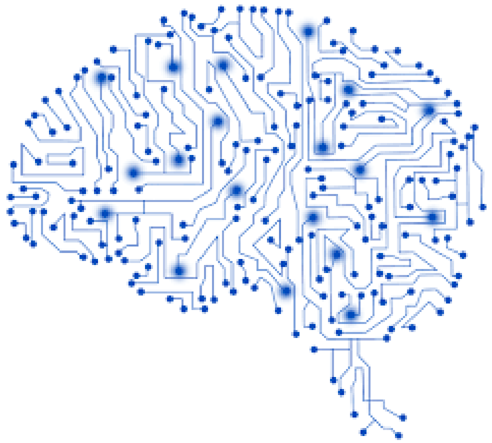
University of Minho
School of Engineering

ISLab



Dados e Aprendizagem Automática

Support Vector Machine

DAA @ MEI/1º ano – 1º Semestre

Bruno Fernandes, Filipe Gonçalves, Victor Alves, Cesar Analide

Part V

Contents

- Support Vector Machine
- Hands On

Support Vector Machine

Support Vector Machine

- Exercise:
  - **Problem:** Development of a Machine Learning Model able to classify if a patient has breast cancer
  - **Classification Approach:** Support Vector Machine approach to solve this problem
  - **Dataset:** table with information regarding the patient ID, diagnosis and real-valued features computed for each cell nucleus, including:
    - Radius (mean of distances from center to points on the perimeter)
    - Texture (standard deviation of gray-scale values)
    - Perimeter
    - Area
    - Smoothness (local variation in radius lengths)
    - Compactness (perimeter^2 / area - 1.0)
    - Concavity (severity of concave portions of the contour)
    - Concave points (number of concave portions of the contour)
    - Symmetry
    - Fractal dimension ("coastline approximation" - 1)

Support Vector Machine

Import Libraries

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
%matplotlib inline
```

Get the Data

We'll use the built in breast cancer dataset from Scikit Learn. We can get with the load function:

```
from sklearn.datasets import load_breast_cancer
```

```
cancer = load_breast_cancer()
```

Support Vector Machine

The data set is presented in a dictionary form:

```
cancer.keys()
```

```
dict_keys(['data', 'target', 'target_names', 'DESCR', 'feature_names', 'filename'])
```

```
cancer['feature_names']
```

```
array(['mean radius', 'mean texture', 'mean perimeter', 'mean area',
       'mean smoothness', 'mean compactness', 'mean concavity',
       'mean concave points', 'mean symmetry', 'mean fractal dimension',
       'radius error', 'texture error', 'perimeter error', 'area error',
       'smoothness error', 'compactness error', 'concavity error',
       'concave points error', 'symmetry error',
       'fractal dimension error', 'worst radius', 'worst texture',
       'worst perimeter', 'worst area', 'worst smoothness',
       'worst compactness', 'worst concavity', 'worst concave points',
       'worst symmetry', 'worst fractal dimension'], dtype='<U23')
```

Support Vector Machine

Set up DataFrame

```
df_feat = pd.DataFrame(cancer['data'],columns=cancer['feature_names'])
df_feat.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 569 entries, 0 to 568
Data columns (total 30 columns):
mean radius                569 non-null float64
mean texture               569 non-null float64
mean perimeter             569 non-null float64
mean area                  569 non-null float64
mean smoothness            569 non-null float64
mean compactness           569 non-null float64
mean concavity             569 non-null float64
mean concave points        569 non-null float64
mean symmetry              569 non-null float64
mean fractal dimension     569 non-null float64
radius error               569 non-null float64
texture error              569 non-null float64
perimeter error            569 non-null float64
area error                 569 non-null float64
smoothness error           569 non-null float64
compactness error          569 non-null float64
concavity error            569 non-null float64
concave points error       569 non-null float64
symmetry error             569 non-null float64
fractal dimension error    569 non-null float64
worst radius               569 non-null float64
worst texture              569 non-null float64
worst perimeter            569 non-null float64
worst area                 569 non-null float64
worst smoothness           569 non-null float64
worst compactness          569 non-null float64
worst concavity            569 non-null float64
worst concave points       569 non-null float64
worst symmetry             569 non-null float64
worst fractal dimension    569 non-null float64
dtypes: float64(30)
memory usage: 133.4 KB
```

```
cancer['target']
```

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

```
df_target = pd.DataFrame(cancer['target'],columns=['Cancer'])
```

Now let's actually check out the dataframe!

