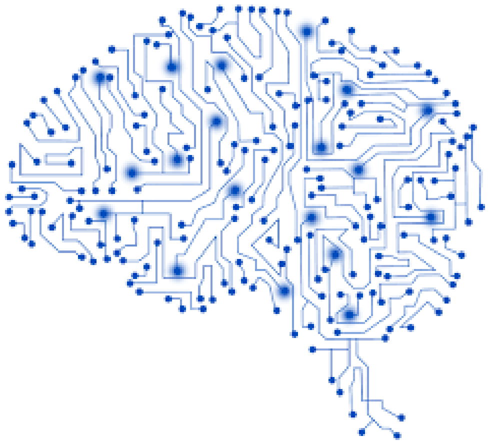
University of Minho
School of Engineering



Dados e Aprendizagem Automática

Support Vector Machine:
Hyperparameter Tuning with GridSearchCV

DAA @ MEI-1º/MiEI-4º – 1º Semestre

Bruno Fernandes, César Analide, Dalila Alves, Filipa Ferraz, Victor Alves

Part VI

Contents

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()
```

The data set is presented in the form of a dictionary:

```
cancer.keys()
```

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

Support Vector Machine

We can obtain information and arrays from this dictionary to set up our data frame and understand the features:

```
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')
```

```
print(cancer['DESCR'])
```

```
.. _breast_cancer_dataset:

Breast cancer wisconsin (diagnostic) dataset
--------------------------------------------

**Data Set Characteristics:**

    :Number of Instances: 569

    :Number of Attributes: 30 numeric, predictive attributes and the class

    :Attribute Information:
        - radius (mean of distances from center to points on the perimeter)
        - texture (standard deviation of gray-scale values)
```

…

Support Vector Machine

Set up the dataframe

```python
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):
 #   Column                   Non-Null Count  Dtype  
---  ------                   --------------  -----  
 0   mean radius              569 non-null    float64
 1   mean texture             569 non-null    float64
 2   mean perimeter           569 non-null    float64
 3   mean area                569 non-null    float64
 4   mean smoothness          569 non-null    float64
 5   mean compactness         569 non-null    float64
 6   mean concavity           569 non-null    float64
 7   mean concave points      569 non-null    float64
 8   mean symmetry            569 non-null    float64
 9   mean fractal dimension   569 non-null    float64
 10  radius error             569 non-null    float64
 11  texture error            569 non-null    float64
 12  perimeter error          569 non-null    float64
 13  area error               569 non-null    float64
 14  smoothness error         569 non-null    float64
 15  compactness error        569 non-null    float64
 16  concavity error          569 non-null    float64
 17  concave points error     569 non-null    float64
 18  symmetry error           569 non-null    float64
 19  fractal dimension error  569 non-null    float64
```

…

Support Vector Machine


```
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 check the dataframe:

```
df_target.head()
```

	Cancer
0	0
1	0
2	0
3	0
4	0

Support Vector Machine

Exploratory Data Analysis

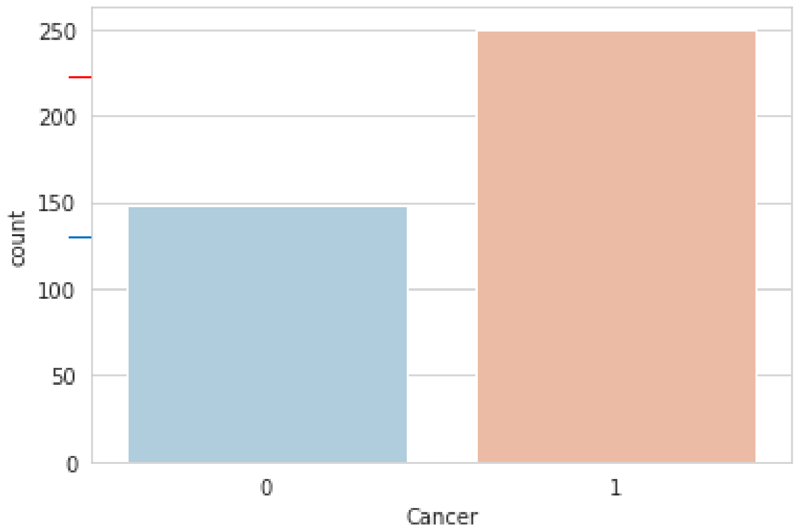
Train Test Split

```
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)
```

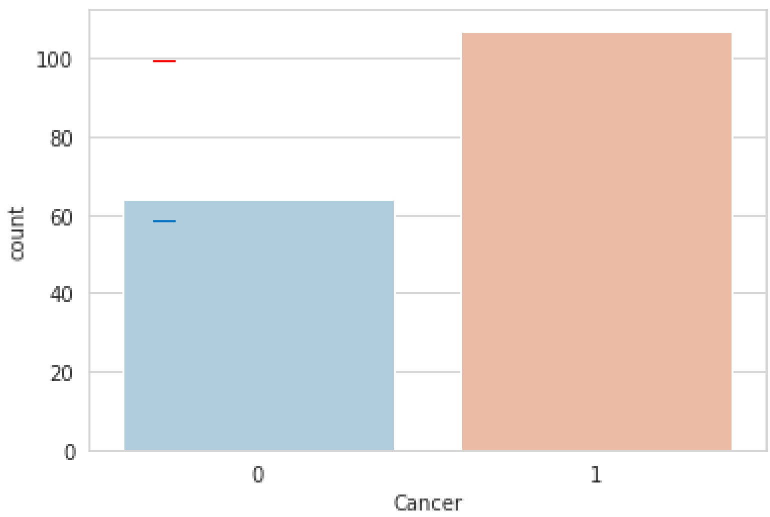
```
sns.set_style('whitegrid')
sns.countplot(x='Cancer', data = pd.DataFrame(y_train,columns=['Cancer']) ,palette='RdBu_r')
```

