Supervised Learning: Multivariable Regressor & Classifiers

AAA-Python Edition

Plan

- 1- Preprocessing Data
- 2- A single variable regressor
- 3- A multivariable regressor
- 4- Regularization
- 5- Logistic Regression Classifier
- 6- Confusion matrix

Introduction

- Before going further, we have to focus in one important task: **preprocessing data**.
- Before starting the learning (and the predicting) process, the used **data must be prepared** or **transformed**.

Learning

Observed Data **preprocessed**

Predict

New Data **preprocessed**

- The different processes that can be applied to the data are:
  - Binarization
  - Mean removal
  - Scaling
  - Normalization

1- Preprocessing Data

AIM

Binarization

Define a **threshold**

Values **> threshold** ==> 1

Values **<= threshold** ==> 0

myD

	Feature1
s1	-0.626009
s2	1.556353
s3	0.450549
s4	0.898856
s5	-0.191403

```
# the value of the threshold
th = 1
# instaciate a Binarizer object with that threshold
myBin = preprocessing.Binarizer(threshold=th)
# transform the feature's values with that Binarizer
myD_B =myBin.transform(myD.Feature1.values.reshape(-1,1))
```

```
array([[0.],
       [1.],
       [0.],
       [0.],
       [0.]])
```

Mean removal

Calculate the "mean" ==> Values - mean

```
myD_mr = preprocessing.scale(myD,with_std=False)
```

Previous mean== **0.502482**

New mean: $new\,mean \simeq 0$

```
array([[-0.11428357],
       [-0.34343912],
       [ 0.13638221],
       [ 1.27876145],
       [-0.95742096]])
```

[By Amina Delali]

Min Max Scaling

Features with different scales of values → Select a range → Determine min and max for each feature → Apply the formula

myD

	Feature1	Feature2	Feature3
s1	-0.626009	100	9
s2	1.556353	1000	7
s3	0.450590	500	-3
s4	0.898856	200	1
s5	-0.191403	80	55

```
# initiate a minmax scaler instance
myScaler=preprocessing.MinMaxScaler(feature_range=(0, 1))
# scale the features in myD
myD_s = myScaler.fit_transform(myD)
```

```
array([[0.        , 0.02173913, 0.20689655],
       [1.        , 1.        , 0.17241379],
       [0.49331825, 0.45652174, 0.        ],
       [0.6987223 , 0.13043478, 0.06896552],
       [0.19914478, 0.        , 1.        ]])
```

The transformation is done using this formula:

```
X_std = (X - X.min(axis=0)) / (X.max(axis=0) - X.min(axis=0))
X_scaled = X_std * (max - min) + min
```

(from Google colab help)

```
min, max = feature_range
```

Ex: X[s1,Feature2]_std=(100 -**80**)/(**1000**-80)= 0.02173913
X[s1,Feature2]_scaled=0.02173913*(**1-0**)+**0**= **0.02173913**

[By Amina Delali]

1- Preprocessing Data

Normalization

Least Absolute Deviations: L1 → **Sum** of **absolute** features values for each sample row will be == **1**

```python
# Normalize myD using L1 normalization
myD_l1 = preprocessing.normalize(myD, norm='l1')
```

|-5.71040582e-03|+
|9.12192288e-01|+
| 8.20973059e-02|==**1**

```
array([[-5.71040582e-03,  9.12192288e-01,  8.20973059e-02],
       [ 1.54314927e-03,  9.91516237e-01,  6.94061366e-03],
       [ 8.95003420e-04,  9.93146120e-01, -5.95887672e-03],
       [ 4.45201136e-03,  9.90595014e-01,  4.95297507e-03],
       [-1.41579269e-03,  5.91753604e-01,  4.06830603e-01]])
```

Least Squares: L2 → **Sum** of **the square of** features values for each sample row will be == **1**

```python
# Normalize myD using L2 normalization
myD_l2 = preprocessing.normalize(myD, norm='l2')
```

