Machine Learning with Python – SciKit-learn
Part 1

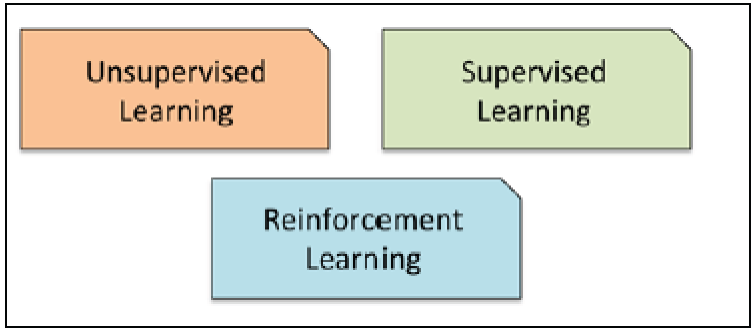
Ramesh Shankar

UConn

What is machine learning?

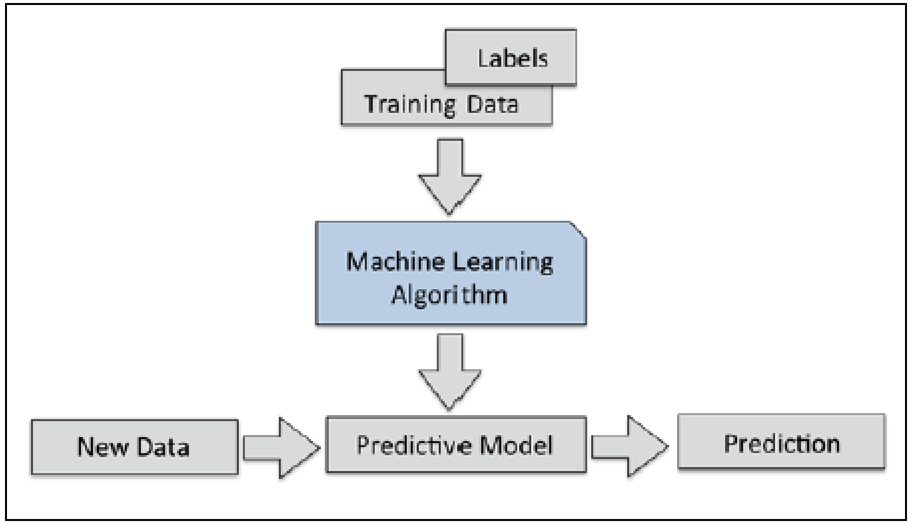
- Artificial Intelligence – use computer intelligence to make human-like judgments
- We have lots of data – structured and unstructured
- Machine Learning – subfield of Artificial Intelligence: self-learning algorithms to gain knowledge from data, make predictions
- Make data-driven decisions
- Applications: spam filters, voice recognition, web search, chess programs, perhaps self-driving cars?

