture 5: Supervised Learning

Elom Amematsro, 05/20/2021

Outline:

Introduction to ML

Regression

Classification

Group Exercises

Outline:

Introduction to ML

Regression

Classification

Group Exercises

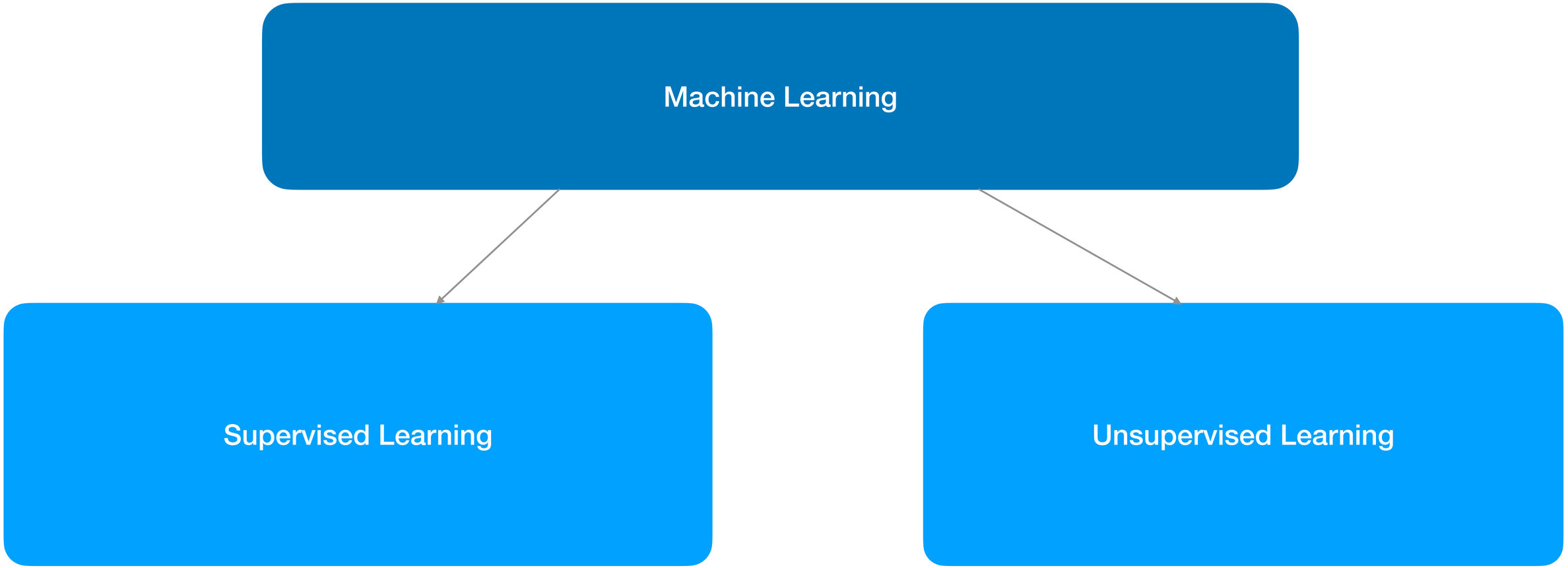
Introduction to ML:

Machine Learning

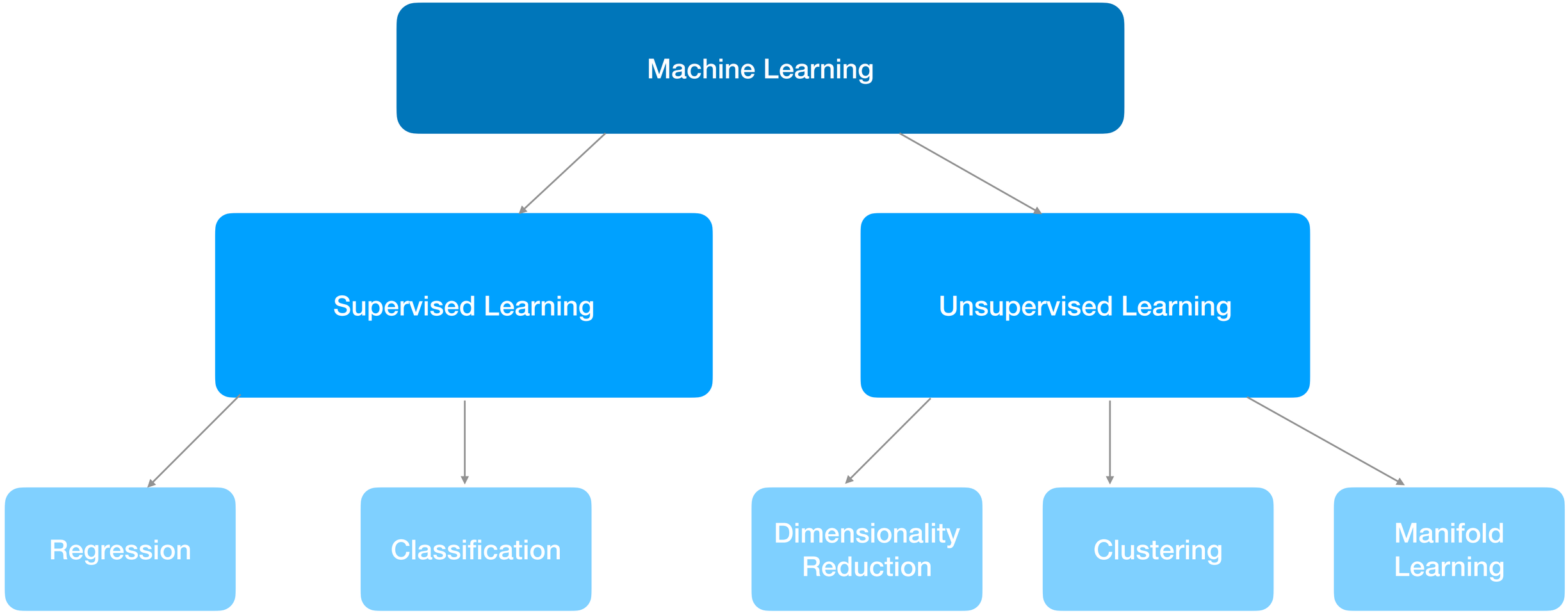
Introduction to SKLEARN

Model Selection

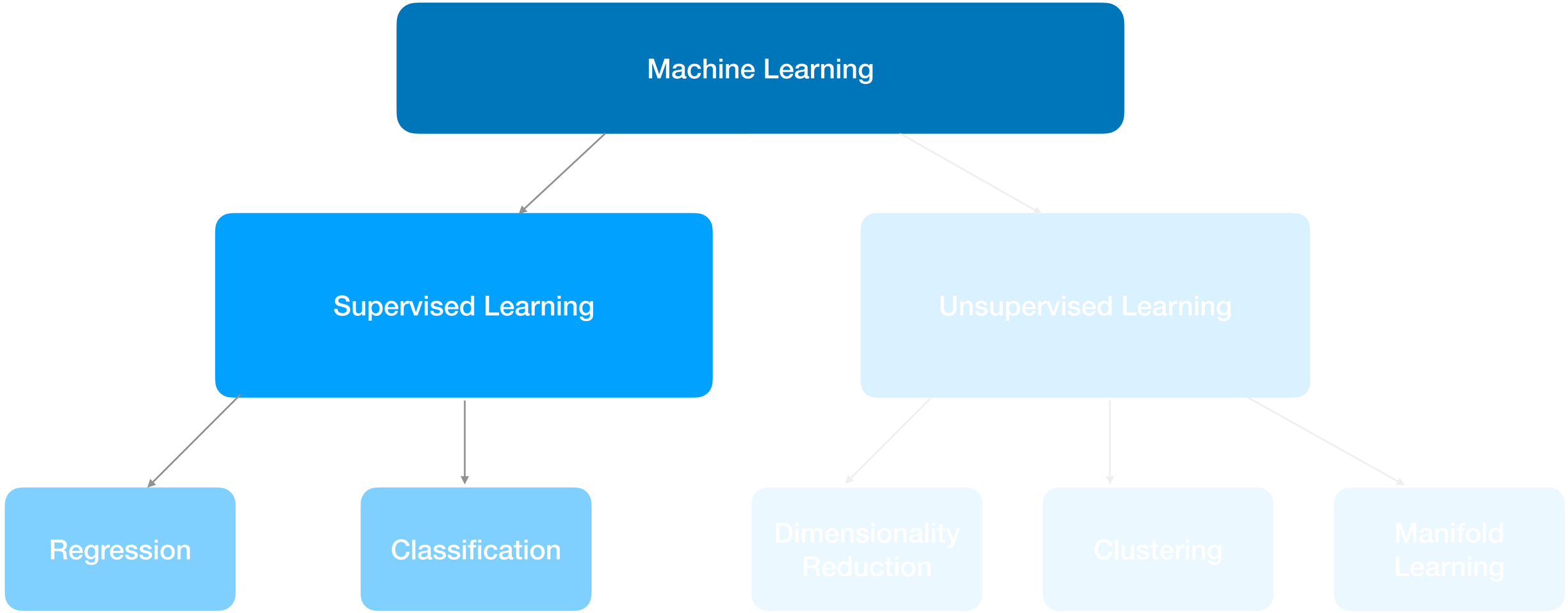
What is ML?


Machine Learning
Supervised Learning
Unsupervised Learning

What is ML?


Machine Learning
Supervised Learning
Unsupervised Learning
Regression
Classification
Dimensionality Reduction
Clustering
Manifold Learning

What is ML?


Machine Learning
Supervised Learning
Unsupervised Learning
Regression
Classification
Dimensionality Reduction
Clustering
Manifold Learning

Introduction to ML:

Machine Learning

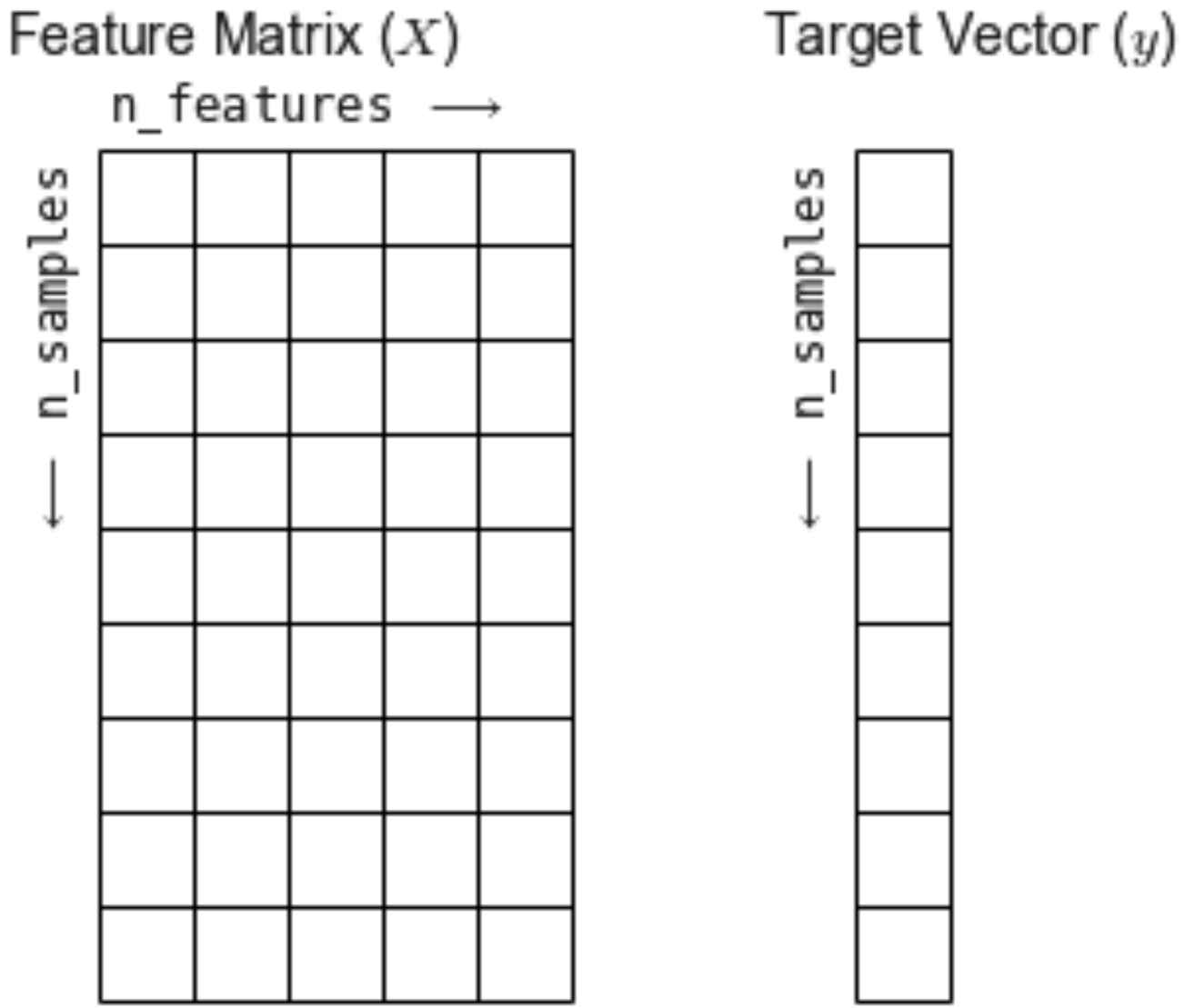
Introduction to SKLEARN

Model Selection

Scikit-Learn

- Consistent:
  - All sklearn models share a common interface with very consistent documentation.
- Customizable:
  - All parameter values of sklearn models are public and easy to modify.
- Composable:
  - Many ML tasks can be thought of as sequences of basic algorithms, and sklearn makes use of this.
- Comparable:
  - Python and sklearn's interface make it particularly easy to compare results from different models or parameter choices.