<AxesSubplot:xlabel='Cancer', ylabel='count'>




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

<AxesSubplot:xlabel='Cancer', ylabel='count'>



Concepts

- **Model Parameters:** a model's (internal) configuration variable whose value is estimated from the training data, i.e., the value is not set manually. 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 the best value for each hyperparameter in advance. Tuning a model consists of finding the best (or at least a good) configuration of the 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

Train the Support Vector Classifier

10-Fold Cross Validation

Let's try the Cross Validation technique with 10 folds:

```
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
```

And now without Cross Validation:

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

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

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

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

Support Vector Machine

Predictions and evaluations

Let's predict using the trained model:

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

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

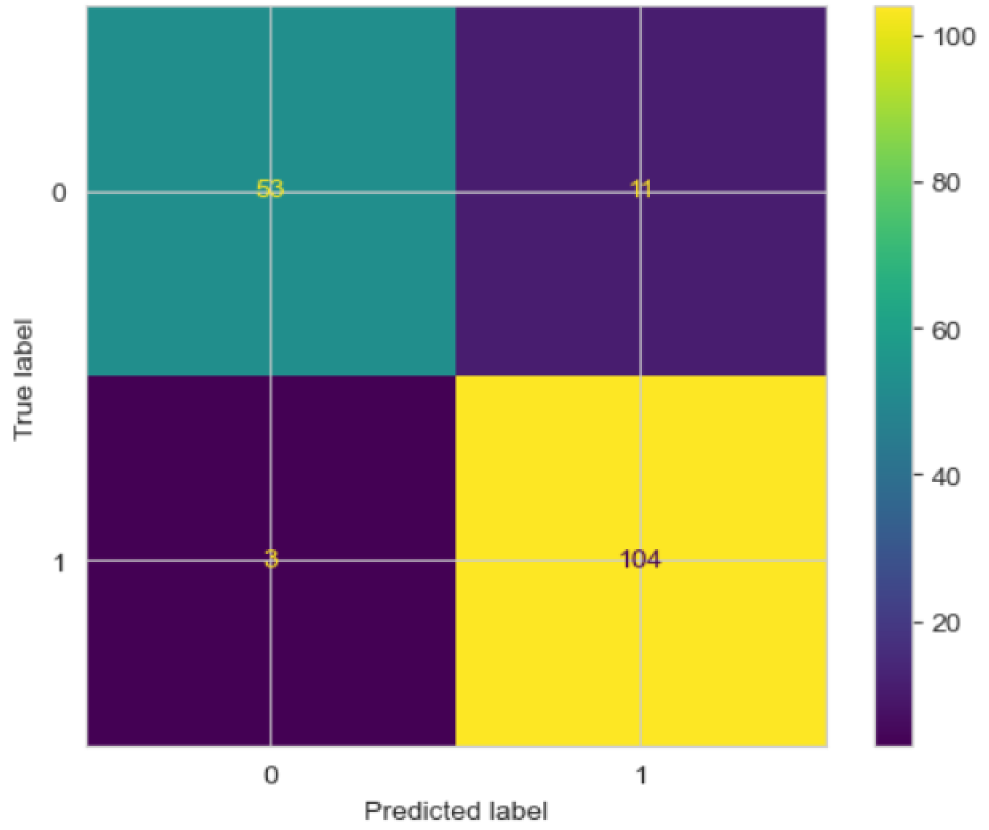
```
print("%0.2f accuracy" % (accuracy_score(y_test, predictions)))
```

```
0.92 accuracy
```

```
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
```

```
ConfusionMatrixDisplay.from_predictions(y_test, predictions)
plt.show()
```



Support Vector Machine

GridSearch

- Finding the right parameters (such as the values of *C* or *gamma* to use) is a complicated task
- The idea of creating a "grid" of parameters and trying out all the possible combinations is called *GridSearch*
  - This method is common enough for Scikit-learnto have this functionality incorporated with *GridSearchCV* (*CV* stands for *Cross-Validation*)
  - *GridSearchCV* receives a dictionary describing the parameters to be tried and the model to be trained
  - The parameter grid is defined as a dictionary in which 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 a meta-estimator
- It takes an estimator like *SVC* and creates a new estimator that behaves in exactly the same way - in this case, like a classifier

Add *refit=True* and choose verbose for the number you want (*verbose* means the text output that describes the process)

Train a model with GridSearchCV

Now its time to train a Support Vector Machine Classifier

Call the *SVC()* model from sklearn and fit the model to the training data:

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

Support Vector Machine

What does fit do:

- 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

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

```
Fitting 5 folds for each of 25 candidates, totalling 125 fits
[CV 1/5] END ........C=0.1, gamma=1, kernel=rbf;, score=0.625 total time=   0.0s
[CV 2/5] END ........C=0.1, gamma=1, kernel=rbf;, score=0.625 total time=   0.0s
[CV 3/5] END ........C=0.1, gamma=1, kernel=rbf;, score=0.625 total time=   0.0s
[CV 4/5] END ........C=0.1, gamma=1, kernel=rbf;, score=0.633 total time=   0.0s
[CV 5/5] END ........C=0.1, gamma=1, kernel=rbf;, score=0.633 total time=   0.0s

  ...

[CV 3/5] END ..C=1000, gamma=0.0001, kernel=rbf;, score=0.912 total time=   0.0s
[CV 4/5] END ..C=1000, gamma=0.0001, kernel=rbf;, score=0.937 total time=   0.0s
[CV 5/5] END ..C=1000, gamma=0.0001, kernel=rbf;, score=0.911 total time=   0.0s
```

```
▸ GridSearchCV
▸ estimator: SVC
    ▸ SVC
```

