CME 466: Design of a Digital System

Winter 2023

Machine Learning – Classification [part 2]

KHAN A. WAHID, PhD, PEng, SMIEEE

Professor
Electrical and Computer Engineering
University of Saskatchewan
Email: khan.wahid@usask.ca

1. Classification using Support Vector Machine (SVM)

- Support vector machines (SVMs) are a set of supervised learning methods used for classification, regression and outlier detection[1]. SVM offers very high accuracy compared to other classifiers such as, logistic regression and decision trees.

- SVM is effective in high dimensional spaces and handling of nonlinear input spaces. Many applications such as face detection, intrusion detection, classification of emails, news articles and web pages, classification of genes, handwriting recognition use SVM as the classifier.

- It uses a set of mathematical functions that are defined as the kernel. The kernel function takes the input data and transforms it into the required form (when needed), where a hyperplane can be drawn to separate the classes.

- Different SVM algorithms use different types of kernel functions. For example: linear, nonlinear, polynomial, radial basis function (RBF), and sigmoid.

$$Linear: K(x_1, x_2) = \sum x_1 . x_2 \qquad Polynomial: K(x_1, x_2) = (a x_1 . x_2 + b)^c \qquad RBF: K(x_1, x_2) = e^{-(\gamma \|x_1 - x_2\|^2)}$$

- One disadvantage of SVM is if the number of features is much greater than the number of samples, it tends to over-fit the model. Therefore, choosing a proper kernel and regularization term is crucial.

- Some examples of different kernels in SVM are given below:

  https://scikit-learn.org/stable/auto_examples/svm/plot_iris_svc.html#sphx-glr-auto-examples-svm-plot-iris-svc-py
  https://scikit-learn.org/stable/auto_examples/svm/plot_svm_kernels.html

[1] https://scikit-learn.org/stable/modules/svm.html

- Creating and instantiating an SVM is similar to other classifiers we have seen so far in scikit-learn. When used as a classifier, it is referred to specifically as SVC in sklearn.svm[2].

  Let us look at some examples of SVM classifier with different kernels.

1.1 SVM on Synthetic Datasets (generated by the user)

- In this section, we will work with synthetic datasets, i.e., generated according to a particular statistical distribution. This is in contrast to previous examples, where we utilized an experimental dataset.

- Synthetic datasets are very useful to study algorithm behavior in a controlled manner, when real datasets are not available, or of insufficient quantity/quality.

- In sklearn, we can use *make_blobs()* to generate pseudo-random datasets with desirable statistics[3]. Here, the *centers* parameter controls the number of classes in the dataset. Parameter *n_features* (default is 2) controls the number of features (i.e., number of columns in X).

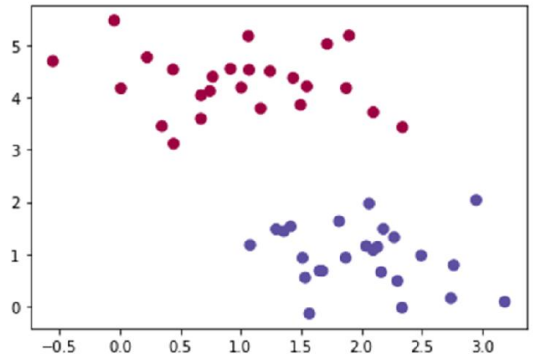
```
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
import seaborn as sns

from sklearn.datasets import make_blobs
X, y = make_blobs(n_samples=50, centers=2, random_state=0, cluster_std=0.6)
```

[2] https://scikit-learn.org/stable/modules/generated/sklearn.svm.SVC.html
[3] https://scikit-learn.org/stable/modules/generated/sklearn.datasets.make_blobs.html

```
print(X.shape,y.shape)

plt.scatter(X[:,0],X[:,1],c=y,s=50,cmap='Spectral');
```



- It can be seen from the plot that the dataset we just generated is linearly separable. Therefore, we can use algorithms like K-Nearest Neighbor, Logistic Regression, or SVM to classify it.
- You can play with the parameters of *make_blobs()* to better understand them.

