Statistical Machine Learning with Python Week #3

Ching-Shih Tsou (Ph.D.), Distinguished Prof. at the Department of Mechanical Engineering/Director of the Center for Artifical Intelligence & Data Science, Ming Chi University of Technology

September/2020 at MCUT

Collecting Data

- Image processing is a difficult task for many types of machine learning algorithms. The relationships linking patterns of pixels to higher concepts are extremely complex and hard to define. For instance, it's easy for a human being to recognize a face, a cat, or the letter "A", but defining these patterns in strict rules is difficult. Furthermore, image data is often *noisy*. There can be many slight variations in how the image was captured, depending on the lighting, orientation, and positioning of the subject.
- According to the documentation provided by Frey and Slate (1991)(http://archive.ics.uci.edu/ml (http://archive.ics.uci.edu/ml)), when the glyphs are scanned into the computer, they are converted into pixels and 16 statistical attributes are recorded.
- The attributes measure such characteristics as the horizontal and vertical dimensions of the glyph, the proportion of black (versus white) pixels, and the average horizontal and vertical position of the pixels.
- Presumably, differences in the concentration of black pixels across various areas of the box should provide a way to differentiate among the 26 letters of the alphabet.

```
import pandas as pd
letters = pd.read_csv("letterdata.csv")
print(letters.dtypes)
```

```
## letter    object
## xbox       int64
## ybox       int64
## width      int64
## height     int64
## onpix      int64
## xbar       int64
## ybar       int64
## x2bar      int64
## y2bar      int64
## xybar      int64
## x2ybar     int64
## xy2bar     int64
## xedge      int64
## xedgey     int64
## yedge      int64
## yedgex     int64
## dtype: object
```

```
print(letters.shape)
```

```
## (20000, 17)
```

Exploring and Preparing the Data

```
print(letters.describe(include = 'all'))
```

```
##        letter          xbox  ...         yedge       yedgex
## count   20000  20000.000000  ...  20000.000000  20000.00000
## unique     26           NaN  ...           NaN          NaN
## top         U           NaN  ...           NaN          NaN
## freq      813           NaN  ...           NaN          NaN
## mean      NaN      4.023550  ...      3.691750      7.80120
## std       NaN      1.913212  ...      2.567073      1.61747
## min       NaN      0.000000  ...      0.000000      0.00000
## 25%       NaN      3.000000  ...      2.000000      7.00000
## 50%       NaN      4.000000  ...      3.000000      8.00000
## 75%       NaN      5.000000  ...      5.000000      9.00000
## max       NaN     15.000000  ...     15.000000     15.00000
##
## [11 rows x 17 columns]
```

```
print(letters['letter'].value_counts())
```

```
## U    813
## D    805
## P    803
## T    796
## M    792
## A    789
## X    787
## Y    786
## N    783
## Q    783
## F    775
## G    773
## E    768
## B    766
## V    764
## L    761
## R    758
## I    755
## O    753
## W    752
## S    748
## J    747
## K    739
## C    736
## H    734
## Z    734
## Name: letter, dtype: int64
```

```
import numpy as np
from sklearn.feature_selection import VarianceThreshold
vt = VarianceThreshold(threshold=0)
print(vt.fit_transform(letters.iloc[:,1:]).shape)
```

```
## (20000, 16)
```

```
print(np.sum(vt.get_support() == False)) # Get a mask of the features selected
```

```
## 0
```

```
print(vt.get_support(indices=True)) # Get integer index of the features select
ed
```

```
## [ 0  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15]
```

```
cor = letters.iloc[:,1:].corr().values
print(cor[:5,:5])
```

```
## [[1.         0.7577928  0.851514   0.67276367 0.61909688]
##  [0.7577928  1.         0.67191188 0.82320706 0.55506655]
##  [0.851514   0.67191188 1.         0.66021536 0.76571612]
##  [0.67276367 0.82320706 0.66021536 1.         0.64436627]
##  [0.61909688 0.55506655 0.76571612 0.64436627 1.        ]]
```

```
import numpy as np
np.fill_diagonal(cor, 0)
threTF = abs(cor) > 0.8
print(threTF[:5,:5])
```

```
## [[False False  True False False]
##  [False False False  True False]
##  [ True False False False False]
##  [False  True False False False]
##  [False False False False False]]
```