Three types of machine learning



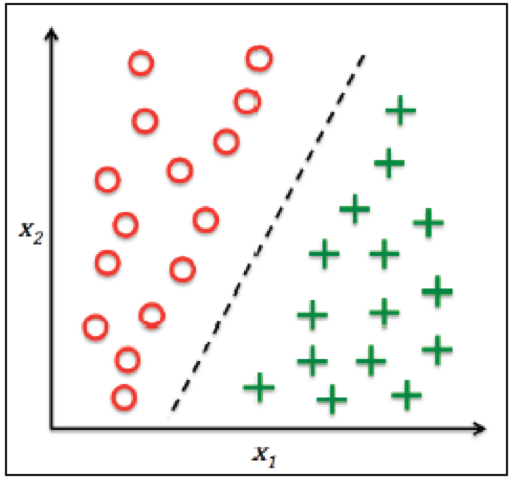
Supervised learning

e.g. classifier

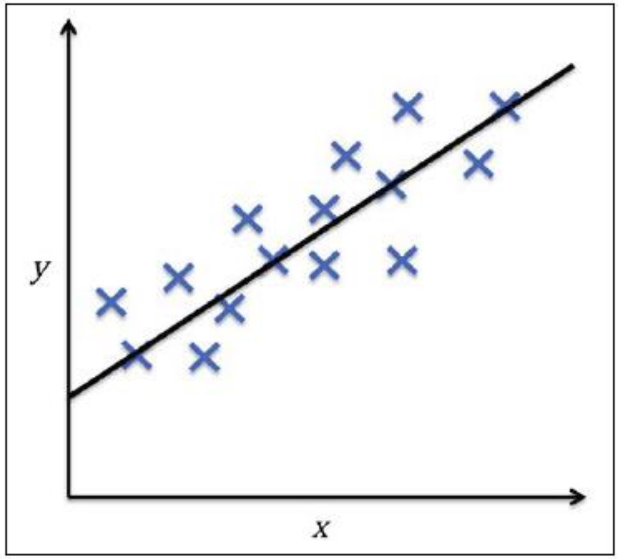


- Learn a model from labeled training data
- Make predictions about unseen or future data
- “supervised” – set of samples where desired output (labels) known
- E.g. Spam filter
- Classification: another name for supervised learning task with discrete class labels
  - Either binary (e.g. spam filter) or multiple classes (handwriting recognition)
- Regression: supervised learning task with continuous outcomes

Binary classification task with two variables:
- ML algorithm learns rule (separating line)



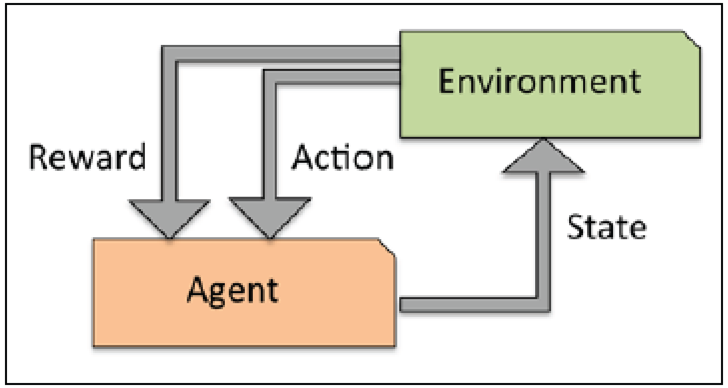
Supervised learning – regression



- Prediction of continuous outcomes
- Predictor (explanatory) variables – X-values
- Continuous response variable (outcome) – Y-value
- We try to find the relationship (line) – use it to predict unknown y-values for future X-values
- Linear or non-linear regression

Reinforcement learning

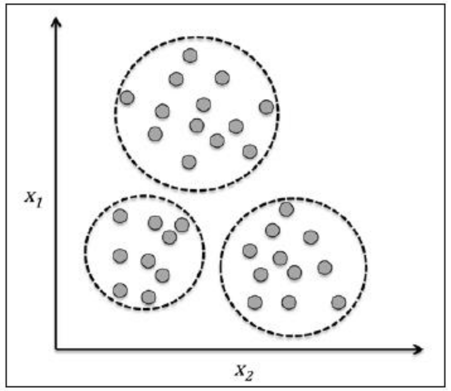
e.g. Chess



- Goal: “develop a system that improves its performance based on interactions with environment”
- Environment contains “reward signal”
- System is programmed to have a “reward function” to evaluate the reward signal
- Through repeated interaction (trial and error, as well as deliberate planning) with environment, system learns which actions maximize reward.

Unsupervised learning

e.g. clustering



- Supervised learning: you know the right answer beforehand
- Reinforcement learning: you know how to measure reward beforehand
- Unsupervised learning: unlabeled data
- Lets us explore structure of data and extract valuable insights without prior guidance
- E.g. clustering – find subgroups without prior knowledge
- Group of objects that share some similarity with each other
- Sometimes called “unsupervised classification”
- E.g. let marketers discover customer-groups