Scikit-Learn requirements



Scikit-Learn

1. Choose a model class

```
[ ]: from sklearn import some_nice_model
```

2. Choose model hyper parameters

```
[ ]: MyModel = some_nice_model(hyper_parameter1 = 0, hyper_parameter2 = 'Blah', ...., hyper_parameter100 = 0)
```

3. Create feature and target matrix

```
[ ]: X = Feature_Matrix
     Y = Target_Matrix
```

4. Fit the model to your data

```
[ ]: MyModel.fit(X,Y)
```

5. Apply model to new data

```
[ ]: Y_predict = MyModel.predict(X)
```

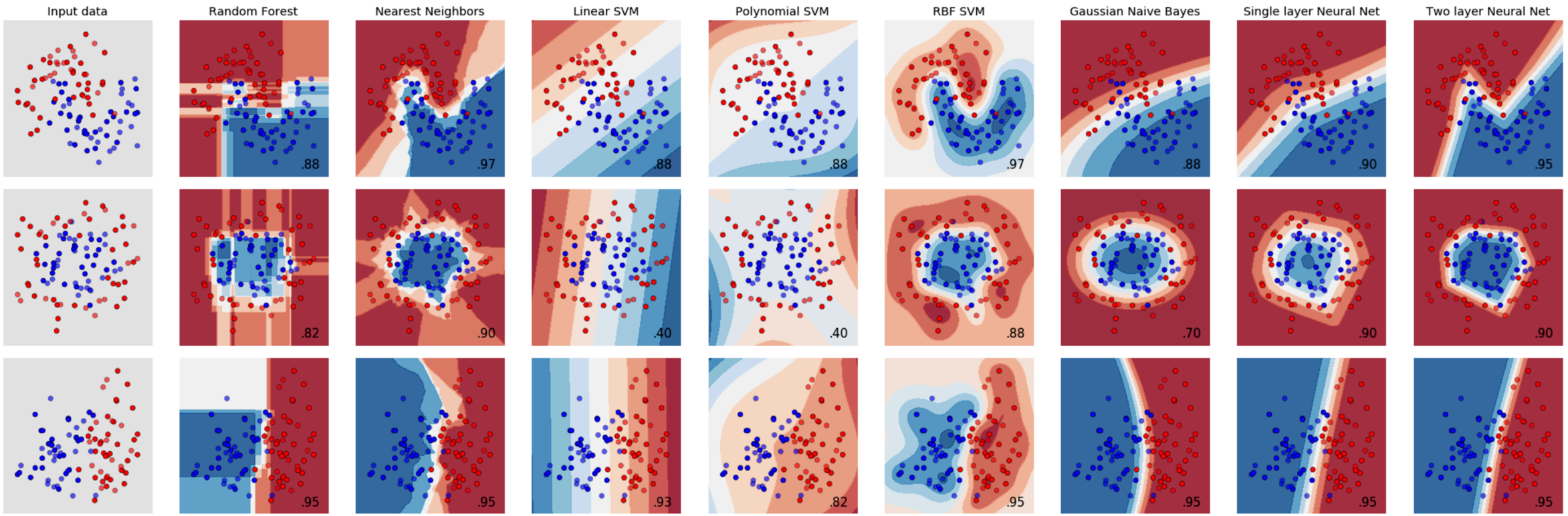
Introduction to ML:

Machine Learning

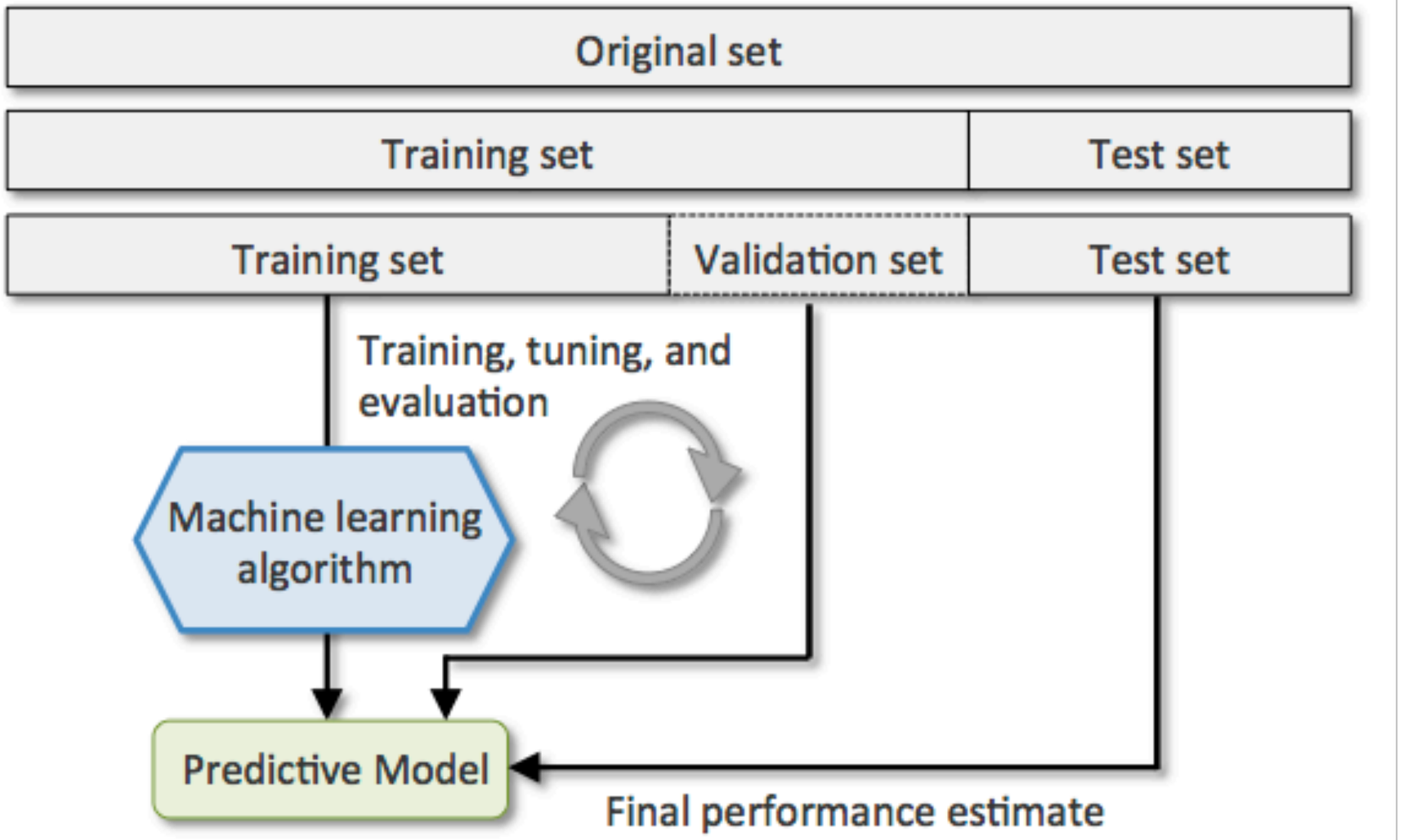
Introduction to SKLEARN

Model Selection

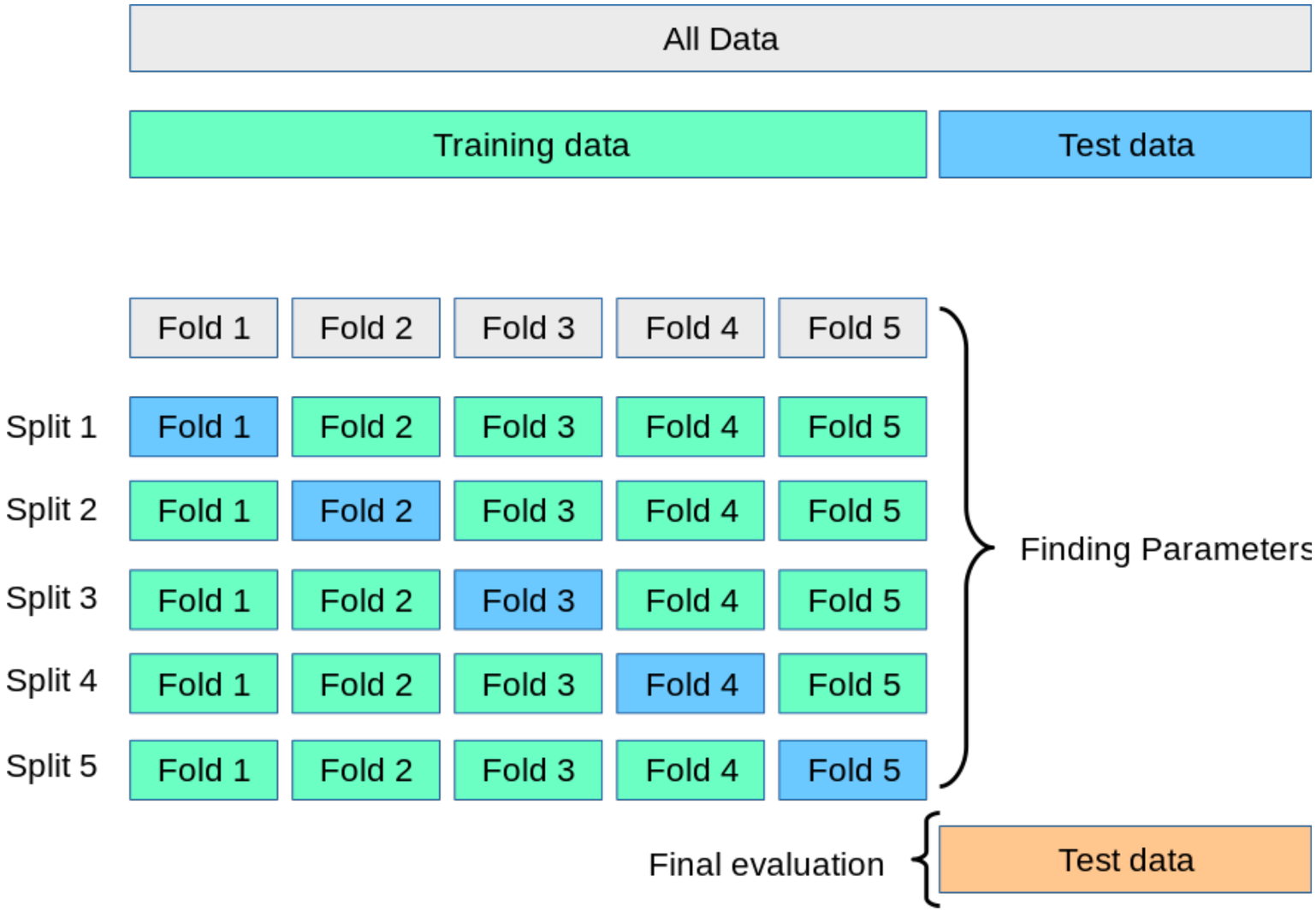
Model Selection



Model Selection



Model Selection



Train/Test

```
[ ]: from sklearn.model_selection import train_test_split
     X_train, X_test, Y_train, Y_test = train_test_split(X,Y,train_size = None)
```

Cross-Validation option 1

```
[ ]: from sklearn.model_selection import cross_validate, cross_val_score
valreports = cross_validate(MyModel,X_train,Y_train, scoring='Whatever scoring method you want to use')
valreports['test_score']
valreports['train_score']
valreports['fit_time']
```

