# Data Analysis

## Practice 2: Supervised learning with scikit-learn lib

Dr. Nataliya K. Sakhnenko

*Please review Lectures 2–3 before this practical*

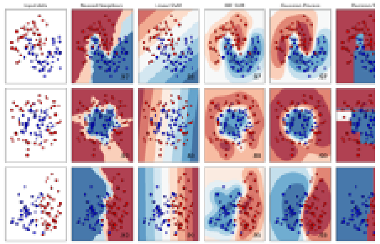
# Scikit-learn

- Simple and efficient tools for data mining and data analysis
- Accessible to everybody, and reusable in various contexts
- Built on NumPy, SciPy, and matplotlib
- Open source, commercially usable

## Classification

Identifying which category an object belongs to.

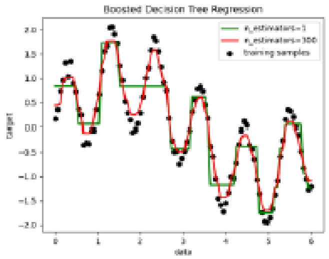
**Applications:** Spam detection, image recognition.
**Algorithms:** SVM, nearest neighbors, random forest, and more...



## Regression

Predicting a continuous-valued attribute associated with an object.

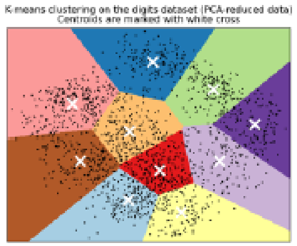
**Applications:** Drug response, Stock prices.
**Algorithms:** SVR, nearest neighbors, random forest, and more...




## Clustering

Automatic grouping of similar objects into sets.

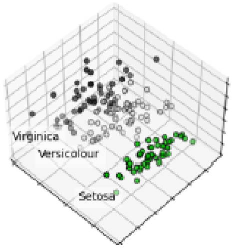
**Applications:** Customer segmentation, Grouping experiment outcomes
**Algorithms:** k-Means, spectral clustering, mean-shift, and more...




## Dimensionality reduction

Reducing the number of random variables to consider.

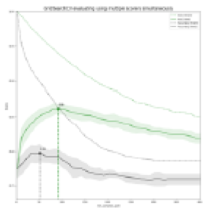
**Applications:** Visualization, Increased efficiency
**Algorithms:** k-Means, feature selection, non-negative matrix factorization, and more...




## Model selection

Comparing, validating and choosing parameters and models.

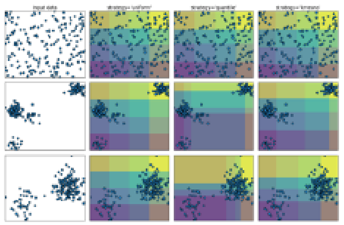
**Applications:** Improved accuracy via parameter tuning
**Algorithms:** grid search, cross validation, metrics, and more...



## Preprocessing

Feature extraction and normalization.

**Applications:** Transforming input data such as text for use with machine learning algorithms.
**Algorithms:** preprocessing, feature extraction, and more...



Data in scikit-learn stored as a 2D array [m_samples, n_features]

# Core interface of Scikit-learn


## Linear Models

```
from sklearn.linear_model import LinearRegression
model = LinearRegression()
model.fit(X_train, y_train)
model.predict(X_test)
```

Normal Equation

```
model.get_params()
```

$$J(\theta) = MSE(\theta) + \frac{\lambda}{2}\sum_{i=1}^{n}\theta_i^2$$

Stochatic gradient descent

```
from sklearn.linear_model import SGDRegressor
model =SGDRegressor(loss='squared_error',penalty='l2',alpha=0.0001,
max_iter=1000,learning_rate='invscaling', eta0=0.01, power_t=0.25)
```

```
from sklearn.linear_model import LogisticRegression
model = LogisticRegression(penalty='l2', class_weight=None,
random_state=None, solver='lbfgs')
```

If we want Logistic Regression trained with gradient descent, we use `SGDClassifier(loss='log_loss')` or choose a solver based on gradient descent (`saga`) in `LogisticRegression`.

