EEE 450- Introduction to Machine Learning Second Application Assignment

DEPARTMENT OF ELECTRICAL AND ELECTRONICS ENGINEERING,
IZMIR KATIP ÇELEBI UNIVERSITY

Turhan Can Kargın | 150403005 | Deadline:12-12-2019

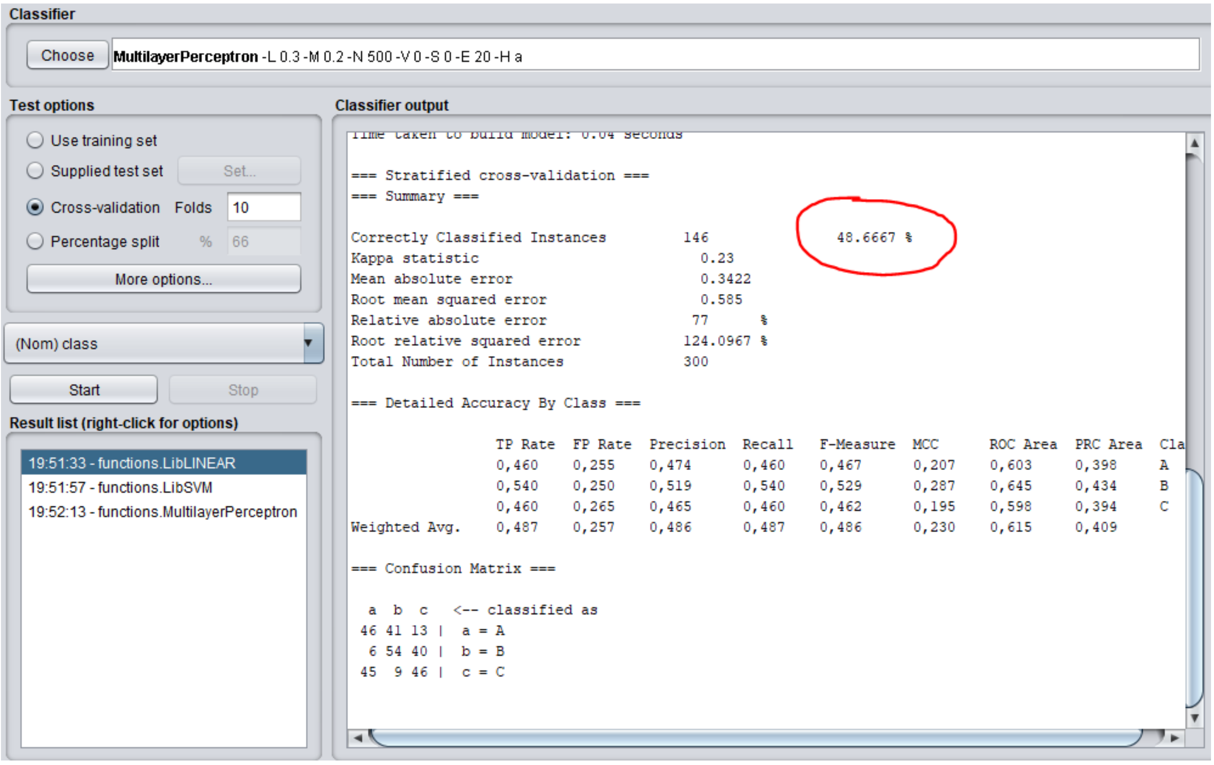
Assignment#1:

Spiral dataset in the Weka tool

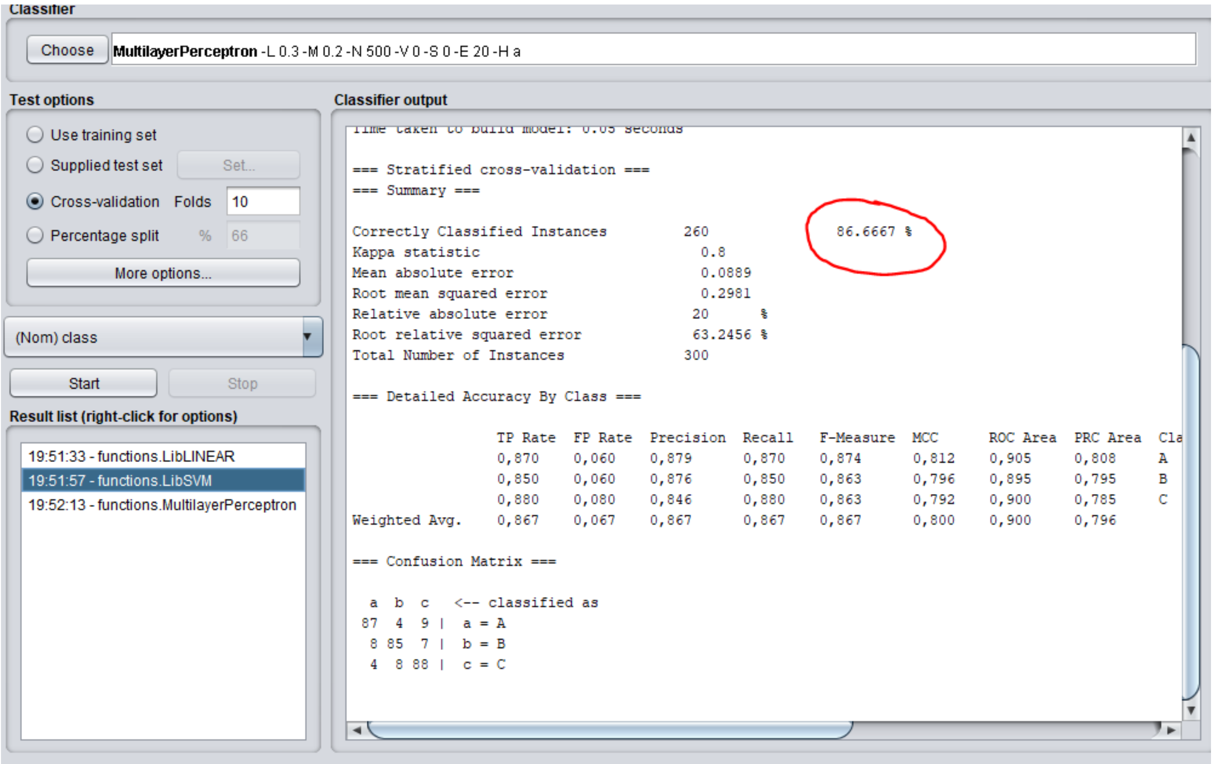
- Classify the Spiral dataset in Weka using the Linear (functions/Liblinear), Neural Network (MultiLayerPerceptron) and Support Vector Machines (functions/LibSVM) algorithms.
- The Spiral dataset can be downloaded from the Datasets page.
- Why does the Linear classifier has much lower accuracy than Neural Network and Support Vector Machine?
- Note that Weka automatically tries to determine the size of the hidden layer. It often happens that it uses too few hidden units to be able to accurately learn the concept. Try to change the hiddenLayers field from a to 72.

Solution:

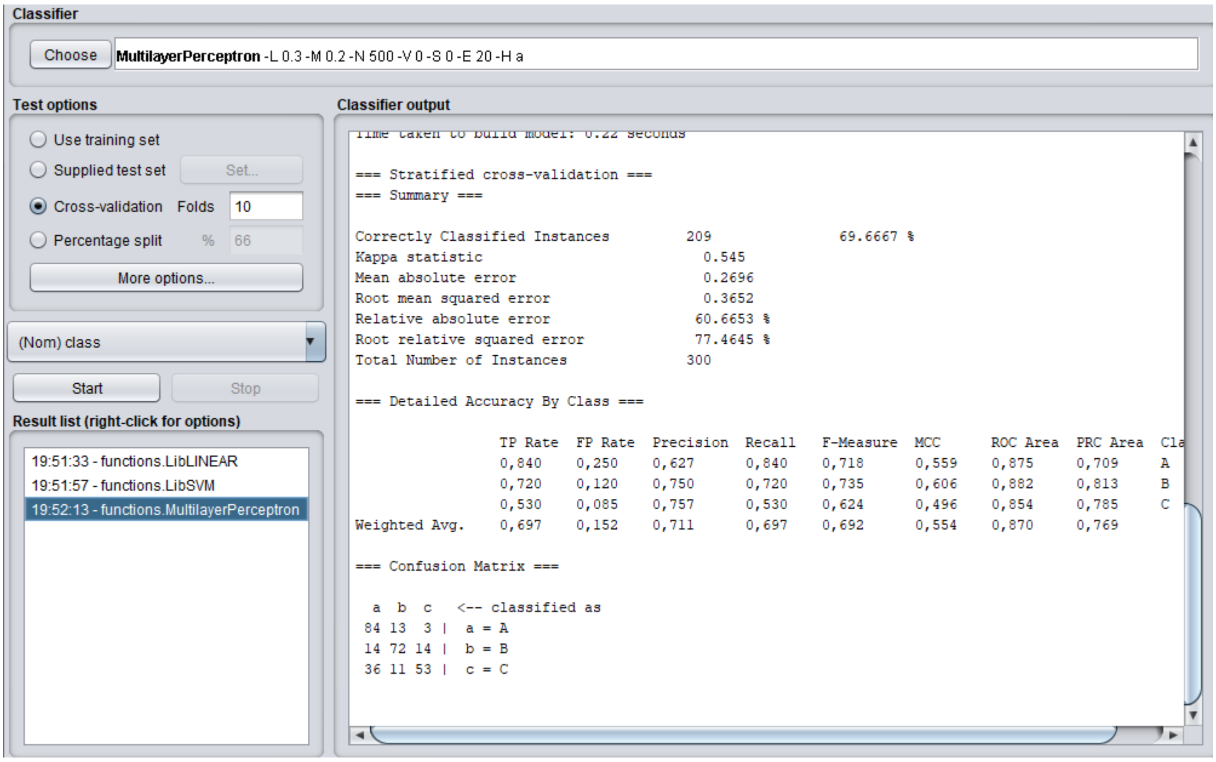
<u>LibLinear</u>



LibSVM

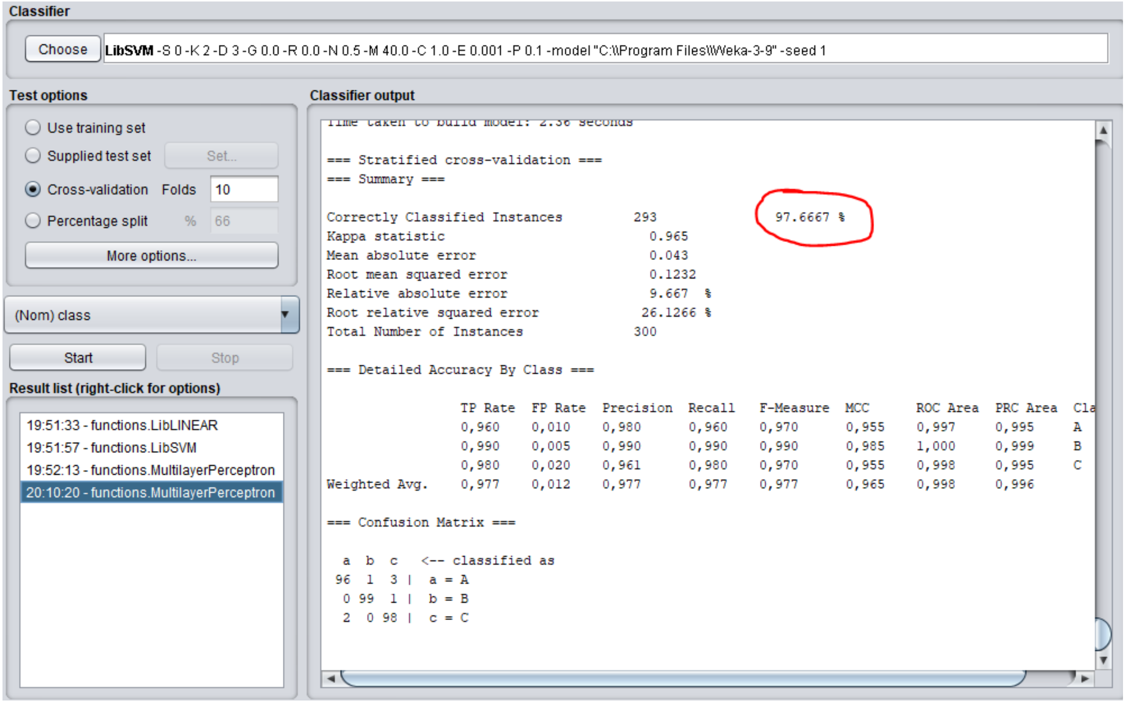


MultiLayerPerceptron



We can see that Liblinear has lowest accuracy because in linear classifier we have a single layer which is the output layer. However, we expand with a layer of hidden nodes for A.Neural Network. For example, you can see that there are 72 hidden layer below and the accuracy is much higher than Liblinear.

MultiLayerPerceptron (HiddenLayer =72)



When we change hidden layers with 72, accuracy increased from 69.6667 to 97.6667.

Assignment#2:

Spiral dataset in Scikit

- Classify the Spiral dataset in Scikit using a Neural Network algorithm
- You need to write code for loading csv dataset files (use the Pandas library)

Solution:

Neural Network Algorithm

```python
import numpy as np
import pandas as pd # library for loading dataset
import seaborn as sns #for plotting graphs
import matplotlib.pyplot as plt
spiral_dataset= pd.read_csv("C:/Program Files/Weka-3-8/data/spiral.csv").values
# Assign data to X variable
X = spiral_dataset[:,0:2]
# Assign data to y variable
y = spiral_dataset[:,2]
from sklearn.model_selection import train_test_split # Train-Test Split
X_train, X_test, y_train, y_test = train_test_split(X,y, test_size= 0.2, random_state=60)
from sklearn.preprocessing import StandardScaler #Before making actual predictions,
#it is always a good practice to scale the features so that all of them can be uniformly evaluated.
scaler = StandardScaler()
scaler.fit(X_train)
X_train = scaler.transform(X_train)
X_test = scaler.transform(X_test)
from sklearn.neural_network import MLPClassifier # Model Training and Predictions
clf = MLPClassifier(hidden_layer_sizes=(100,100,100), max_iter=1000)
clf.fit(x_train, y_train)
y_pred = clf.predict(x_test)
from sklearn.metrics import classification_report, confusion_matrix #evaluating the the Algorithm
print(confusion_matrix(y_test,y_pred))
print(classification_report(y_test,y_pred))
```

Output:

```
[[20  0  0]
 [ 0 18  0]
 [ 1  2 19]]
              precision    recall  f1-score   support

         0.0       0.95      1.00      0.98        20
         1.0       0.90      1.00      0.95        18
         2.0       1.00      0.86      0.93        22

    accuracy                           0.95        60
   macro avg       0.95      0.95      0.95        60
weighted avg       0.95      0.95      0.95        60
```

Assignment#3:

Diabetes dataset in Scikit

- Classify the Diabetes datasets in Scikit using the Neural Network and Xgboost algorithms
- You need to write code for loading csv dataset files (use the Pandas library)

Solution:

Neural Network Algorithm

```python
import numpy as np
import pandas as pd # library for loading dataset
diabetes_dataset= pd.read_csv("C:/Program Files/Weka-3-8/data/diabetes.csv") #loading dataset to pandas dataframe
diabetes_dataset.head() # information about data set
# Assign data from first four columns to X variable
X = diabetes_dataset.iloc[:, 0:4]
# Assign data from first fifth columns to y variable
y = diabetes_dataset.select_dtypes(include=[object])
from sklearn.model_selection import train_test_split # Train-Test Split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.20)
from sklearn.preprocessing import StandardScaler # Before making actual predictions
#it is always a good practice to scale the features so that all of them can be uniformly evaluated
scaler = StandardScaler()
scaler.fit(X_train)
X_train = scaler.transform(X_train)
X_test = scaler.transform(X_test)
from sklearn.neural_network import MLPClassifier # Applying Model
mlp = MLPClassifier(hidden_layer_sizes=(10,10,10), max_iter=1000)
mlp.fit(X_train, y_train.values.ravel())
predictions = mlp.predict(X_test)
from sklearn.metrics import classification_report, confusion_matrix #evaluating the performance of the model.
print(confusion_matrix(y_test,predictions))
print(classification_report(y_test,predictions))
```

Output

```
[[92 16]
 [13 33]]
              precision    recall  f1-score   support

          NO       0.88      0.85      0.86       108
         YES       0.67      0.72      0.69        46

    accuracy                           0.81       154
   macro avg       0.77      0.78      0.78       154
weighted avg       0.82      0.81      0.81       154
```

Xgboost algorithm

```python
import numpy as np
import pandas as pd # library for loading dataset
diabetes_dataset= pd.read_csv("C:/Program Files/Weka-3-8/data/diabetes.csv") #loading dataset to pandas dataframe
 # Assign data from first four columns to X variable
X = diabetes_dataset.iloc[:, 0:4]
# Assign data from first fifth columns to y variable
y = diabetes_dataset.select_dtypes(include=[object])
from sklearn.model_selection import train_test_split # Train-Test Split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.20)
from xgboost import XGBClassifier #Applying Model
model = XGBClassifier()
model.fit(X_train, y_train)
print(model)
# make predictions for test data
predictions = model.predict(X_test)
# evaluate predictions
from sklearn.metrics import accuracy_score
accuracy = accuracy_score(y_test, predictions)
print("Accuracy: %.2f%%" % (accuracy * 100.0))
```

Output

```
XGBClassifier(base_score=0.5, booster='gbtree', colsample_bylevel=1,
              colsample_bynode=1, colsample_bytree=1, gamma=0,
              learning_rate=0.1, max_delta_step=0, max_depth=3,
              min_child_weight=1, missing=None, n_estimators=100, n_jobs=1,
              nthread=None, objective='binary:logistic', random_state=0,
              reg_alpha=0, reg_lambda=1, scale_pos_weight=1, seed=None,
              silent=None, subsample=1, verbosity=1)
Accuracy: 74.68%
```

Assignment#4:

Diabetes dataset in Keras

- Write code for loading, training and evaluating the Diabetes dataset using a neural network classifier in Keras.