$(-6.23476844e\text{-}03)^2+$
$(9.95955081e\text{-}01)^2+$
$(8.96359573e\text{-}02)^2==\mathbf{1}$

```
array([[-6.23476844e-03,  9.95955081e-01,  8.96359573e-02],
       [ 1.55631299e-03,  9.99974290e-01,  6.99982003e-03],
       [ 9.01163413e-04,  9.99981594e-01, -5.99988957e-03],
       [ 4.49417844e-03,  9.99977401e-01,  4.99988701e-03],
       [-1.97154737e-03,  8.24040323e-01,  5.66527722e-01]])
```

[By Amina Delali]

2- A Single Variable Regressor

Training and testing set

Learning : regression (with 1 feature)

build

Mathematical model with parameters

Fit : determine the value of parameters

Observed **Labeled** Data **X: described by 1 feature** and the labels **y.**
The **data** is **split** into **: training** and **testing set**
Parameters determined using **training** set
Prediction is **made** using the **testing** set
Compute the **error** using **predicted** and **test labels**

```python
from sklearn.model_selection import train_test_split
# srpliting the data into test and training sets
x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.2)
```

```python
# create an instance of a Linear regressor
myModel = linear_model.LinearRegression()
# train the model using the training sets
myModel.fit(x_train, y_train)
```

```python
# predict using testing set
y_test_pred = myModel.predict(x_test)
```

```python
# Compute mean absolute error
print("Mean absolute error =", sm.mean_absolute_error(y_test,y_test_pred))
# Compute mean squared error
print("Mean squared error =", sm.mean_squared_error(y_test,y_test_pred))
```

Mean of absolute differences between test labels and predicted labels

Mean of square of the differences between test labels and predicted labels

[By Amina Delali]

2- A Single Variable Regressor

Loading and saving models

Saving the model

```
# creating a new file
f= open("myModel.okl", 'wb')
# saving the model to that file
pickle.dump(myModel, f)
```

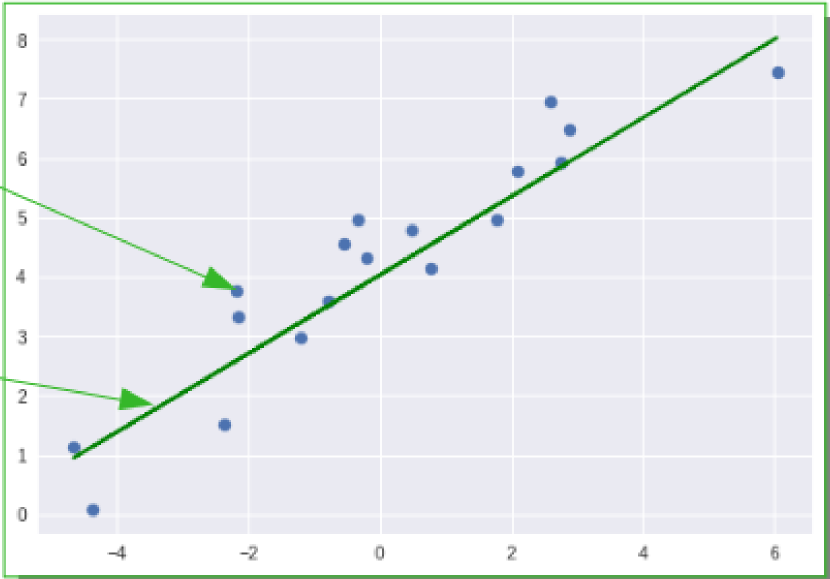
Loading the model

```
# opening (loading) the file
f= open("myModel.okl", 'rb')
#loading the model from that file
myLModel = pickle.load(f)
```

```
# try other splits
x1,x2,y1,y2= train_test_split(x, y, test_size=0.3)
```

Real (original) values

Predicted (estimated) values



30% of the values will be used for test (x2 , y2) and 70% should be used for training (x1,y1)
Since the model is Already trained, we will just use the data to test the model

[By Amina Delali]

3- A Multivariable Regressor

Multiple features

Learning : regression (with multiple features)

- build

Mathematical model with parameters

Fit : determine the value of parameters

Observed **Labeled** Data **X:**
- described by **1 feature,** then **new features** are **generated:** like **polynomial features**
- described by **multiple features:** they are **used** as **they are**
- described by **multiple features,** and **new features** are **generated**