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_
```

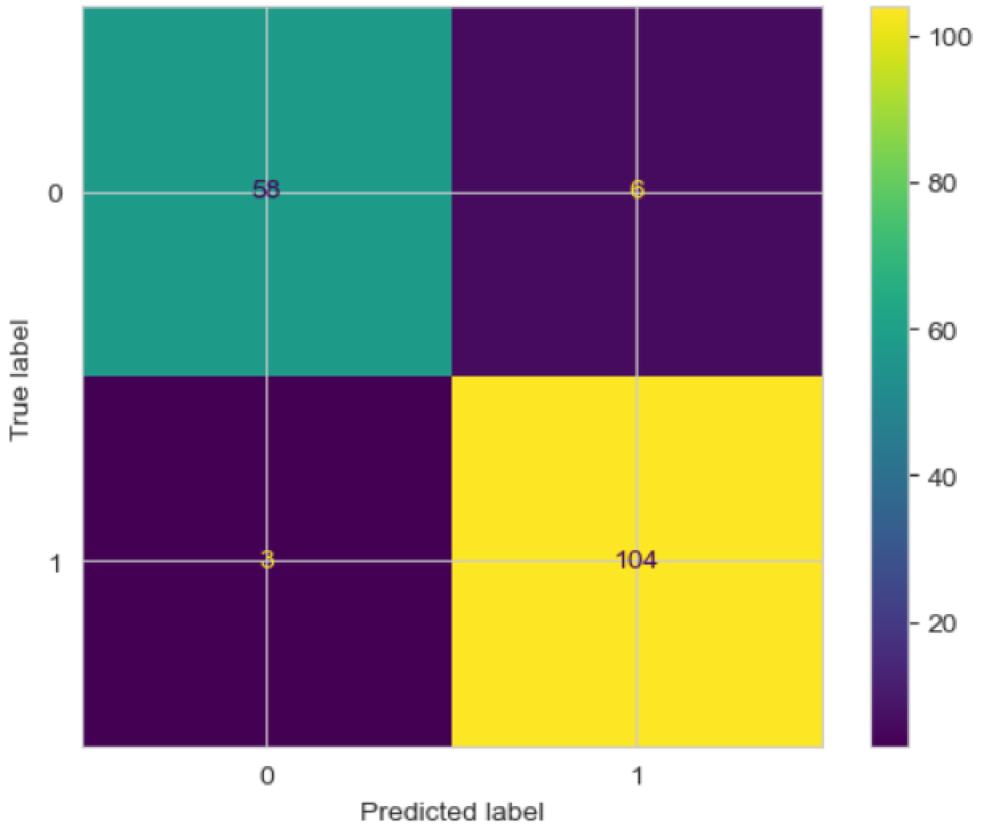
```
SVC
SVC(C=1, gamma=0.0001, random_state=2021)
```

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

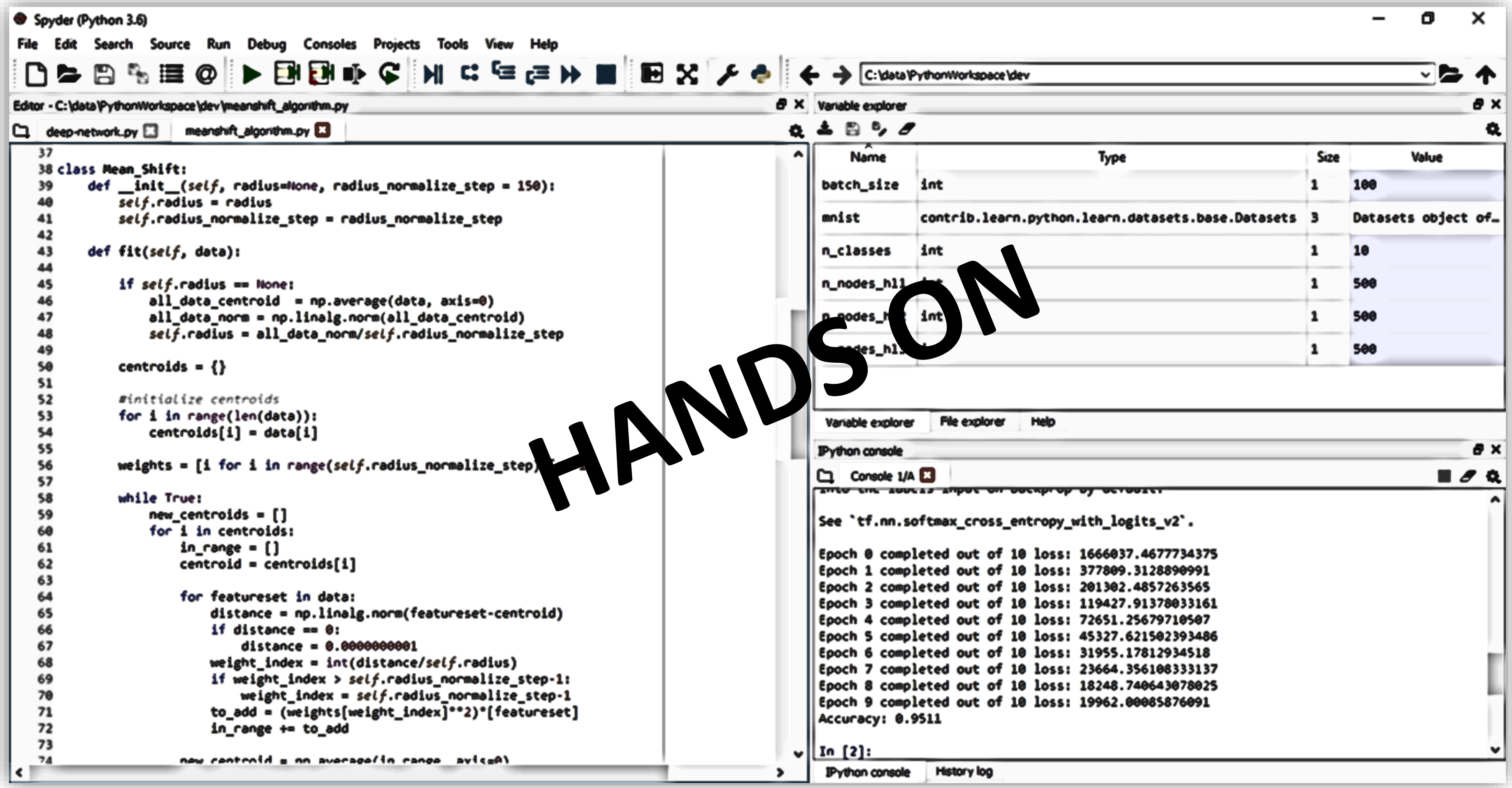
```
grid_predictions = grid.predict(X_test)
print(classification_report(y_test,grid_predictions))
```

