# ML.py Documentation

## CSC 448 Machine Learning Library

Author: Tim Hays

## Contents



## Using ML.py

1. Import ML
2. Create class instances and use functions as described on the following pages

# Perceptron

The `Perceptron` class implements the perceptron method of machine learning that consists of making a guess, checking whether a particular point is classified correctly, and adjusting if it isn't. The details behind the idea of a perceptron are available on Wikipedia: https://en.wikipedia.org/wiki/Perceptron

## Methods

### Perceptron()

| Parameter | Default | Description |
|---|---|---|
| **rate** | 0.01 | How much to adjust the guess when a point is misclassified |
| **niter** | 10 | Number of iterations |

### Perceptron.fit()

| Parameter | Default | Description |
|---|---|---|
| **X** | | Training input |
| **y** | | Outputs associated with training input |

Adjusts the class's weights based on the given training data, using the Perceptron algorithm

### Perceptron.net_input()

| Parameter | Default | Description |
|---|---|---|
| **X** | | Input to predict |

Returns the raw (not restricted to {-1, 1}) prediction of the given input(s)

### Perceptron.predict()

| Parameter | Default | Description |
|---|---|---|
| **X** | | Input to predict |

Returns the predicted label (either -1 or 1) of the given input(s)

## Example Usage

```
import pandas as pd, numpy as np
from ML import Perceptron
df = pd.read_csv('https://archive.ics.uci.edu/ml/machine-learning-'
                 'databases/iris/iris.data', header=None)
y = df.iloc[0:100, 4].values
y = np.where(y == 'Iris-setosa', -1, 1)

X = df.iloc[0:100, [0, 2]].values

pn = Perceptron(0.1, 10)
pn.fit(X, y)

print(pn.weight)
print(pn.errors)
```

Returns:

```
[-0.4  -0.68  1.82]
[2. 2. 3. 2. 1. 0. 0. 0. 0. 0.]
```

## Stochastic Soft SVM

The `StochasticSoftSVM` class implements the Stochastic Soft Support Vector Machine method of machine learning to determine which of two sets a point belongs to. It differs from Strong SVM in that Strong SVM requires the dataset to be separable, while Soft SVM does not. The goal with SVM in general is to maximize the *margin*, or distance from the dividing line to any of the input points. This class implements SVM using Stochastic Gradient Descent; it makes a guess, checks if a randomly-chosen point is classified correctly, and if it's not, the guess is adjusted. The amount to adjust the guess by is determined by Lambda, a parameter to the class constructor.

### Methods

#### StochasticSoftSVM()

| Parameter | Default | Description |
|---|---|---|
| **_lambda** | | How much to adjust the guess when a point is misclassified |
| **niter** | | Number of iterations |

#### StochasticSoftSVM.fit()

| Parameter | Default | Description |
|---|---|---|
| **X** | | Training input |
| **y** | | Outputs associated with training input |

Adjusts the class's weights based on the given training data, using Stochastic Gradient Descent

#### StochasticSoftSVM.net_input()

| Parameter | Default | Description |
|---|---|---|
| **X** | | Input to predict |

Returns the raw (not restricted to {-1, 1}) prediction of the given input(s)

#### StochasticSoftSVM.predict()

| Parameter | Default | Description |
|---|---|---|
| **X** | | Input to predict |

Returns the predicted label (either -1 or 1) of the given input(s)

### Example Usage

```
import pandas as pd, numpy as np
from ML import StochasticSoftSVM
df = pd.read_csv('https://archive.ics.uci.edu/ml/machine-learning-'
                 'databases/iris/iris.data', header=None)
y = df.iloc[0:100, 4].values
y = np.where(y == 'Iris-setosa', -1, 1)
X = df.iloc[0:100, [0, 2]].values

svm = StochasticSoftSVM(1, 10000)
svm.fit(X, y)

print(svm.weight)
```

Returns something like:

```
[-0.5  -1.4  2.9]
```

# Linear Regression

The `LinearRegression` class implements the linear regression method of machine learning with the least squares algorithm. The theory behind the method involves some linear algebra, but it boils down to the following equation, where $X$ and $y$ are the sample inputs and outputs respectively:

$$\vec{\beta} = (X^T X)^{-1} X^T \vec{y}$$

This gives $\vec{\beta}$, which is the least squares solution to

$$\vec{y} = \beta_0 + \beta_1 X_1 + \beta_2 X_2 + \ldots$$

Further details behind the idea of least squares are available on Wikipedia:
https://en.wikipedia.org/wiki/Ordinary_least_squares

## Methods

### LinearRegression.fit()

| Parameter | Default | Description |
|---|---|---|
| **X** | | Training input |
| **y** | | Outputs associated with training input |

Adjusts the class's weights based on the given training data, using the Least Squares algorithm

## Example Usage

```
import pandas as pd, numpy as np
from ML import LinearRegression
lr = LinearRegression()

X = np.array([0, 1, 2, 3, 4])
y = np.array([4, 6, 8, 10, 12])
lr.fit(X, y)
print(lr.weight)
```