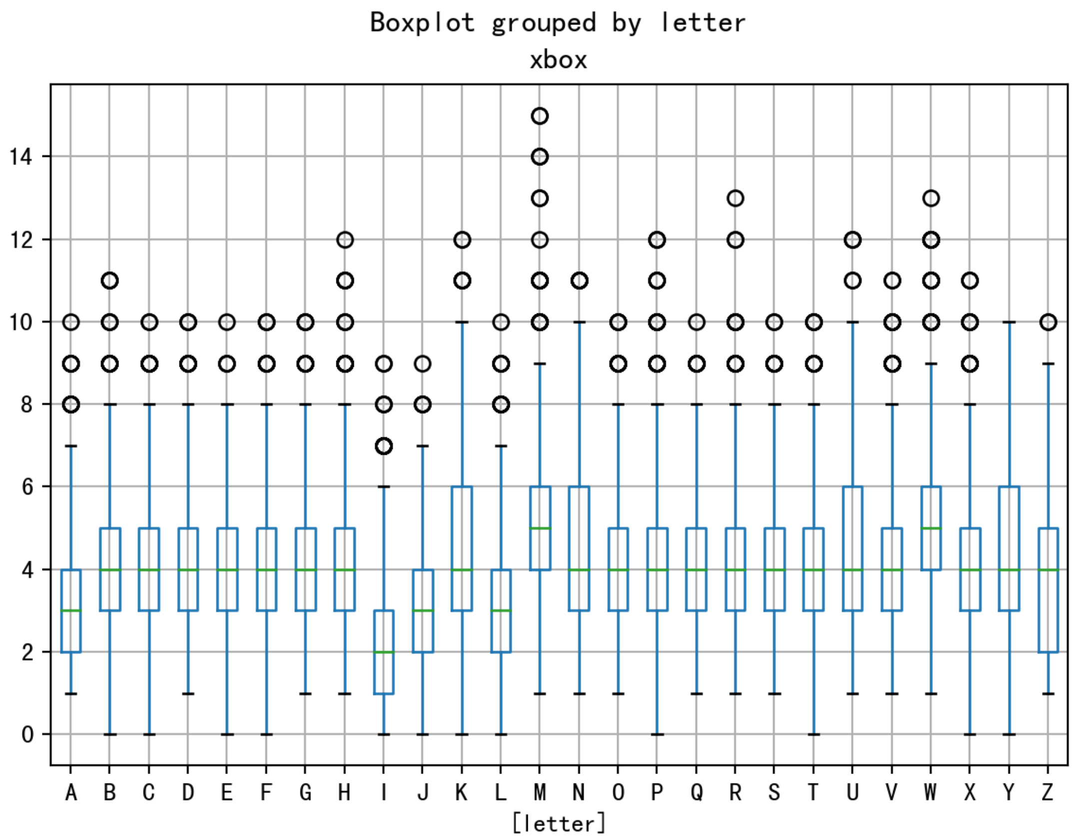
```
print(np.argwhere(threTF == True))
```

```
## [[0 2]
##  [1 3]
##  [2 0]
##  [3 1]]
```