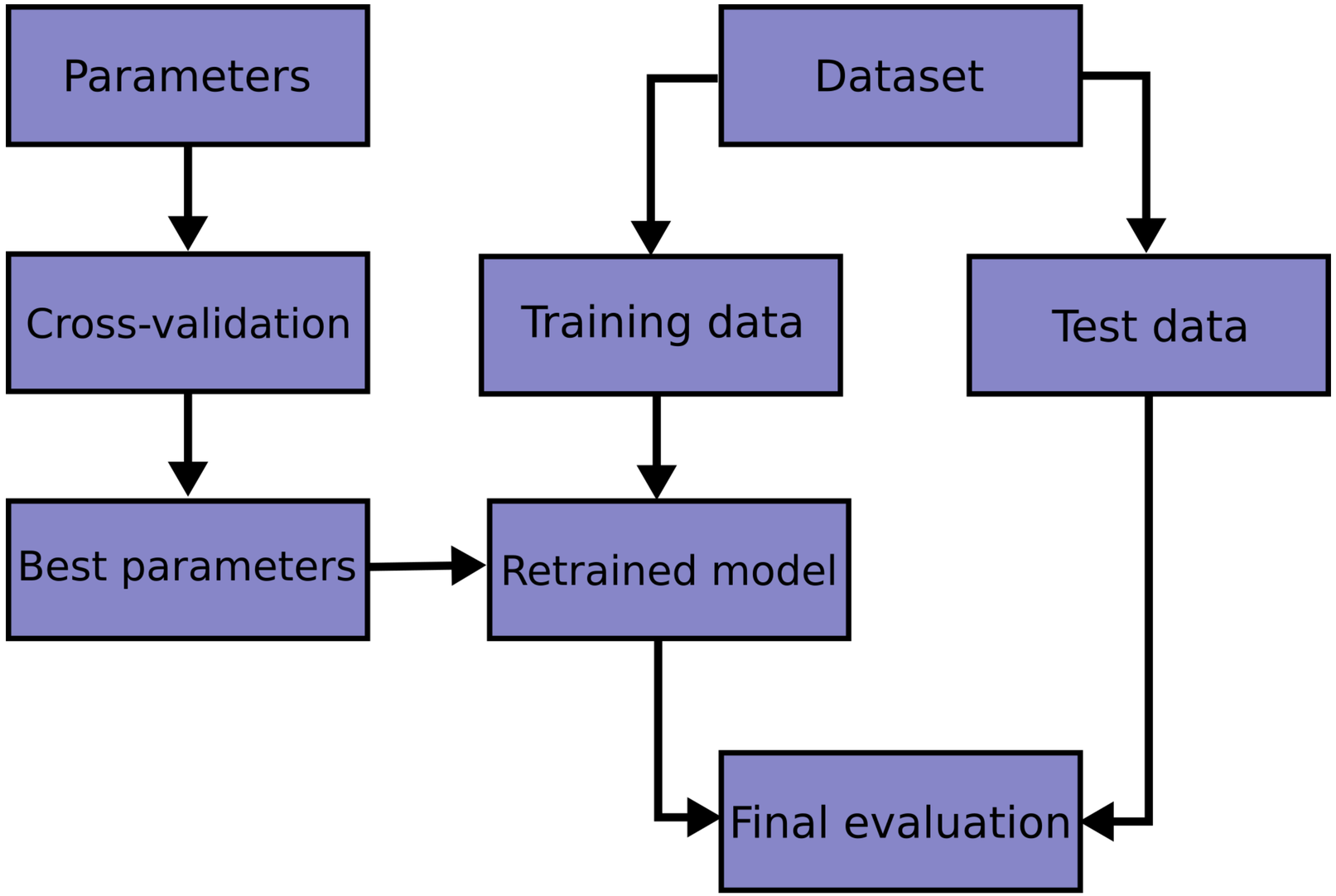
Some common scoring methods are:

- ‘R2’: R2 score
- ‘neg_mean_squared_error’
- ‘max_error’

If you only care about the scores you can also use:

```
[ ]: valscores = cross_val_score(MyModel,X_train, Y_train, scoring='...')
```

GridSearchCV



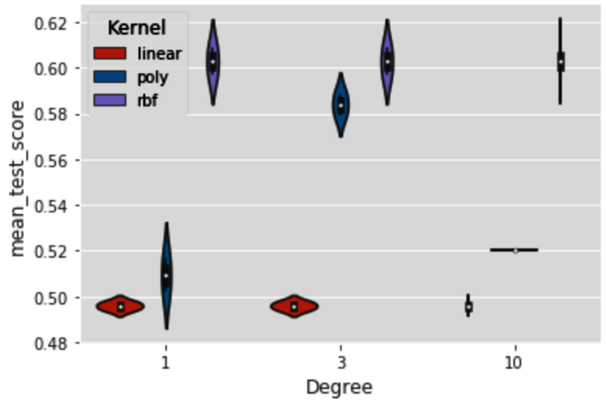
Model Selection

[ ]:
```
from sklearn.model_selection import GridSearchCV
parameters = {'parameter_1': [0,1,10] ,
              'parameter_2': ['Option 1', 'Option 2'],}
grid = GridSearchCV(MyModel,paramgrid=parameters, scoring='...')
grid.fit(X_train,Y_train)
best_model = grid.best_estimator_
grid.best_params
grid.best_score
```

Model Selection

[58]:
```
parameters = {'kernel': ['linear','poly','rbf'],
              'degree': [1,3,10],
              'C': [1,10]}
svm_grid = GridSearchCV(svm.SVC(), param_grid=parameters, n_jobs=5)
svm_grid.fit(X_train,Y_train)
print(svm_grid.best_params_)
best_model = svm_grid.best_estimator_
df = pd.DataFrame(svm_grid.cv_results_)
df.rename(columns = {'param_degree': 'Degree', 'param_kernel': 'Kernel'},inplace=True)
plt.figure()
sns.violinplot(x='Degree', y='mean_test_score',hue='Kernel',data=df)
plt.show()
print(svm_grid.cv_results_.keys())
```

```
{'C': 10, 'degree': 1, 'kernel': 'rbf'}
```




```
dict_keys(['mean_fit_time', 'std_fit_time', 'mean_score_time', 'std_score_time', 'param_C', 'param_degree', 'param_kernel', 'params', 'split0_test_score', 'split1_test_score', 'split2_test_score', 'split3_test_score', 'split4_test_score', 'mean_test_score', 'std_test_score', 'rank_test_score'])
```

Outline:

Introduction to ML

Regression

Classification

Group Exercises

Regression:

Linear Regression

Polynomial Regression

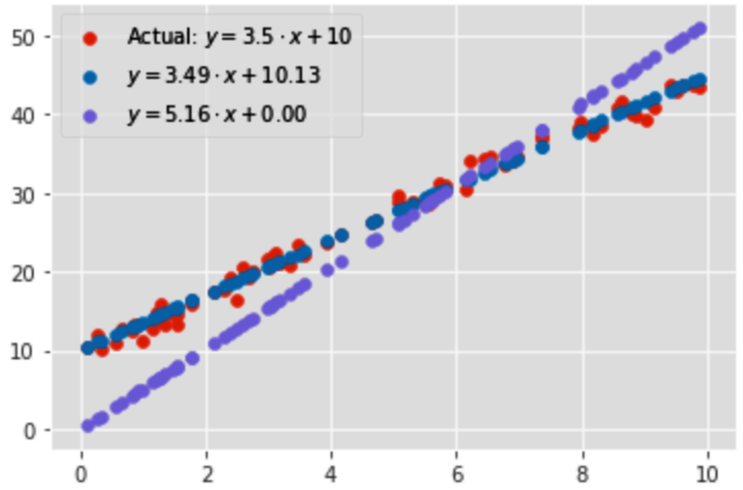
Multilayer Perceptron Regression

Linear Regression

[145]:
```
n_samples = 75
x_train = 10*np.random.rand(n_samples,1)
y_train = 3.5*x_train + 10 + np.random.randn(n_samples,1)
x_test = 10*np.random.rand(n_samples,1)
y_test = 3.5*x_test + 10 + np.random.randn(n_samples,1)
plt.scatter(x_train[:,0] , y_train)
plt.title('Training Data')
plt.show()
```

Linear Regression

[146]:
```
from sklearn.linear_model import LinearRegression
MyModel = LinearRegression()
MyModel_ni = LinearRegression(fit_intercept = False)
MyModel.fit(x_train,y_train)
MyModel_ni.fit(x_train,y_train)
predict = MyModel.predict(x_test)
predict_ni = MyModel_ni.predict(x_test)
plt.scatter(x_test,y_test,label=r'Actual: $y = 3.5 \cdot x + 10$')
plt.scatter(x_test,predict,label=r'$y= {:.2f} \cdot x + {:.2f} $'.format(MyModel.coef_[0][0],MyModel.intercept_[0]))
plt.scatter(x_test,predict_ni,label=r'$y= {:.2f} \cdot x + {:.2f} $'.format(MyModel_ni.coef_[0][0],MyModel_ni.intercept_))
plt.legend()
plt.show()
```



Regression:

Linear Regression

Polynomial Regression

Multilayer Perceptron Regression

Polynomial Regression

$$y = a + a_1 x^1 + a_2 x^2 + \ldots + a_n x^n$$

$$b_n = x^n$$ Kernel

$$y = a + a_1 b_1 + a_2 b_2 + \ldots + a_n b_n$$

Polynomial Regression

[159]:
```
from sklearn.preprocessing import PolynomialFeatures

nums = np.array([1,2,3,4,5])

poly_transform = PolynomialFeatures(3,include_bias = False)
polys = poly_transform.fit_transform(nums[:,np.newaxis])
print(polys)
```

```
[[  1.   1.   1.]
 [  2.   4.   8.]
 [  3.   9.  27.]
 [  4.  16.  64.]
 [  5.  25. 125.]]
```

Polynomial Regression

[160]:
```
from sklearn.preprocessing import PolynomialFeatures

nums = np.array([1,2,3,4,5])

poly_transform = PolynomialFeatures(3,include_bias = True)
polys = poly_transform.fit_transform(nums[:,np.newaxis])
print(polys)
```

```
[[  1.   1.   1.   1.]
 [  1.   2.   4.   8.]
 [  1.   3.   9.  27.]
 [  1.   4.  16.  64.]
 [  1.   5.  25. 125.]]
```

Polynomial Regression

[160]:
```
from sklearn.preprocessing import PolynomialFeatures

nums = np.array([1,2,3,4,5])

poly_transform = PolynomialFeatures(3,include_bias = True)
polys = poly_transform.fit_transform(nums[:,np.newaxis])
print(polys)
```

```
[[  1.   1.   1.   1.]
 [  1.   2.   4.   8.]
 [  1.   3.   9.  27.]
 [  1.   4.  16.  64.]
 [  1.   5.  25. 125.]]
```

Polynomial Regression

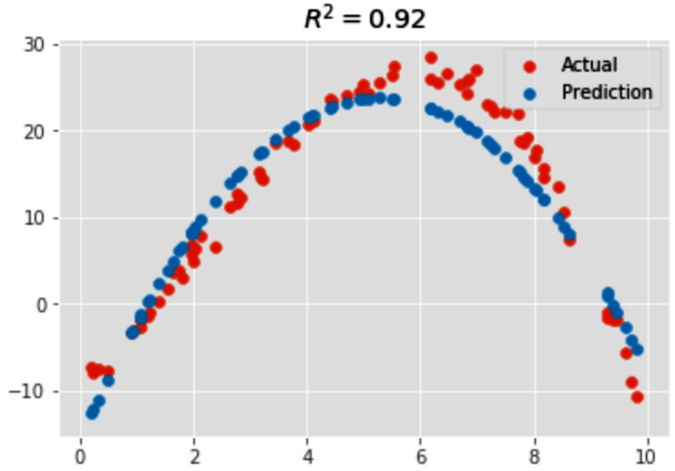
[169]:
```
from sklearn.pipeline import Pipeline
from sklearn.preprocessing import PolynomialFeatures
my_pipeline = Pipeline([('poly_transform',PolynomialFeatures()), ('linear_regression',LinearRegression())])
```

Polynomial Regression

[181]:
```
def poly(x):
    X = x.squeeze()
    return -.15*X**3 + .75*X**2 + 7*X - 10 + np.random.randn(x.shape[0])
x_train = 10*np.random.rand(n_samples,1)
y_train = poly(x_train)
x_test = 10*np.random.rand(n_samples,1)
y_test = poly(x_test)
plt.scatter(x_train[:,0] , y_train)
plt.title('Training Data')
plt.show()
```

Polynomial Regression

[182]:
```
from sklearn.pipeline import Pipeline
from sklearn.preprocessing import PolynomialFeatures
my_pipeline = Pipeline([('poly_transform',PolynomialFeatures()), ('linear_regression',LinearRegression())])
my_pipeline.fit(x_train,y_train)
prediction = my_pipeline.predict(x_test)
plt.title(r'$R^2 ={:.2f}$'.format(my_pipeline.score(x_test,y_test)))
plt.scatter(x_test[:,0],y_test,label='Actual: ')
plt.scatter(x_test[:,0],prediction,label='Prediction')
plt.legend()
plt.show()
```

