Advanced Artificial Intelligence

Lab 06

Outline

- More details on the content in lecture 06
- Exercise

Details

- **Linear Regression Example**
- Cross Validation Example
- Gradient Descent

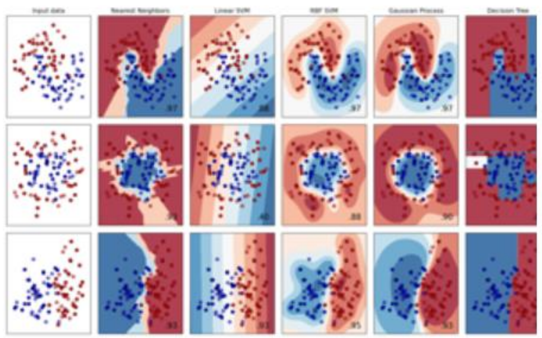
Sklearn library

The sklearn library contains a lot of efficient tools for machine learning modeling including classification, regression, clustering and dimensionality reduction

Classification

Identifying which category an object belongs to.

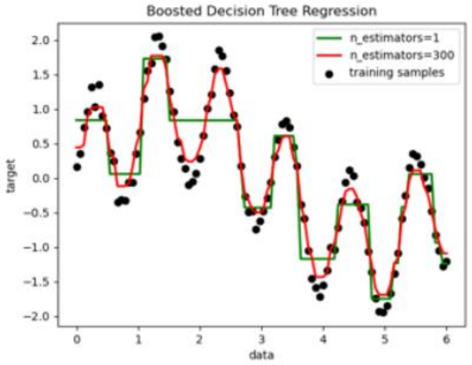
Applications: Spam detection, image recognition.
Algorithms: SVM, nearest neighbors, random forest, and more...



Regression

Predicting a continuous-valued attribute associated with an object.

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



Diabetes dataset

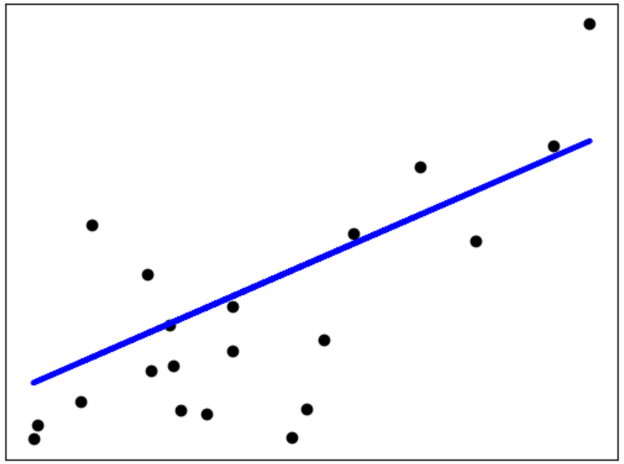
Number of Instances:	442
Number of Attributes:	First 10 columns are numeric predictive values
Target:	Column 11 is a quantitative measure of disease progression one year after baseline
Attribute Information:	• age age in years • sex • bmi body mass index • bp average blood pressure • s1 tc, total serum cholesterol • s2 ldl, low-density lipoproteins • s3 hdl, high-density lipoproteins • s4 tch, total cholesterol / HDL • s5 ltg, possibly log of serum triglycerides level • s6 glu, blood sugar level

Linear Model

$$\hat{y}(w, x) = w_0 + w_1 x_1 + \ldots + w_p x_p$$

we designate the vector $w = (w_1, \ldots, w_p)$ as `coef_` and w_0 as `intercept_`.

Linear Regression



LinearRegression fits a linear model with coefficients $w = (w_1, \ldots, w_p)$ to minimize the residual sum of squares between the observed targets in the dataset, and the targets predicted by the linear approximation. Mathematically it solves a problem of the form:

$$\min_{w} || X w - y||_2^2$$

```python
from sklearn import linear_model
l_reg = linear_model.LinearRegression()

X1 = [[0,0],[1,1],[2,2]]
y1 = [0,1,2]

l_reg.fit(X1,y1)
print(l_reg.coef_,l_reg.intercept_)
```

```
[0.5 0.5] 2.220446049250313e-16
```