```
              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
```

```
ConfusionMatrixDisplay.from_predictions(y_test, grid_predictions)
plt.show()
```



Hands On



HANDS ON
Spyder (Python 3.6)
File Edit Search Source Run Debug Consoles Projects Tools View Help
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]
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
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

Decision Tree Classifier vs. Support Vector Classifier


```
from sklearn.svm import SVC
from sklearn.tree import DecisionTreeClassifier
from sklearn import tree
```

```
dt_model = DecisionTreeClassifier(random_state=2022)
```

```
svc_model = SVC(random_state=2022)
```

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

```
▾ DecisionTreeClassifier
DecisionTreeClassifier(random_state=2022)
```

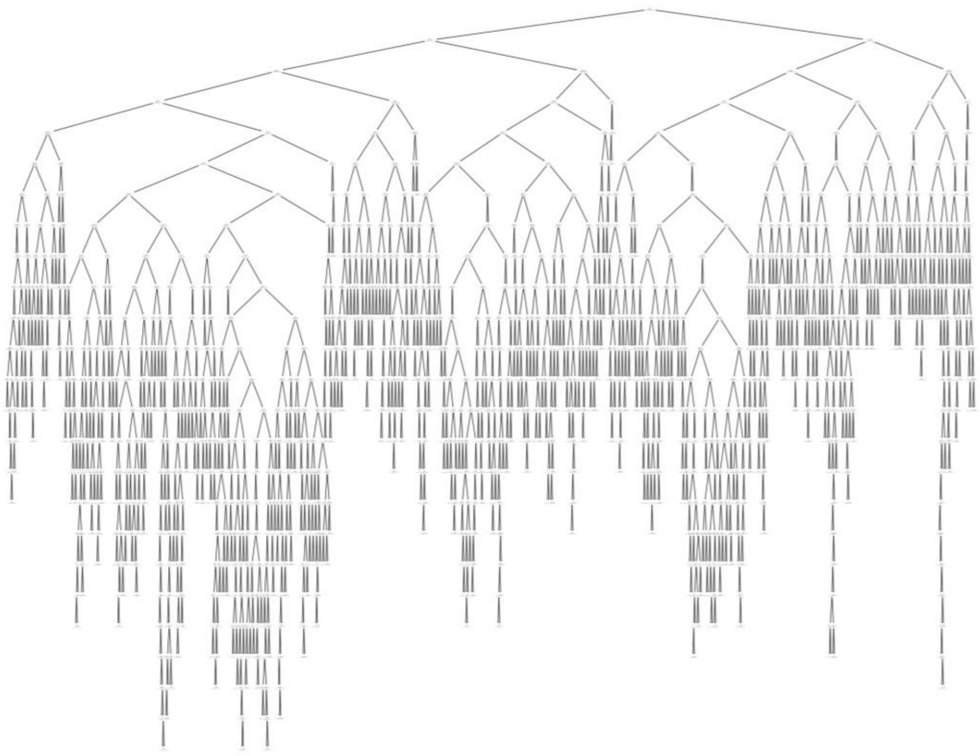
```
svc_model.fit(X_train,y_train)
```

```
▾ SVC
SVC(random_state=2022)
```

Decision Tree Classifier vs. Support Vector Classifier

1. Plot the resultant tree (plot_tree) and save it as figure (.png)

```
fig = plt.figure(figsize=(25,20))
tree.plot_tree(dt_model)
plt.show()

fig.savefig("dt_plot.png")
```



2. Using export_text to represent the tree. Save it in a log file

```
text_representation = tree.export_text(dt_model)
print(text_representation)