- Let us now split the dataset, but take only 6% of it (i.e., only 3 samples) for training and see what happens.

```
from sklearn import metrics
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=.94,
random_state=0)
print(X_train.shape, X_test.shape, y_train.shape, y_test.shape)

from sklearn.svm import SVC
clf_svc = SVC(kernel='linear')
# train with only the training set
clf_svc.fit(X_train,y_train)

# test with only the testing set
print(metrics.accuracy_score(y_test, clf_svc.predict(X_test)))
```

```
1.0
```

- Do you notice what happened? We used only 6% of the data for training (i.e., only 3 samples for training), and the SVM model is able to predict the remaining test data (i.e., 47 samples) with 100% accuracy. How about other models?

```
# Logistic Regression
from sklearn.linear_model import LogisticRegression
clf_lr = LogisticRegression()
clf_lr.fit(X_train,y_train)
print(metrics.accuracy_score(y_test,clf_lr.predict(X_test)))

# kNN
from sklearn.neighbors import KNeighborsClassifier
clf_knn = KNeighborsClassifier(n_neighbors=3)
clf_knn.fit(X_train,y_train)
print(metrics.accuracy_score(y_test,clf_knn.predict(X_test)))
```

- What observations can you make regarding instance-based (such as, kNN) vs. model-based (LR or SVM) learning, given these results?

- Let us check the decision boundary plot.

```
# general code to plot decision boundary
from matplotlib.colors import ListedColormap
# Create color maps
cmap_light = ListedColormap(["orange", "cyan", "cornflowerblue"])
cmap_bold = ListedColormap(["darkorange", "c", "darkblue"])
```

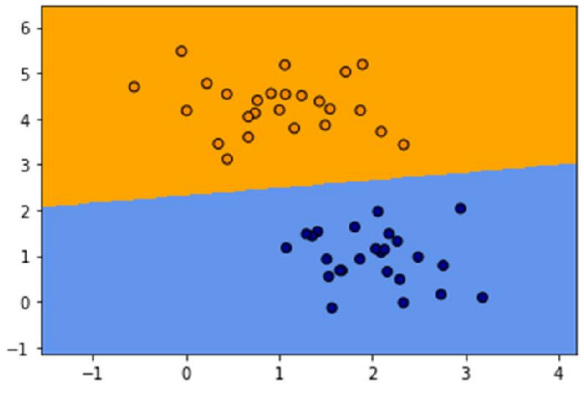
```
h = .02
# change the clf model
clf_svc.fit(X, y)

# Plot the decision boundary. For that, we will assign a color to each
# point in the mesh [x_min, x_max]x[y_min, y_max].
x_min, x_max = X[:, 0].min() - 1, X[:, 0].max() + 1
y_min, y_max = X[:, 1].min() - 1, X[:, 1].max() + 1
xx, yy = np.meshgrid(np.arange(x_min, x_max, h), np.arange(y_min, y_max, h))

# change the clf model
Z = clf_svc.predict(np.c_[xx.ravel(), yy.ravel()])

# Put the result into a color plot
Z = Z.reshape(xx.shape)
plt.figure()
plt.pcolormesh(xx, yy, Z, cmap=cmap_light)

# Plot also the training points
plt.scatter(X[:, 0], X[:, 1], c=y, cmap=cmap_bold,
edgecolor="k");
plt.xlim(xx.min(), xx.max())
plt.ylim(yy.min(), yy.max())
```



1.2 Classifier for Non-linear Datasets

- So far, we have considered only the linear version of the SVM. It will outperform many classifiers for a linearly separable dataset. In fact, now we will see the true strength of SVM when it comes to deal with a non-linear dataset.

- We can use *make_circles()* to generate a dataset containing a large circle encompassing a smaller circle in 2D. It is a simple toy dataset to visualize clustering and classification algorithms. We can play with parameters like, *noise* and *factor* to see how they affect the dataset.