Load the diabetes dataset

```
from sklearn import datasets
diabetes_X, diabetes_y = datasets.load_diabetes(return_X_y=True)
print(diabetes_X.shape,diabetes_y.shape)
print(diabetes_X[:3])
print(diabetes_y[:3])
```

```
(442, 10) (442,)
[[ 0.03807591  0.05068012  0.06169621  0.02187235 -0.0442235  -0.03482076
  -0.04340085 -0.00259226  0.01990842 -0.01764613]
 [-0.00188202 -0.04464164 -0.05147406 -0.02632783 -0.00844872 -0.01916334
   0.07441156 -0.03949338 -0.06832974 -0.09220405]
 [ 0.08529891  0.05068012  0.04445121 -0.00567061 -0.04559945 -0.03419447
  -0.03235593 -0.00259226  0.00286377 -0.02593034]]
[151.  75. 141.]
```

Split the diabetes dataset

```python
import numpy as np
# Use only one feature
diabetes_X = diabetes_X[:, np.newaxis, 2]

# Split the data into training/testing sets
diabetes_X_train = diabetes_X[:-20]
diabetes_X_test = diabetes_X[-20:]

# Split the targets into training/testing sets
diabetes_y_train = diabetes_y[:-20]
diabetes_y_test = diabetes_y[-20:]
```

```python
import numpy as np
print(diabetes_X[:3])
print('******')
print(diabetes_X[:3,2])
print('******')
print(diabetes_X[:3, np.newaxis, 2])
```

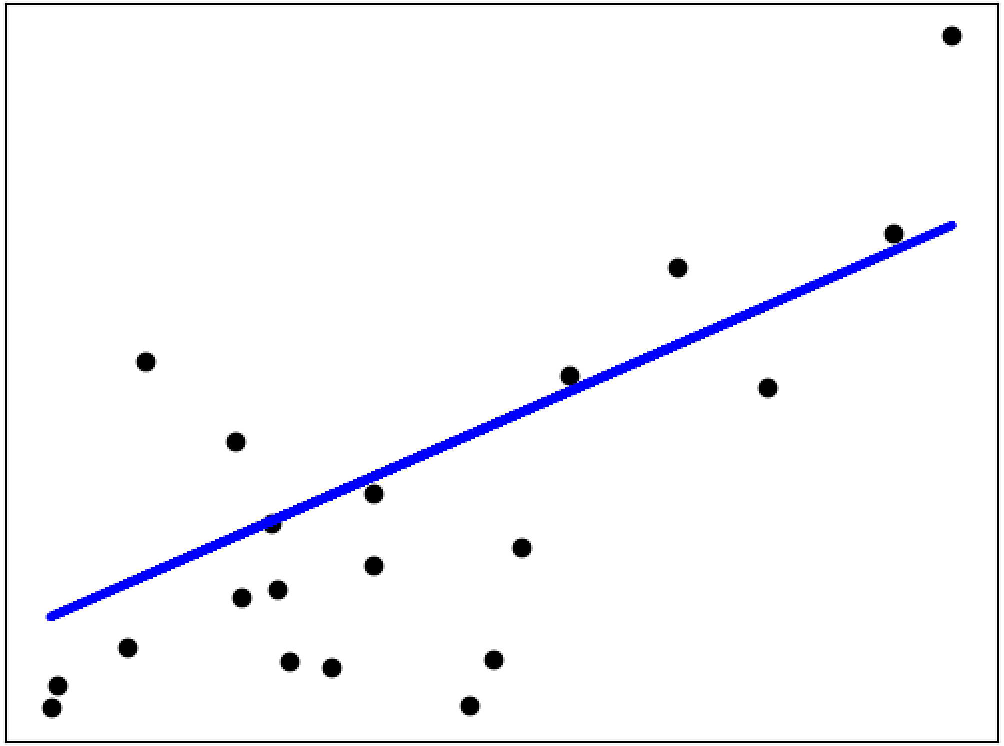
```
[[ 0.03807591  0.05068012  0.06169621  0.02187235 -0.0442235  -0.03482076
  -0.04340085 -0.00259226  0.01990842 -0.01764613]
 [-0.00188202 -0.04464164 -0.05147406 -0.02632783 -0.00844872 -0.01916334
   0.07441156 -0.03949338 -0.06832974 -0.09220405]
 [ 0.08529891  0.05068012  0.04445121 -0.00567061 -0.04559945 -0.03419447
  -0.03235593 -0.00259226  0.00286377 -0.02593034]]
******
[ 0.06169621 -0.05147406  0.04445121]
******
[[ 0.06169621]
 [-0.05147406]
 [ 0.04445121]]
```