The data here is described by **3 features**

	col0	col1	col2	col3
0	2.06	3.48	7.21	15.69
1	6.37	3.01	7.27	15.34
2	1.18	1.20	5.42	0.66
3	7.37	3.81	-1.95	38.37
4	6.16	1.39	7.39	9.96

```
# move to AAA-Ped-Week4 folder
%cd AAA-Ped-Week4
# loading the data:
# "col0""col1""col2" are x features values: 3 features
# and "col3" are the labels y
myD3 = pd.read_csv("A3P-w4-data_multivar_regr.txt",header=-1,prefix="col")
```

[By Amina Delali]

3- A Multivariable Regressor

Simple Linear Regression (with **multiple** features)

Learning : Simple linear regression (with multiple features)

build

Mathematical model==

$$y = a_1 * x_1 + a_2 * x_2 + ... + a_n * x_n + b$$

with **n+1** parameters: $\mathbf{a_i}$ and **b**

Fit : determine the value of $\mathbf{a_i}$ and **b**

Observed **Labeled** Data: **X** described by **multiple features**
the **labels: y**

```python
# Features values
myX= myD3.iloc[:,:3]
# labels
myY = myD3.iloc[:,3]
myY = myY.values.reshape(-1,1)
# train again the linear regression model
myModel.fit(myX,myY)
print("ai are in ",myModel.coef_)
print ("b is in ",myModel.intercept_)
```

Selecting the **3** first **columns** corresponding to **3** features

The **fourth** column is the **labels** column

```
b is in  [5.77068196]
```

b

```
ai are in  [[ 1.97288257  3.91615283 -1.75291766]]
```

$\mathbf{a_1, a_2,}$ **and** $\mathbf{a_3}$

[By Amina Delali]

AIM

3- A Multivariable Regressor

Polynomial Basis Function (with **multiple** features)

Learning: Linear regression (with 3 features, order == 2)

build

Mathematical model==

$$y= a_1*x_1+a_2*x_2+ a_3*x_3+a_4*x_1^2+ a_5*x_1*x_2 +a_6*x_1*x_3+a_7*x_2^2+a_8*x_2*x_3+a_9*x_3^2+ b$$

with **10** parameters: $\mathbf{a_i}$ and **b**

Fit : determine the value of $\mathbf{a_i}$ and **b**

Observed **Labeled** Data: **X** described by **multiple features + Polynomial features are generated**
the **labels: y**

```
from sklearn.preprocessing import PolynomialFeatures
# polynomial features generator of degree 2
polynomial = PolynomialFeatures(degree=2, include_bias=False)
# generate the new features
x_poly = polynomial.fit_transform(myX)

# fit the model
myModel.fit(x_poly,myY)
```

```
ai are in  [[ 2.24816066e+00  4.01398972e+00 -1.95742341e+00 -3.11506706e-02
   -3.60516775e-02  3.01335037e-02 -5.82113466e-03  2.13183791e-02
   -4.29337992e-04]]
b is in  [5.67788315]
```

[By Amina Delali]

4- Regularization

AIM

Introduction

Mathematical model **overfit** data ==**Fits** very well to the training set. But, **doesn’t perform** well on the **testing** or the **new data** sets

Introduce penalties to the model

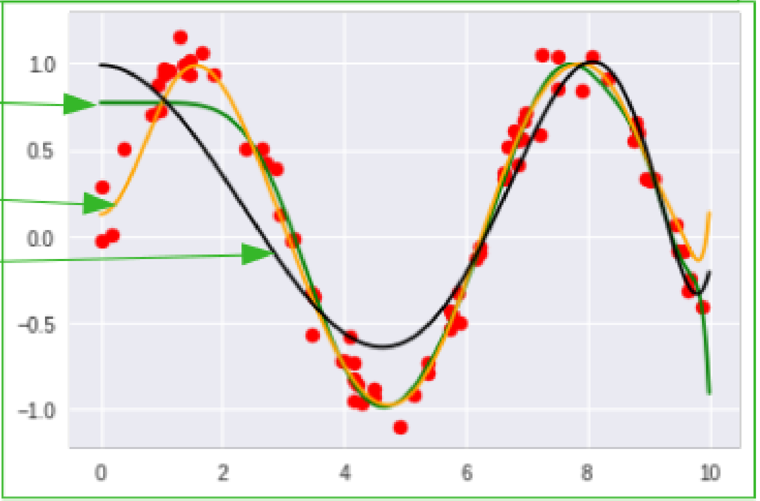
Lasso Regression (L1 regularization): Penalizes the sum of absolute values (1-norms) of regression parameters: $\mathbf{a}_i$ and $\mathbf{b}$