# Scikit-learn

## kNN

```
from sklearn.neighbors import KNeighborsClassifier
model = KNeighborsClassifier(n_neighbors=5, weights ='uniform’)
```

```
from sklearn.neighbors import KNeighborsRegressor
model = KNeighborsRegressor(n_neighbors=5, weights ='uniform')
```

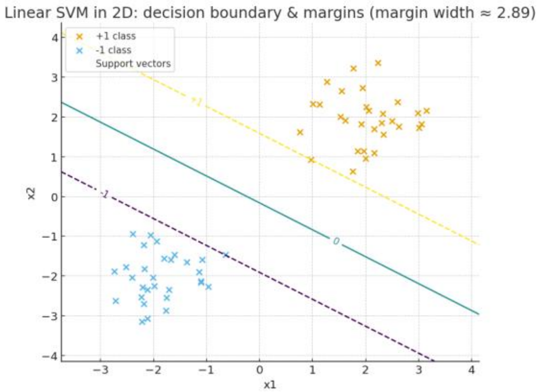
## Decision Tree

```
from sklearn.tree import DecisionTreeClassifier
model = DecisionTreeClassifier(criterion="gini", max_depth=None,
min_samples_split=2)
```

```
from sklearn.tree import DecisionTreeRegressor
model = DecisionTreeRegressor(criterion="squared_error")
```

# SVM in sklearn lib

```python
from sklearn.svm import LinearSVC
from sklearn.linear_model import SGDClassifier
from sklearn.svm import SVC
```




```python
SGDClassifier(loss='hinge')
SVC(kernel='rbf')
```

| Class | Time complexity | Scaling required | Kernel trick |
| --- | --- | --- | --- |
| LinearSVC | O(mxn) | Yes | No |
| SGDClassifier | O(mxn) | Yes | No |
| SVC | $O(m^2xn)$ to $O(m^3xn)$ | Yes | Yes |

SVMs are sensitive to the feature scaling!!!

```python
from sklearn.svm import LinearSVR
from sklearn.svm import SVR
```

SVM software packages:

- libsvm – most commonly used implementation of kernalized svm, sklearn uses wrapper over it
- liblinear – gradient descent based implementation of linear SVM

# Scikit-learn-ensemble

```
from sklearn.ensemble import RandomForestClassifier,
      RandomForestRegressor
from sklearn.ensemble import AdaBoostClassifier, AdaBoostRegressor
```

Scikit-learn also provides the **`ensemble`** module, which implements ensemble learning methods

These models combine the predictions of multiple "weak learners" (often decision trees) to build a stronger, more accurate model.

✓ **RandomForest** – bagging of many decision trees

✓ **AdaBoost** – boosting that reweights errors to focus on harder samples

# sklearn.model_selection

The model_selection module provides tools for splitting datasets (train_test_split), cross-validation (KFold, cross_val_score), and hyperparameter tuning (GridSearchCV).

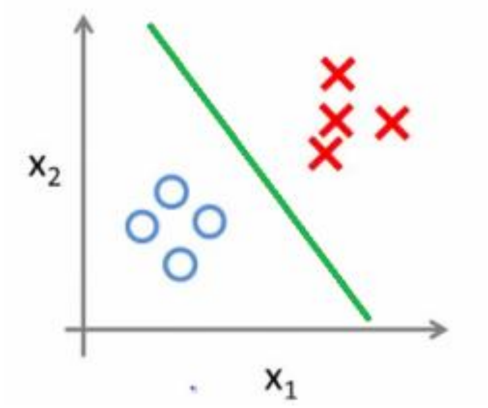
## Train Test split

```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, shuffle=True,
stratify=None, test_size = 0.2)
```

## GridSearch

```
param_grid = {'C': [0.1, 1, 10, 100, 1000], 'gamma':
[1,0.1,0.01,0.001,0.0001], 'kernel': ['rbf']}
grid = GridSearchCV(SVC(), param_grid,
cv=5,refit=True)
grid.fit(X_train,y_train)
```