Choosing a classification algorithm

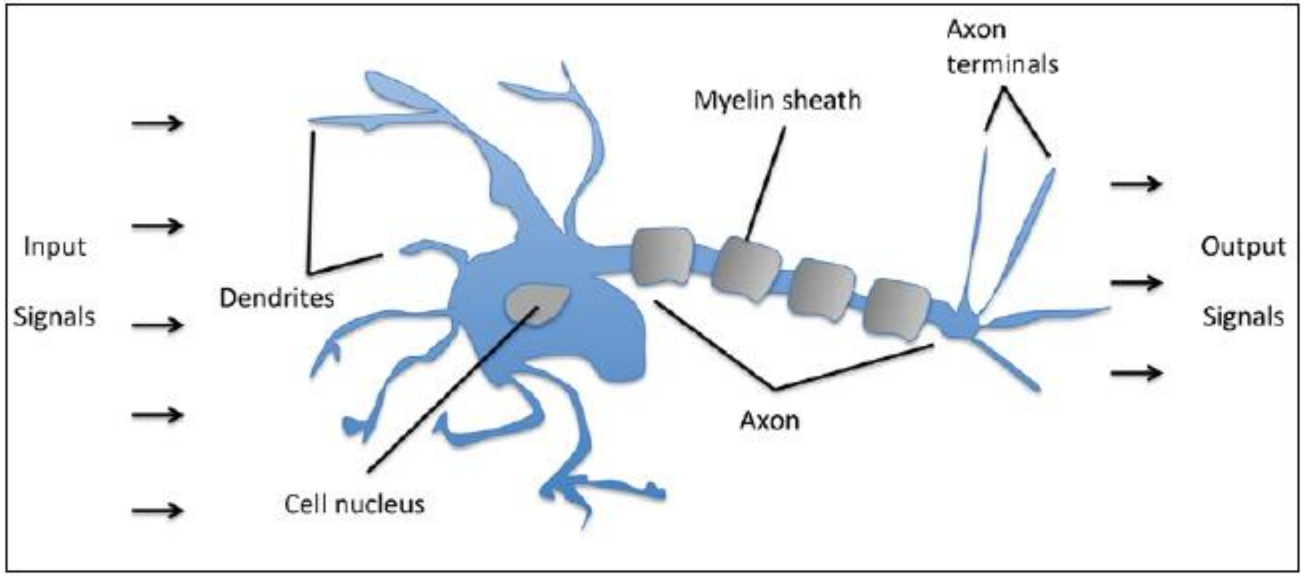
- Each algorithm has strengths and limitations
- No single algorithm works best on all tasks / scenarios
- Compare performance of a few algorithms – select best model for your problem
- Performance of a model
  - Computational power
  - Predictive power
  - Depends on underlying data available for learning

Main steps in training an ML algorithm

- Feature selection
- Choice of performance metric
- Choice of classifier and optimization algorithm
- Evaluating performance of model
- Tuning the algorithm

Perceptron

Brain cell:



All figures and code from:
"Python Machine Learning", Sebastian Raschka, Packt Publishing.

- Multiple inputs arrive
- Integrated into cell body
- If accumulated signal exceeds threshold, output signal generated

Perceptron

$$\phi(z): activation\ function$$
$$where\ z = \boldsymbol{w}^T . \boldsymbol{x}$$

$$\boldsymbol{w} = \begin{bmatrix} w_1 \\ \cdots \\ w_m \end{bmatrix}, \boldsymbol{x} = \begin{bmatrix} x_1 \\ \cdots \\ x_m \end{bmatrix}$$

- Learn weight coefficients $\boldsymbol{w}$
- Multiplied with input features $\boldsymbol{x}$
- Decide whether $\phi(z)$ = 1 or -1 (output)
- Can be used as a binary classifier