```
df_target.head()
```

	Cancer
0	0
1	0
2	0
3	0
4	0

Support Vector Machine

Train the Support Vector Classifier

10-Fold Cross Validation

```
from sklearn.model_selection import cross_val_score
from sklearn.svm import SVC
```

```
cross_valid_model = SVC(random_state=2021)
scores = cross_val_score(cross_valid_model, df_feat, np.ravel(df_target), cv=10)
scores
```

```
array([0.89473684, 0.84210526, 0.89473684, 0.92982456, 0.92982456,
       0.92982456, 0.94736842, 0.92982456, 0.92982456, 0.91071429])
```

```
print("%0.2f accuracy with a standard deviation of %0.2f" % (scores.mean(), scores.std()))
```

```
0.91 accuracy with a standard deviation of 0.03
```

Support Vector Machine

Exploratory Data Analysis

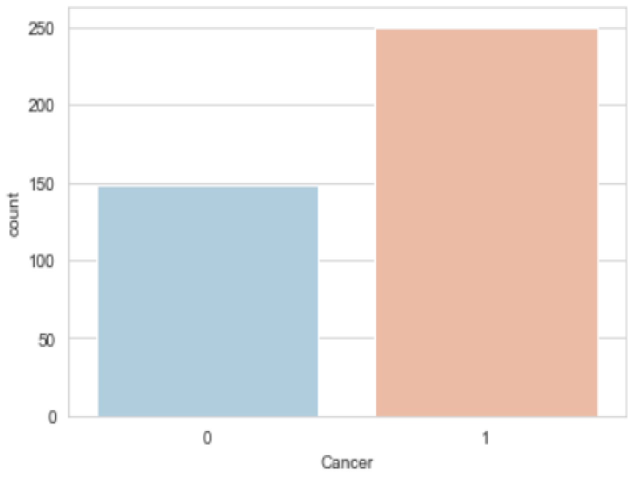
Train Test Split

```
from sklearn.model_selection import train_test_split
```

```
X_train, X_test, y_train, y_test = train_test_split(df_feat, np.ravel(df_target), test_size=0.30, random_state=2021)
```

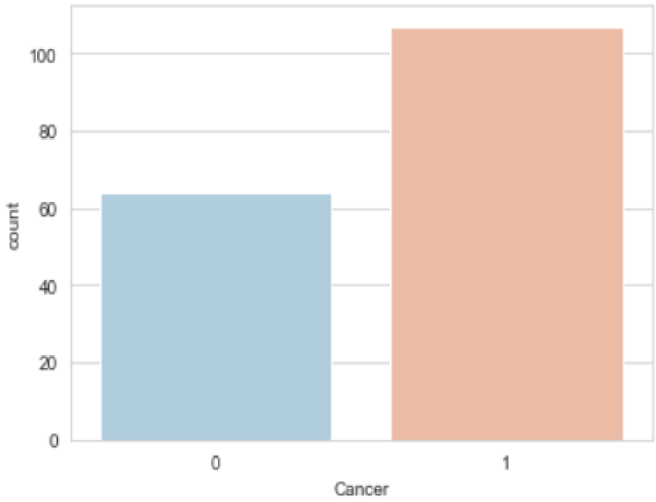
```
sns.set_style('whitegrid')
sns.countplot(x='Cancer', data = pd.DataFrame(y_train,columns=['Cancer']) ,palette='RdBu_r')
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x17f179831c8>
```




```
sns.countplot(x='Cancer', data = pd.DataFrame(y_test,columns=['Cancer']) ,palette='RdBu_r')
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x17f17d29988>
```



Support Vector Machine

Hold-out

```
from sklearn.svm import SVC
```

```
model = SVC(random_state=2021)
```

```
model.fit(X_train,y_train)
```

```
SVC(random_state=2021)
```

Predictions and Evaluations

Now let's predict using the trained model.

```
predictions = model.predict(X_test)
```

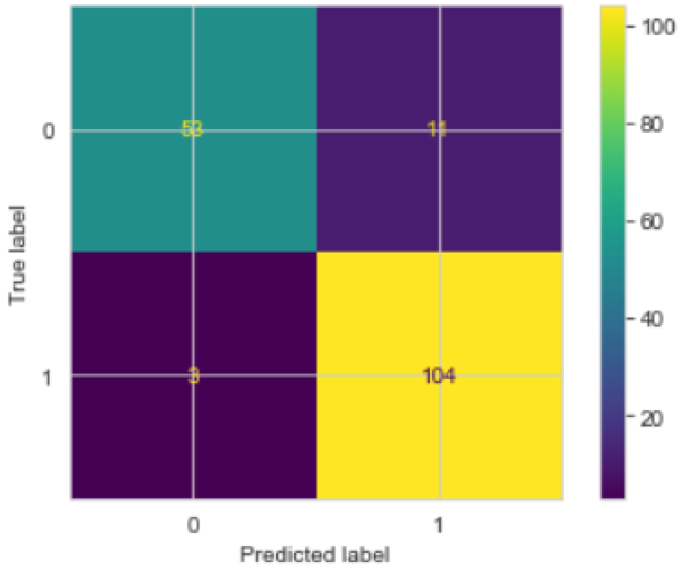
```
from sklearn.metrics import classification_report, plot_confusion_matrix, accuracy_score
```