with open("dt_text.log", "w") as fout:
    fout.write(text_representation)
```

```
|--- feature_1 <= 25.00
|   |--- feature_4 <= 1020.50
|   |   |--- feature_5 <= 91.50
|   |   |   |--- feature_4 <= 1011.50
|   |   |   |   |--- feature_3 <= 15.50
|   |   |   |   |   |--- feature_3 <= 11.50
|   |   |   |   |   |   |--- feature_6 <= 1.50
|   |   |   |   |   |   |   |--- feature_4 <= 1002.00
|   |   |   |   |   |   |   |   |--- class: 1
|   |   |   |   |   |   |   |--- feature_4 >  1002.00
|   |   |   |   |   |   |   |   |--- feature_3 <= 10.50
|   |   |   |   |   |   |   |   |   |--- feature_2 <= 1.50
|   |   |   |   |   |   |   |   |   |   |--- feature_7 <= 1.50
|   |   |   |   |   |   |   |   |   |   |   |--- truncated branch of depth 6
|   |   |   |   |   |   |   |   |   |   |--- feature_7 >  1.50
|   |   |   |   |   |   |   |   |   |   |   |--- truncated branch of depth 3
|   |   |   |   |   |   |   |   |   |--- feature_2 >  1.50
|   |   |   |   |   |   |   |   |   |   |--- class: 1
|   |   |   |   |   |   |   |   |--- feature_3 >  10.50
|   |   |   |   |   |   |   |   |   |--- feature_4 <= 1005.00
```

...

Decision Tree Classifier vs. Support Vector Classifier

Model evaluation

Decision Tree

	precision	recall	f1-score	support
1	0.75	0.74	0.74	632
2	0.58	0.55	0.56	201
3	0.72	0.73	0.73	179
4	0.46	0.49	0.47	313
5	0.40	0.42	0.41	175
accuracy			0.62	1500
macro avg	0.58	0.58	0.58	1500
weighted avg	0.62	0.62	0.62	1500

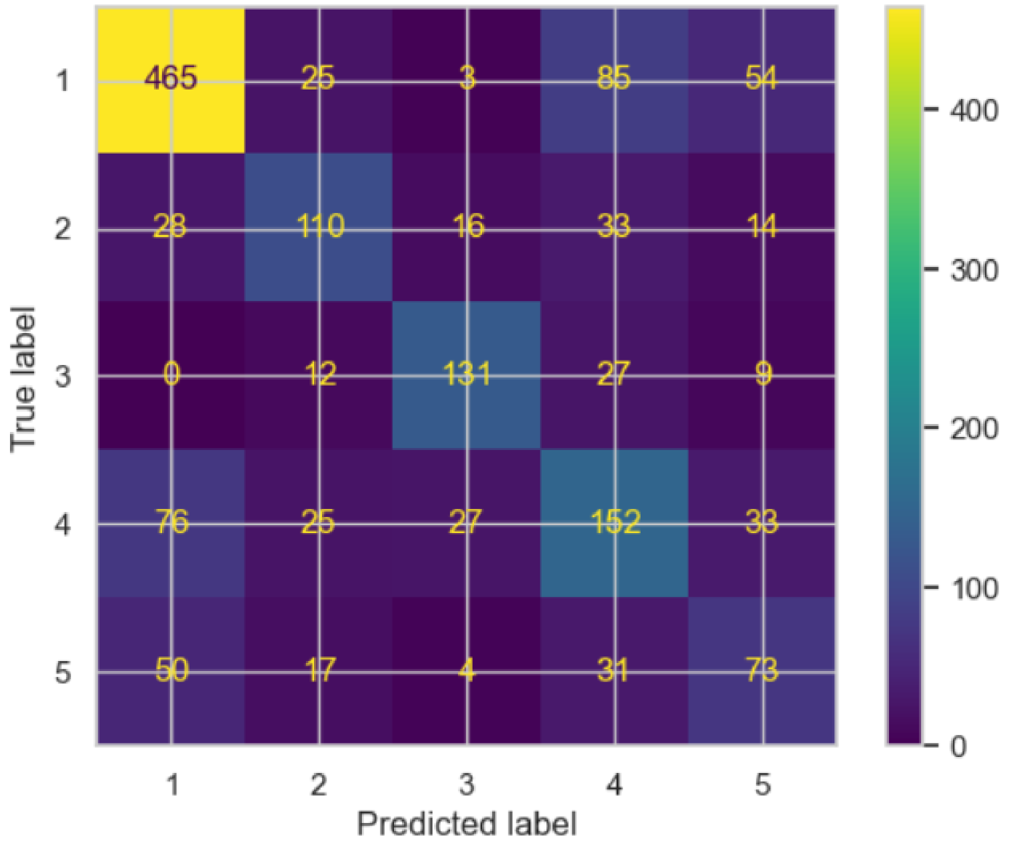
Support Vector Machine

	precision	recall	f1-score	support
1	0.57	0.98	0.72	632
2	0.43	0.41	0.42	201
3	0.77	0.71	0.74	179
4	0.47	0.08	0.13	313
5	0.00	0.00	0.00	175
accuracy			0.57	1500
macro avg	0.45	0.44	0.40	1500
weighted avg	0.49	0.57	0.47	1500

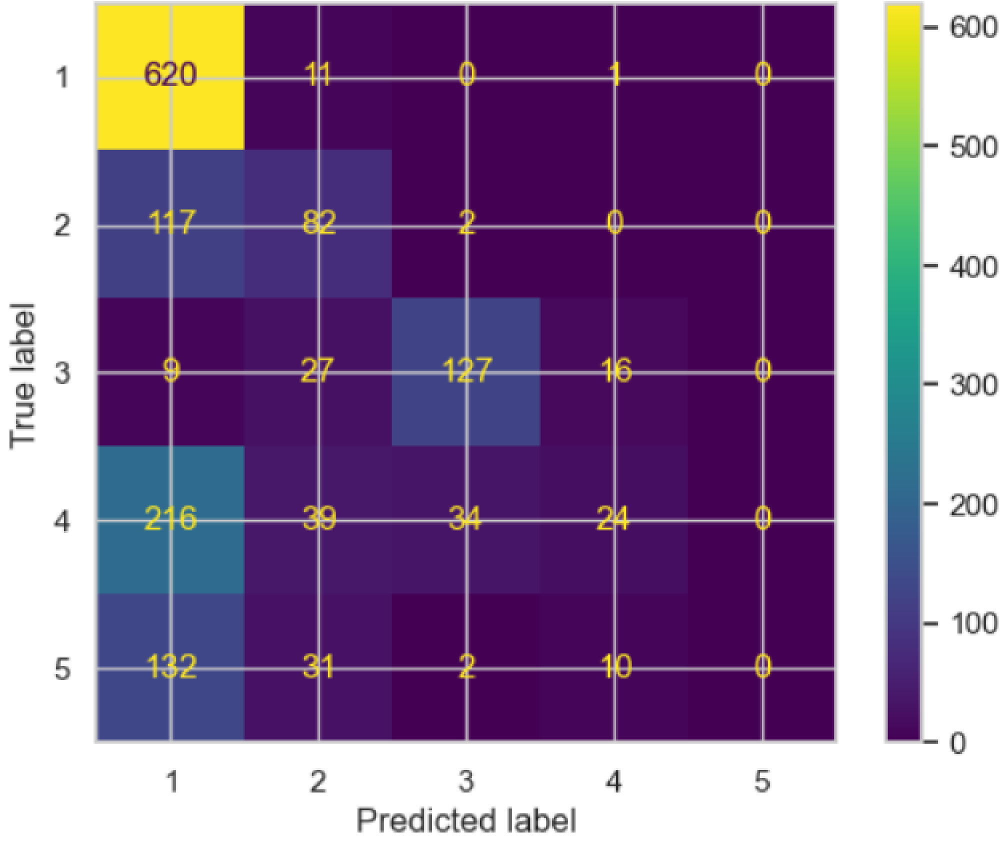
Decision Tree Classifier vs. Support Vector Classifier

Model evaluation

Decision Tree



Support Vector Machine



Decision Tree Classifier vs. Support Vector Classifier

GridSearch

Decision Tree