$$\phi(z) = \begin{cases} 1 & if\ z \geq \theta \\ -1 & otherwise \end{cases}$$

Perceptron

- Equivalently,

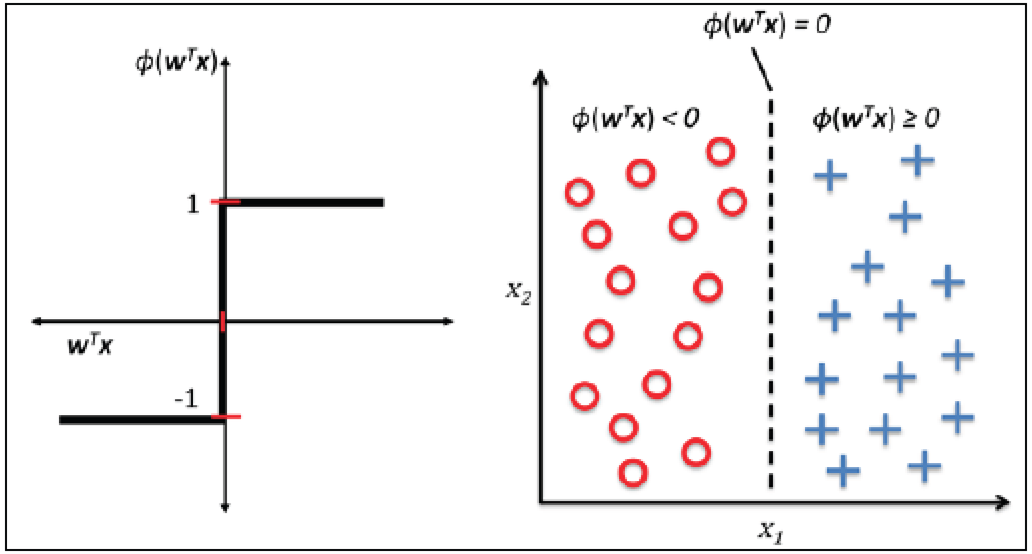
$$\phi(z) = \begin{cases} 1 & if\ z \geq 0 \\ -1 & otherwise \end{cases}$$

Φ(z) is also known as “y” or output

$$w_0 = -\theta \text{ and } x_0 = 1$$

Where $\boldsymbol{w} = \begin{bmatrix} w_0 \\ \cdots \\ w_m \end{bmatrix}, \boldsymbol{x} = \begin{bmatrix} x_0 \\ \cdots \\ x_m \end{bmatrix}$

$$z = w_0 x_0 + w_1 x_1 + \ldots + w_m x_m = \boldsymbol{w}^T \boldsymbol{x}$$



Algorithm

1. Initialize weights to 0 or small random numbers
2. Compute output y-hat: $\phi(z = w_0x_0+w_1x_1+w_2x_2+\ldots)$
3. Update weights:

$$w_{j(t)} = w_{j(t-1)}+ \eta(y_j-\hat{y}_{j(t)})x_j$$

4. Repeat step 2

   - until all rows of the training sample are completed

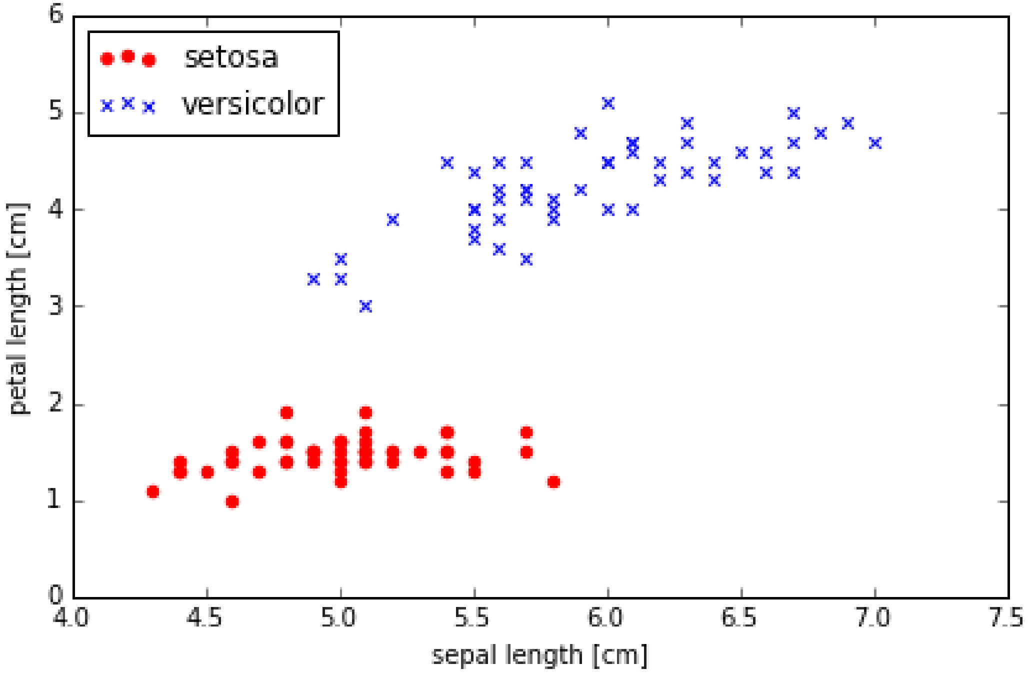
- That completes one iteration.
- Next iteration – repeat all of above steps