Returns:

```
[4, 2]
```

which represents y = 4 + 2x

# Linear Regression for Polynomials

The `PolynomialLinReg` class implements the linear regression method of machine learning with the least squares algorithm to approximate d-degree polynomials. The theory behind the method involves some linear algebra, but it boils down to the following, where $x$ and $y$ are the sample inputs and outputs respectively:

$$X = \begin{bmatrix} 1 & x_1 & x_1^2 & & x_1^d \\ 1 & x_2 & x_2^2 & \cdots & x_2^d \\ 1 & x_3 & x_3^2 & & x_3^d \\ & \vdots & & \ddots & \vdots \\ 1 & x_n & x_n^2 & \cdots & x_n^d \end{bmatrix}$$

$$\vec{\beta} = (X^T X)^{-1} X^T \vec{y}$$

This gives $\vec{\beta}$, which is the least squares solution to

$$\vec{y} = \beta_0 + \beta_1 X_1 + \beta_2 X_2 + \ldots$$

Further details behind the idea of least squares are available on Wikipedia:
https://en.wikipedia.org/wiki/Polynomial_regression

## Methods

### PolynomialLinReg.fit()

| Parameter | Default | Description |
|---|---|---|
| **x** | | Training vector |
| **y** | | Outputs associated with training input |
| **dim** | | Order of the polynomial: 1 = linear, 2 = quadratic, etc |

Adjusts the class's weights based on the given training data, using the Least Squares algorithm

## Example Usage

```
import pandas as pd, numpy as np
from ML import PolynomialLinReg
plr = PolynomialLinReg()

x = np.array([0, 1, 2, 3, 4])
y = np.array([3, 6, 11, 18, 27])
plr.fit(x, y)
print(plr.weight)
```

Returns:

```
[3, 2, 1]
```

which represents $y = 3 + 2x + 1x^2$

# Threshold

The `Threshold` class implements an empirical risk minimization for thresholding functions. A thresholding function has an output value of 0 if the input is less than a predetermined value, *a,* and is 1 if the input is larger than or equal to *a*.

## Methods

### Threshold.ERM()

| Parameter | Default | Description |
|---|---|---|
| **x** | | Training vector |
| **y** | | Target vector |

Returns the threshold x-value to use; All points with a smaller x-value will have y=0, and all points with a higher x-value will have y=1.

## Example Usage

```
import pandas as pd, numpy as np
from ML import Threshold

thres = Threshold()

x = np.array([0, 1, 2, 2.1, 3, 4])
y = np.array([0, 0, 0, 1, 1, 1])

resp = thres.ERM(x, y)
print(resp)
```

Returns:

```
2.05
```

## Decision Stumps

The `DecisionStump` class implements an empirical risk minimization for decision stumps. A decision stump is a machine learning model consisting of a one-level decision tree. In other words, it asks if a specific feature of a given datapoint is above or below a certain threshold and classifies into one of two classes based only on that.

### Methods

#### DecisionStump.ERM()

| Parameter | Default | Description |
|---|---|---|
| **X** | | Training input |
| **y** | | Outputs associated with training input |
| **D** | | Probability vector |

Adjusts the class's weights based on the given training data, using the Least Squares algorithm

Some notes on the inputs:

X[i][j] = the i'th coordinate of the j'th input point.

y should be decreasing – that is, y should be any number of 1's followed by any number of -1's. It is valid for y to be all 1's or all -1's.

D[i] = the probability of the i'th point in X to appear.

### Example Usage

```
import pandas as pd, numpy as np
from ML import DecisionStump

ds = DecisionStump()

# Notice only the second coordinate changes, and y = -1 when X[1] > 2.5
X = np.array([[5, 5, 5, 5, 5], [0, 1, 2, 3, 4], [1, 1, 1, 1, 1]])
y = np.array([1, 1, 1, -1, -1])

resp = ds.ERM(X, y, [0.2]*5)
print(resp)
```

Returns:

```
(1, 2.5)
```

which represents the question "Is x[1] > 2.5?"

# Nearest Neighbor (1-NN)

The `NearestNeighbor` class finds the nearest neighbor to a point in n-dimensional space using a norm of choice.

## Methods

### NearestNeighbor()

| Parameter | Default | Description |
|---|---|---|
| **X** | | Data set. Note: X[i][j] = the i'th coordinate of the j'th input point |
| **norm** | 2 | Norm to use. For example, 2 = Euclidean (standard), 1 = Taxicab distance |

Initialize with a dataset and a specific norm to use

### NearestNeighbor.nearest()

| Parameter | Default | Description |
|---|---|---|
| **x** | | Point to find nearest neighbor to |

Finds the point in the input dataset closest to x, using the norm specified on initialization.

## Example Usage

```
import pandas as pd, numpy as np
from ML import NearestNeighbor

X = np.array([[1, 1], [0, 1.7]])
nn1 = NearestNeighbor(X, 1)
nn2 = NearestNeighbor(X, 2)

print(nn1.nearest([0, 0])) # Dist to [1, 1] is 2; dist to [0, 1.7] is 1.7
print(nn2.nearest([0, 0])) # Dist to [1, 1] is 1.414; dist to [0, 1.7] is 1.7
```

Returns:

```
[0  1.7]
[1  1]
```

## Other Functions

### plot_decision_regions()

| Parameter | Default | Description |
|---|---|---|
| **X** | | Input data, ideally from the same source that training data was taken from |
| **y** | | Output values associated with the input data |
| **classifier** | | The class used to classify the data. Requires a "predict" class method |
| **resolution** | 0.02 | How accurate the dividing line will be. Lower number = more accurate |

Given a classifier and a training set with two real inputs and one output with five or fewer possible values, this function will display which values will be given to points across the plane.

### scatter_plot()

| Parameter | Default | Description |
|---|---|---|
| **X** | | Input data, with X[:, 0] = x-values and X[:, 1] = y-values |

Uses matplotlib to create a simple 2-D scatter plot.

---

## Testing

See unittests.py for tests involving each module. This is also a good source of sample code, to see how each module is used.