Train the model and predict

```
from sklearn import linear_model
from sklearn.metrics import mean_squared_error

# Create linear regression object
regr = linear_model.LinearRegression()

# Train the model using the training sets
regr.fit(diabetes_X_train, diabetes_y_train)

# Make predictions using the testing set
diabetes_y_pred = regr.predict(diabetes_X_test)

print('Mean squared error: %.2f'
      % mean_squared_error(diabetes_y_test, diabetes_y_pred))
```

```
Mean squared error: 2548.07
```

1-Dimension feature



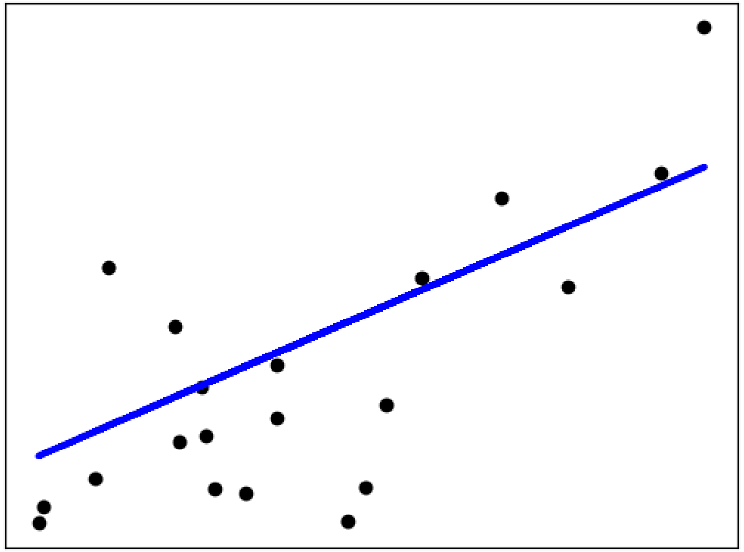
We only use the one feature of the diabetes dataset, in order to illustrate the data points within the two-dimensional plot.

[Linear Regression Example — scikit-learn 1.0.1 documentation]

Visualize the model

```
print('Coefficients:', regr.coef_,'intercept:',regr.intercept_)
import matplotlib.pyplot as plt
plt.scatter(diabetes_X_test, diabetes_y_test,  color='black')
plt.plot(diabetes_X_test, diabetes_y_pred, color='blue', linewidth=3)

plt.xticks(())
plt.yticks(())

plt.show()
```