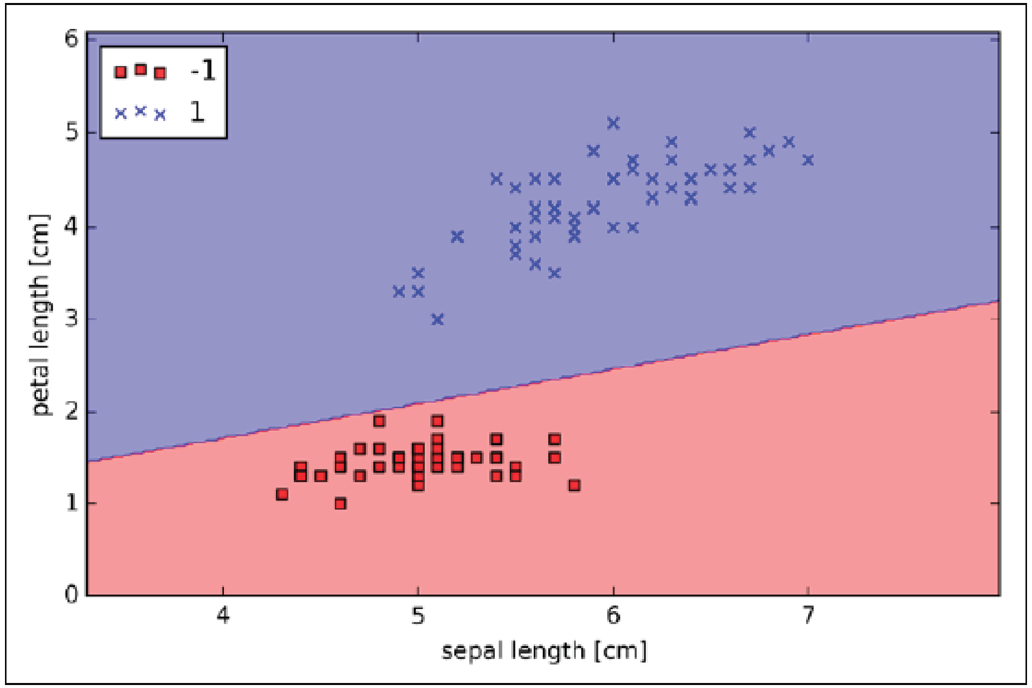
```
In [1]: import pandas as pd

In [2]: df = pd.read_csv('https://archive.ics.uci.edu/ml/'
   ...: 'machine-learning-databases/iris/iris.data',header=None)

In [3]: df.head()
Out[3]:
     0    1    2    3            4
0  5.1  3.5  1.4  0.2  Iris-setosa
1  4.9  3.0  1.4  0.2  Iris-setosa
2  4.7  3.2  1.3  0.2  Iris-setosa
3  4.6  3.1  1.5  0.2  Iris-setosa
4  5.0  3.6  1.4  0.2  Iris-setosa

In [4]: df.tail()
Out[4]:
       0    1    2    3               4
145  6.7  3.0  5.2  2.3  Iris-virginica
146  6.3  2.5  5.0  1.9  Iris-virginica
147  6.5  3.0  5.2  2.0  Iris-virginica
148  6.2  3.4  5.4  2.3  Iris-virginica
149  5.9  3.0  5.1  1.8  Iris-virginica
```


setosa
versicolor
petal length [cm]
sepal length [cm]
0
1
2
3
4
5
6
4.0
4.5
5.0
5.5
6.0
6.5
7.0
7.5