Solution:

What is Keras?

- Keras is a high-level neural network API which is written in Python.
- It is capable of running on top of Tensorflow, CNTK or Theano.
- Keras can be used as a deep learning library. Support Convolutional and Recurrent Neural Networks
- Prototyping with keras is fast and easy
- Runs seamlessly on CPU and GPU

I will build a neural network classification for diabetes dataset classification with keras.

Neural Network Algorithm:

```
# Import required libraries
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import sklearn
# Import necessary modules
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_squared_error
from math import sqrt
# Keras specific
import keras
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense
from keras.utils import to_categorical
df = pd.read_csv("C:/Program Files/Weka-3-8/data/diabetes.csv") #loading dataset to pandas dataframe
#To understand the data better, let's view the dataset details.
df.head(3)
#we need to check what type of data we have in the dataset
print(df.shape)
df.describe()
#Creating Arrays for the Features and the Response Variable.
target_column = ['Diabetes']
predictors = list(set(list(df.columns))-set(target_column))
df[predictors] = df[predictors]/df[predictors].max()
df.describe()
#Creating the Training and Test Datasets
X = df[predictors].values
y = df[target_column].values
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.30, random_state=40)
print(X_train.shape); print(X_test.shape)
# one hot encode outputs
y_train = to_categorical(y_train)
y_test = to_categorical(y_test)
count_classes = y_test.shape[1]
print(count_classes)
#Define, Compile, and Fit the Keras Classification Model
model = Sequential()
model.add(Dense(500, activation='relu', input_dim=8))
model.add(Dense(100, activation='relu'))
model.add(Dense(50, activation='relu'))
model.add(Dense(2, activation='softmax'))
# Compile the model
model.compile(optimizer='adam',loss='categorical_crossentropy', metrics=['accuracy'])
```

```python
# build the model
model.fit(X_train, y_train,batch_size=10,epochs=100)
#Predict on the Test Data and Compute Evaluation Metrics;
pred_train= model.predict(X_train)
scores = model.evaluate(X_train, y_train, verbose=0)
print('Accuracy on training data: {}% \n Error on training data: {}'.format(scores[1], 1 - scores[1]))
pred_test= model.predict(X_test)
scores2 = model.evaluate(X_test, y_test, verbose=0)
print('Accuracy on test data: {}% \n Error on test data: {}'.format(scores2[1], 1 - scores2[1]))
```

Outputs:

	0TimesPregnant	PlasmaGlucoseConc	DiastolicBloodPressure	TricepsSkinFoldThickness	SerumInsulin	BMI	DiabetesPedigreeFunction	Age	Diabetes
0	6	148	72	35	0	33.6	0.627	50	1
1	1	85	66	29	0	26.6	0.351	31	0
2	8	183	64	0	0	23.3	0.672	32	1

	0TimesPregnant	PlasmaGlucoseConc	DiastolicBloodPressure	TricepsSkinFoldThickness	SerumInsulin	BMI	DiabetesPedigreeFunction	Age	Diabetes
count	768.000000	768.000000	768.000000	768.000000	768.000000	768.000000	768.000000	768.000000	768.000000
mean	3.845052	120.894531	69.105469	20.536458	79.799479	31.992578	0.471876	33.240885	0.348958
std	3.369578	31.972618	19.355807	15.952218	115.244002	7.884160	0.331329	11.760232	0.476951
min	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.078000	21.000000	0.000000
25%	1.000000	99.000000	62.000000	0.000000	0.000000	27.300000	0.243750	24.000000	0.000000
50%	3.000000	117.000000	72.000000	23.000000	30.500000	32.000000	0.372500	29.000000	0.000000
75%	6.000000	140.250000	80.000000	32.000000	127.250000	36.600000	0.626250	41.000000	1.000000
max	17.000000	199.000000	122.000000	99.000000	846.000000	67.100000	2.420000	81.000000	1.000000

```python
#Creating the Training and Test Datasets
X = df[predictors].values
y = df[target_column].values

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.30, random_state=40)
print(X_train.shape); print(X_test.shape)
```

```
(537, 8)
(231, 8)
```

```python
# one hot encode outputs
y_train = to_categorical(y_train)
y_test = to_categorical(y_test)

count_classes = y_test.shape[1]
print(count_classes)
```

```
2
```

```
537/537 [==============================] - 1s 2ms/sample - loss: 0.2322 - accuracy: 0.9069
Epoch 93/100
537/537 [==============================] - 1s 1ms/sample - loss: 0.2184 - accuracy: 0.9069
Epoch 94/100
537/537 [==============================] - 1s 2ms/sample - loss: 0.2066 - accuracy: 0.9181
Epoch 95/100
537/537 [==============================] - 0s 917us/sample - loss: 0.2019 - accuracy: 0.9199
Epoch 96/100
537/537 [==============================] - 0s 836us/sample - loss: 0.2040 - accuracy: 0.9255
Epoch 97/100
537/537 [==============================] - 1s 1ms/sample - loss: 0.2021 - accuracy: 0.9218
Epoch 98/100
537/537 [==============================] - 0s 867us/sample - loss: 0.2023 - accuracy: 0.9199
Epoch 99/100
537/537 [==============================] - 0s 650us/sample - loss: 0.2042 - accuracy: 0.9106 - loss: 0.1768 - accura
Epoch 100/100
537/537 [==============================] - 1s 982us/sample - loss: 0.2074 - accuracy: 0.9125
```

```
Accuracy on training data: 0.9199255108833313
 Error on training data: 0.0800744891166687
Accuracy on test data: 0.701298713684082%
 Error on test data: 0.29870128631591797
```

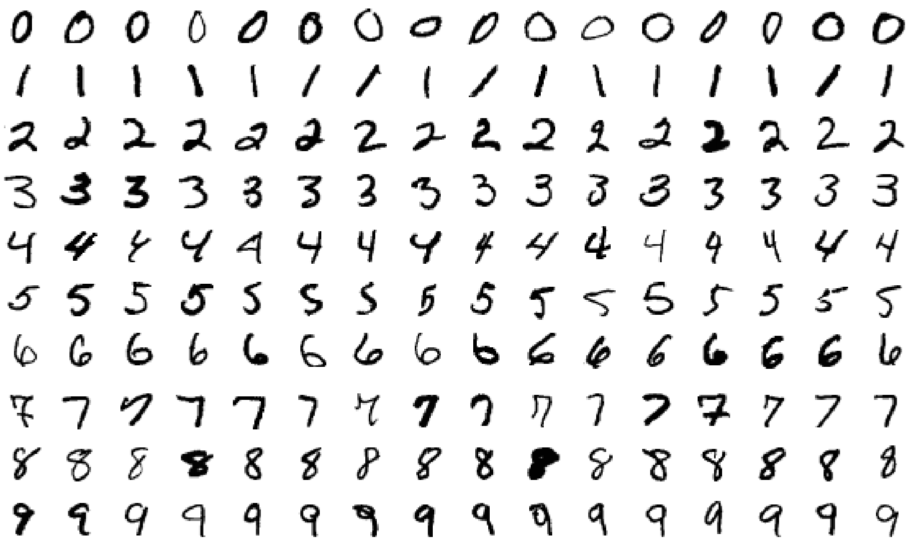
Assignment#5:

MNSIT dataset in Keras

- Write code for loading, training and evaluating the MNIST dataset using a Linear, Neural Network and a ConvNet classifier in Keras.

Solution:

We're going to tackle a classic machine learning problem: MNIST handwritten digit classification. It's simple: given an image, classify it as a digit.



Sample images from the MNIST dataset

Each image in the MNIST dataset is 28x28 and contains a centered, grayscale digit. We'll flatten each 28x28 into a 784 dimensional vector, which we'll use as input to our neural network. Our output will be one of 10 possible classes: one for each digit.