Lasso Regression (L1 regularization): Penalizes the sum of squares (2-norms) of the regressions parameters: $\mathbf{a}_i$ and $\mathbf{b}$

Linear model without regularization

Ridge model

Lasso model



[By Amina Delali]

4- Regularization

Ridge Regression (L2 regularization)

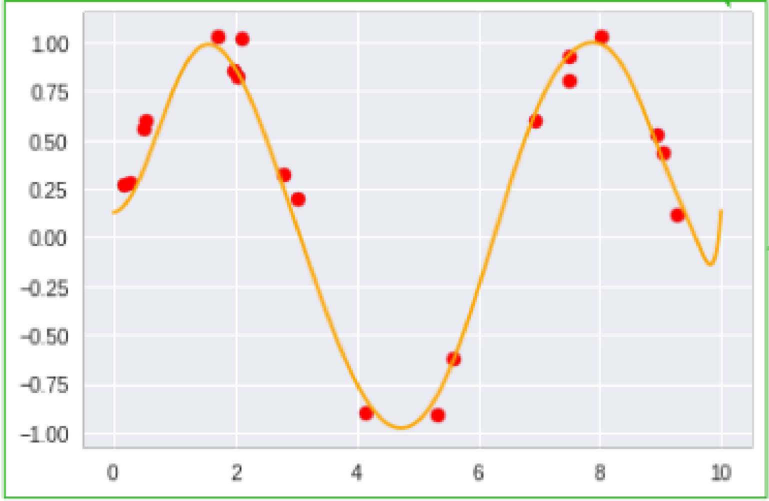
Penalty formula

$$p = \alpha * \sum_{n=1}^{N} \theta_n^2$$

```
# a Ridge regressor: linear regressor with L2 regularization
regressor2 =Ridge(alpha=0.1)
```

```
# a Ridge regressor with polynomial features
myRModel= make_pipeline(PolynomialFeatures(17), regressor2)
myRModel.fit(x_tr,y_tr)
y_pred2 = myRModel.predict(x_plot)
```

```
y_test_pred = myRModel.predict(x_plot)
plt.scatter(x_te, y_te, color="red")
plt.plot(x_plot, y_pred2, color="orange")
plt.show()
```

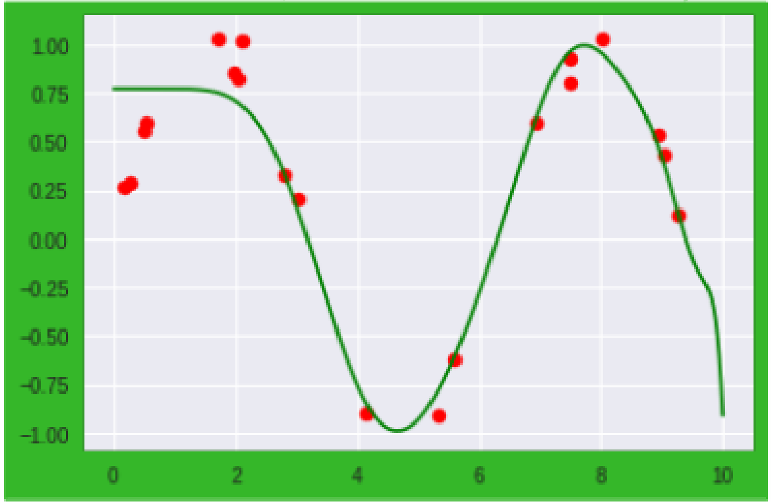


[By Amina Delali]

The first linear model without regularization and the test values

Smoother and more fitting to the test data

```
y_test_pred = myModel2.predict(x_plot)
plt.scatter(x_te, y_te, color="red")
plt.plot(x_plot, y_pred, color="green")
#x_tes= np.linspace(0, 10, 1000)
plt.show()
```



4- Regularization

Lasso Regression (L1 regularization)