```
print(dt_model.get_depth())
print(dt_model.get_n_leaves())
```

```
24
1189
```

```
param_grid_dt = {'criterion': ['gini', 'entropy'], 'max_depth': [1,2,3,4,5,6,7,8,9,10]}
estimator_dt = DecisionTreeClassifier(random_state=2022)
grid_dt = GridSearchCV(estimator_dt, param_grid_dt, refit=True, verbose=2)
```

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

```
Fitting 5 folds for each of 20 candidates, totalling 100 fits
[CV] END .......................criterion=gini, max_depth=1; total time=   0.0s
[CV] END .......................criterion=gini, max_depth=1; total time=   0.0s
[CV] END .......................criterion=gini, max_depth=1; total time=   0.0s
[CV] END .......................criterion=gini, max_depth=1; total time=   0.0s
[CV] END .......................criterion=gini, max_depth=1; total time=   0.0s
...
```

```
▸ GridSearchCV
▸ estimator: DecisionTreeClassifier
  ▸ DecisionTreeClassifier
```

Support Vector Machine

```
param_grid_svc = {'C': [0.1,1, 10, 100], 'gamma': [1,0.1,0.01,0.001], 'kernel': ['rbf']}
estimator_svc = SVC(random_state=2022)
grid_svc = GridSearchCV(estimator_svc, param_grid_svc, refit=True, verbose=2)
```

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

```
Fitting 5 folds for each of 16 candidates, totalling 80 fits
[CV] END .........................C=0.1, gamma=1, kernel=rbf; total time=   1.3s
[CV] END .........................C=0.1, gamma=1, kernel=rbf; total time=   1.4s
[CV] END .........................C=0.1, gamma=1, kernel=rbf; total time=   1.5s
[CV] END .........................C=0.1, gamma=1, kernel=rbf; total time=   1.3s
[CV] END .........................C=0.1, gamma=1, kernel=rbf; total time=   1.3s
[CV] END .......................C=0.1, gamma=0.1, kernel=rbf; total time=   1.0s
[CV] END .......................C=0.1, gamma=0.1, kernel=rbf; total time=   1.0s
[CV] END .......................C=0.1, gamma=0.1, kernel=rbf; total time=   1.0s
[CV] END .......................C=0.1, gamma=0.1, kernel=rbf; total time=   1.1s
[CV] END .......................C=0.1, gamma=0.1, kernel=rbf; total time=   1.0s
...
```