Linear

In [1]:
```
import numpy as np
import mnist
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense
from keras.utils import to_categorical
```
```
Using TensorFlow backend.
```

In [2]:
```
train_images = mnist.train_images()
train_labels = mnist.train_labels()
test_images = mnist.test_images()
test_labels = mnist.test_labels()
```

In [3]:
```
# Normalize the images.
train_images = (train_images / 255) - 0.5
test_images = (test_images / 255) - 0.5
```

In [4]:
```
# Flatten the images.
train_images = train_images.reshape((-1, 784))
test_images = test_images.reshape((-1, 784))
```

In [5]:
```
# Build the model.
model = Sequential([
  Dense(10, activation='softmax'),
])
```

In [6]:
```
# Compile the model.
model.compile(
  optimizer='adam',
  loss='categorical_crossentropy',
  metrics=['accuracy'],
```

In [7]:
```
# Train the model.
model.fit(
  train_images,
  to_categorical(train_labels),
  epochs=5,
  batch_size=32,
)
```

```
Train on 60000 samples
Epoch 1/5
60000/60000 [==============================] - 188s 3ms/sample - loss: 0.4924 - accuracy: 0.8652
Epoch 2/5
60000/60000 [==============================] - 188s 3ms/sample - loss: 0.3323 - accuracy: 0.9037
Epoch 3/5
60000/60000 [==============================] - 187s 3ms/sample - loss: 0.3113 - accuracy: 0.9098
Epoch 4/5
60000/60000 [==============================] - 187s 3ms/sample - loss: 0.3010 - accuracy: 0.9137
Epoch 5/5
60000/60000 [==============================] - 187s 3ms/sample - loss: 0.2937 - accuracy: 0.9172
```

Out[7]: <tensorflow.python.keras.callbacks.History at 0x209e7a6ee48>

Neural Network

In [1]:
```
import numpy as np
import mnist
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense
from keras.utils import to_categorical
```

```
Using TensorFlow backend.
```

In [2]:
```
train_images = mnist.train_images()
train_labels = mnist.train_labels()
test_images = mnist.test_images()
test_labels = mnist.test_labels()
```

In [3]:
```
# Normalize the images.
train_images = (train_images / 255) - 0.5
test_images = (test_images / 255) - 0.5
```

In [4]:
```
# Flatten the images.
train_images = train_images.reshape((-1, 784))
test_images = test_images.reshape((-1, 784))
```

In [5]:
```
# Build the model.
model = Sequential([
  Dense(64, activation='relu', input_shape=(784,)),
  Dense(64, activation='relu'),
  Dense(10, activation='softmax'),
])
```

In [6]:
```
# Compile the model.
model.compile(
  optimizer='adam',
  loss='categorical_crossentropy',
  metrics=['accuracy'],
)
```

In [7]:
```
# Train the model.
model.fit(
  train_images,
  to_categorical(train_labels),
  epochs=5,
  batch_size=32,
)
```

```
Train on 60000 samples
Epoch 1/5
60000/60000 [==============================] - 189s 3ms/sample - loss: 0.3666 - accuracy: 0.8901
Epoch 2/5
60000/60000 [==============================] - 172s 3ms/sample - loss: 0.1911 - accuracy: 0.9417
Epoch 3/5
60000/60000 [==============================] - 184s 3ms/sample - loss: 0.1493 - accuracy: 0.9538
Epoch 4/5
60000/60000 [==============================] - 178s 3ms/sample - loss: 0.1262 - accuracy: 0.9617
Epoch 5/5
60000/60000 [==============================] - 166s 3ms/sample - loss: 0.1112 - accuracy: 0.9655
```

Out[7]: <tensorflow.python.keras.callbacks.History at 0x28a02c04048>

In [8]:
```
# Evaluate the model.
model.evaluate(
  test_images,
  to_categorical(test_labels)
)
```

```
3s 330us/sample - loss: 0.0723 - accuracy: 0.9620
```

ConvNet

In [1]:
```
#Loading required libraries
import numpy as np
import mnist
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Conv2D, MaxPooling2D, Dense, Flatten
from keras.utils import to_categorical
```

```
Using TensorFlow backend.
```

In [2]:
```
train_images = mnist.train_images()
train_labels = mnist.train_labels()
test_images = mnist.test_images()
test_labels = mnist.test_labels()
```

In [3]:
```
# Normalize the images.
train_images = (train_images / 255) - 0.5
test_images = (test_images / 255) - 0.5
```

In [4]:
```
# Reshape the images.
train_images = np.expand_dims(train_images, axis=3)
test_images = np.expand_dims(test_images, axis=3)
num_filters = 8
filter_size = 3
pool_size = 2
```

In [5]:
```
# Build the model.
model = Sequential([
  Conv2D(num_filters, filter_size, input_shape=(28, 28, 1)),
  MaxPooling2D(pool_size=pool_size),
  Flatten(),
  Dense(10, activation='softmax'),
])
```

In [6]:
```
# Compile the model.
model.compile(
  'adam',
  loss='categorical_crossentropy',
  metrics=['accuracy'],
)
```

In [7]:
```
# Train the model.
model.fit(
  train_images,
  to_categorical(train_labels),
  epochs=3,
  validation_data=(test_images, to_categorical(test_labels)),
)
```

```
Train on 60000 samples, validate on 10000 samples
Epoch 1/3
60000/60000 [==============================] - 210s 4ms/sample - loss: 0.3673 - accuracy: 0.8932 - val_loss: 0.2291 - val_accur
acy: 0.9335
Epoch 2/3
60000/60000 [==============================] - 205s 3ms/sample - loss: 0.2001 - accuracy: 0.9418 - val_loss: 0.1622 - val_accur
acy: 0.9535
Epoch 3/3
60000/60000 [==============================] - 197s 3ms/sample - loss: 0.1453 - accuracy: 0.9585 - val_loss: 0.1395 - val_accur
acy: 0.9606
```

Out[7]:
```
<tensorflow.python.keras.callbacks.History at 0x1fe7c369f08>
```

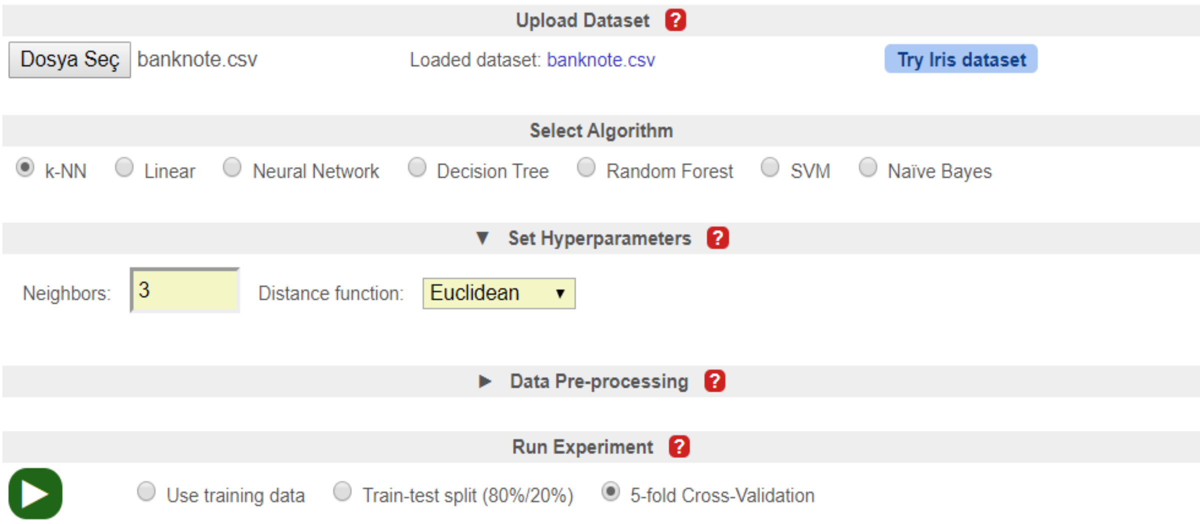
Assignment#6:

Banknote dataset in Web ML Experimenter

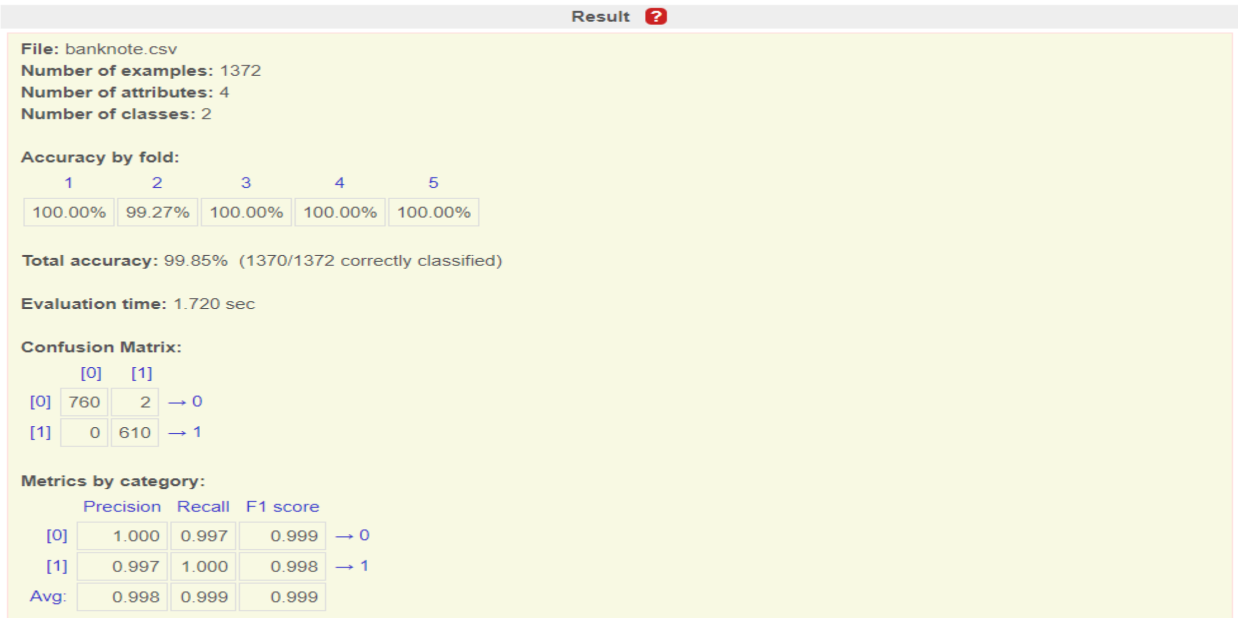
- Download the Banknote dataset from the Datasets page
- Upload the dataset in the Web ML Experimenter
- Try classifying the Banknote dataset using different classifiers. Test with different hyperparameter settings.
- Which classifier had the highest accuracy?

Solution:

k-NN



N=3



N=7

Result ?

File: banknote.csv
Number of examples: 1372
Number of attributes: 4
Number of classes: 2

Accuracy by fold:

1	2	3	4	5
100.00%	99.27%	99.64%	100.00%	100.00%

Total accuracy: 99.78% (1369/1372 correctly classified)

Evaluation time: 1.482 sec

Confusion Matrix:

	[0]	[1]	
[0]	759	3	→ 0
[1]	0	610	→ 1

Metrics by category:

	Precision	Recall	F1 score	
[0]	1.000	0.996	0.998	→ 0
[1]	0.995	1.000	0.998	→ 1
Avg:	0.998	0.998	0.998	

Linear

Upload Dataset ?

Dosya Seç banknote.csv   Loaded dataset: banknote.csv   Try Iris dataset

Select Algorithm

k-NN   Linear   Neural Network   Decision Tree   Random Forest   SVM   Naïve Bayes

▼ Set Hyperparameters ?

Training iterations: 1000   Learning rate: 0.5   Batch size: 0

L2 regularization: 0   Momentum: 0.1

▶ Data Pre-processing ?

Run Experiment ?

Use training data   Train-test split (80%/20%)   5-fold Cross-Validation

Training iteration : 1000 – Batch Size: 0

Result

File: banknote.csv
Number of examples: 1372
Number of attributes: 4
Number of classes: 2

Accuracy by fold:

1	2	3	4	5
97.45%	99.64%	97.81%	97.81%	99.64%

Total accuracy: 98.47% (1351/1372 correctly classified)

Evaluation time: 3.469 sec

Confusion Matrix:

	[0]	[1]	
[0]	744	18	→ 0
[1]	3	607	→ 1

Metrics by category:

	Precision	Recall	F1 score	
[0]	0.996	0.976	0.986	→ 0
[1]	0.971	0.995	0.983	→ 1
Avg:	0.984	0.986	0.985	

Training iteration : 1000 – Batch Size: 25

Result

File: banknote.csv
Number of examples: 1372
Number of attributes: 4
Number of classes: 2

Accuracy by fold:

1	2	3	4	5
97.45%	98.91%	97.81%	97.81%	99.64%

Total accuracy: 98.32% (1349/1372 correctly classified)

Evaluation time: 1.742 sec

Confusion Matrix:

	[0]	[1]	
[0]	742	20	→ 0
[1]	3	607	→ 1

Metrics by category:

	Precision	Recall	F1 score	
[0]	0.996	0.974	0.985	→ 0
[1]	0.968	0.995	0.981	→ 1
Avg:	0.982	0.984	0.983	

Training iteration : 500 – Batch Size: 25

Result

File: banknote.csv
Number of examples: 1372
Number of attributes: 4
Number of classes: 2

Accuracy by fold:

1	2	3	4	5
97.08%	98.91%	97.81%	97.81%	98.91%

Total accuracy: 98.10% (1346/1372 correctly classified)

Evaluation time: 0.890 sec

Confusion Matrix:

	[0]	[1]	
[0]	740	22	→ 0
[1]	4	606	→ 1

Metrics by category:

	Precision	Recall	F1 score	
[0]	0.995	0.971	0.983	→ 0
[1]	0.965	0.993	0.979	→ 1
Avg:	0.980	0.982	0.981	

Neural Network

Upload Dataset

Dosya Seç banknote.csv Loaded dataset: banknote.csv Try Iris dataset

Select Algorithm

k-NN Linear Neural Network Decision Tree Random Forest SVM Naïve Bayes

▼ **Set Hyperparameters**

Training iterations: 500 Hidden layers: 32 Learning rate: 0.8 Batch size: 0

L2 regularization: 0 Momentum: 0.1 Dropout: 0.2

▶ **Data Pre-processing**

Run Experiment

Use training data Train-test split (80%/20%) 5-fold Cross-Validation

Training iteration : 500 – Batch Size: 0 – Hidden Layers: 32

Result ?

File: banknote.csv
Number of examples: 1372
Number of attributes: 4
Number of classes: 2

Accuracy by fold:

1	2	3	4	5
100.00%	100.00%	100.00%	100.00%	100.00%

Total accuracy: 100.00% (1372/1372 correctly classified)

Evaluation time: 18.739 sec

Confusion Matrix:

	[0]	[1]	
[0]	762	0	→ 0
[1]	0	610	→ 1

Metrics by category:

	Precision	Recall	F1 score	
[0]	1.000	1.000	1.000	→ 0
[1]	1.000	1.000	1.000	→ 1
Avg:	1.000	1.000	1.000	

Training iteration : 500 – Batch Size: 25 – Hidden Layers: 32

Result ?

File: banknote.csv
Number of examples: 1372
Number of attributes: 4
Number of classes: 2

Accuracy by fold:

1	2	3	4	5
98.91%	100.00%	98.54%	100.00%	100.00%

Total accuracy: 99.49% (1365/1372 correctly classified)

Evaluation time: 7.867 sec

Confusion Matrix:

	[0]	[1]	
[0]	755	7	→ 0
[1]	0	610	→ 1

Metrics by category:

	Precision	Recall	F1 score	
[0]	1.000	0.991	0.995	→ 0
[1]	0.989	1.000	0.994	→ 1
Avg:	0.994	0.995	0.995	

Training iteration : 100 – Batch Size: 25 – Hidden Layers: 32

Result

File: banknote.csv
Number of examples: 1372
Number of attributes: 4
Number of classes: 2

Accuracy by fold:

1	2	3	4	5
98.18%	99.64%	97.81%	98.54%	98.91%

Total accuracy: 98.62% (1353/1372 correctly classified)

Evaluation time: 1.654 sec

Confusion Matrix:

	[0]	[1]	
[0]	743	19	→ 0
[1]	0	610	→ 1

Metrics by category:

	Precision	Recall	F1 score	
[0]	1.000	0.975	0.987	→ 0
[1]	0.970	1.000	0.985	→ 1
Avg:	0.985	0.988	0.986	

Training iteration : 100 – Batch Size: 25 – Hidden Layers: 16

Result

File: banknote.csv
Number of examples: 1372
Number of attributes: 4
Number of classes: 2

Accuracy by fold:

1	2	3	4	5
98.18%	99.27%	98.18%	98.18%	98.91%

Total accuracy: 98.54% (1352/1372 correctly classified)

Evaluation time: 0.882 sec

Confusion Matrix:

	[0]	[1]	
[0]	745	17	→ 0
[1]	3	607	→ 1

Metrics by category:

	Precision	Recall	F1 score	
[0]	0.996	0.978	0.987	→ 0
[1]	0.973	0.995	0.984	→ 1
Avg:	0.984	0.986	0.985	

Decision Tree

Upload Dataset ?