# Recap: Binary Classifier




$$f : X \to Y, \;\; Y = \{0,1\}$$

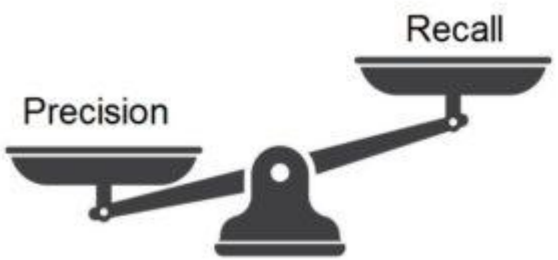
$$N = 0, \; P = 1$$

$$precision = \frac{TP}{TP + FP}$$

$$recall = \frac{TP}{TP + FN}$$

$$F1 = \frac{2 \times precision \times recall}{precision + recall}$$

$$accuracy = \frac{TP + TN}{TP + FN + TN + FP}$$

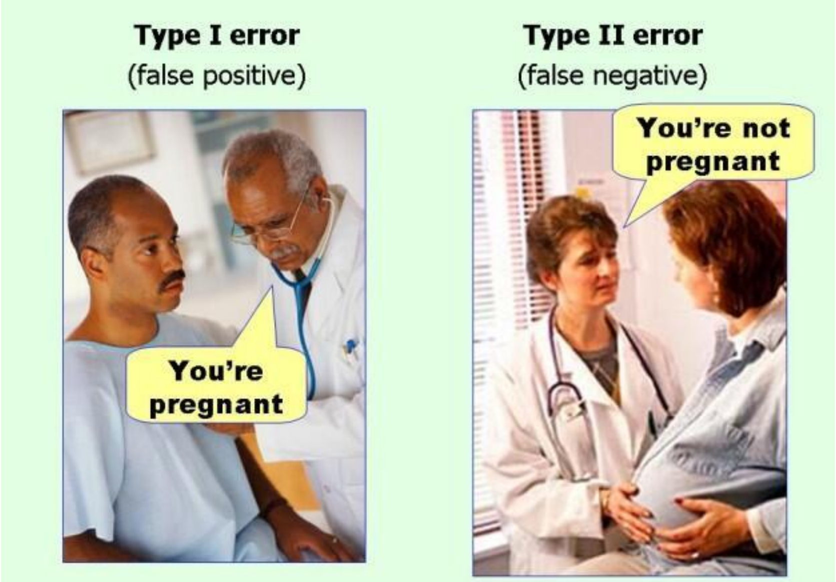



- Low precision, high recall: predict almost everything as positive
- High precision, low recall: predict positive when very sure
- If we want to find an optimal blend of precision and recall we can combine the two metrics using what is called the F1 score

## Confusion matrix

| | | Predicted Class | |
|---|---|---|---|
| | | No | Yes |
| Observed Class | No | TN | FP |
| | Yes | FN | TP |

| | |
|---|---|
| TN | True Negative |
| FP | False Positive |
| FN | False Negative |
| TP | True Positive |



# sklearn.metrics

## Model Evaluation

```
from sklearn.metrics import classification_report
from sklearn.metrics import confusion_matrix
```

```
y_true = [0, 0, 0, 1, 1, 2, 2, 2]
y_pred = [0, 0, 1, 1, 1, 2, 1, 2]
print(classification_report(y_true, y_pred))
```

```
              precision    recall  f1-score   support

           0       1.00      0.67      0.80         3
           1       0.50      1.00      0.67         2
           2       1.00      0.67      0.80         3

    accuracy                           0.75         8
   macro avg       0.83      0.78      0.76         8
weighted avg       0.88      0.75      0.77         8
```