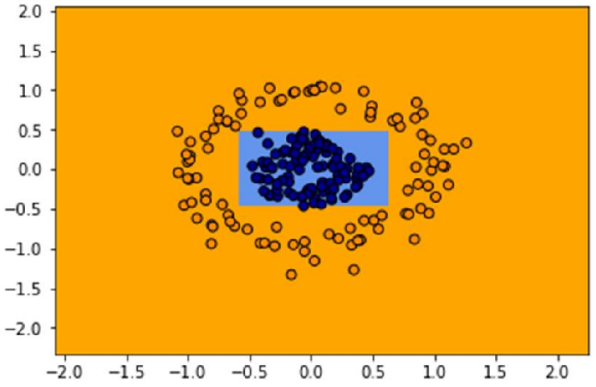
```
from sklearn.datasets import make_circles
X, y = make_circles(200, factor=0.3, noise=0.1)
plt.scatter(X[:, 0], X[:, 1], c=y, cmap='Spectral');
```

1.3 Decision Tree Classifier and Random Forest Classifier

- Decision Trees (DTs) are a non-parametric supervised learning method used for classification and regression. The goal is to create a model that predicts the value of a target variable by learning simple decision rules inferred from the data features.

```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.80,
random_state=0)
print(X_train.shape, X_test.shape, y_train.shape, y_test.shape)
```

```
from sklearn import tree
from sklearn import metrics
clf_dt = tree.DecisionTreeClassifier()
clf_dt_f = clf_dt.fit(X_train,y_train)
print(metrics.accuracy_score(y_test,
clf_dt.predict(X_test)))
```



- We should plot the decision boundary and see the results. Once the model is built, we can plot the tree with the *plot_tree()* function.

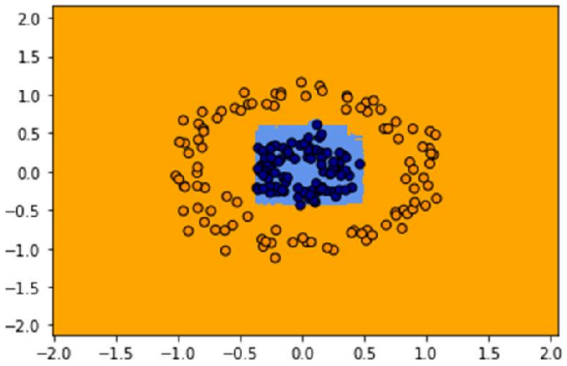
```
tree.plot_tree(clf_dt_f)
from sklearn.metrics import confusion_matrix
confusion_matrix(y_test, clf_dt.predict(X_test))
```

```
array([[58, 20],
       [ 0, 82]], dtype=int64)
```

- DTs are simple to understand and can be easily visualized. However, it may overfit the data by creating a complex tree. Predictions in DTs are neither smooth nor continuous, but piecewise constant approximations as seen in the above figure. Therefore, they are not good at extrapolation.

- In addition, DTs can be unstable because small variations in the data might result in a completely different tree being generated. This problem is mitigated by using decision trees within an ensemble, known as Random Forest.

- A Random Forest is an ensembled estimator that fits a number of decision tree classifiers on various sub-samples of the dataset and uses averaging to improve the predictive accuracy and control over-fitting[4]. A proper discussion on Random Forest classifier is beyond the scope of CME466. Here, we will only use it to assess the comparative performance of several classifiers.




```
# random forest classifier
from sklearn.ensemble import RandomForestClassifier
clf_rf = RandomForestClassifier(n_estimators=50)
clf_rf.fit(X_train, y_train)
print(metrics.accuracy_score(y_test,clf_rf.predict(X_test)))
```

- Interesting, if you run the above code of random forest algorithm several times, it is possible to get a different accuracy value each time. It is due to the randomness of the algorithm.

1.4 SVM Classifier with RBF Kernel

- Lastly, in this section, we will go back to SVM and try out a Radial Basis Function (RBF) kernel. At the end , we will compare the performance of SVC, DTs and RF algorithms.