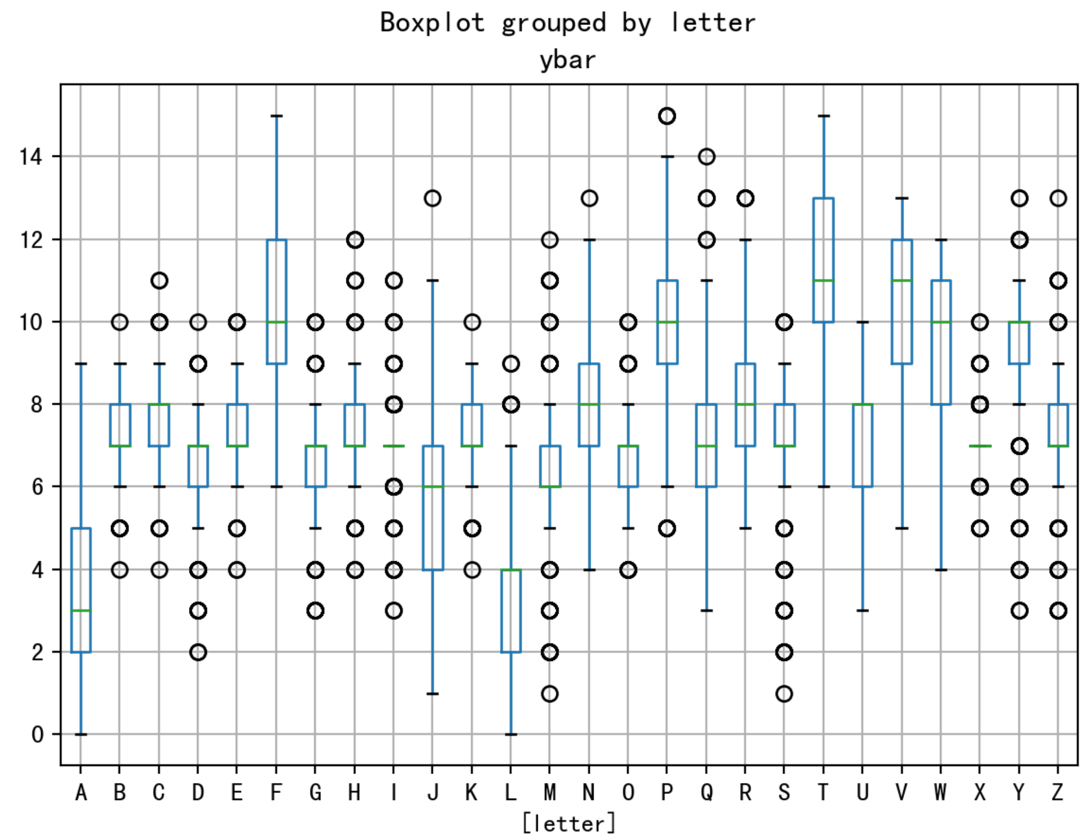
```
print(letters.columns[1:5])
```

```
## Index(['xbox', 'ybox', 'width', 'height'], dtype='object')
```

```
import matplotlib.pyplot as plt
ax1 = letters[['xbox', 'letter']].boxplot(by = 'letter')
fig1 = ax1.get_figure()
plt.show()
# fig1.savefig('./_img/xbox_boxplot.png')
```




```
ax2 = letters[['ybar', 'letter']].boxplot(by = 'letter')
fig2 = ax2.get_figure()
plt.show()
# fig2.savefig('./_img/ybar_boxplot.png')
```



- Recall that SVM learners require all features to be numeric, and moreover, that each feature is scaled to a fairly small interval. In this case, every feature is an integer, so we do not need to convert any factors into numbers. On the other hand, some of the ranges for these integer variables appear fairly wide. This indicates that we need to normalize or standardize the data.
- Given that the data preparation has been largely done for us, we can move directly to the training and testing phases of the machine learning process. In the previous analyses, we randomly divided the data between the training and testing sets.
- Although we could do so here, Frey and Slate have already randomized the data, and therefore suggest using the first 16,000 records (80 percent) to build the model and the next 4,000 records (20 percent) to test. Following their advice, we can create training and testing data frames as follows:

```
# create training and testing data
from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(
letters.iloc[:, 1:], letters['letter'], test_size=0.2,
random_state=0)
```

```
# StandardScaler
from sklearn.preprocessing import StandardScaler
sc = StandardScaler()
sc.fit(X_train)
```

```
## StandardScaler(copy=True, with_mean=True, with_std=True)
```

```
X_train_std = sc.transform(X_train)
X_test_std = sc.transform(X_test)
```