Penalty formula

$$p = \alpha * \sum_{n=1}^{N} |\theta_n|$$

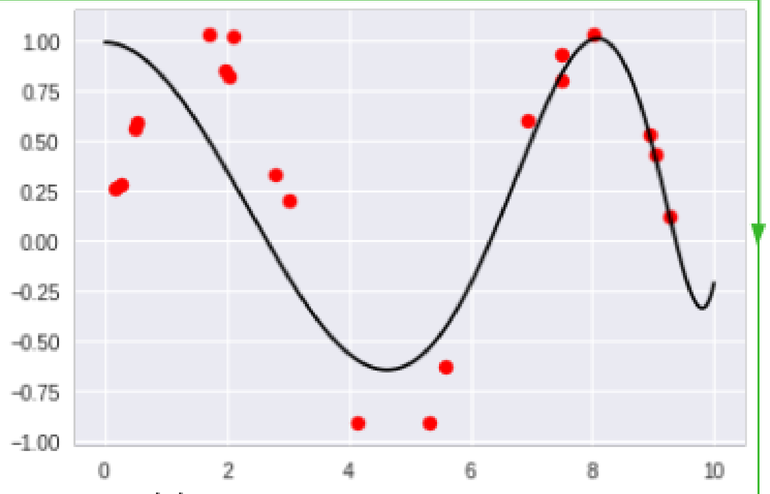
The first linear model without regularization and the test values

```
# a Lasso regressor: linear regressor with L1 regularization
regressor3 =Lasso(alpha=0.1)
```

More smoother but less fitting to the test data

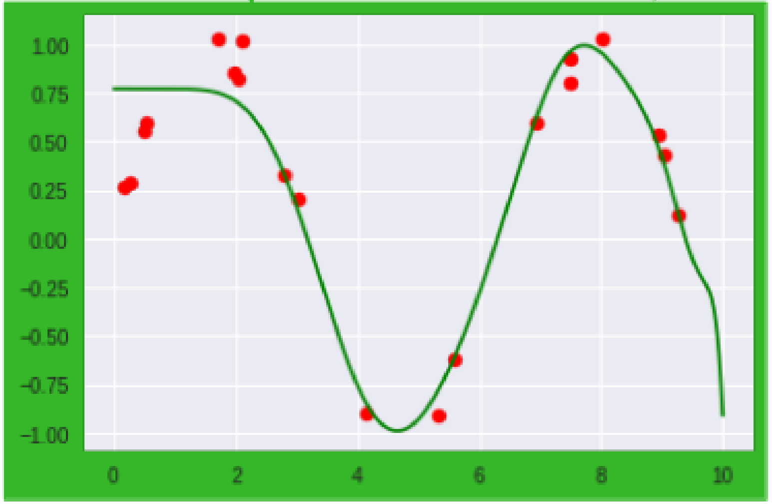
```
# a Lasso regressor with polynomial features
myLModel = make_pipeline(PolynomialFeatures(17), regressor3)
myLModel.fit(x_tr,y_tr)
y_pred3 = myLModel.predict(x_plot)
```

```
y_test_pred = myLModel.predict(x_plot)
plt.scatter(x_te, y_te, color="red")
plt.plot(x_plot, y_pred3, color="black")
plt.show()
```



[By Amina Delali]

```
y_test_pred = myModel2.predict(x_plot)
plt.scatter(x_te, y_te, color="red")
plt.plot(x_plot, y_pred, color="green")
#x_tes= np.linspace(0, 10, 1000)
plt.show()
```



5- Logistic Regressor Classifier

Definition

Classification with a logistic regression

build

Mathematical model == a **logistic Regression defined by** a **regression model** and **the sigmoid** function **S:**

$$s(z)=\frac{1}{1+e^{-z}}$$

where z is :the prediction value obtained by a regression model

with

Tunable parameters: The regression model parameter

Fit

Observed **Labeled** Data: a defined cost function is used for the logistic Regression. The S values are the probabilities of the samples data belonging to a Certain class.