-1
1
petal length [cm]
sepal length [cm]
0
1
2
3
4
5
6
4
5
6
7

Perceptron model on Iris dataset

- X: petal length, petal width of 150 samples
- Y: class labels of flowers

```
In [3]: from sklearn import datasets
   ...: import numpy as np
   ...:
   ...: iris = datasets.load_iris()
   ...: X = iris.data[:, [2, 3]]
   ...: y = iris.target
   ...:
   ...: print('Class labels:', np.unique(y))
   ...:
('Class labels:', array([0, 1, 2]))
```

Y- values: Iris-Setosa, Iris-Versicolor, Iris-Virginica
Stored as (0,1,2)

```
In [14]: np.unique(y)
Out[14]: array([0, 1, 2])
```

Separate data into training and test datasets

- Check version, download appropriate method
- Split data into train (70%) and test (30%) data
- Test data – see how well model performs on unseen data

```
In [13]: from sklearn.model_selection import train_test_split

In [14]: X_train, X_test, y_train, y_test = train_test_split(X,y,test_size=0.3,random_state=0)
```

Standardizing data

- Subtract mean from each data point
- Divide by standard deviation
- Should be ideally normal distribution, but we ignore this requirement
- Usually required for most machine learning estimators in scikit-learn

```
from sklearn.preprocessing import StandardScaler

sc = StandardScaler()
sc.fit(X_train)
X_train_std = sc.transform(X_train)
X_test_std = sc.transform(X_test)
```

1. Load StandardScaler class from preprocessing module
2. Initialize new StandardScaler object – assign variable sc
3. Use fit method to estimate mean and standard deviation for each feature dimension
4. Use above fit to standardize training data
5. Use same fit to standardize test data as well

Train a perceptron model

- Multi-class classification

```
In [10]: from sklearn.linear_model import Perceptron

In [11]: ppn = Perceptron(max_iter=40, eta0=0.1, random_state=0)

In [12]: ppn.fit(X_train_std, y_train)
Out[12]:
Perceptron(alpha=0.0001, class_weight=None, eta0=0.1, fit_intercept=True,
      max_iter=40, n_iter=None, n_jobs=1, penalty=None, random_state=0,
      shuffle=True, tol=None, verbose=0, warm_start=False)
```

1. Load perceptron class from linear_model module
2. Initialize perceptron object
    - n_iter: number of iterations (epochs or passes over training set)
    - Eta0: **learning rate** – same as the “eta” in excel example
        - Too small – very slow; too large – may overshoot optimum
    - “random_state” parameter: shuffle initial training dataset after each epoch (a number, like “seed”)
3. Train the model using the “fit” method

Make predictions

```
In [25]: y_pred = ppn.predict(X_test_std)
    ...: print('Misclassified samples: %d' % (y_test != y_pred).sum())
    ...:
Misclassified samples: 4
```

- Perceptron misclassifies 4 out of 45 flower samples
- Misclassification error on test dataset is 8.9% (= 4/45)
- Accuracy = 1 – misclassification = 91.1%

```
In [26]: from sklearn.metrics import accuracy_score
    ...:
    ...: print('Accuracy: %.2f' % accuracy_score(y_test, y_pred))
    ...:
Accuracy: 0.91
```

- y_test are the true class labels
- y_pred are the predicted class labels

Preparing to plot the results

```
In [27]:     Z = classifier.predict(np.array([xx1.ravel(), xx2.ravel()]).T)
    ...:     Z = Z.reshape(xx1.shape)
    ...:     plt.contourf(xx1, xx2, Z, alpha=0.4, cmap=cmap)
    ...:     plt.xlim(xx1.min(), xx1.max())
    ...:     plt.ylim(xx2.min(), xx2.max())
    ...:
    ...:     for idx, cl in enumerate(np.unique(y)):
    ...:         plt.scatter(x=X[y == cl, 0], y=X[y == cl, 1],
    ...:                     alpha=0.8, c=cmap(idx),
    ...:                     marker=markers[idx], label=cl)
    ...:
    ...:     # highlight test samples
    ...:     if test_idx:
    ...:         # plot all samples
    ...:         if not versiontuple(np.__version__) >= versiontuple('1.9.0'):
    ...:             X_test, y_test = X[list(test_idx), :], y[list(test_idx)]
    ...:             warnings.warn('Please update to NumPy 1.9.0 or newer')
    ...:         else:
    ...:             X_test, y_test = X[test_idx, :], y[test_idx]
    ...:
    ...:         plt.scatter(X_test[:, 0],
    ...:                     X_test[:, 1],
    ...:                     c='',
    ...:                     alpha=1.0,
    ...:                     linewidths=1,
    ...:                     marker='o',
    ...:                     s=55, label='test set')
```