Training a Model on the Data

```
# SVC: Support Vector Classification
# SVR: Support Vector Regression
# OneClassSVM: (Outlier Detection
from sklearn.svm import SVC
svm = SVC(kernel='linear')
svm.fit(X_train_std, y_train)
```

```
## SVC(C=1.0, cache_size=200, class_weight=None, coef0=0.0,
##     decision_function_shape='ovr', degree=3, gamma='auto_deprecated',
##     kernel='linear', max_iter=-1, probability=False, random_state=None,
##     shrinking=True, tol=0.001, verbose=False)
```

```
tr_pred = svm.predict(X_train_std)
y_pred = svm.predict(X_test_std)
print(tr_pred[:5])
```

```
## ['I' 'M' 'Z' 'D' 'G']
```

```
print(y_train[:5])
```

```
## 17815    I
## 18370    M
## 1379     Z
## 14763    D
## 7346     L
## Name: letter, dtype: object
```

```
print(y_pred[:5])
```

```
## ['Y' 'B' 'K' 'T' 'Q']
```

```
print(y_test[:5].tolist())
```

```
## ['Y', 'B', 'K', 'Y', 'Q']
```

```
err_tr = (y_train != tr_pred).sum()/len(y_train)
print("train set error: %f" % err_tr)
```

```
## train set error: 0.133250
```

```
err = (y_test != y_pred).sum()/len(y_test)
print("test set error: %f" % err)
```

```
## test set error: 0.134500
```

Improving Model Performance

- Our previous SVM model used the simple linear kernel function. By using a more complex kernel function, we can map the data into a higher dimensional space, and potentially obtain a better, probably a simple, model fit.
- It can be challenging, however, to choose from the many different kernel functions. A popular convention is to begin with the Gaussian RBF kernel, which has been shown to perform well for many types of data.

```
svm = SVC(kernel='rbf', random_state=0, gamma=0.2, C=1.0)
svm.fit(X_train_std, y_train)
```

```
## SVC(C=1.0, cache_size=200, class_weight=None, coef0=0.0,
##     decision_function_shape='ovr', degree=3, gamma=0.2, kernel='rbf',
##     max_iter=-1, probability=False, random_state=0, shrinking=True, tol=0.0
01,
##     verbose=False)
```

```
tr_pred = svm.predict(X_train_std)
y_pred = svm.predict(X_test_std)
print(tr_pred[:5])
```

```
## ['I' 'M' 'Z' 'D' 'L']
```

```
print(y_train[:5])
```

```
## 17815    I
## 18370    M
## 1379     Z
## 14763    D
## 7346     L
## Name: letter, dtype: object
```

```
print(y_pred[:5])
```

```
## ['Y' 'B' 'K' 'X' 'Q']
```

```
print(y_test[:5].tolist())
```

```
## ['Y', 'B', 'K', 'Y', 'Q']
```

```
err_tr = (y_train.values != tr_pred).sum()/len(y_train)
print("train set error: %f" % err_tr)
```

```
## train set error: 0.011750
```

```
err = (y_test != y_pred).sum()/len(y_test)
print("test set error: %f" % err)
```

```
## test set error: 0.027500
```