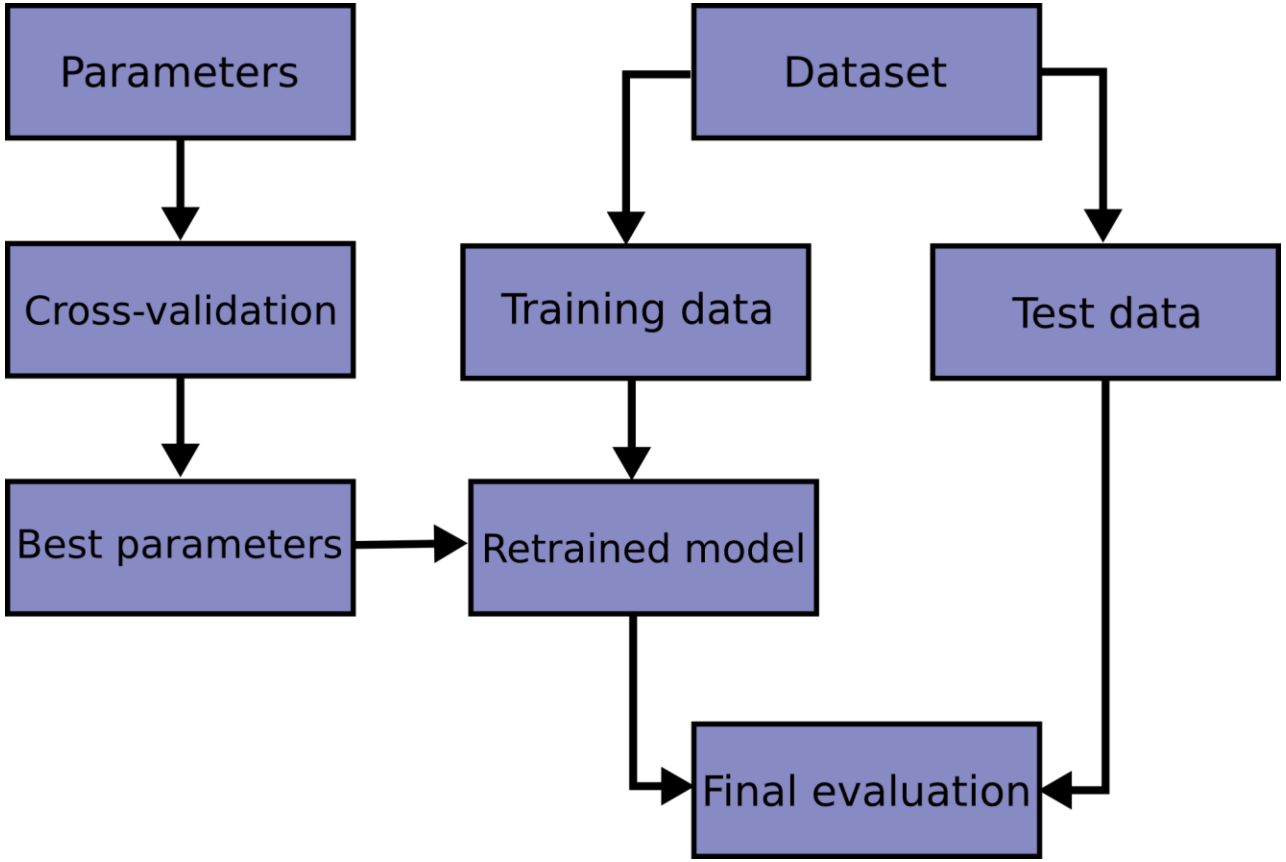
```
Coefficients: [938.23786125] intercept: 152.91886182616167
```



Details

- Linear Regression Example
- **Cross Validation Example**
- Gradient Descent

Cross Validation



We use CV to select best hyperparameters

Load the dataset

Let's load the iris data set to fit a linear support vector machine on it

```
>>> import numpy as np
>>> from sklearn.model_selection import train_test_split
>>> from sklearn import datasets
>>> from sklearn import svm

>>> X, y = datasets.load_iris(return_X_y=True)
>>> X.shape, y.shape
((150, 4), (150,))
```