Preparing to plot the results

```
In [28]: X_combined_std = np.vstack((X_train_std, X_test_std))
    ...: y_combined = np.hstack((y_train, y_test))
    ...:
    ...: plot_decision_regions(X=X_combined_std, y=y_combined,
    ...:                       classifier=ppn, test_idx=range(105, 150))
    ...: plt.xlabel('petal length [standardized]')
    ...: plt.ylabel('petal width [standardized]')
    ...: plt.legend(loc='upper left')
    ...:
    ...: plt.tight_layout()
    ...: # plt.savefig('./figures/iris_perceptron_scikit.png', dpi=300)
    ...: plt.show()
    ...:
```

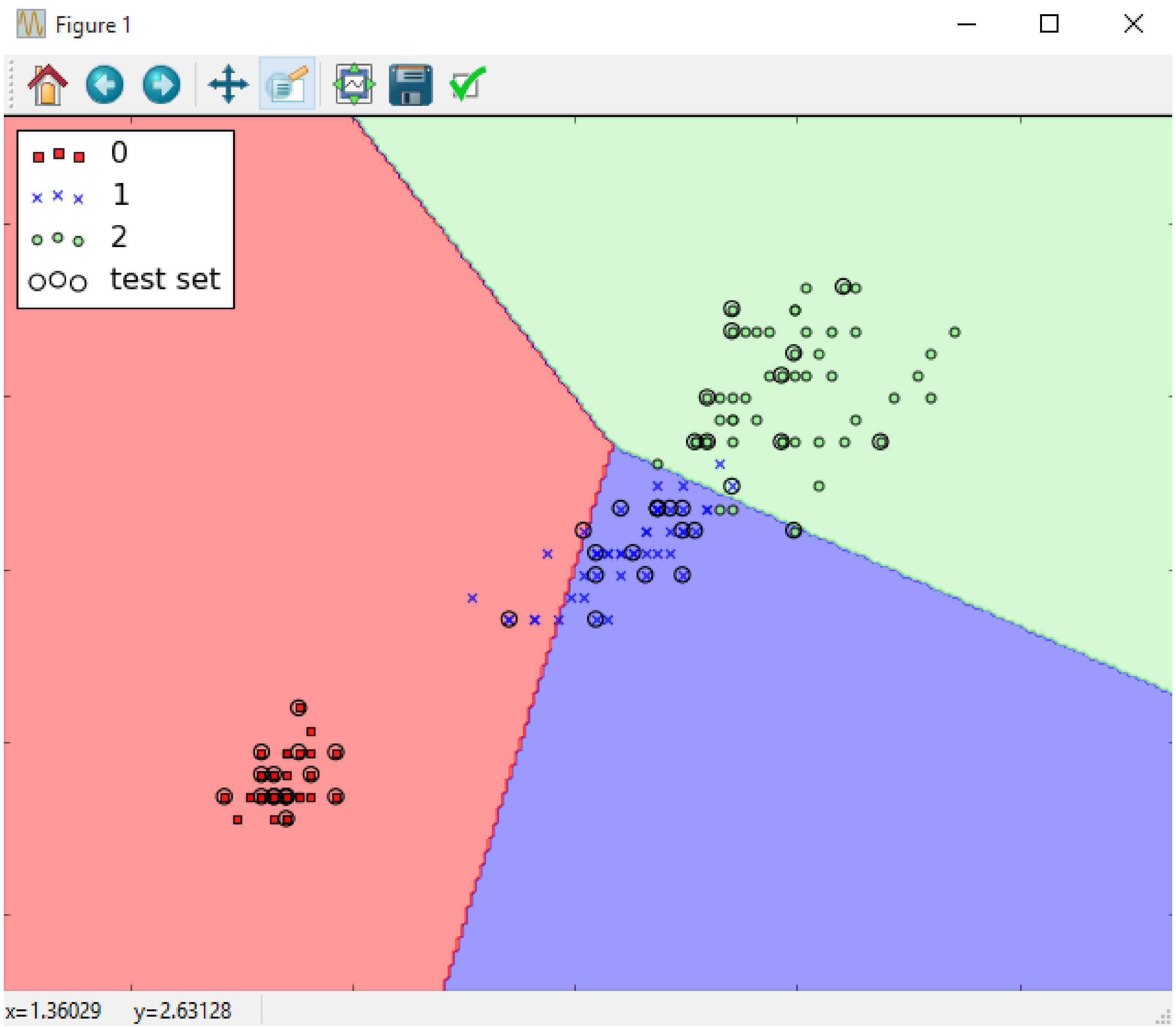


Figure 1
0
1
2
test set
x=1.36029 y=2.63128

		Predicted		
		1	2	3
Actual	1	1 16	4 0	7 0
	2	2 2	5 15	8 1
	3	3 0	6 1	9 10

1,5,9 – true positives (tp)
2,3 – false positives (fp) for 1
4,6 – false positives for 2
7,8 – false positives for 3
4,7 – false negatives (fn) for 1
2,8 – false negatives for 2
3,6 – false negatives for 3

Precision = tp/(tp+fp)
Recall = tp/(tp+fn)
F-measure: weighted harmonic mean of precision and recall

$$F_1 = 2 \cdot \frac{1}{\frac{1}{\mathrm{recall}} + \frac{1}{\mathrm{precision}}} = 2 \cdot \frac{\mathrm{precision} \cdot \mathrm{recall}}{\mathrm{precision} + \mathrm{recall}}.$$

```
from sklearn.metrics import classification_report, confusion_matrix
print(confusion_matrix(y_test,y_pred))
print(classification_report(y_test,y_pred))
```

```
[[16  0  0]
 [ 2 15  1]
 [ 0  1 10]]
             precision    recall  f1-score   support

          0       0.89      1.00      0.94        16
          1       0.94      0.83      0.88        18
          2       0.91      0.91      0.91        11