pandas_ml need：

- pip install scikit-learn==0.21.1
- pip install pandas==0.24.2
- pip install pandas_ml

```
import pandas_ml as pdml
cm = pdml.ConfusionMatrix(y_test.values, y_pred)
cm_df = cm.to_dataframe(normalized=False, calc_sum=True,
sum_label='all')
print(cm_df.iloc[:12, :12])
```

```
## Predicted    A    B    C    D    E    F    G    H    I    J    K    L
## Actual
## A          147    0    0    0    0    0    0    0    0    0    0    0
## B            0  153    0    0    0    0    0    0    0    0    0    0
## C            0    0  152    0    0    0    3    0    0    0    0    0
## D            1    1    0  166    0    0    0    2    0    0    0    0
## E            0    1    0    0  141    0    1    0    0    0    0    1
## F            0    1    0    1    0  163    0    0    0    0    0    0
## G            0    1    0    2    0    0  175    0    0    0    0    1
## H            0    1    0    2    0    0    0  111    0    0    2    0
## I            0    0    0    0    0    1    0    0  118    8    0    0
## J            0    0    0    0    0    1    0    0    1  156    0    0
## K            0    0    0    0    0    0    0    3    0    0  136    0
## L            0    0    1    0    1    0    0    0    0    0    1  156
```

```
perf_indx = cm.stats()
```

```
## /opt/anaconda3/lib/python3.7/site-packages/pandas_ml/confusion_matrix/stat
s.py:60: FutureWarning: supplying multiple axes to axis is deprecated and will
be removed in a future version.
##   num = df[df > 1].dropna(axis=[0, 1], thresh=1).applymap(lambda n: choose
(n, 2)).sum().sum() - np.float64(nis2 * njs2) / n2
## /opt/anaconda3/lib/python3.7/site-packages/pandas_ml/confusion_matrix/bcm.p
y:344: RuntimeWarning: divide by zero encountered in double_scalars
##   return(np.float64(self.LRP) / self.LRN)
## /opt/anaconda3/lib/python3.7/site-packages/pandas_ml/confusion_matrix/bcm.p
y:330: RuntimeWarning: divide by zero encountered in double_scalars
##   return(np.float64(self.TPR) / self.FPR)
```

```
print(type(perf_indx))
```

```
## <class 'collections.OrderedDict'>
```

```
print(perf_indx.keys())
```

```
## odict_keys(['cm', 'overall', 'class'])
```

```
print(type(perf_indx['overall']))
# perf_indx['overall'].keys()
```

```
## <class 'collections.OrderedDict'>
```

```
print(" acc:{}".format(perf_indx['overall']
['Accuracy']))
```

```
##  acc:0.9725
```

```
print(" acc95%:\n{}".format(perf_indx
['overall']['95% CI']))
```

```
##  acc95%:
## (0.9669490685534711, 0.9773453558266993)
```

```
print("Kappa:\n{}".format(perf_indx['overall']
['Kappa']))
```

```
## Kappa:
## 0.9713890027910028
```

```
print(type(perf_indx['class']))
```

```
## <class 'pandas.core.frame.DataFrame'>
```