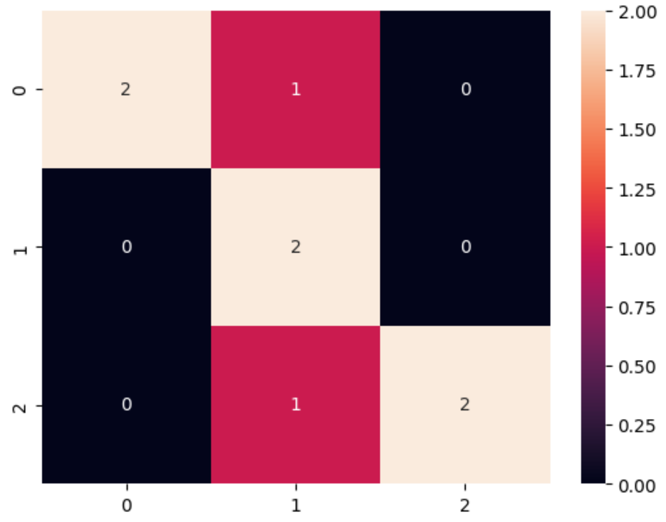
$$precision = \frac{TP}{TP + FP}$$

$$recall = \frac{TP}{TP + FN}$$

```
print(confusion_matrix(y_true, y_pred))
```

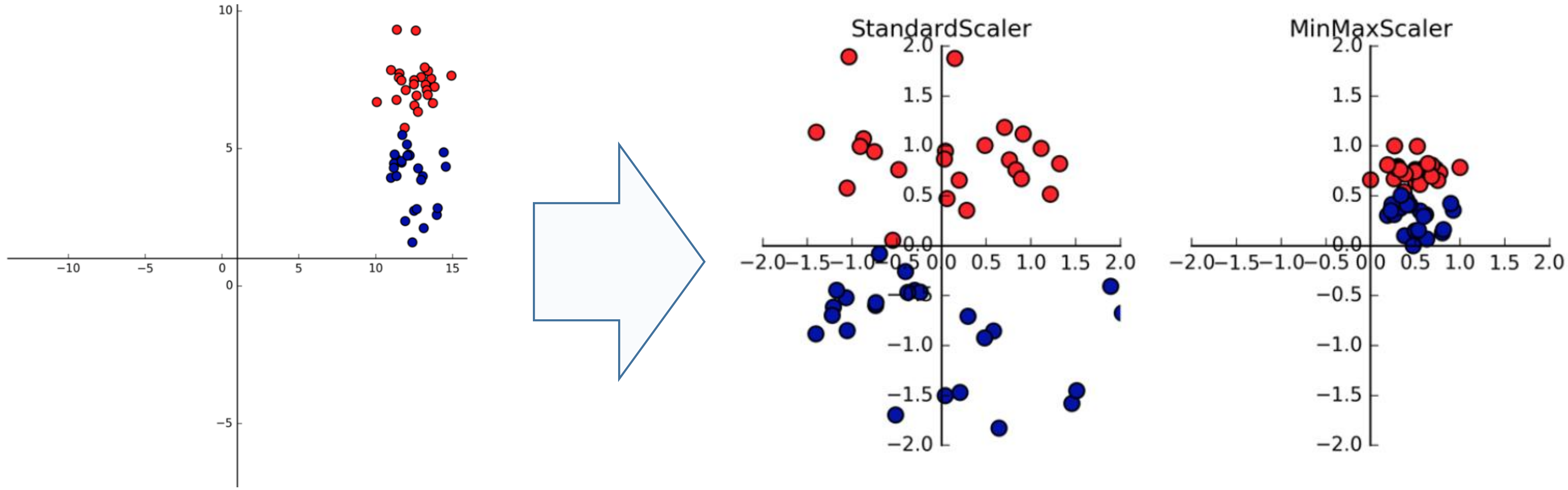
```
[[2 1 0]
 [0 2 0]
 [0 1 2]]
```

```
import seaborn as sns
sns.heatmap(confusion_matrix(y_true, y_pred), annot = True)
```



# sklearn.preprocessing

## Data Normalization




```
from sklearn.preprocessing import
StandardScaler
```

Standardize features by removing the mean and scaling to unit variance

```
from sklearn.preprocessing import
MinMaxScaler
```

Transform features by scaling each feature to a given range

**Usage**

- ✓ **fit(X)** – compute statistics (mean / min-max) from the data
- ✓ **transform(X)** – apply scaling to the data
- ✓ **fit_transform(X)** – do both in one step