Split Dataset, Train and Test

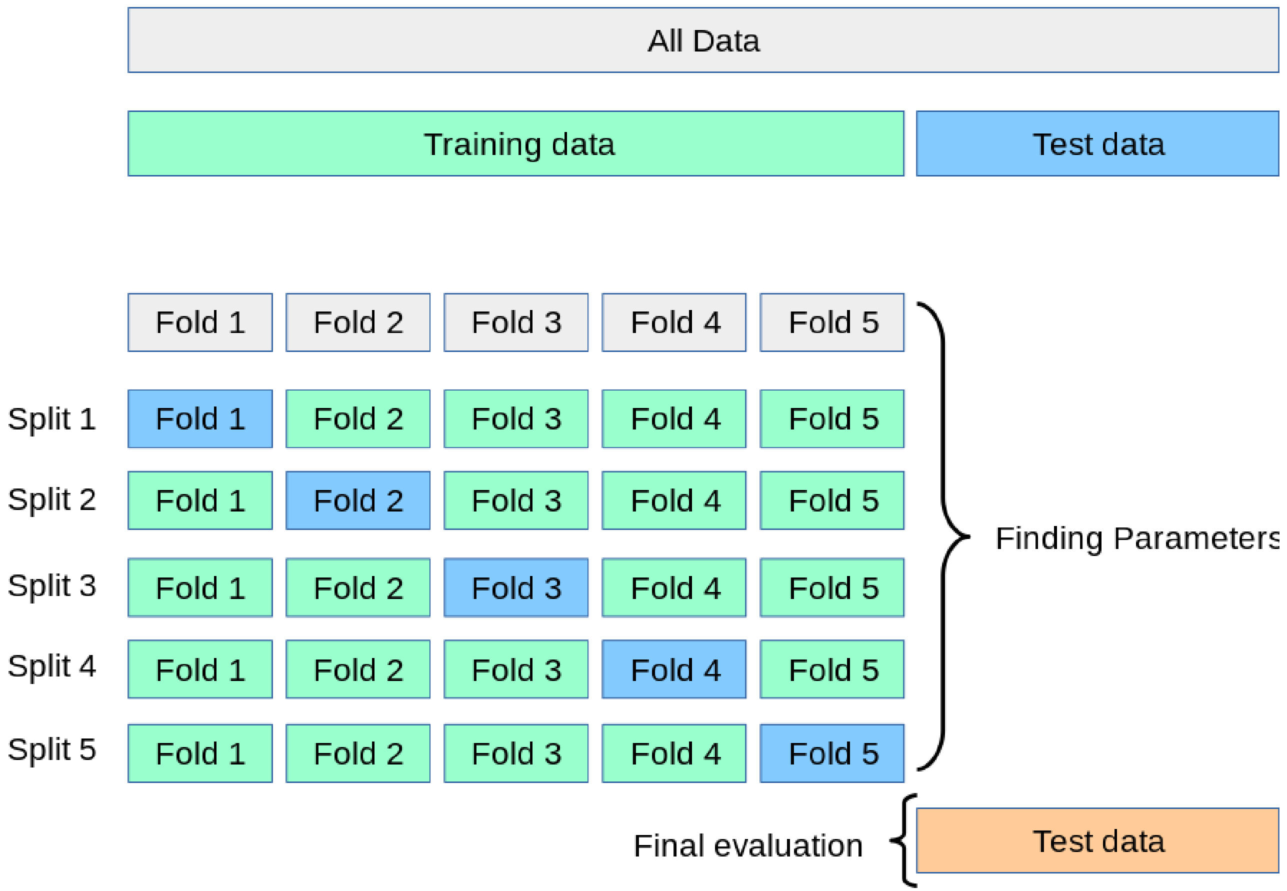
```
>>> X_train, X_test, y_train, y_test = train_test_split(
...     X, y, test_size=0.4, random_state=0)

>>> X_train.shape, y_train.shape
((90, 4), (90,))
>>> X_test.shape, y_test.shape
((60, 4), (60,))

>>> clf = svm.SVC(kernel='linear', C=1).fit(X_train, y_train)
>>> clf.score(X_test, y_test)
0.96...
```

We can now quickly sample a training set while holding out 40% of the data for testing (evaluating) our classifier

Validation Set

By partitioning the available data into three sets, we drastically reduce the number of samples which can be used for learning the model

K-Fold cross validation



- A model is trained using k−1 of the folds as training data;
- the resulting model is validated on the remaining part of the data

5-fold CV

```
>>> from sklearn.model_selection import cross_val_score
>>> clf = svm.SVC(kernel='linear', C=1, random_state=42)
>>> scores = cross_val_score(clf, X, y, cv=5)
>>> scores
array([0.96..., 1.  , 0.96..., 0.96..., 1.  ])
```

Estimate the accuracy by splitting the data, fitting a model, and computing the score for 5 consecutive times (with different splits each time)

Leave One Out (LOO)

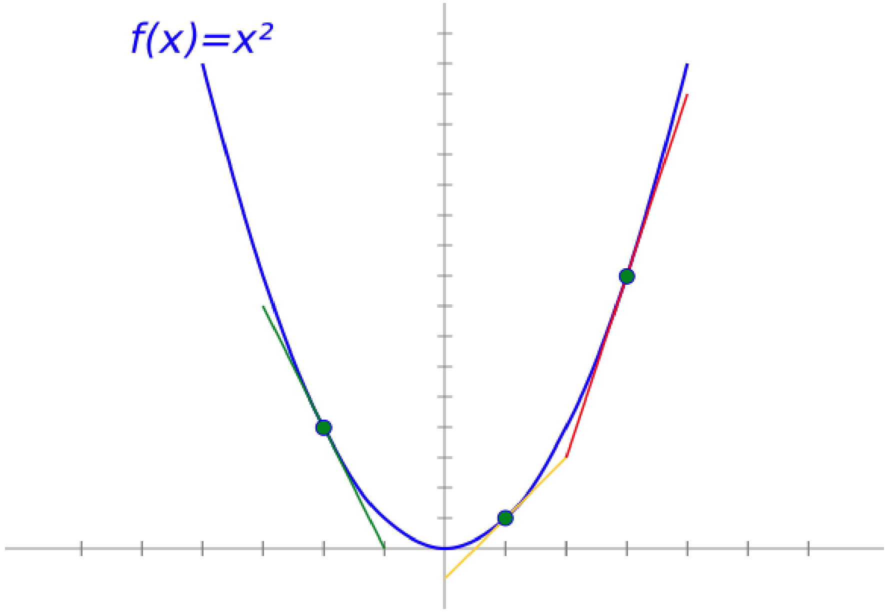
```
>>> from sklearn.model_selection import LeaveOneOut

>>> X = [1, 2, 3, 4]
>>> loo = LeaveOneOut()
>>> for train, test in loo.split(X):
...     print("%s %s" % (train, test))
[1 2 3] [0]
[0 2 3] [1]
[0 1 3] [2]
[0 1 2] [3]
```