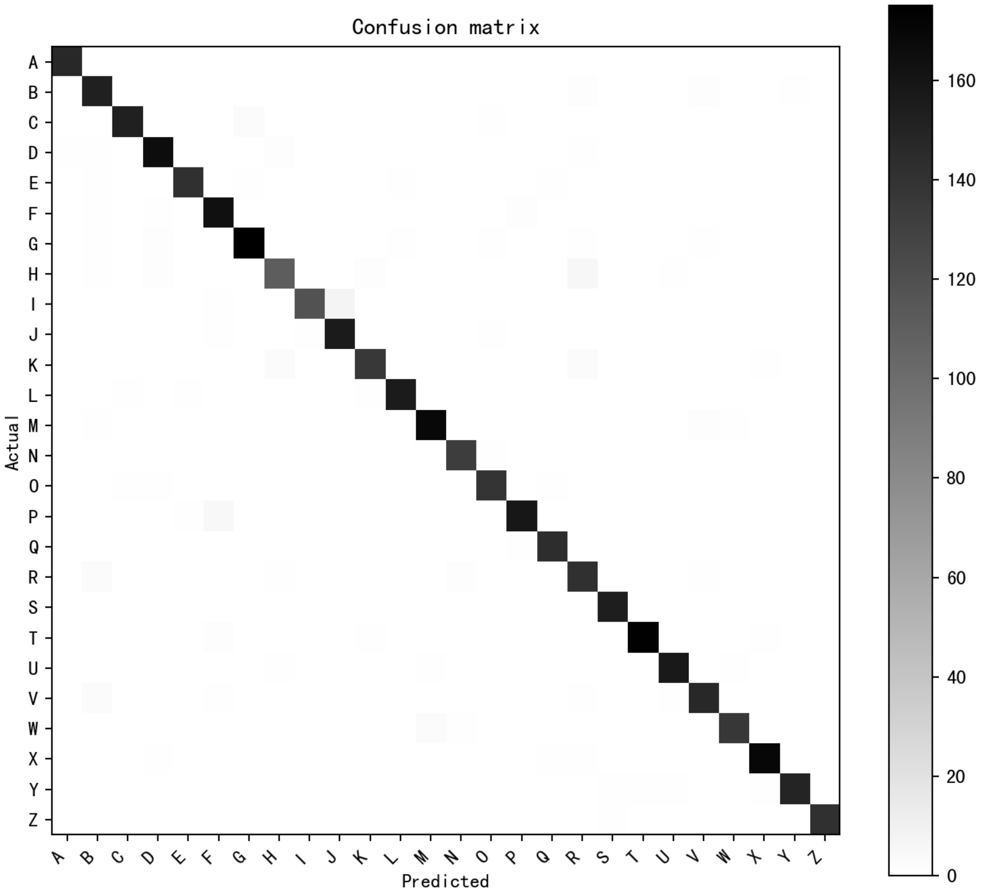
```
print(perf_indx['class'].shape)
```

```
## (26, 26)
```

```
print(perf_indx['class'])
```

```
## Classes                                             A  ...            Z
## Population                                       4000  ...         4000
## P: Condition positive                             147  ...          143
## N: Condition negative                            3853  ...         3857
## Test outcome positive                             148  ...          142
## Test outcome negative                            3852  ...         3858
## TP: True Positive                                 147  ...          142
## TN: True Negative                                3852  ...         3857
## FP: False Positive                                  1  ...            0
## FN: False Negative                                  0  ...            1
## TPR: (Sensitivity, hit rate, recall)                1  ...     0.993007
## TNR=SPC: (Specificity)                        0.99974  ...            1
## PPV: Pos Pred Value (Precision)              0.993243  ...            1
## NPV: Neg Pred Value                                 1  ...     0.999741
## FPR: False-out                            0.000259538  ...            0
## FDR: False Discovery Rate                  0.00675676  ...            0
## FNR: Miss Rate                                      0  ...   0.00699301
## ACC: Accuracy                                 0.99975  ...      0.99975
## F1 score                                      0.99661  ...     0.996491
## MCC: Matthews correlation coefficient        0.996487  ...     0.996368
## Informedness                                  0.99974  ...     0.993007
## Markedness                                   0.993243  ...     0.999741
## Prevalence                                    0.03675  ...      0.03575
## LR+: Positive likelihood ratio                   3853  ...          inf
## LR-: Negative likelihood ratio                      0  ...   0.00699301
## DOR: Diagnostic odds ratio                        inf  ...          inf
## FOR: False omission rate                            0  ...  0.000259202
##
## [26 rows x 26 columns]
```

```
import matplotlib.pyplot as plt
ax = cm.plot()
fig = ax.get_figure()
plt.show()
# fig.savefig('./_img/svc_rbf.png')
```


Confusion matrix
Actual
Predicted
A
B
C
D
E
F
G
H
I
J
K
L
M
N
O
P
Q
R
S
T
U
V
W
X
Y
Z
160
140
120
100
80
60
40
20
0