```
print("%0.2f accuracy" % (accuracy_score(y_test, predictions)))
```

```
0.92 accuracy
```

Support Vector Machine


```
plot_confusion_matrix(model, X_test, y_test)
```




```
print(classification_report(y_test,predictions))
```

```
              precision    recall  f1-score   support

           0       0.95      0.83      0.88        64
           1       0.90      0.97      0.94       107

    accuracy                           0.92       171
   macro avg       0.93      0.90      0.91       171
weighted avg       0.92      0.92      0.92       171
```

- Notice that we are classifying everything into a single class! This means our model needs to have it parameters adjusted.
- We can search for these parameters using a GridSearch

Concepts

But first some concepts…

- Model Parameters: a model's (internal) configuration variable whose value is estimated from training data, i.e., the value is not set manually. Some examples include:
  - Weights in Artificial Neural Networks
  - Support vectors in Support Vector Machines

- Model Hyperparameters: a model's (external) configuration variable whose value can be set manually. It is difficult to know, beforehand, the best value of each hyperparameter. Tuning a model consists in finding the best (or, at least, a good) configuration of hyperparameters. Examples include:
  - Optimizer and learning rate in Artificial Neural Networks
  - C and gamma in Support Vector Machines
  - Quality measure and pruning method in Decision Trees

Support Vector Machine

Gridsearch

- Finding the right parameters (like what C or gamma values to use) is a tricky task;
- The idea of creating a 'grid' of parameters and trying out all the possible combinations is called a Gridsearch;
  - This method is common enough that Scikit-learn has this functionality built in with GridSearchCV (CV stands for cross-validation).
  - GridSearchCV takes a dictionary that describes the parameters that should be tried and a model to train.
  - The grid of parameters is defined as a dictionary, where the keys are the parameters and the values are the settings to be tested.

```
param_grid = {'C': [0.1,1, 10, 100, 1000], 'gamma': [1,0.1,0.01,0.001,0.0001], 'kernel': ['rbf']}
```

```
from sklearn.model_selection import GridSearchCV
```

- GridSearchCV is that it is a meta-estimator.
- It takes an estimator like SVC, and creates a new estimator, that behaves exactly the same - in this case, like a classifier.
- You should add refit=True and choose verbose to whatever number you want (verbose means the text output describing the process).

```
grid = GridSearchCV(SVC(random_state=2021),param_grid,refit=True,verbose=3)
```

Support Vector Machine

What fit does is a bit more involved then usual:

- Runs the same loop with cross-validation, to find the best parameter combination.
- Once it has the best combination, it runs fit again on all data passed to fit (without cross-validation), to built a single new model using the best parameter setting.

```
# May take awhile!
grid.fit(X_train,y_train)
```

```
Fitting 3 folds for each of 25 candidates, totalling 75 fits
[CV] C=0.1, gamma=1, kernel=rbf .....................................
[CV]  C=0.1, gamma=1, kernel=rbf, score=0.631578947368421, total=   0.0s
[CV] C=0.1, gamma=1, kernel=rbf .....................................
[CV]  C=0.1, gamma=1, kernel=rbf, score=0.631578947368421, total=   0.0s
[CV] C=0.1, gamma=1, kernel=rbf .....................................
[CV]  C=0.1, gamma=1, kernel=rbf, score=0.6363636363636364, total=   0.0s
[CV] C=0.1, gamma=0.1, kernel=rbf ...................................
[CV]  C=0.1, gamma=0.1, kernel=rbf, score=0.631578947368421, total=   0.0s
[CV] C=0.1, gamma=0.1, kernel=rbf ...................................
[CV]  C=0.1, gamma=0.1, kernel=rbf, score=0.631578947368421, total=   0.0s
[CV] C=0.1, gamma=0.1, kernel=rbf ...................................
[CV]  C=0.1, gamma=0.1, kernel=rbf, score=0.6363636363636364, total=   0.0s
[CV] C=0.1, gamma=0.01, kernel=rbf ..................................
[CV]  C=0.1, gamma=0.01, kernel=rbf, score=0.631578947368421, total=   0.0s
[CV] C=0.1, gamma=0.01, kernel=rbf ..................................
[CV]  C=0.1, gamma=0.01, kernel=rbf, score=0.631578947368421, total=   0.0s
[CV] C=0.1, gamma=0.01, kernel=rbf ..................................
[CV]  C=0.1, gamma=0.01, kernel=rbf, score=0.6363636363636364, total=   0.0s
```

Support Vector Machine

You can inspect the best parameters found by GridSearchCV in the best_params_ attribute, and the best estimator in the best_estimator_ attribute:

```
grid.best_params_
```

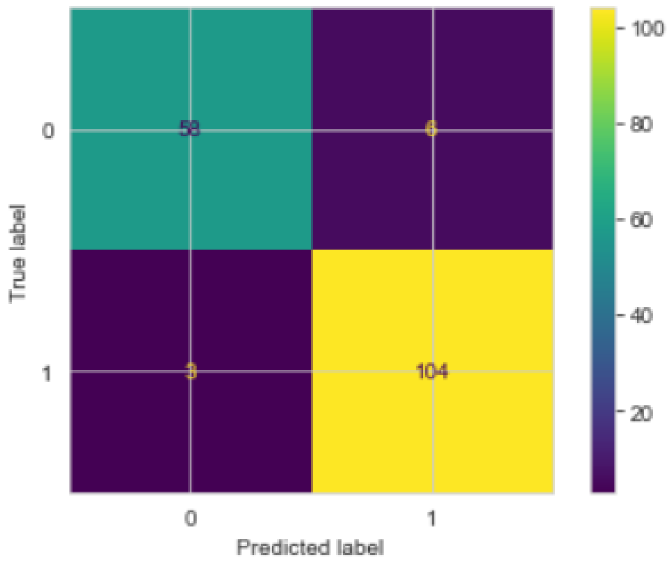
```
{'C': 1, 'gamma': 0.0001, 'kernel': 'rbf'}
```

```
grid.best_estimator_
```

```
SVC(C=1, gamma=0.0001, random_state=2021)
```

Then you can re-run predictions on this grid object just like you would with a normal model.

```
plot_confusion_matrix(grid, X_test, y_test)
```



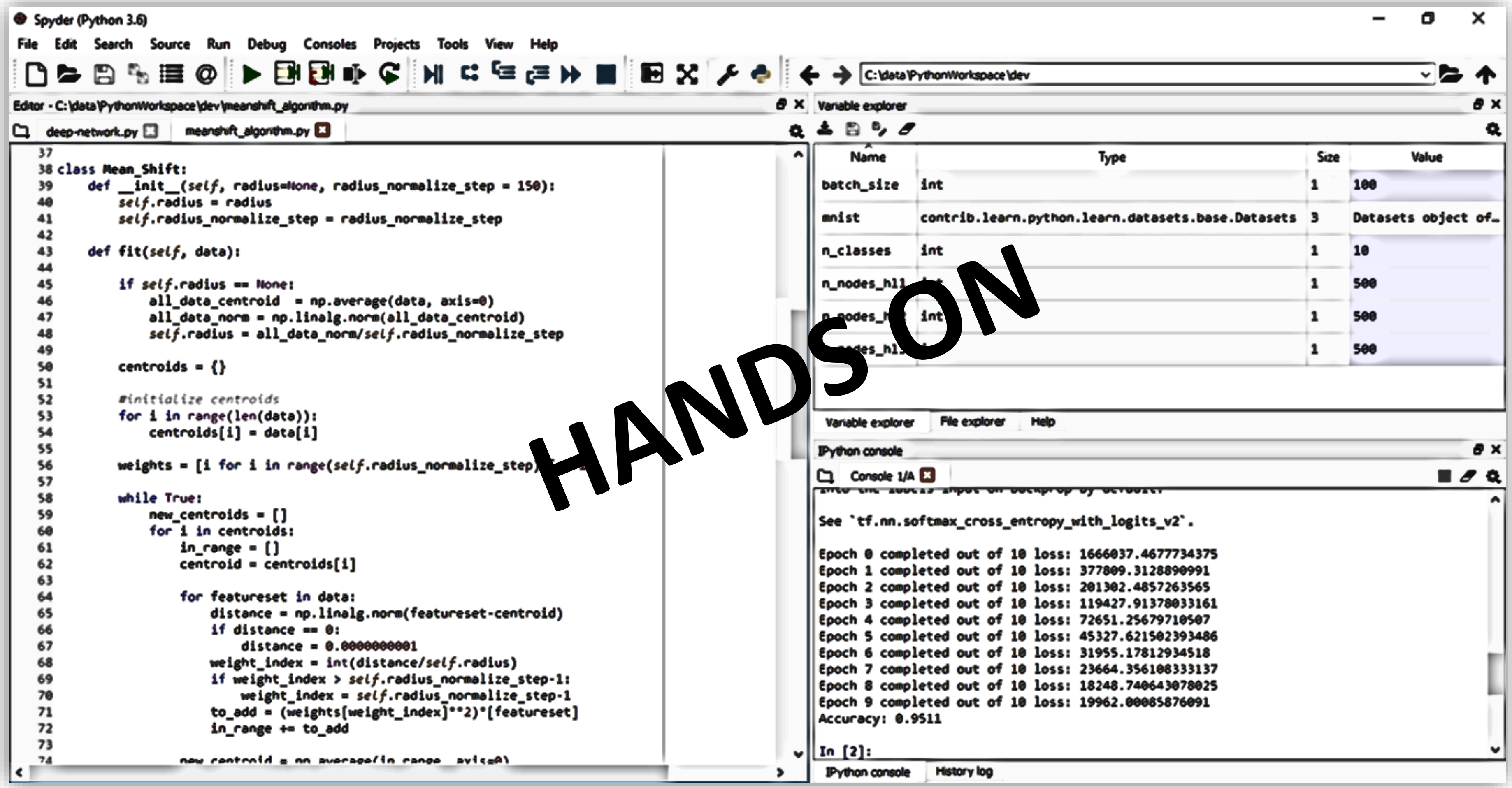