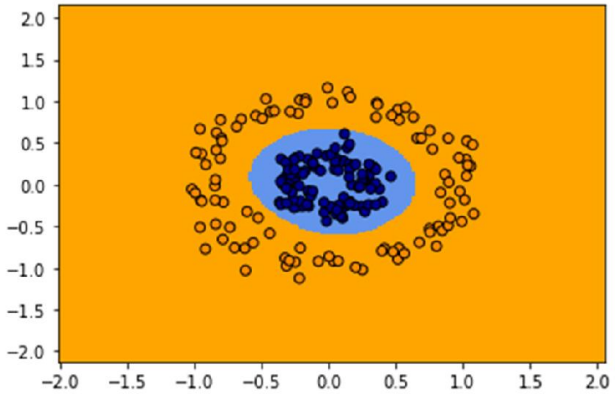
- There are many parameters that can be used to optimize the Radial Basis Function (RBF) kernel[5]. A low *c* (regularization parameter) makes the decision surface smooth, while a high *c* aims at classifying all training examples correctly.

[4] https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestClassifier.html
[5] https://scikit-learn.org/stable/auto_examples/svm/plot_rbf_parameters.html#sphx-glr-auto-examples-svm-plot-rbf-parameters-py

- On the other hand, *gamma* (kernel coefficient with default is 1.0) defines how much influence a single training example has. The larger the *gamma* is, the closer other examples must be to be affected.

```
from sklearn.svm import SVC
clf_svc = SVC(kernel='rbf')
clf_svc.fit(X_train,y_train)
print(metrics.accuracy_score(y_test,
clf_svc.predict(X_test)))

confusion_matrix(y_test, clf_svc.predict(X_test))
```



2. Classification using Digit Dataset

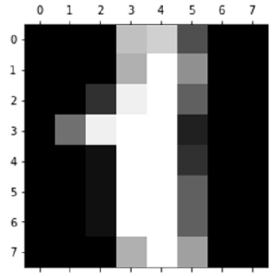
- In this section, we will evaluate the performance of some of the classifiers to recognize digital numbers optically using a "handwritten digits dataset" from the Toy dataset of scikit-learn[6]. The data set contains 1,797 gray scale images of hand-written digits: 10 classes where each class refers to a digit.

- Each image is of 8x8 resolution with integer pixel values scaled between 0 to 16, where '0' means black and '16' means white. All other values in between represent different shades of gray. Since each digit is an 8x8 matrix, there are 64 attributes assigned to each digit.

- **Pixel in an image:** In digital imaging, a pixel (or pel or picture element) is the smallest addressable element in a raster image. It is also the smallest element that can be processed by a computer program or software.

[6] https://scikit-learn.org/stable/datasets/toy_dataset.html#optical-recognition-of-handwritten-digits-dataset

- Recognizing handwritten text is a problem that can be traced long time back to the early 20th century by an automatic machine. OCR (Optical Character Recognition) software is a classic example.



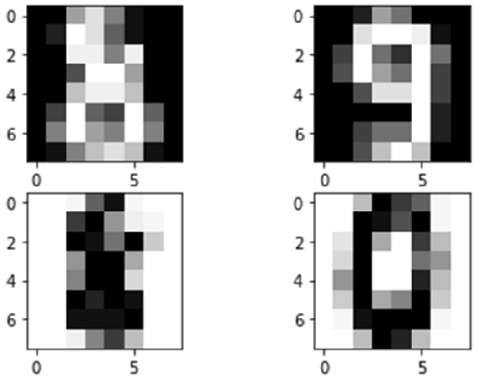
```
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
import seaborn as sns

from sklearn.datasets import load_digits
digits = load_digits()
print(digits.images.shape)
print(np.unique(digits.target))
print(digits.images[1])
plt.matshow(digits.images[1], cmap='gray');
```

```
[[ 0.  0.  0. 12. 13.  5.  0.  0.]
 [ 0.  0.  0. 11. 16.  9.  0.  0.]
 [ 0.  0.  3. 15. 16.  6.  0.  0.]
 [ 0.  7. 15. 16. 16.  2.  0.  0.]
 [ 0.  0.  1. 16. 16.  3.  0.  0.]
 [ 0.  0.  1. 16. 16.  6.  0.  0.]
 [ 0.  0.  1. 16. 16.  6.  0.  0.]
 [ 0.  0.  0. 11. 16. 10.  0.  0.]]
```