Dosya Seç banknote.csv Loaded dataset: banknote.csv Try Iris dataset

Select Algorithm

○ k-NN ○ Linear ○ Neural Network ◉ Decision Tree ○ Random Forest ○ SVM ○ Naïve Bayes

▼ **Set Hyperparameters** ?

Max depth: 7 Min samples split: 5

▶ **Data Pre-processing** ?

Run Experiment ?

○ Use training data ○ Train-test split (80%/20%) ◉ 5-fold Cross-Validation

Max Depth: 7 – Min samples split: 5

Result ?

File: banknote.csv
Number of examples: 1372
Number of attributes: 4
Number of classes: 2

Accuracy by fold:

1	2	3	4	5
98.18%	100.00%	100.00%	100.00%	100.00%

Total accuracy: 99.64% (1367/1372 correctly classified)

Evaluation time: 19.321 sec

Confusion Matrix:

	[0]	[1]	
[0]	760	2	→ 0
[1]	3	607	→ 1

Metrics by category:

	Precision	Recall	F1 score	
[0]	0.996	0.997	0.997	→ 0
[1]	0.997	0.995	0.996	→ 1
Avg:	0.996	0.996	0.996	

Max Depth: 3 – Min samples split: 5

Result

File: banknote.csv
Number of examples: 1372
Number of attributes: 4
Number of classes: 2

Accuracy by fold:

1	2	3	4	5
95.26%	93.80%	93.80%	91.97%	96.38%

Total accuracy: 94.24% (1293/1372 correctly classified)

Evaluation time: 14.734 sec

Confusion Matrix:

	[0]	[1]	
[0]	728	34	→ 0
[1]	45	565	→ 1

Metrics by category:

	Precision	Recall	F1 score	
[0]	0.942	0.955	0.949	→ 0
[1]	0.943	0.926	0.935	→ 1
Avg:	0.943	0.941	0.942	

Random Forest

Upload Dataset

Dosya Seç banknote.csv Loaded dataset: banknote.csv Try Iris dataset

Select Algorithm

k-NN Linear Neural Network Decision Tree Random Forest SVM Naïve Bayes

▼ **Set Hyperparameters**

No estimators: 15 Max depth: 7 Min samples split: 5 Subset sample size: 0.9

▶ **Data Pre-processing**

Run Experiment

Use training data Train-test split (80%/20%) 5-fold Cross-Validation

Max Depth: 7 – Min samples split: 5

Result

File: banknote.csv
Number of examples: 1372
Number of attributes: 4
Number of classes: 2

Accuracy by fold:

1	2	3	4	5
99.27%	100.00%	100.00%	100.00%	99.64%

Total accuracy: 99.78% (1369/1372 correctly classified)

Evaluation time: 14.858 sec

Confusion Matrix:

	[0]	[1]	
[0]	760	2	→ 0
[1]	1	609	→ 1

Metrics by category:

	Precision	Recall	F1 score	
[0]	0.999	0.997	0.998	→ 0
[1]	0.997	0.998	0.998	→ 1
Avg:	0.998	0.998	0.998	

Max Depth: 3 – Min samples split: 5

Result

File: banknote.csv
Number of examples: 1372
Number of attributes: 4
Number of classes: 2

Accuracy by fold:

1	2	3	4	5
94.89%	92.70%	93.43%	92.70%	96.74%

Total accuracy: 94.10% (1291/1372 correctly classified)

Evaluation time: 11.667 sec

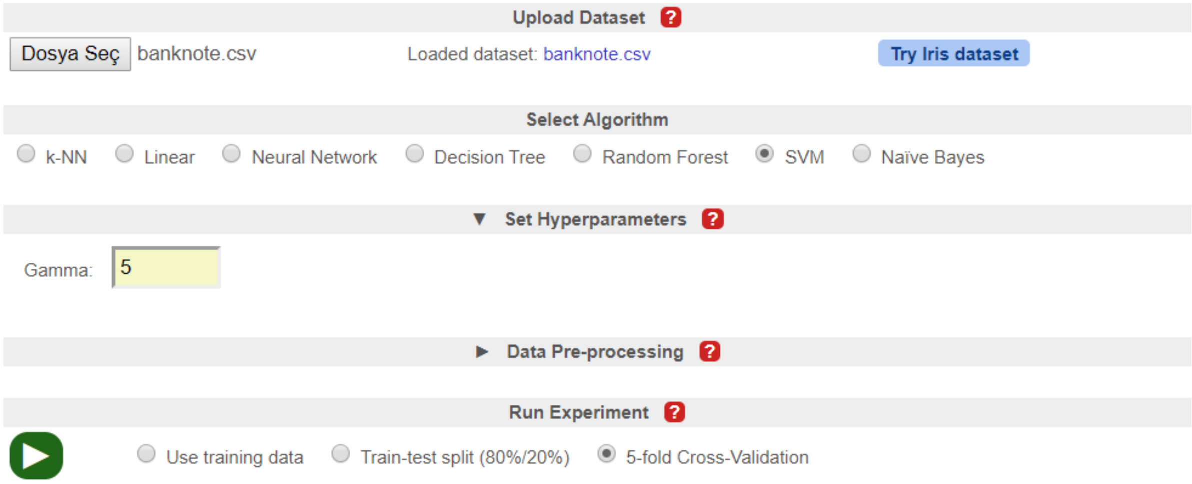
Confusion Matrix:

	[0]	[1]	
[0]	705	57	→ 0
[1]	24	586	→ 1

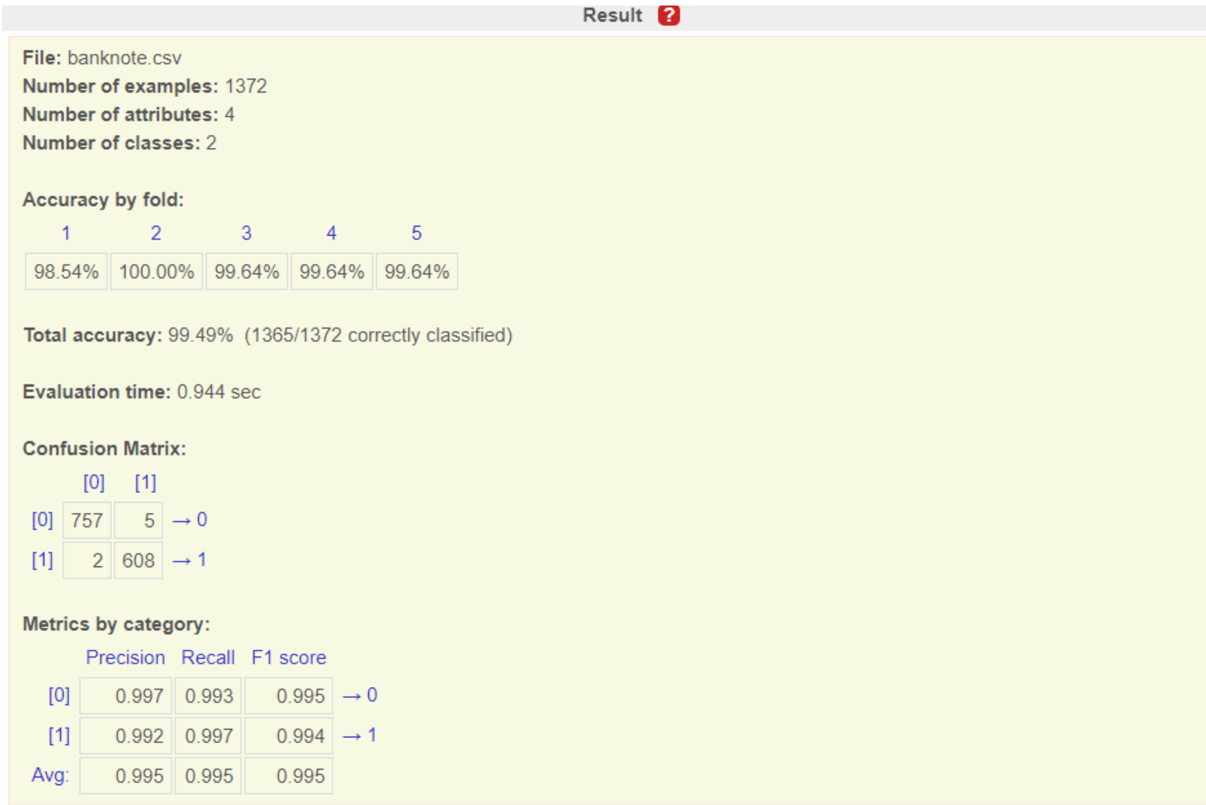
Metrics by category:

	Precision	Recall	F1 score	
[0]	0.967	0.925	0.946	→ 0
[1]	0.911	0.961	0.935	→ 1
Avg:	0.939	0.943	0.941	

SVM

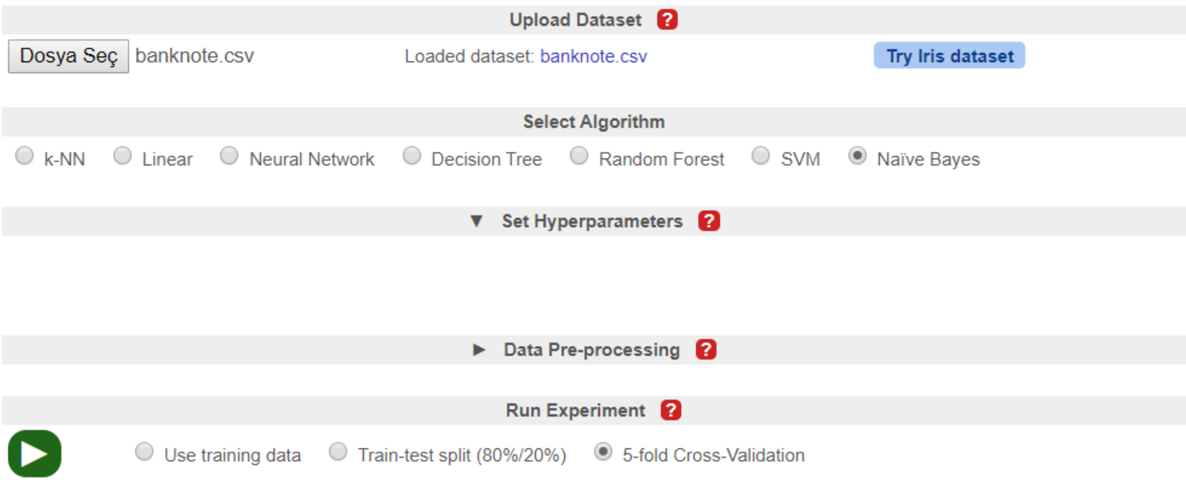

Upload Dataset
Dosya Seç
banknote.csv
Loaded dataset: banknote.csv
Try Iris dataset
Select Algorithm
k-NN
Linear
Neural Network
Decision Tree
Random Forest
SVM
Naïve Bayes
Set Hyperparameters
Gamma:
5
Data Pre-processing
Run Experiment
Use training data
Train-test split (80%/20%)
5-fold Cross-Validation

Gamma = 5


Result
File: banknote.csv
Number of examples: 1372
Number of attributes: 4
Number of classes: 2
Accuracy by fold:
1 2 3 4 5
98.54% 100.00% 99.64% 99.64% 99.64%
Total accuracy: 99.49% (1365/1372 correctly classified)
Evaluation time: 0.944 sec
Confusion Matrix:
[0] [1]
[0] 757 5 → 0
[1] 2 608 → 1
Metrics by category:
Precision Recall F1 score
[0] 0.997 0.993 0.995 → 0
[1] 0.992 0.997 0.994 → 1
Avg: 0.995 0.995 0.995

Gamma = 7

Naive Bayes



Result

File: banknote.csv
Number of examples: 1372
Number of attributes: 4
Number of classes: 2

Accuracy by fold:

1	2	3	4	5
86.86%	81.75%	80.66%	83.21%	90.22%

Total accuracy: 84.55% (1160/1372 correctly classified)

Evaluation time: 0.051 sec

Confusion Matrix:

	[0]	[1]	
[0]	662	100	→ 0
[1]	112	498	→ 1

Metrics by category:

	Precision	Recall	F1 score	
[0]	0.855	0.869	0.862	→ 0
[1]	0.833	0.816	0.825	→ 1
Avg:	0.844	0.843	0.843	

According to results, we can easily see that Naive Bayes Classification has the worst accuracy. Besides, we can see that Neural Network Classification has the highest accuracy because it is the only one which we get 100% percent of accuracy.

Assignment#7:

Diabetes dataset in R

- Classify the diabetes dataset in R using Neural Networks, SVM and RandomForest.
- Split the dataset into 80% training and 20% testing, and evaluate accuracy on the test dataset
- Which classifier had the highest accuracy?

Solution:

Support Vector Machine

```r
# Read the Data
dataset <- read.csv("C:\\Program Files\\Weka-3-8\\data\\diabetes.csv", sep = ',', header = FALSE)
str(dataset)
head(dataset)
#80% training and 20% testing
intrain <- createDataPartition(y = dataset$V9, p= 0.8, list = FALSE)
training <- dataset[intrain,]
testing <- dataset[-intrain,]
dim(training);
dim(testing);
anyNA(dataset)
summary(dataset)
training[["V9"]] = factor(training[["V9"]])
trctrl <- trainControl(method = "repeatedcv", number = 10, repeats = 3)
#Model
svm_Linear <- train(V9 ~., data = training, method = "svmLinear",
                    trControl=trctrl,
                    preProcess = c("center", "scale"),
                    tuneLength = 10)
svm_Linear
test_pred <- predict(svm_Linear, newdata = testing)
test_pred
#Evaluation
confusionMatrix(table(test_pred, testing$V9))
```

Output:

```
> str(dataset)
'data.frame':   769 obs. of  9 variables:
 $ V1: Factor w/ 18 levels "0","1","10","11",..: 18 14 2 16 2 1 13 11 3 10 ...
 $ V2: Factor w/ 137 levels "0","100","101",..: 137 50 122 85 126 39 18 115 17 97 ...
 $ V3: Factor w/ 48 levels "0","100","102",..: 48 31 28 26 28 13 32 17 1 30 ...
 $ V4: Factor w/ 52 levels "0","10","11",..: 52 27 21 1 15 27 1 24 1 37 ...
 $ V5: Factor w/ 187 levels "0","100","105",..: 187 1 1 1 183 41 1 178 1 141 ...
 $ V6: Factor w/ 249 levels "0.0","18.2","18.4",..: 249 124 63 31 78 210 54 104 141 100 ...
 $ V7: Factor w/ 518 levels "0.078","0.084",..: 518 351 197 369 54 515 81 119 24 45 ...
 $ V8: Factor w/ 53 levels "21","22","23",..: 53 30 11 12 1 13 10 6 9 33 ...
 $ V9: Factor w/ 3 levels "Diabetes","NO",..: 1 3 2 3 2 3 2 3 2 3 ...
> head(dataset)
             V1               V2                    V3                        V4            V5   V6
1 NoTimesPregnant PlasmaGlucoseConc DiastolicBloodPressure TricepsSkinFoldThickness SerumInsulin  BMI
2             6              148                    72                        35             0 33.6
3             1               85                    66                        29             0 26.6
4             8              183                    64                         0             0 23.3
5             1               89                    66                        23            94 28.1
6             0              137                    40                        35           168 43.1
                       V7  V8       V9
1 DiabetesPedigreeFunction Age Diabetes
2                    0.627  50      YES
3                    0.351  31       NO
4                    0.672  32      YES
5                    0.167  21       NO
6                    2.288  33      YES
```

```
> dim(training);
[1] 616   9
> dim(testing);
[1] 153   9
> anyNA(dataset)
[1] FALSE
> summary(dataset)
       V1           V2           V3           V4           V5           V6           V7           V8
 1      :135   100    : 17   70     : 57   0      :227   0      :374   32.0   : 13   0.254  :  6   22     : 72
 0      :111   99     : 17   74     : 52   32     : 31   105    : 11   31.2   : 12   0.258  :  6   21     : 63
 2      :103   106    : 14   68     : 45   30     : 27   130    :  9   31.6   : 12   0.207  :  5   25     : 48
 3      : 75   111    : 14   78     : 45   27     : 23   140    :  9   0.0    : 11   0.238  :  5   24     : 46
 4      : 68   125    : 14   72     : 44   23     : 22   120    :  8   32.4   : 10   0.259  :  5   23     : 38
 5      : 57   129    : 14   64     : 43   18     : 20   100    :  7   33.3   : 10   0.261  :  5   28     : 35
 (Other):220   (Other):679   (Other):483   (Other):419   (Other):351   (Other):701   (Other):737   (Other):467
       V9
 Diabetes:  1
 NO      :500
 YES     :268
```

```
> svm_Linear
Support Vector Machines with Linear Kernel

616 samples
  8 predictor
  3 classes: 'Diabetes', 'NO', 'YES'

Pre-processing: centered (1254), scaled (1254)
Resampling: Cross-Validated (10 fold, repeated 3 times)
Summary of sample sizes: 554, 554, 554, 554, 553, 555, ...
Resampling results:

  Accuracy   Kappa
  0.5691608  0.1112582
```

```
Confusion Matrix and Statistics

test_pred  Diabetes NO YES
  Diabetes        0  0   0
  NO              0 79  32
  YES             0 21  21

Overall Statistics

               Accuracy : 0.6536
                 95% CI : (0.5725, 0.7286)
    No Information Rate : 0.6536
    P-Value [Acc > NIR] : 0.5373

                  Kappa : 0.1958

 Mcnemar's Test P-Value : NA

Statistics by Class:

                     Class: Diabetes Class: NO Class: YES
Sensitivity                       NA    0.7900     0.3962
Specificity                        1    0.3962     0.7900
Pos Pred Value                    NA    0.7117     0.5000
Neg Pred Value                    NA    0.5000     0.7117
Prevalence                         0    0.6536     0.3464
Detection Rate                     0    0.5163     0.1373
Detection Prevalence               0    0.7255     0.2745
Balanced Accuracy                 NA    0.5931     0.5931
```