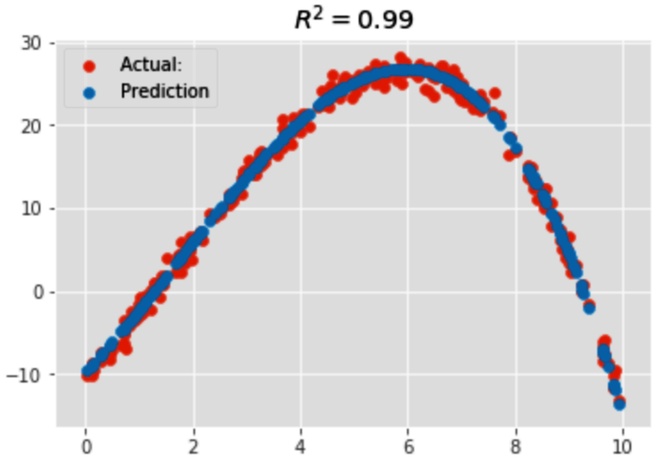


[194]:
```
print(my_pipeline['poly_transform'].get_feature_names())
print(my_pipeline['linear_regression'].coef_)
```

```
['1', 'x0', 'x0^2']
[ 0.         14.87190599 -1.40582506]
```

Polynomial Regression

[204]:
```
my_pipeline = Pipeline([('poly_transform',PolynomialFeatures()), ('linear_regression',LinearRegression())])
params = {'poly_transform__degree': np.arange(5)+1}
grid = GridSearchCV(my_pipeline,param_grid = params,scoring='neg_mean_squared_error')
grid.fit(x_train,y_train)
best_model = grid.best_estimator_
prediction = best_model.predict(x_test)
plt.title(r'$R^2 ={:.2f}$'.format(best_model.score(x_test,y_test)))
plt.scatter(x_test[:,0],y_test,label='Actual: ')
plt.scatter(x_test[:,0],prediction,label='Prediction')
plt.legend()
plt.show()
print(best_model['poly_transform'].get_feature_names())
print(best_model['linear_regression'].coef_)
```




```
['1', 'x0', 'x0^2', 'x0^3']
[ 0.          6.85748271  0.79506764 -0.15363529]
```

Regression:

Linear Regression

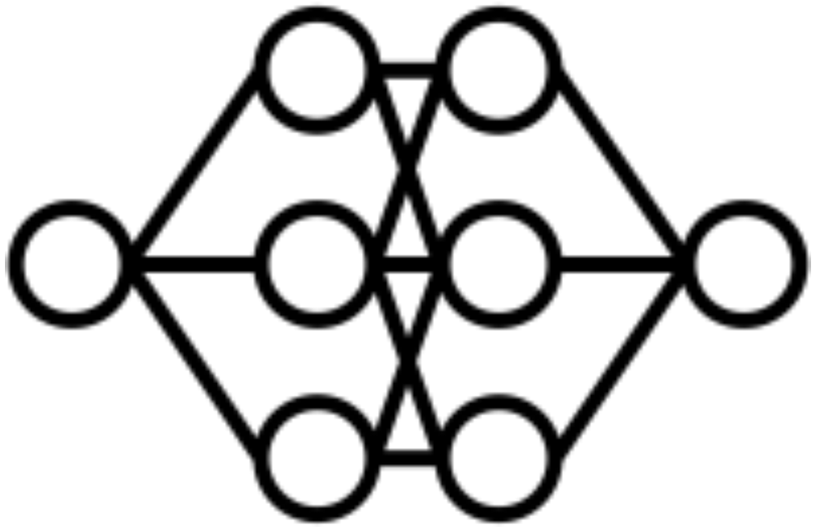
Polynomial Regression

Multilayer Perceptron Regression

Multilayer Perceptrons

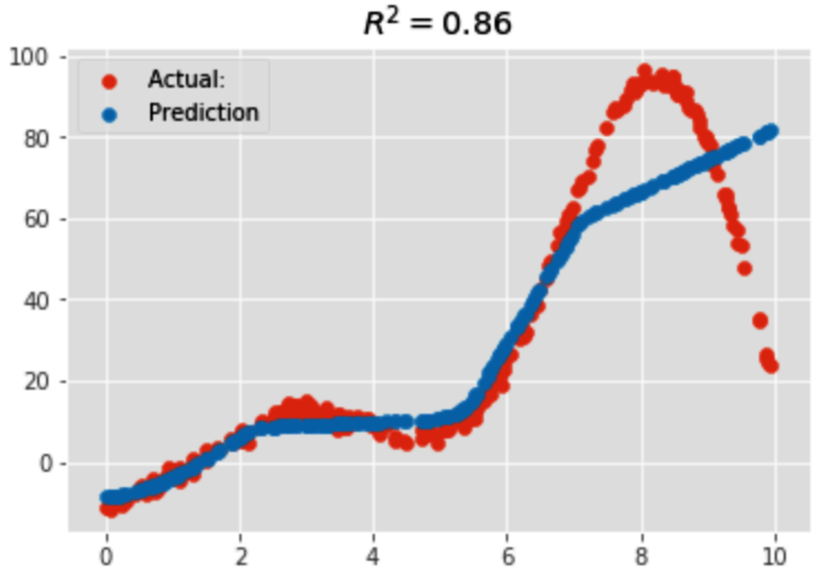
[205]:
```
def poly(x):
    X = x.squeeze()
    return -np.cos(.15*X**3) + .75*X**2 * np.sin(X) + 7*X - 10 + np.random.randn(x.shape[0])
n_samples = 250
x_train = 10*np.random.rand(n_samples,1)
y_train = poly(x_train)
x_test = 10*np.random.rand(n_samples,1)
y_test = poly(x_test)
plt.scatter(x_train[:,0] , y_train)
plt.title('Training Data')
plt.show()
```

Multilayer Perceptrons



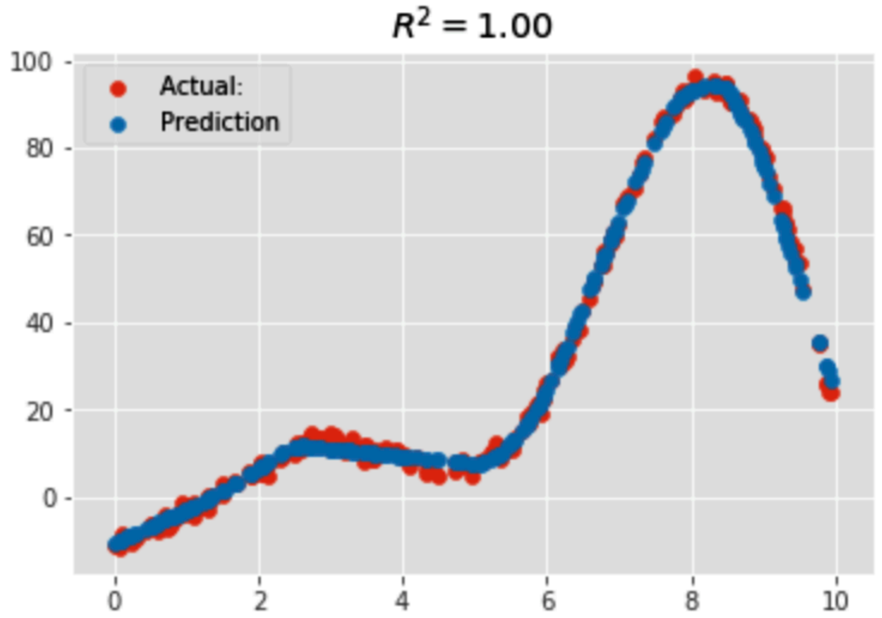
Multilayer Perceptrons

[212]:
```
from sklearn.neural_network import MLPRegressor
Network = MLPRegressor(hidden_layer_sizes=(50,25,10),max_iter=2500)
Network.fit(x_train,y_train)
prediction = Network.predict(x_test)
plt.title(r'$R^2 ={:.2f}$'.format(Network.score(x_test,y_test)))
plt.scatter(x_test[:,0],y_test,label='Actual: ')
plt.scatter(x_test[:,0],prediction,label='Prediction')
plt.legend()
plt.show()
```



Multilayer Perceptrons

[213]:
```
from sklearn.neural_network import MLPRegressor
Network = MLPRegressor(hidden_layer_sizes=(50,50,50,25),max_iter=2500)
Network.fit(x_train,y_train)
prediction = Network.predict(x_test)
plt.title(r'$R^2 ={:.2f}$'.format(Network.score(x_test,y_test)))
plt.scatter(x_test[:,0],y_test,label='Actual: ')
plt.scatter(x_test[:,0],prediction,label='Prediction')
plt.legend()
plt.show()
```



Outline:

Introduction to ML

Regression

Classification

Group Exercises

Classification:

Random Forests

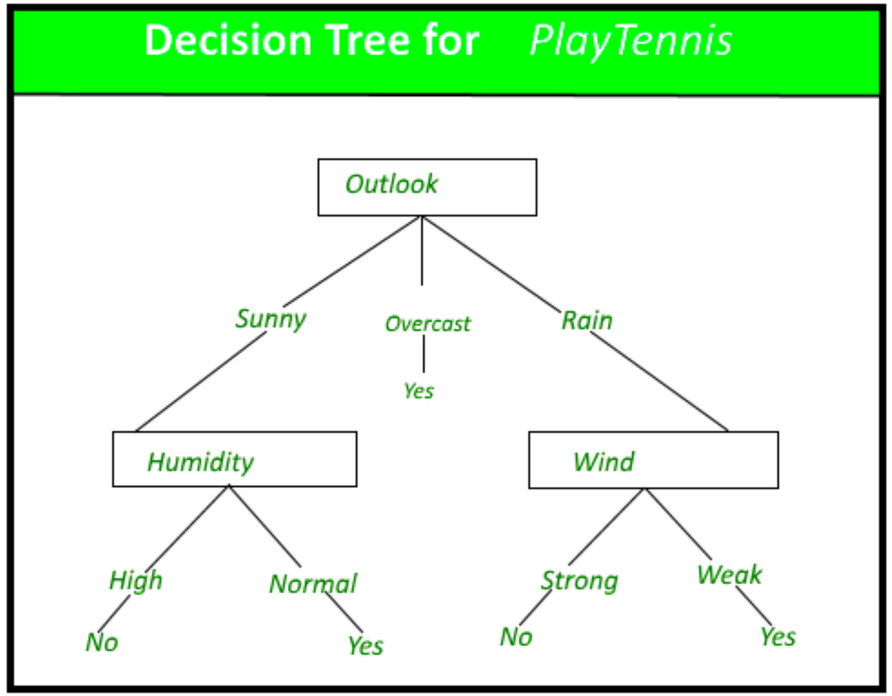
K-Nearest Neighbors

Support Vector Machines

Multilayer Perceptrons

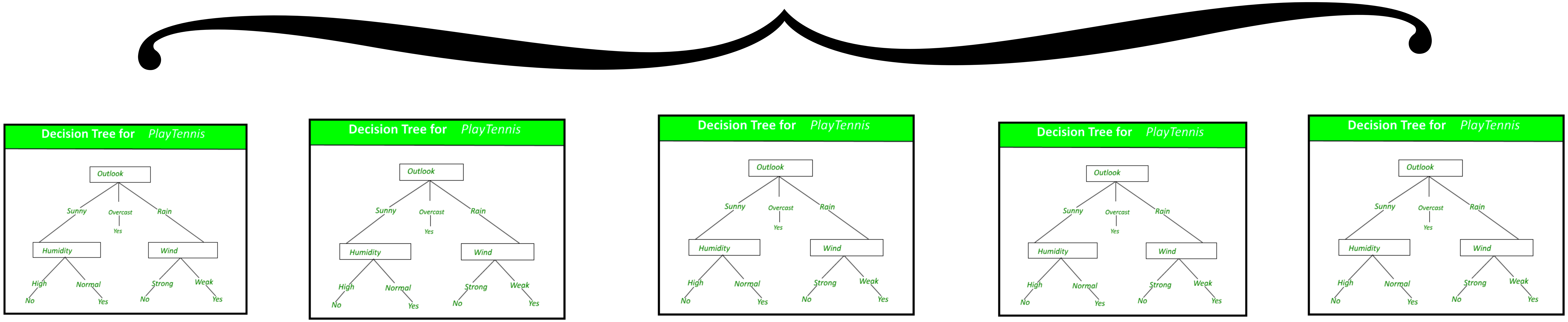
Gaussian Naive Bayes

Random Forest



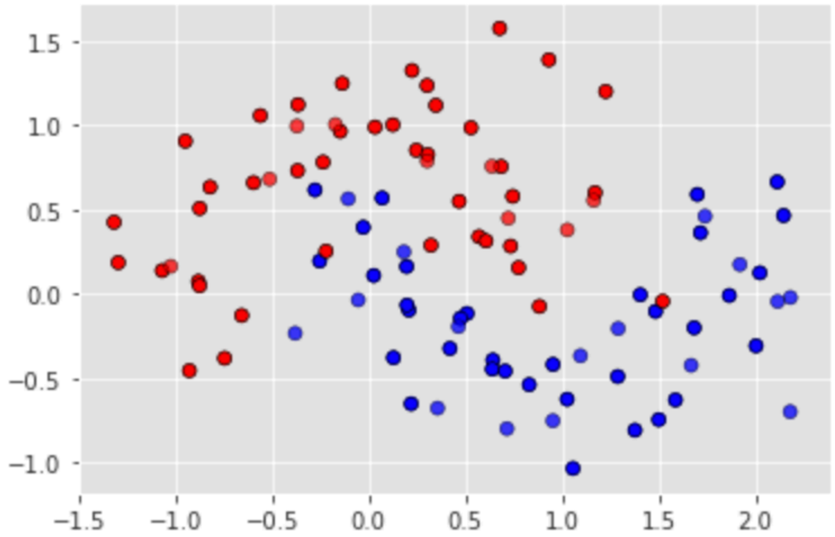
Random Forest

xK



Random Forest

[219]:
```
from sklearn.datasets import make_moons
Data = make_moons(noise=0.3, random_state=0)
X, y = Data
X_train, X_test ,Y_train , Y_test = train_test_split(X, y)
plt.scatter(X_train[:, 0], X_train[:, 1], c=Y_train, cmap=cm_bright,
            edgecolors='k')
# Plot the testing points
plt.scatter(X_test[:, 0], X_test[:, 1], c=Y_test, cmap=cm_bright, alpha=0.6,
            edgecolors='k')
plt.show()
```



Random Forest

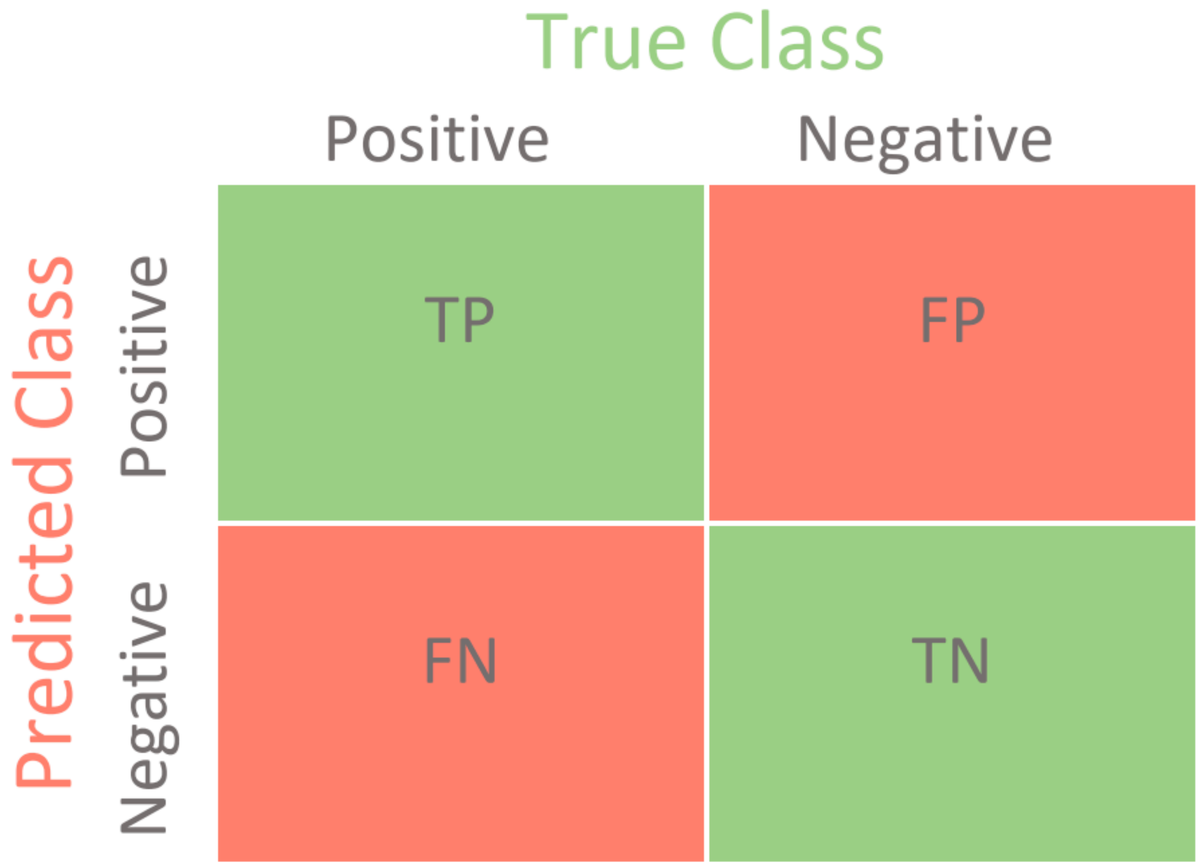
[225]:
```
from sklearn.ensemble import RandomForestClassifier

forest = RandomForestClassifier(n_estimators = 10, max_depth =5)
forest.fit(X_train, Y_train)
print(['True Labels: ', Y_test])
print(['Predicted Labels: ', forest.predict(X_test)])
forest.predict_proba(X_test)
```

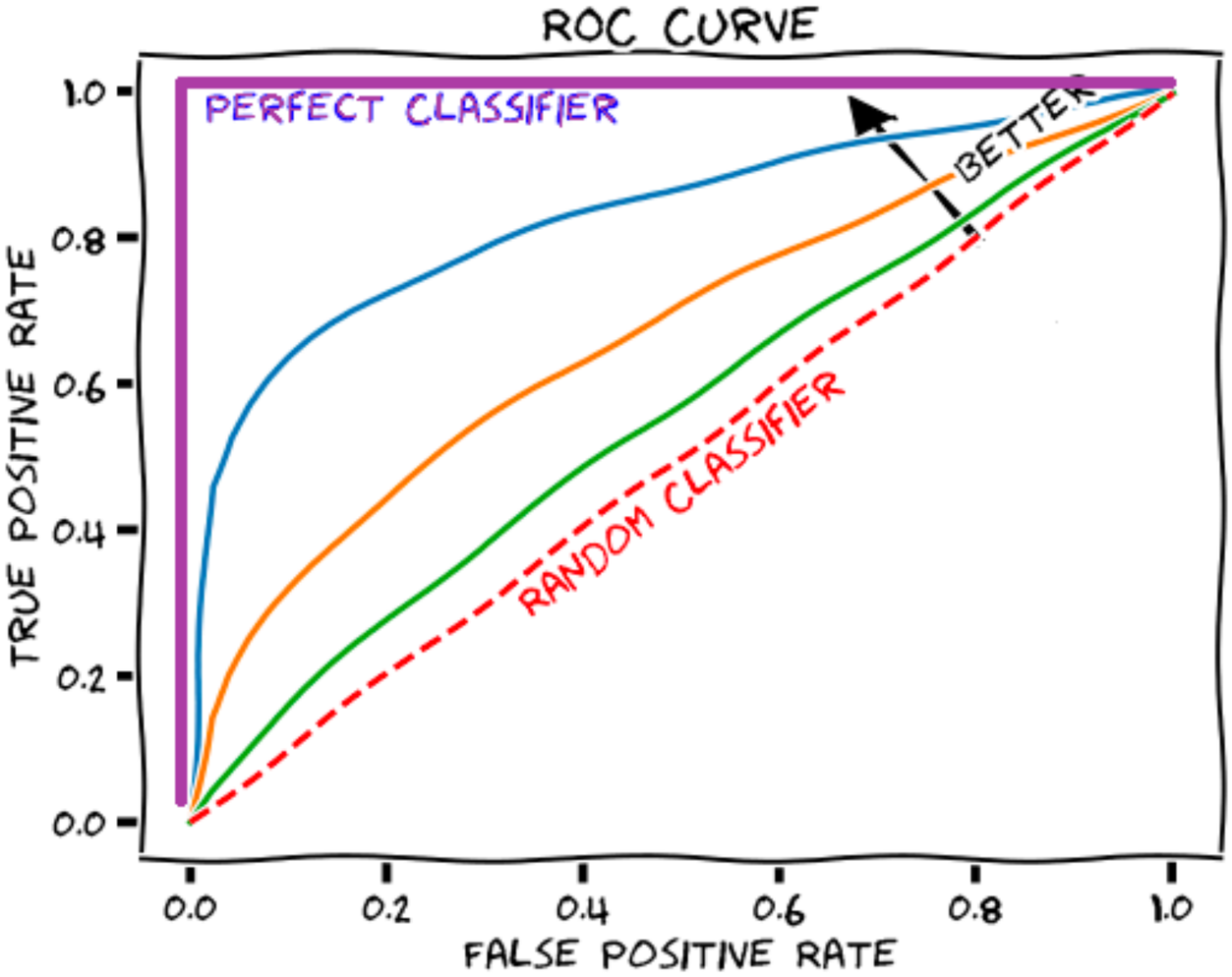
```
['True Labels: ', array([1, 1, 0, 0, 1, 0, 0, 0, 1, 1, 0, 1, 1, 1, 1, 1, 0, 1, 0, 1, 1, 1,
       0, 1, 1])]
['Predicted Labels: ', array([1, 1, 0, 0, 1, 0, 0, 0, 1, 1, 0, 1, 1, 0, 1, 1, 0, 1, 0, 1, 1, 1,
       0, 0, 1])]
```

[225]:
```
array([[0.05714286, 0.94285714],
       [0.25108974, 0.74891026],
       [0.71588412, 0.28411588],
       [0.99333333, 0.00666667],
       [0.        , 1.        ],
       [0.9       , 0.1       ],
       [1.        , 0.        ],
       [0.97      , 0.03      ],
       [0.10833333, 0.89166667],
       [0.025     , 0.975     ],
       [0.86040793, 0.13959207],
       [0.09333333, 0.90666667],
       [0.325     , 0.675     ],
       [0.82874126, 0.17125874],
       [0.48333333, 0.51666667],
       [0.07      , 0.93      ],
       [0.98333333, 0.01666667],
       [0.09333333, 0.90666667],
       [0.97      , 0.03      ],
       [0.025     , 0.975     ],
       [0.        , 1.        ],
       [0.10833333, 0.89166667],
       [0.52421745, 0.47578255],
       [0.58540793, 0.41459207],
       [0.        , 1.        ]])
```