- We can plot some sample digits and see how they look.

```
plt.subplot(221)
plt.imshow(digits.images[1796], cmap='gray')
plt.subplot(222)
plt.imshow(digits.images[1795], cmap='gray')
plt.subplot(223)
plt.imshow(digits.images[1794], cmap=plt.cm.Greys)
plt.subplot(224)
plt.imshow(digits.images[1793], cmap=plt.cm.Greys)
```



- The first two images have been plotted using a typical gray scale colormap. The other two are the compliment (just to show how it can be done; here '0' is white and '16' is black).

- Now we will apply some classifiers that we tried before to train the dataset. We will also print the first few digits from the test dataset to see the prediction.

```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(digits.data, digits.target,
test_size=0.4, random_state=0)
print(X_train.shape, X_test.shape, y_train.shape, y_test.shape)

from sklearn import tree
from sklearn import metrics

# decision tree classifier
clf_dt = tree.DecisionTreeClassifier()
clf_dt_f = clf_dt.fit(X_train,y_train)
print(metrics.accuracy_score(y_test, clf_dt.predict(X_test)))

# print first 40 entries and see the error
print(np.vstack([clf_dt.predict(X_test[0:40,:]), y_test[0:40]]))

[[2 8 2 6 6 7 1 9 8 5 2 8 6 6 6 6 1 0 5 8 8 7 8 4 7 5 4 9 2 9 4 7 6 8 9 4
  3 8 0 9]
 [2 8 2 6 6 7 1 9 8 5 2 8 6 6 6 6 1 0 5 8 8 7 8 4 7 5 4 9 2 9 4 7 6 8 9 4
  3 1 0 1]]


from sklearn.metrics import confusion_matrix
```

```
confusion_matrix(y_test, clf_dt.predict(X_test))
```

```
array([[59,  0,  0,  0,  0,  1,  0,  0,  0,  0],
       [ 1, 59,  2,  1,  0,  0,  1,  2,  4,  3],
       [ 1,  3, 53,  3,  1,  0,  0,  1,  7,  2],
       [ 0,  0,  1, 60,  0,  4,  1,  1,  0,  3],
       [ 1,  1,  0,  0, 53,  2,  0,  3,  2,  1],
       [ 0,  2,  0,  0,  3, 76,  0,  0,  8,  0],
       [ 1,  0,  0,  0,  0,  1, 73,  1,  0,  0],
       [ 0,  0,  0,  0,  1,  0,  0, 61,  1,  2],
       [ 0, 12,  4,  4,  1,  1,  0,  3, 45,  8],
       [ 1,  2,  0,  3,  0,  3,  0,  2,  1, 62]], dtype=int64)
```

```
# random forest classifier
from sklearn.ensemble import RandomForestClassifier
clf_rf = RandomForestClassifier(n_estimators=50)
clf_rf.fit(X_train, y_train)
print(metrics.accuracy_score(y_test,clf_rf.predict(X_test)))

# SVM different kernels (linear and rbf)
from sklearn.svm import SVC
clf_svc = SVC(kernel='rbf')
clf_svc.fit(X_train,y_train)
print(metrics.accuracy_score(y_test, clf_svc.predict(X_test)))
```

- Therefore, it can be concluded that different machine learning models (even with only 50% to 60% training data) can achieve an accuracy of 96-98% on the handwritten dataset.

- One thing we should note here is that for the digit dataset, one sample (or digit) has 64 features. This is too many. Later we will see how to apply processing techniques, like Principal Component Analysis (PCA) to reduce the dimensionality of the data, so that using only a few features (2-5), we can achieve similar performance.

- More examples can be found here: https://scikit-learn.org/stable/auto_examples/classification/plot_digits_classification.html