```
▸ GridSearchCV
▸ estimator: SVC
  ▸ SVC
```

Decision Tree Classifier vs. Support Vector Classifier

GridSearch

Decision Tree

	precision	recall	f1-score	support
1	0.68	0.93	0.78	632
2	0.55	0.71	0.62	201
3	0.79	0.75	0.77	179
4	0.52	0.22	0.31	313
5	0.40	0.16	0.23	175
accuracy			0.64	1500
macro avg	0.59	0.55	0.54	1500
weighted avg	0.61	0.64	0.60	1500

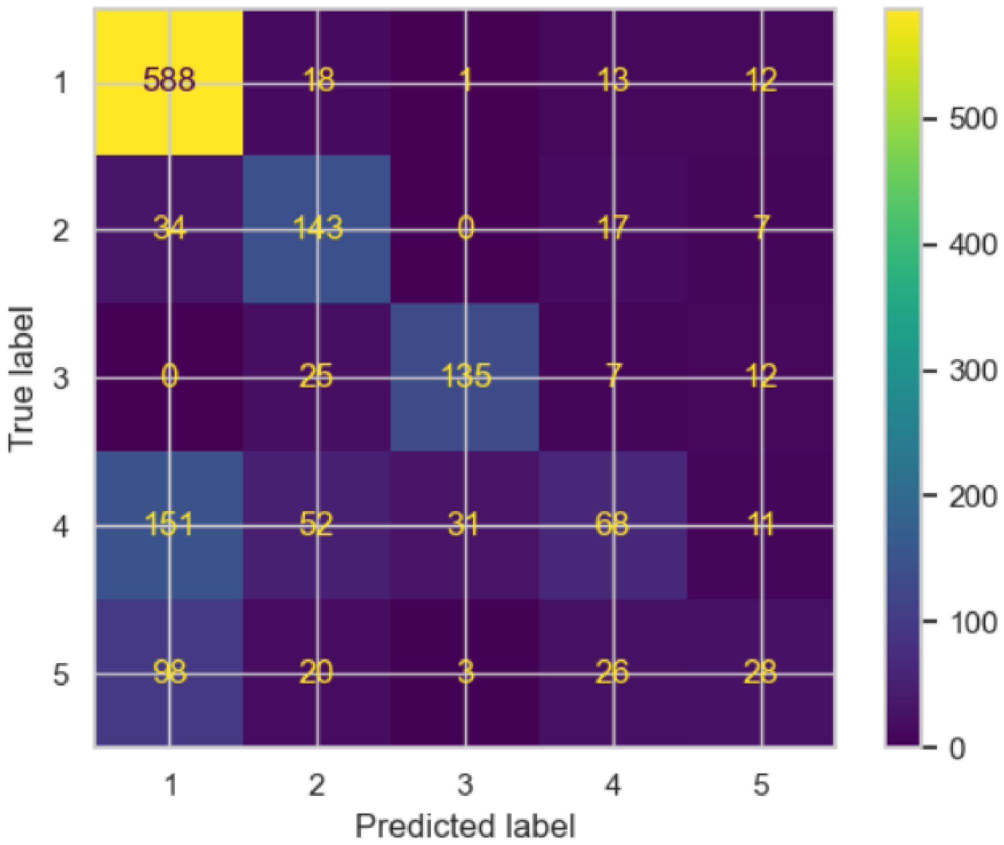
Support Vector Machine

	precision	recall	f1-score	support
1	0.70	0.88	0.78	632
2	0.57	0.46	0.51	201
3	0.59	0.82	0.68	179
4	0.44	0.32	0.37	313
5	0.38	0.15	0.21	175
accuracy			0.61	1500
macro avg	0.54	0.53	0.51	1500
weighted avg	0.58	0.61	0.58	1500

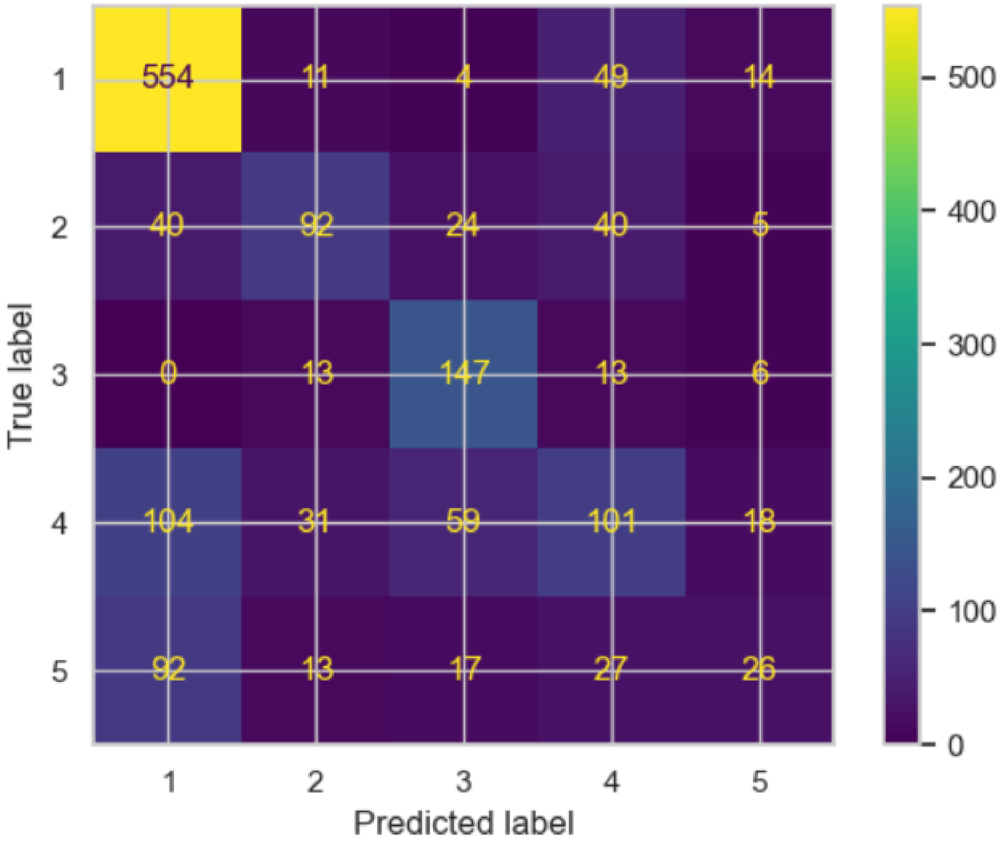
Decision Tree Classifier vs. Support Vector Classifier

GridSearch

Decision Tree



Support Vector Machine



Decision Tree Classifier and Pruning

A Decision Tree is pruned by replacing an entire subtree with a leaf node. If the expected error rate in the subtree is higher than that of a single leaf, it is replaced.

```
print(dt_model.get_depth())
print(dt_model.get_n_leaves())
```

```
24
1189
```

Best Depth Tree

```
max_depth = dt_model.get_depth()
max_depth
```

```
24
```

```
param_grid = {'max_depth': [max_depth for max_depth in range(1, max_depth + 1)]}
```

```
estimator = DecisionTreeClassifier(random_state=42)
```

```
max_depth_grid_search = GridSearchCV(estimator, param_grid)
```

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

```
▸ GridSearchCV
▸ estimator: DecisionTreeClassifier
    ▸ DecisionTreeClassifier
```

Decision Tree Classifier and Pruning

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

```
max_depth_grid_search.best_params_
```

```
{'ccp_alpha': 0.0008316875137151618}
```

```
max_depth_tree = max_depth_grid_search.best_estimator_
print(max_depth_tree)
```

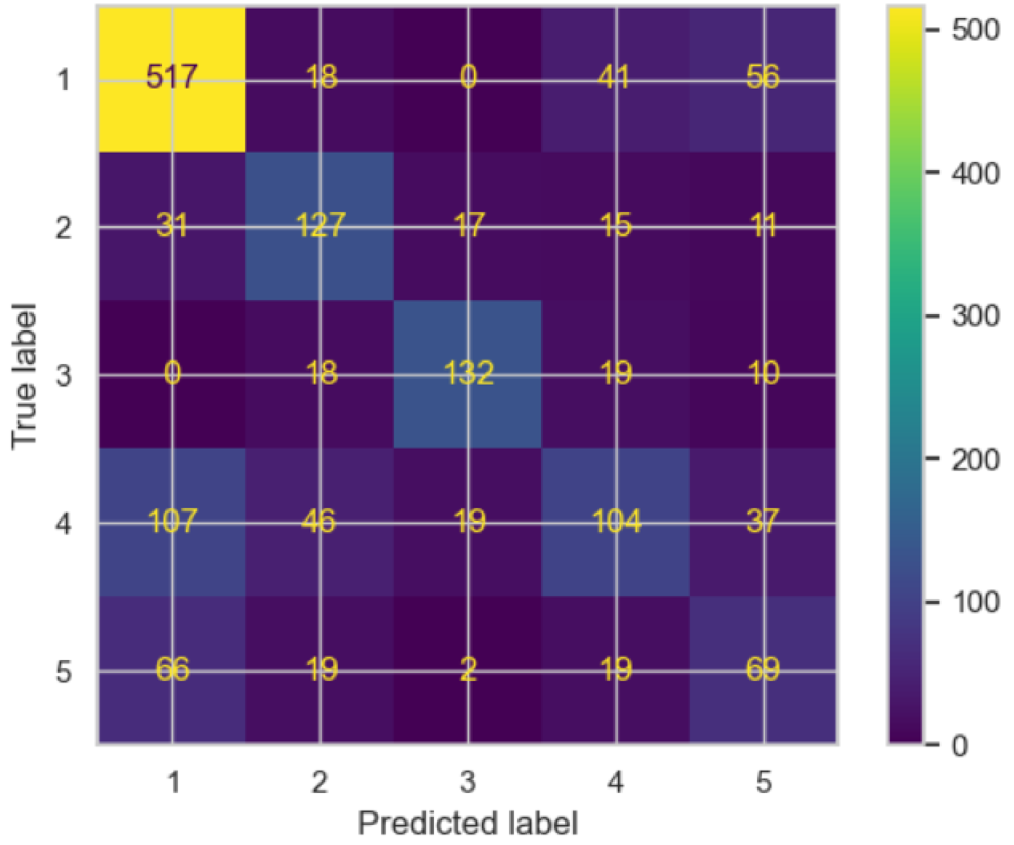
```
DecisionTreeClassifier(ccp_alpha=0.0008316875137151618, random_state=42)
```