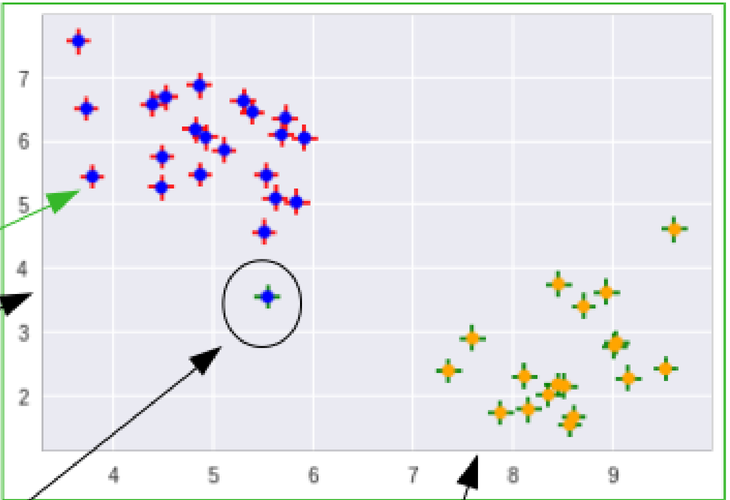
Predict the knows Categories

New Data: each sample is classed into one of the known categories. If the probability value is higher than a certain threshold, the sample Is affected to that class. And if it is not, it is affected to the other class.

5- Logistic Regressor Classifier

Classification with 2 **classes**

```
# instance of a logistic regressor
myModel_reg = linear_model.LogisticRegression()
# train the model
myModel_reg.fit(x2_train, y2_train)
# predict for test values:
y2_test_pred= myModel_reg.predict(x2_test)
```



The “**+**” marks **colors** represent the **predicted** classes. The “**.**” marks **colors** represent the **real** values

y axis : second feature values

This sample was predicted to be in class2 (2) , but in fact, it belongs to calss1 (1)

x axis : first feature values

```
# the coefficients calculated of the regression model
a = myModel_reg.coef_[0]
```

```
# the intercept of the rgression model
b= myModel_reg.intercept_
```

```
# the corresponding probabilities predicted the 3 first samples
y2_test_pr = myModel_reg.predict_proba(x2_test)[0:3]
y2_test_pr
```

```
array([[0.0145635 , 0.9854365 ],
       [0.99701667, 0.00298333],
       [0.00170004, 0.99829996]])
```

Probabilities for belonging to class2

Calculated parameters for the regression model

```
array([-0.12507469])
```

```
array([ 1.34836231, -2.05135853])
```

[By Amina Delali]

5- Logistic Regressor Classifier

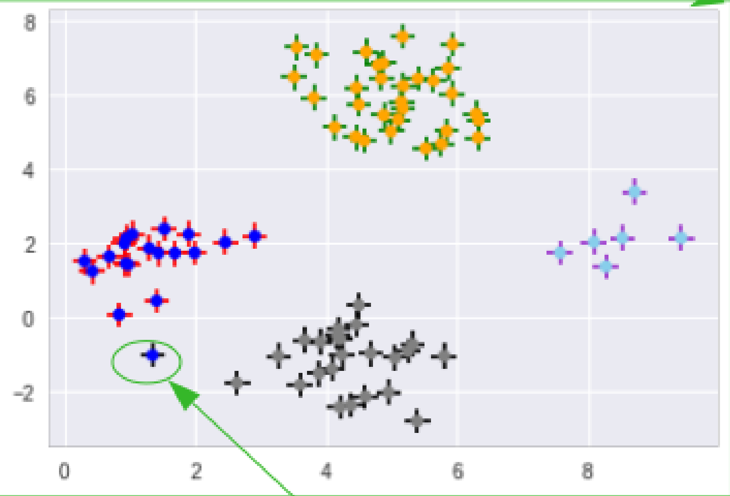
Classification with **more than 2 classes: One vs all**