```
grid_predictions = grid.predict(X_test)
print(classification_report(y_test,grid_predictions))
```

```
              precision    recall  f1-score   support

           0       0.95      0.91      0.93        64
           1       0.95      0.97      0.96       107

    accuracy                           0.95       171
   macro avg       0.95      0.94      0.94       171
weighted avg       0.95      0.95      0.95       171
```

Hands On


Spyder (Python 3.6)
File Edit Search Source Run Debug Consoles Projects Tools View Help
C:\data\PythonWorkspace\dev
Editor - C:\data\PythonWorkspace\dev\meanshift_algorithm.py
deep-network.py
meanshift_algorithm.py
class Mean_Shift:
def __init__(self, radius=None, radius_normalize_step = 150):
self.radius = radius
self.radius_normalize_step = radius_normalize_step
def fit(self, data):
if self.radius == None:
all_data_centroid = np.average(data, axis=0)
all_data_norm = np.linalg.norm(all_data_centroid)
self.radius = all_data_norm/self.radius_normalize_step
centroids = {}
#initialize centroids
for i in range(len(data)):
centroids[i] = data[i]
weights = [i for i in range(self.radius_normalize_step)
while True:
new_centroids = []
for i in centroids:
in_range = []
centroid = centroids[i]
for featureset in data:
distance = np.linalg.norm(featureset-centroid)
if distance == 0:
distance = 0.0000000001
weight_index = int(distance/self.radius)
if weight_index > self.radius_normalize_step-1:
weight_index = self.radius_normalize_step-1
to_add = (weights[weight_index]**2)*[featureset]
in_range += to_add
Variable explorer
Name
Type
Size
Value
batch_size
int
1
100
mnist
contrib.learn.python.learn.datasets.base.Datasets
3
Datasets object of…
n_classes
int
1
10
500
500
500
Variable explorer
File explorer
Help
IPython console
Console 1/A
See `tf.nn.softmax_cross_entropy_with_logits_v2`.
Epoch 0 completed out of 10 loss: 1666037.4677734375
Epoch 1 completed out of 10 loss: 377809.3128890991
Epoch 2 completed out of 10 loss: 201302.4857263565
Epoch 3 completed out of 10 loss: 119427.91378033161
Epoch 4 completed out of 10 loss: 72651.25679710507
Epoch 5 completed out of 10 loss: 45327.621502393486
Epoch 6 completed out of 10 loss: 31955.17812934518
Epoch 7 completed out of 10 loss: 23664.356108333137
Epoch 8 completed out of 10 loss: 18248.740643078025
Epoch 9 completed out of 10 loss: 19962.00085876091
Accuracy: 0.9511
In [2]:
IPython console
History log
HANDS ON