avg / total       0.91      0.91      0.91        45
```

http://scikit-learn.org/stable/modules/generated/sklearn.metrics.precision_recall_fscore_support.html

Kaggle loans data

```
import pandas as pd
import numpy as np
d = pd.read_csv("./kaggle/loans.csv")
# d['outcome'] = d.loan_status.astype('category').cat.codes
# d['gcode'] = d.Gender.astype('category').cat.codes
# d['ecode'] = d.education.astype('category').cat.codes
d.loc[(d.Gender=='male'),'gcode'] = 0
d.loc[(d.Gender=='female'),'gcode'] = 1

d.loan_status.unique()
d.loc[(d.loan_status=='PAIDOFF'),'outcome'] = 0
d.loc[(d.loan_status=='COLLECTION'),'outcome'] = 1
d.loc[(d.loan_status=='COLLECTION_PAIDOFF'),'outcome'] = 2

d.education.unique()
d.loc[(d.education=='college'),'ecode'] = 0
d.loc[(d.education=='High School or Below'),'ecode'] = 1
d.loc[(d.education=='Bechalor'),'ecode'] = 2
d.loc[(d.education=='Master or Above'),'ecode'] = 3
```

```python
y = d.outcome
X = d[['age','ecode','gcode']]

from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X,y,test_size=0.3, random_state=0)

from sklearn.preprocessing import StandardScaler
sc = StandardScaler()
sc.fit(X_train)
X_train_std = sc.transform(X_train)
X_test_std = sc.transform(X_test)

from sklearn.linear_model import Perceptron
ppn = Perceptron(max_iter=40,eta0=0.1,random_state=0)
ppn.fit(X_train_std,y_train)
y_pred = ppn.predict(X_test_std)

from sklearn.metrics import accuracy_score
print('Misclassified samples: %d' % (y_test != y_pred).sum())
print('Accuracy: %.2f' % accuracy_score(y_test, y_pred))
```

```
Misclassified samples: 100
Accuracy: 0.26
```