Random Forest



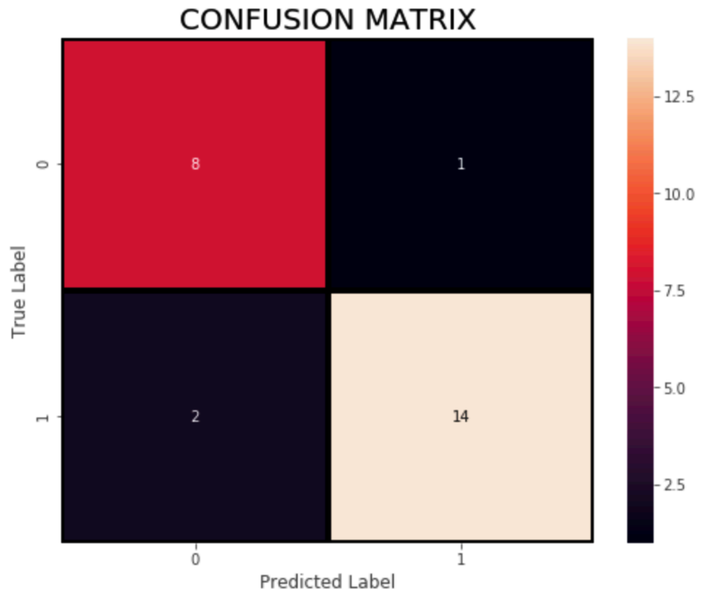
Random Forest

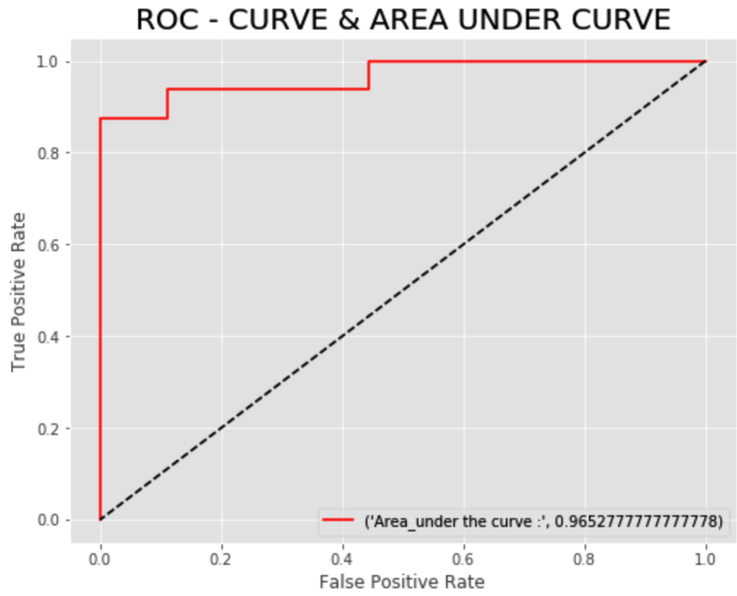


Random Forest

[222]:
```
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import confusion_matrix, roc_curve
forest = RandomForestClassifier(n_estimators = 10, max_depth =5)
forest.fit(X_train, Y_train)
prediction = forest.predict(X_test)
plt.figure(figsize=(18, 14))
plt.subplot(221)
sns.heatmap(confusion_matrix(Y_test, prediction), annot=True, fmt = "d", linecolor="k", linewidths=3)
plt.title("CONFUSION MATRIX", fontsize=20)
plt.xlabel('Predicted Label')
plt.ylabel('True Label')

predicted_probs = forest.predict_proba(X_test)[:, 1]
fpr, tpr, thresholds = metrics.roc_curve(Y_test, predicted_probs)
plt.subplot(222)
plt.plot(fpr, tpr, label = ("Area_under the curve :", metrics.auc(fpr, tpr)), color = "r")
plt.plot([1,0], [1,0], linestyle = "dashed", color ="k")
plt.legend(loc = "best")
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title("ROC - CURVE & AREA UNDER CURVE",fontsize=20)
plt.show()
```





Classification:

Random Forests

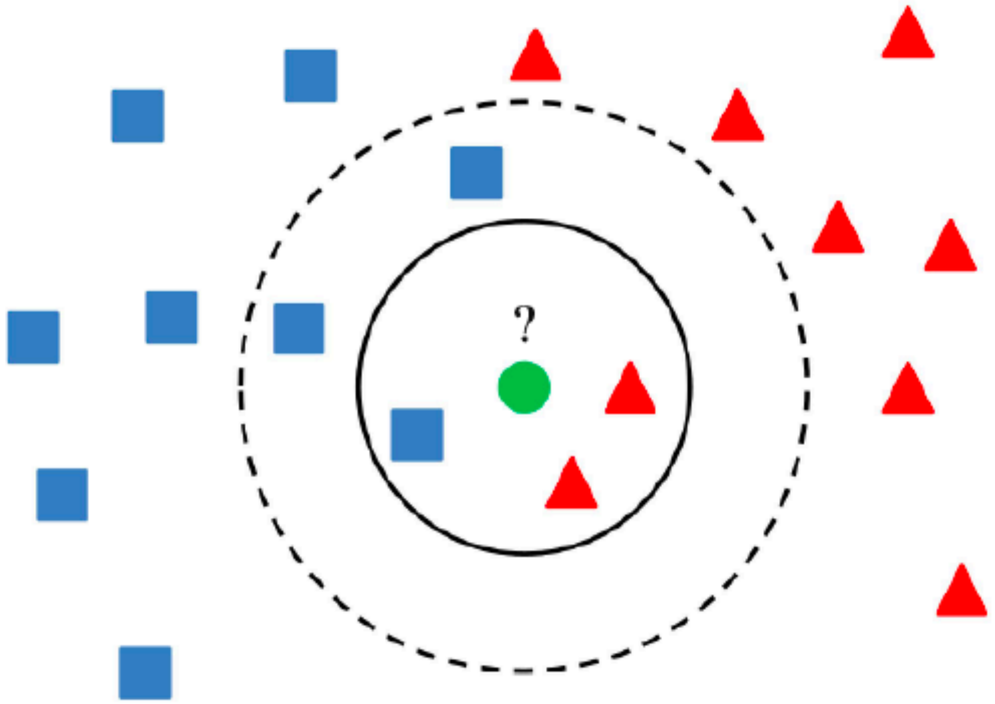
K-Nearest Neighbors

Support Vector Machines

Multilayer Perceptrons

Gaussian Naive Bayes

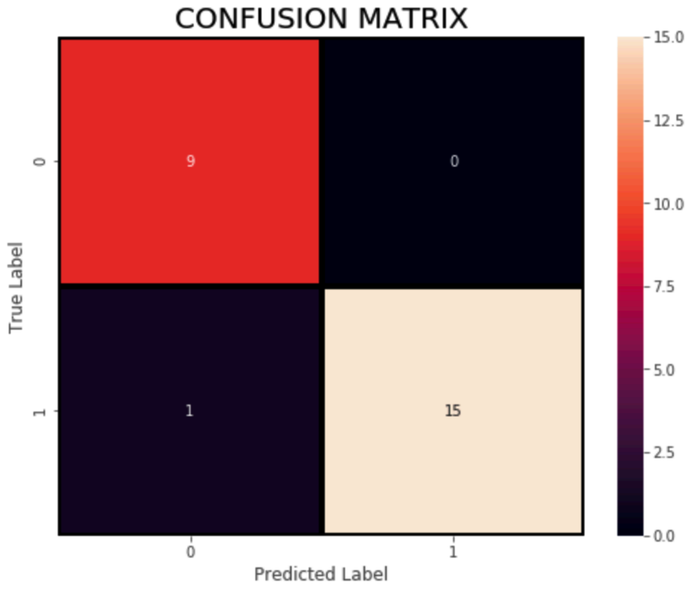
k-Nearest Neighbors

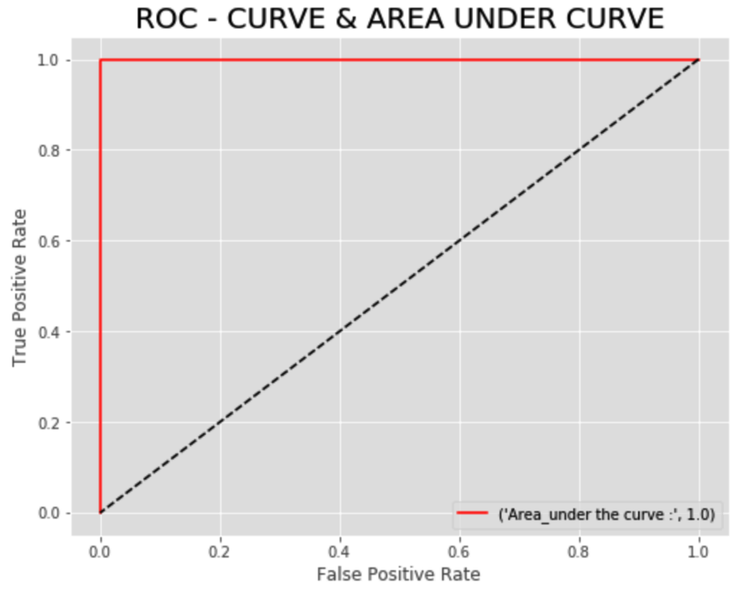


k-Nearest Neighbors

[229]:
```
from sklearn.neighbors import KNeighborsClassifier
neighbors = KNeighborsClassifier(n_neighbors = 5,metric='minkowski')
neighbors.fit(X_train, Y_train)
prediction = neighbors.predict(X_test)
plt.figure(figsize=(18, 14))
plt.subplot(221)
sns.heatmap(metrics.confusion_matrix(Y_test, prediction), annot=True, fmt = "d", linecolor="k", linewidths=3)
plt.title("CONFUSION MATRIX", fontsize=20)
plt.xlabel('Predicted Label')
plt.ylabel('True Label')

predicted_probs = neighbors.predict_proba(X_test)[:, 1]
fpr, tpr, thresholds = metrics.roc_curve(Y_test, predicted_probs)
plt.subplot(222)
plt.plot(fpr, tpr, label = ("Area_under the curve :", metrics.auc(fpr, tpr)), color = "r")
plt.plot([1,0], [1,0], linestyle = "dashed", color ="k")
plt.legend(loc = "best")
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title("ROC - CURVE & AREA UNDER CURVE",fontsize=20)
plt.show()
```





Classification:

Random Forests

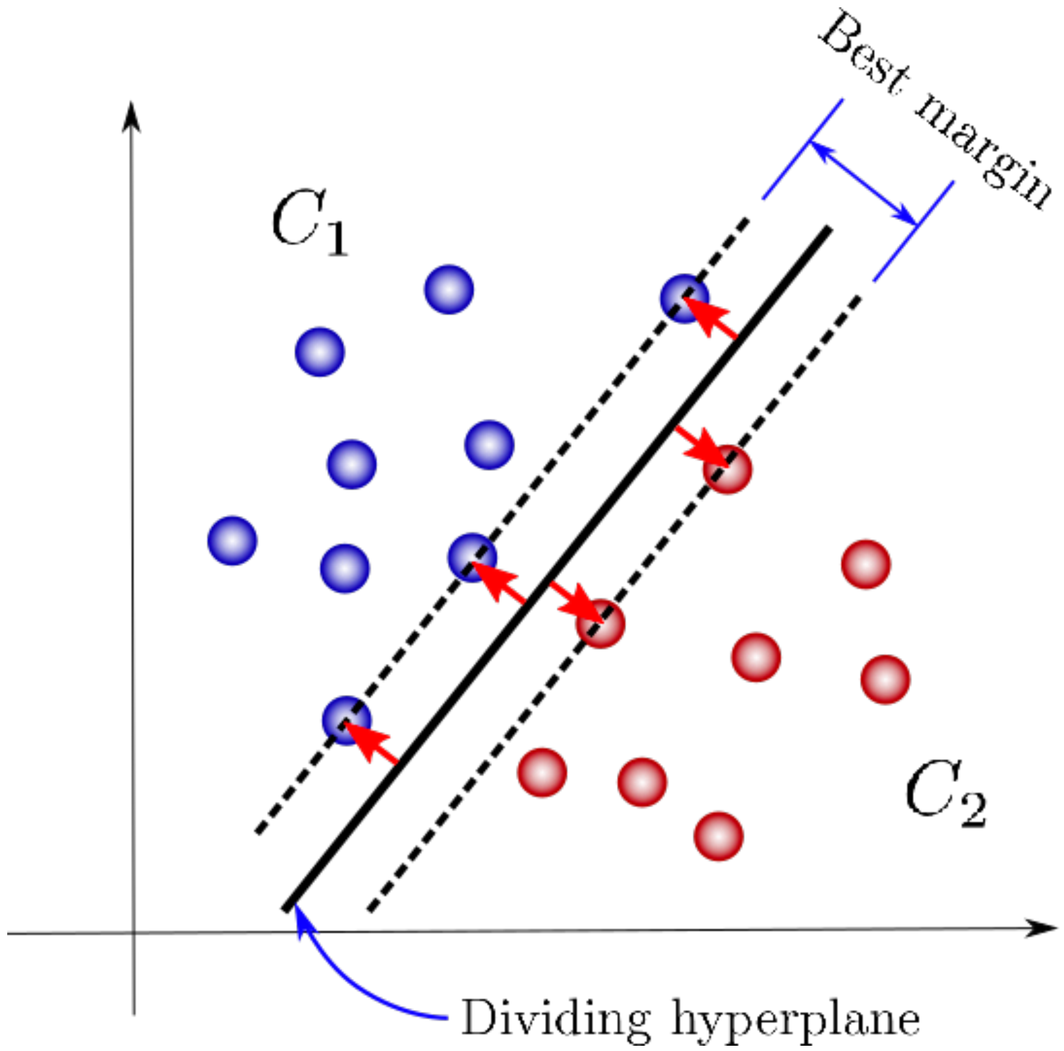
K-Nearest Neighbors

Support Vector Machines

Multilayer Perceptrons

Gaussian Naive Bayes

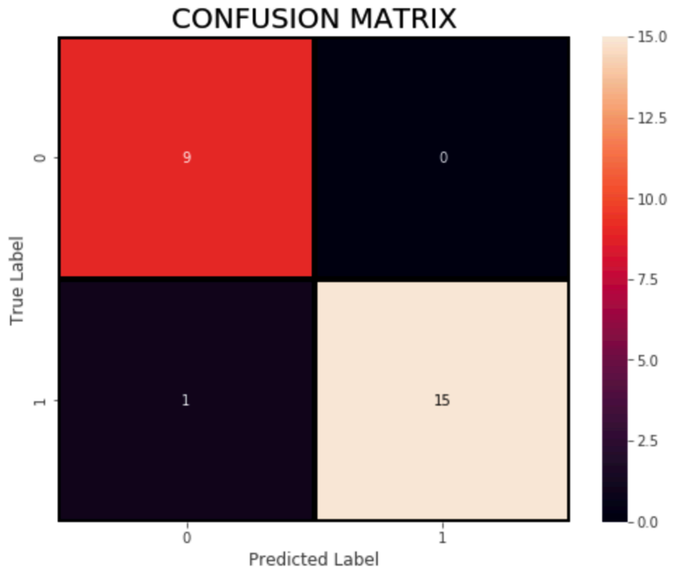
Support Vector Machines

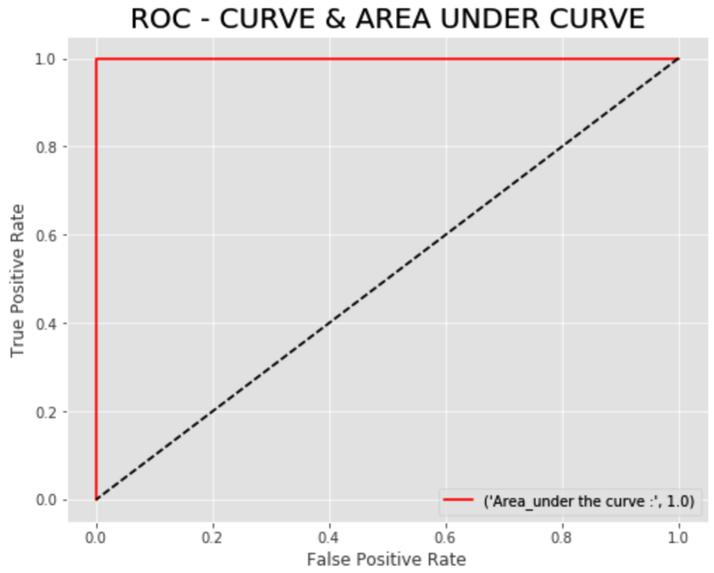


Support Vector Machines

[233]:
```
from sklearn.svm import SVC
classifier = SVC(C = 2,kernel='rbf',probability=True)
classifier.fit(X_train, Y_train)
prediction = classifier.predict(X_test)
plt.figure(figsize=(18, 14))
plt.subplot(221)
sns.heatmap(metrics.confusion_matrix(Y_test, prediction), annot=True, fmt = "d", linecolor="k", linewidths=3)
plt.title("CONFUSION MATRIX", fontsize=20)
plt.xlabel('Predicted Label')
plt.ylabel('True Label')

predicted_probs = classifier.predict_proba(X_test)[:, 1]
fpr, tpr, thresholds = metrics.roc_curve(Y_test, predicted_probs)
plt.subplot(222)
plt.plot(fpr, tpr, label = ("Area_under the curve :", metrics.auc(fpr, tpr)), color = "r")
plt.plot([1,0], [1,0], linestyle = "dashed", color ="k")
plt.legend(loc = "best")
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title("ROC - CURVE & AREA UNDER CURVE",fontsize=20)
plt.show()
```





Classification:

Random Forests

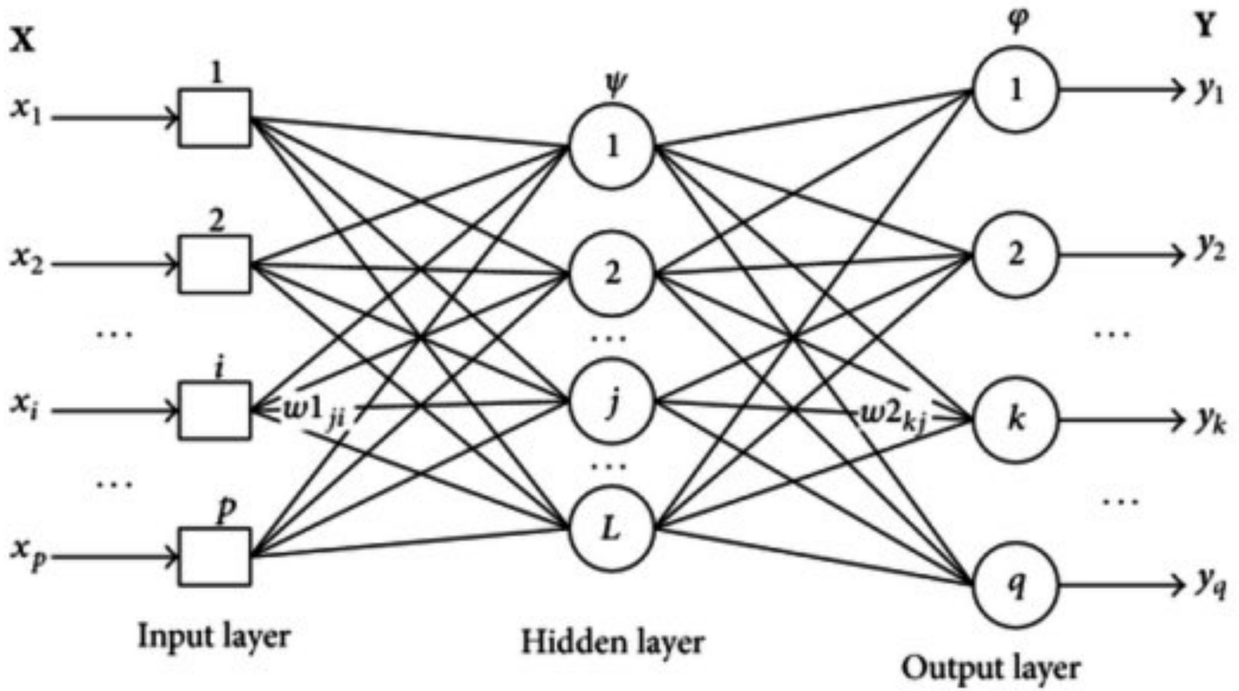
K-Nearest Neighbors

Support Vector Machines

Multilayer Perceptrons

Gaussian Naive Bayes

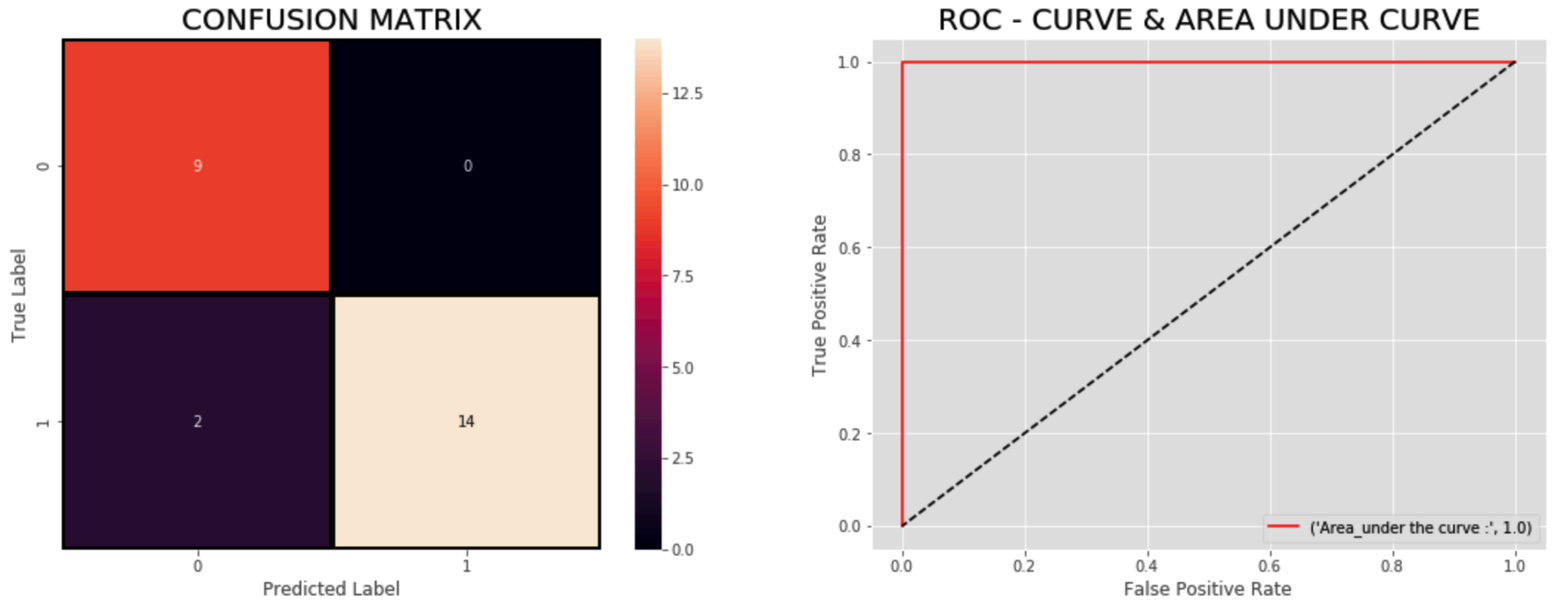
Multilayer Perceptrons



Multilayer Perceptrons

[238]:
```
from sklearn.neural_network import MLPClassifier
classifier = MLPClassifier(hidden_layer_sizes=(100),max_iter = 1500)
classifier.fit(X_train, Y_train)
prediction = classifier.predict(X_test)
plt.figure(figsize=(18, 14))
plt.subplot(221)
sns.heatmap(metrics.confusion_matrix(Y_test, prediction), annot=True, fmt = "d", linecolor="k", linewidths=3)
plt.title("CONFUSION MATRIX", fontsize=20)
plt.xlabel('Predicted Label')
plt.ylabel('True Label')

predicted_probs = classifier.predict_proba(X_test)[:, 1]
fpr, tpr, thresholds = metrics.roc_curve(Y_test, predicted_probs)
plt.subplot(222)
plt.plot(fpr, tpr, label = ("Area_under the curve :", metrics.auc(fpr, tpr)), color = "r")
plt.plot([1,0], [1,0], linestyle = "dashed", color ="k")
plt.legend(loc = "best")
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title("ROC - CURVE & AREA UNDER CURVE",fontsize=20)
plt.show()
```



Classification:

Random Forests

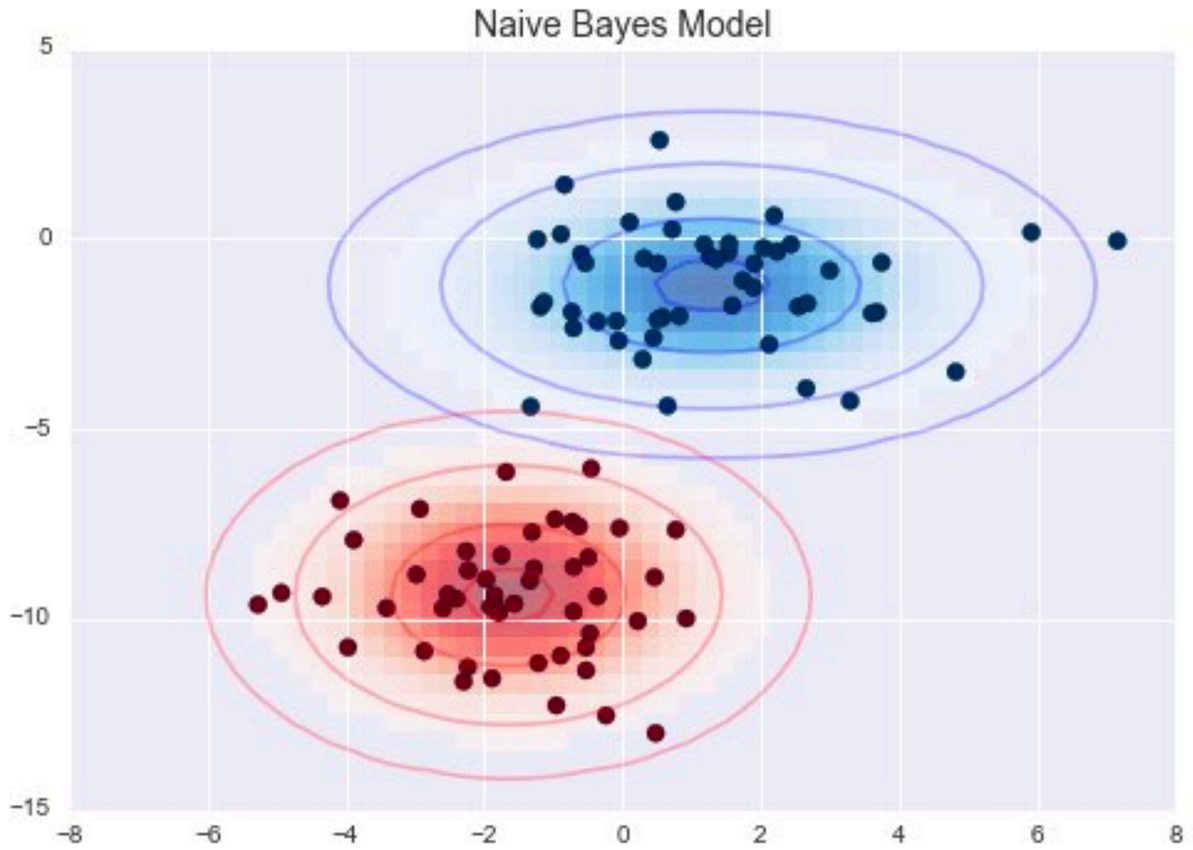
K-Nearest Neighbors

Support Vector Machines

Multilayer Perceptrons

Gaussian Naive Bayes

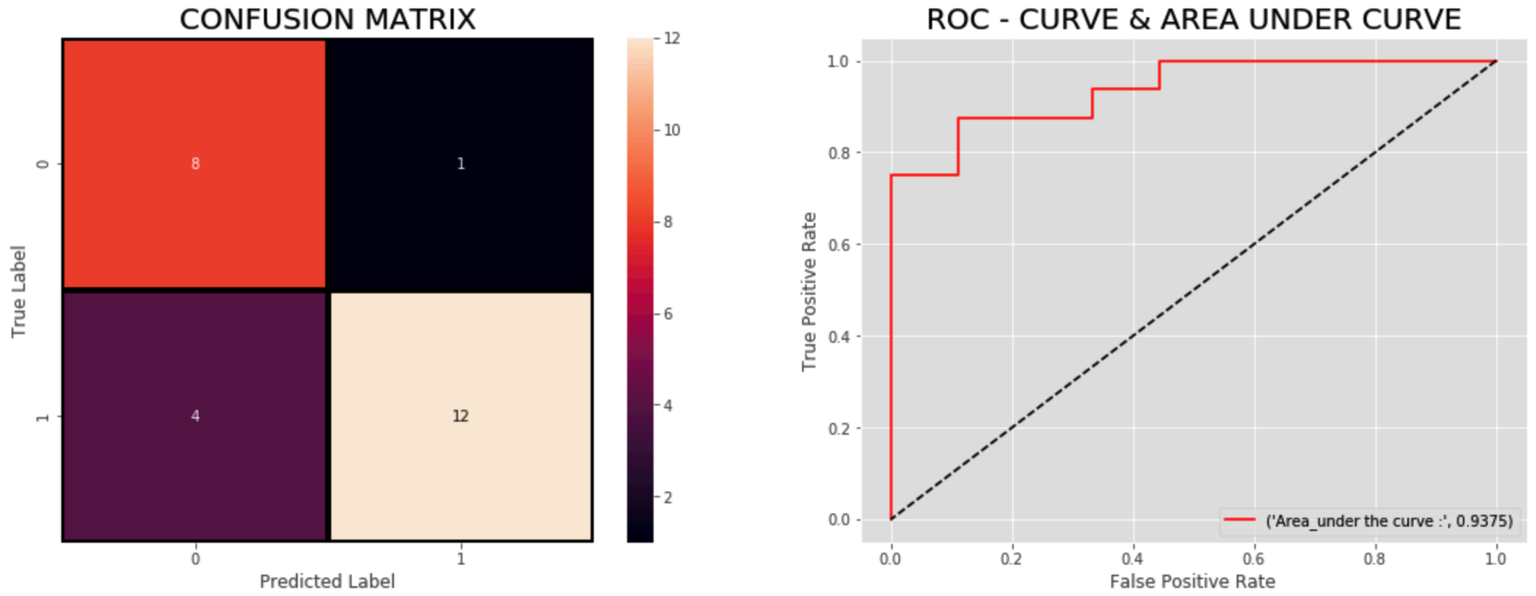
Gaussian Naive Bayes



Gaussian Naive Bayes

[240]:
```
from sklearn.naive_bayes import GaussianNB
classifier = GaussianNB(priors=None)
classifier.fit(X_train, Y_train)
prediction = classifier.predict(X_test)
plt.figure(figsize=(18, 14))
plt.subplot(221)
sns.heatmap(metrics.confusion_matrix(Y_test, prediction), annot=True, fmt = "d", linecolor="k", linewidths=3)
plt.title("CONFUSION MATRIX", fontsize=20)
plt.xlabel('Predicted Label')
plt.ylabel('True Label')

predicted_probs = classifier.predict_proba(X_test)[:, 1]
fpr, tpr, thresholds = metrics.roc_curve(Y_test, predicted_probs)
plt.subplot(222)
plt.plot(fpr, tpr, label = ("Area_under the curve :", metrics.auc(fpr, tpr)), color = "r")
plt.plot([1,0], [1,0], linestyle = "dashed", color ="k")
plt.legend(loc = "best")
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title("ROC - CURVE & AREA UNDER CURVE",fontsize=20)
plt.show()
```



Outline:

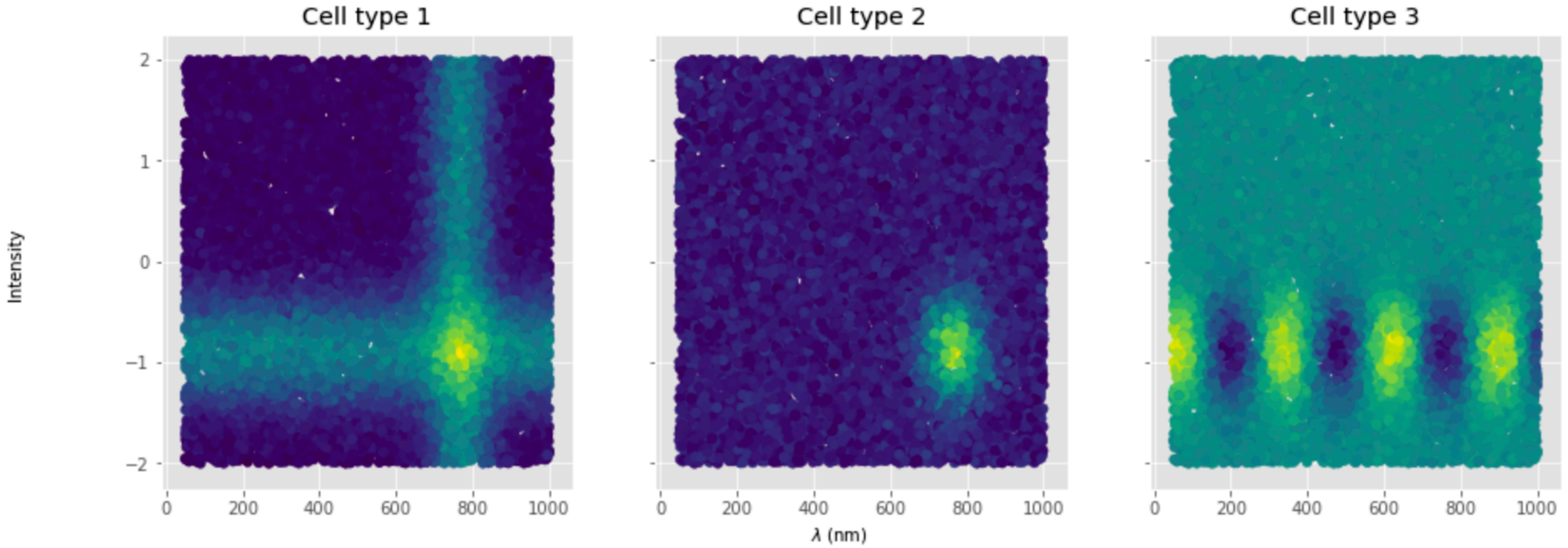
Introduction to ML

Regression

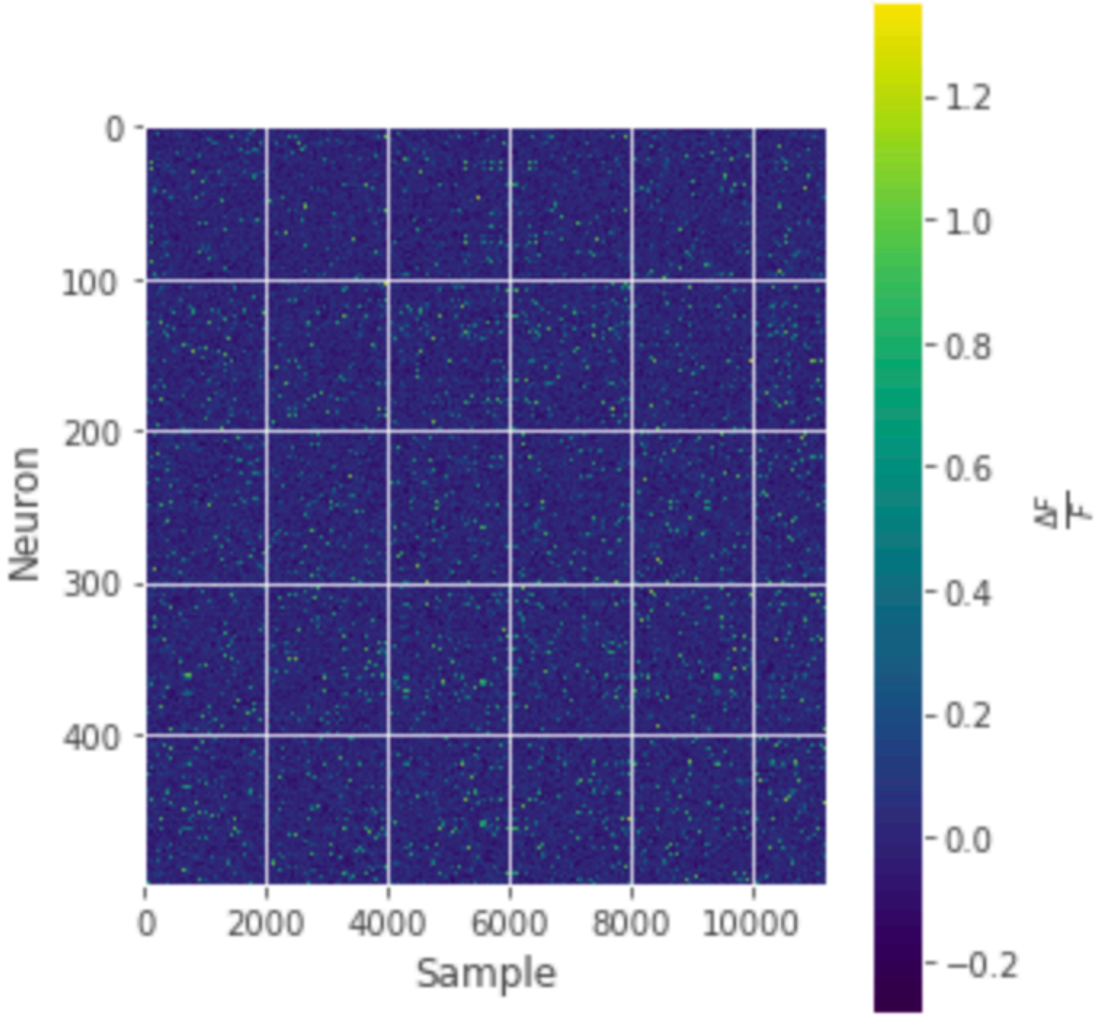
Classification

Group Exercises

Exercise 1:


Cell type 1
Cell type 2
Cell type 3
Intensity
2
1
0
−1
−2
0
200
400
600
800
1000
λ (nm)

Exercise 2:


0
100
200
300
400
Neuron
0
2000
4000
6000
8000
10000
Sample
1.2
1.0
0.8
0.6
0.4
0.2
0.0
−0.2
ΔF/F

Exercise 3:

Molecular classification of cancer: class discovery and class prediction by gene expression monitoring

T R Golub [1], D K Slonim, P Tamayo, C Huard, M Gaasenbeek, J P Mesirov, H Coller, M L Loh, J R Downing, M A Caligiuri, C D Bloomfield, E S Lander

Affiliations + expand

PMID: 10521349 DOI: 10.1126/science.286.5439.531

Free article

Abstract

Although cancer classification has improved over the past 30 years, there has been no general approach for identifying new cancer classes (class discovery) or for assigning tumors to known classes (class prediction). Here, a generic approach to cancer classification based on gene expression monitoring by DNA microarrays is described and applied to human acute leukemias as a test case. A class discovery procedure automatically discovered the distinction between acute myeloid leukemia (AML) and acute lymphoblastic leukemia (ALL) without previous knowledge of these classes. An automatically derived class predictor was able to determine the class of new leukemia cases. The results demonstrate the feasibility of cancer classification based solely on gene expression monitoring and suggest a general strategy for discovering and predicting cancer classes for other types of cancer, independent of previous biological knowledge.