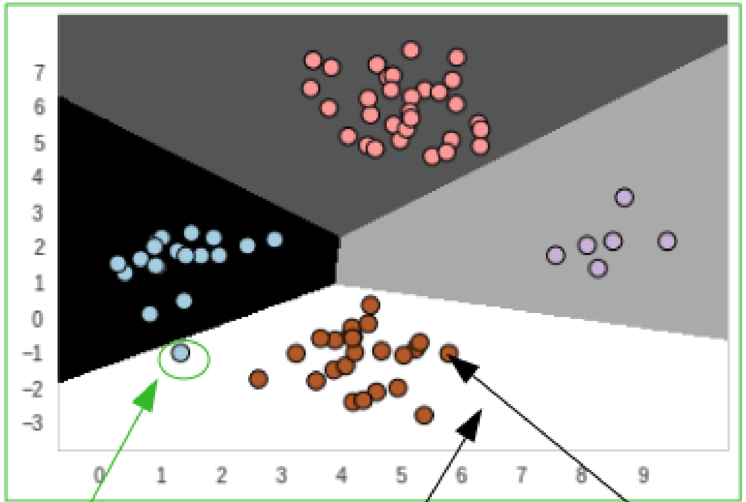
- The logistic regression is run for each class against all the other classes (considered as one class).
- Select the class for which the probability of belonging is the **greatest**.

An other way for visualization (the code is from :Joshi Prateek. Artificial intelligence with Python. Packt Publishing, 2017 link)

```
# instance of a logistic regressor
myModel_reg = linear_model.LogisticRegression()
# train the model
myModel_reg.fit(x2_train, y2_train)
# predict for test values:
y2_test_pred= myModel_reg.predict(x2_test)
```

Same code as logistic regression with 2 classes





[By Amina Delali]

Wrong predicted class

Predicted class

Real class

Definition

True Negatives: number of samples predicted to be in class **0** and they are in class **0**	**False Positives:** number of samples predicted to be in class **0** but they are in class **1**
False Negatives: number of samples predicted to be in class **1** but they are in class **0**	**True Positives:** number of samples predicted to be in class **1** and they are in class **1**

Well Predicted

Predicted wrong

```
true_classes = [1,1,1,0,1,0,0,1,0,1,0]
pred_classes = [1,0,1,0,1,0,1,1,1,1,1]
# Create confusion matrix
myConfusionMat = confusion_matrix(true_classes, pred_classes)
```

```
array([[2, 3],
       [1, 5]])
```

2 samples are known to be in class **0**, and they were predicted to be in class **0 : True** (the good prediction) **Negatives** (prediction in class 0)

3 samples are known to be in class **0**, and they were predicted to be in class **1 : False** (the wrong prediction) **Positives** (prediction in class 1)

[By Amina Delali]

More than 2 classes

In scikit-learn

Number of classes == **n+1**

Samples are known to be in (the rows number):

(0 , 0)	(0 , 1)	(0 , 2)	…	(0 , n)
(1 , 0)	(1 , 1)	(1 , 2)	…	(1 , n)
(2 , 0)	(1 , 1)	(2, 2)	…	(2 , n)
…	…	…	…	…
(n , 0)	(n , 1)	(n , 2)	…	(n , n)

Samples predicted to be in (columns number)

[By Amina Delali]

6- Confusion Matrix

Confusion Matrix Report

Number of samples that truly belongs to class 4

```
print(classification_report(true_classes, pred_classes))
```

Confusion matrix

```
array([[3, 2, 1, 1, 1],
       [2, 6, 0, 0, 0],
       [1, 2, 1, 0, 0],
       [1, 0, 0, 1, 0],
       [0, 1, 0, 0, 2]])
```

	precision	recall	f1-score	support
0	0.43	0.38	0.40	8
1	0.55	0.75	0.63	8
2	0.50	0.25	0.33	4
3	0.50	0.50	0.50	2
4	0.67	0.67	0.67	3
avg / total	0.51	0.52	0.50	25

0.43 == 3/ (3 + 2 +1 +1 +0)==
True positives / (true positives /false positives)

0.63 == 2* (0.55 * 0.75) /(0.55 + 0.75) == 2* (precision * recall) / (precision + recall)

0.25 == 2 /(1+2+1+0+0) ==
True Positives /(True Positives + False Negatives)

[By Amina Delali]

References

- Joshi Prateek. Artificial intelligence with Python. Packt Publishing, 2017.
- Jake VanderPlas. Python data science handbook: essential tools for working with data. O’Reilly Media, Inc, 2017.
- Scikit-learn.org. scikit-learn, machine learning in python. On-line at scikit-learn.org/stable/. Accessed on 03-11-2018

Thank you!

FOR ALL YOUR TIME