Each learning set is created by taking all the samples except one, the test set being the sample left out.

Details

- Linear Regression Example
- Cross Validation Example
- **Gradient Descent**

$f(x) = x*x$



$f(x) = x^2 \quad f'(x) = 2x \qquad step = -0.1$

```python
def f(x):
    return x*2

def df(x):
    return 2*x

step = -0.1

x = 3

for i in range(20):
    print(x)
    x = df(x)*step+x
```

```
3
2.4
1.92
1.536
1.2288000000000001
0.9830400000000001
0.7864320000000001
0.6291456000000001
0.5033164800000001
0.40265318400000005
0.32212254720000005
0.25769803776000005
0.20615843020800004
0.16492674416640002
0.13194139533312002
0.10555311626649602
0.08444249301319681
0.06755399441055746
0.05404319552844596
0.04323455642275677
```

$f(x, y) = 2x^2 + y^2$

$$\frac{\partial f}{\partial x} = 4x$$
$$\frac{\partial f}{\partial y} = 2y$$

$$\nabla f(x,y) = \left[\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}\right]^T$$
$$\nabla f(x,y) = [4x, 2y]^T$$

```
def f(x,y):
    return 2*x*x+y*y

def dx(x):
    return 4*x

def dy(y):
    return 2*y

step = -0.1

x = 3
y = 5

for i in range(20):
    print(f(x,y))
    x = dx(x)*step+x
    y = dy(y)*step+y
```

```
43
22.479999999999997
12.5728
7.393408
4.496634879999999
2.7931936768
1.7571690004480003
1.1136171773132804
0.7087654396100611
0.4521880419570812З
0.2888884846709232
0.1847043598040117
0.11814445293583098
0.07558856843698666
0.04836808648057122
0.03095248031268949
0.019808473187565876
0.012677021723522221
0.008113149501107041
0.005192363696007523
```

n-variable function

In the same way, if we get a function with 4 variables, we would get a gradient vector with 4 partial derivatives.

Generally, an n-variable function results in an n-dimensional gradient vector.

$$f(x_1, x_2, \ldots, x_n) = \nabla f = \left[\frac{\partial}{\partial x_1}, \frac{\partial}{\partial x_2}, \ldots, \frac{\partial}{\partial x_n}\right]$$

Performance measure

$$\underbrace{Err = \frac{1}{N}\sum_{\mu}(\underset{\text{Ground truth}}{\overline{y^{\mu}}} - \underbrace{(W_1 X^{\mu} + W_0)}_{\text{Predicted values}})^2}_{\text{MSE}}$$

Partial derivatives

$$Err = \underbrace{\frac{1}{N}\sum_{\mu}(\overbrace{y^{\mu}}^{\text{Ground truth}} - (\underbrace{W_1 X^{\mu} + W_0}_{\text{Predicted values}}))^2}_{\text{MSE}}$$

$$\frac{\partial}{\partial W_0} = -\frac{2}{N}\sum_{\mu}(y^{\mu} - (W_1 X^{\mu} + W_0))$$

$$\frac{\partial}{\partial W_1} = -\frac{2}{N}\sum_{\mu} X \times (y^{\mu} - (W_1 \times X^{\mu} + W_0))$$

The gradient

$$\nabla Err = \left[\frac{\partial}{\partial W_0}, \frac{\partial}{\partial W_1}\right]^T$$