```
best_max_depth = max_depth_tree.get_depth()
print(best_max_depth)
```

```
17
```

Model evaluation

	precision	recall	f1-score	support
1	0.72	0.82	0.76	632
2	0.56	0.63	0.59	201
3	0.78	0.74	0.76	179
4	0.53	0.33	0.41	313
5	0.38	0.39	0.39	175
accuracy			0.63	1500
macro avg	0.59	0.58	0.58	1500
weighted avg	0.62	0.63	0.62	1500



Decision Tree Classifier and Pruning

Cost Complexity Pruning

Another pruning technique

```
ccp_alphas = dt_model.cost_complexity_pruning_path(X_train, y_train)["ccp_alphas"]
```

```
ccp_alphas
```

```
array([0.00000000e+00, 1.90476190e-05, 2.38095238e-05, 3.78151261e-05,
       4.76190476e-05, 6.66666667e-05, 7.14285714e-05, 7.14285714e-05,
       7.61904762e-05, 7.61904762e-05, 8.57142857e-05, 9.52380952e-05,
       9.52380952e-05, 9.52380952e-05, 9.52380952e-05, 9.52380952e-05,
       1.08843537e-04, 1.14285714e-04, 1.22448980e-04, 1.36645963e-04,
       1.39097744e-04, 1.40394089e-04, 1.42857143e-04, 1.58730159e-04,
       1.59663866e-04, 1.61904762e-04, 1.63265306e-04, 1.70068027e-04,
       1.71428571e-04, 1.71428571e-04, 1.71428571e-04, 1.72397220e-04,
       1.74603175e-04, 1.74603175e-04, 1.75824176e-04, 1.77777778e-04,
       1.79271709e-04, 1.80952381e-04, 1.82539683e-04, 1.82857143e-04,
       1.90476190e-04, 1.90476190e-04, 1.90476190e-04, 1.90476190e-04,
       1.90476190e-04, 1.90476190e-04, 1.90476190e-04, 1.90476190e-04,
       1.90476190e-04, 1.90476190e-04, 1.90476190e-04, 1.90476190e-04,
       2.11640212e-04, 2.11640212e-04, 2.11764706e-04, 2.14285714e-04,
       2.14285714e-04, 2.14285714e-04, 2.14285714e-04, 2.14285714e-04,
       2.14285714e-04, 2.14285714e-04, 2.14285714e-04, 2.14285714e-04,
       2.14285714e-04, 2.14285714e-04, 2.14285714e-04, 2.14285714e-04,
       2.14285714e-04, 2.14285714e-04, 2.15873016e-04, 2.27106227e-04,
       2.27891156e-04, 2.28571429e-04, 2.28571429e-04, 2.28571429e-04,
       2.28571429e-04, 2.28571429e-04, 2.28571429e-04, 2.28571429e-04,
       2.28571429e-04, 2.28571429e-04, 2.28571429e-04, 2.33333333e-04,
       2.33486943e-04, 2.38095238e-04, 2.38095238e-04, 2.38095238e-04,
        ...
```

Decision Tree Classifier and Pruning


```
estimator.get_params().keys()
```

```
dict_keys(['ccp_alpha', 'class_weight', 'criterion', 'max_depth', 'max_feature
s', 'max_leaf_nodes', 'min_impurity_decrease', 'min_samples_leaf', 'min_sample
s_split', 'min_weight_fraction_leaf', 'random_state', 'splitter'])
```

```
param_grid = {'ccp_alpha': [alpha for alpha in ccp_alphas]}
```

```
estimator_dt = DecisionTreeClassifier(random_state=42)
ccp_alpha_grid_search = GridSearchCV(estimator_dt, param_grid)
```

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

```
▸ GridSearchCV
▸ estimator: DecisionTreeClassifier
  ▸ DecisionTreeClassifier
```

```
ccp_alpha_grid_search.best_params_
```

```
{'ccp_alpha': 0.0008316875137151618}
```

```
best_ccp_alpha_tree = ccp_alpha_grid_search.best_estimator_
print(best_ccp_alpha_tree)
```

```
DecisionTreeClassifier(ccp_alpha=0.0008316875137151618, random_state=42)
```

Decision Tree Classifier and Pruning

Model evaluation

	precision	recall	f1-score	support
1	0.72	0.82	0.76	632
2	0.56	0.63	0.59	201
3	0.78	0.74	0.76	179
4	0.53	0.33	0.41	313
5	0.38	0.39	0.39	175
accuracy			0.63	1500
macro avg	0.59	0.58	0.58	1500
weighted avg	0.62	0.63	0.62	1500