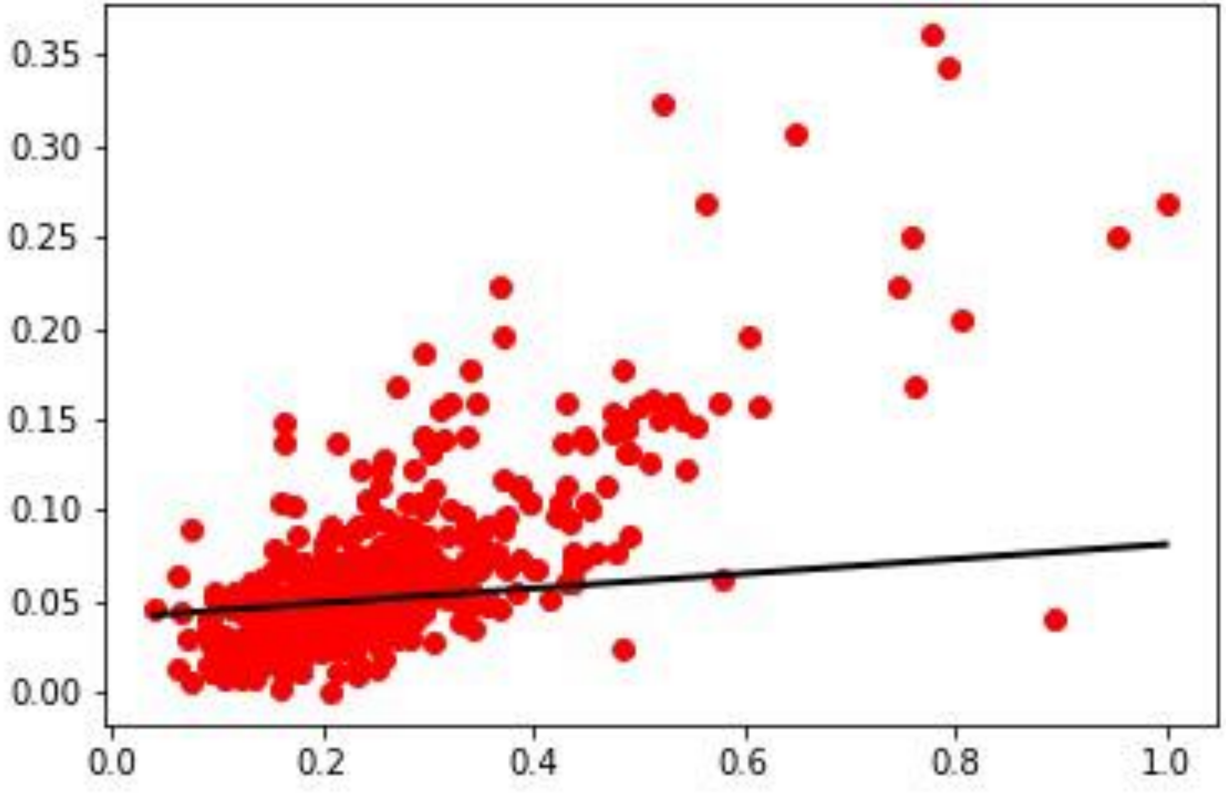
$$W0 = W0 - \eta(\frac{\partial}{\partial W0})$$

$$W1 = W1 - \eta(\frac{\partial}{\partial W1})$$

The gradient

```
for epoch in range(total_epochs):
    for X_batch, y_batch in next_batch(Xs_train, Ys_train, batch_size=batch_size):

        # linearly combine input and weights
        train_pred = W0 + np.dot(X_batch, W1)

        # calculate the SSE between predicted and true values
        train_err = mean_squared_error(y_batch, train_pred)

        # calculate the gradients with respect to W0 and W1
        DW0 = -(2/batch_size) * sum(y_batch.squeeze() - train_pred.squeeze())
        DW1 = -(2/batch_size) * sum(X_batch.squeeze() * (y_batch.squeeze() - train_pred.squeeze()))

        # update W0 and W1 in the opposite direction to the gradient
        W0 = W0 - lr * DW0
        W1 = W1 - lr * DW1
```

The gradient



Exercise

1. Compare linear regression and SVM using the sklearn library

2. Try to implement linear regression from scratch with Python