Random Forest

```r
# Read the Data
dataset <- read.csv("C:\\Program Files\\Weka-3-8\\data\\diabetes.csv", sep = ',', header = TRUE)
# Test-Training Split = 80 : 20
set.seed(100)
train <- sample(nrow(dataset), 0.8*nrow(dataset), replace = FALSE)
TrainSet <- dataset[train,]
ValidSet <- dataset[-train,]
# Create a Random Forest model with default parameters
model1 <- randomForest(Diabetes ~ ., data = TrainSet, importance = TRUE)
model1
# Predicting on train set
predTrain <- predict(model1, TrainSet, type = "class")
# Checking classification accuracy
table(predTrain, TrainSet$Diabetes)
# Predicting on Validation set
predValid <- predict(model1, ValidSet, type = "class")
table(predValid,ValidSet$Diabetes)
# Checking classification accuracy
mean(predValid == ValidSet$Diabetes)
```

Output:

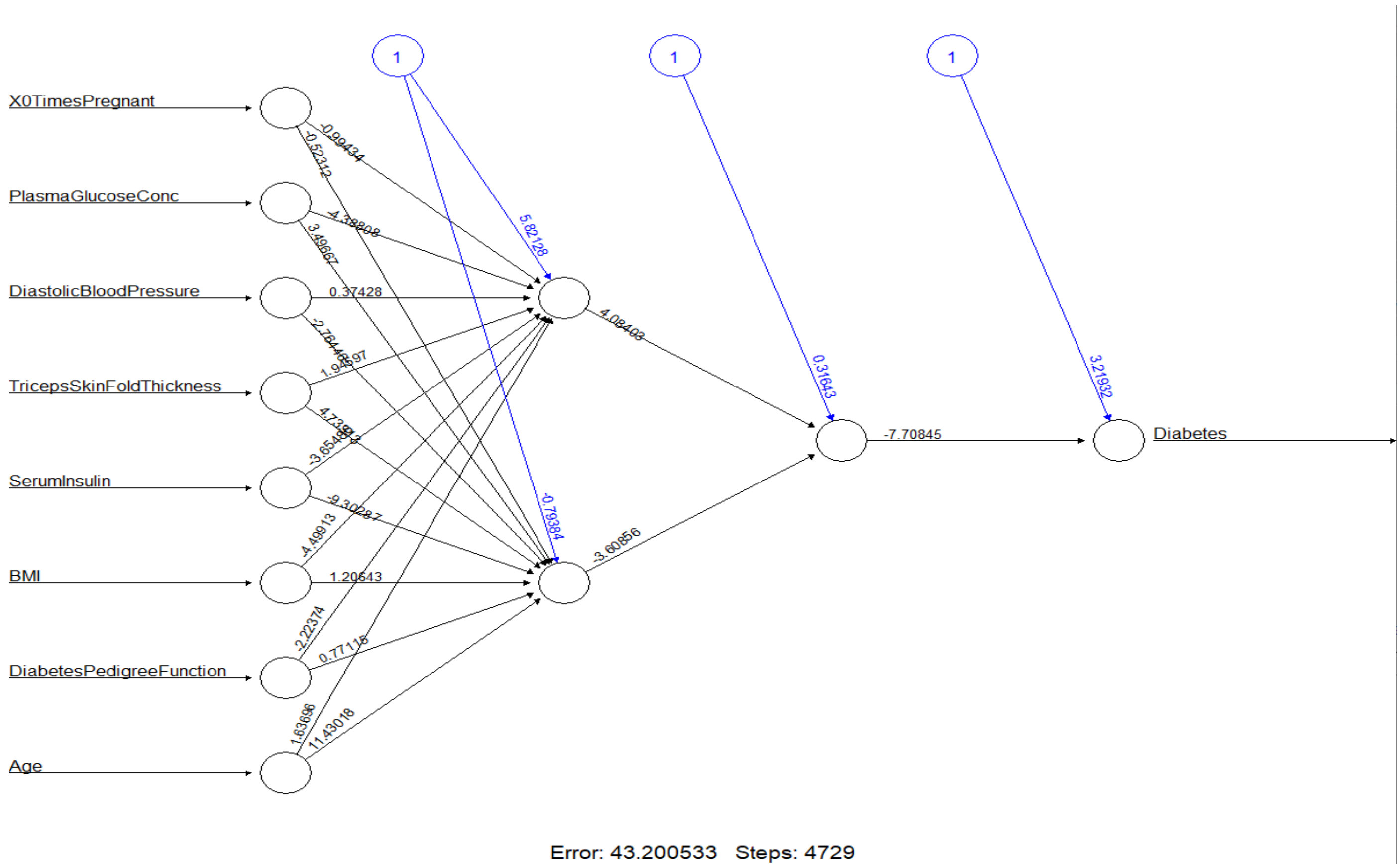
```
> model1

Call:
 randomForest(formula = Diabetes ~ ., data = TrainSet, importance = TRUE)
               Type of random forest: classification
                     Number of trees: 500
No. of variables tried at each split: 2

        OOB estimate of  error rate: 23.13%
Confusion matrix:
     NO YES class.error
NO  348  54   0.1343284
YES  88 124   0.4150943
```

```
> # Predicting on train set
> predTrain <- predict(model1, TrainSet, type = "class")
> # Checking classification accuracy
> table(predTrain, TrainSet$Diabetes)

predTrain  NO YES
      NO  402   0
      YES   0 212
> # Predicting on Validation set
> predValid <- predict(model1, ValidSet, type = "class")
> # Checking classification accuracy
> table(predValid,ValidSet$Diabetes)

predValid NO YES
      NO  86  25
      YES 12  31
> mean(predValid == ValidSet$Diabetes)
[1] 0.7597403
```

Neural Network

```r
# Read the Data
diabetes <- read.csv("C:\\Program Files\\Weka-3-8\\data\\diabetes-.csv", sep = ',', header = TRUE)
attach(diabetes)

#Scaled Normalization
scaleddata<-scale(diabetes)

#Max-Min Normalization
normalize <- function(x) {
  return ((x - min(x)) / (max(x) - min(x)))
}
maxmindf <- as.data.frame(lapply(diabetes, normalize))

# Training and Test Data %80-%20
trainset <- maxmindf[1:614, ]
testset <- maxmindf[615:768, ]

#Training a Neural Network Model using neuralnet
library(neuralnet)
nn <- neuralnet(Diabetes ~ X0TimesPregnant + PlasmaGlucoseConc + DiastolicBloodPressure +
TricepsSkinFoldThickness + SerumInsulin + BMI + DiabetesPedigreeFunction +Age, data=trainset, hidden=c(2,1), linear.output=FALSE, threshold=0.01)
nn$result.matrix
plot(nn)

#Testing The Accuracy Of The Model
temp_test <- subset(testset, select = c("X0TimesPregnant","PlasmaGlucoseConc","DiastolicBloodPressure", "TricepsSkinFoldThickness",
"SerumInsulin", "BMI","DiabetesPedigreeFunction","Age"))
head(temp_test)
nn.results <- compute(nn, temp_test)
results <- data.frame(actual = testset$Diabetes, prediction = nn.results$net.result)

#Confusion Matrix
roundedresults<-sapply(results,round,digits=0)
roundedresultsdf=data.frame(roundedresults)
attach(roundedresultsdf)
table(actual,prediction)
```

Output:



```
> head(temp_test)
    X0TimesPregnant PlasmaGlucoseConc DiastolicBloodPressure TricepsSkinFoldThickness SerumInsulin       BMI
615       0.6470588         0.6934673              0.6065574                0.2626263    0.1702128 0.5380030
616       0.1764706         0.5326633              0.5901639                0.0000000    0.0000000 0.3845007
617       0.3529412         0.5879397              0.7868852                0.0000000    0.0000000 0.4277198
618       0.1176471         0.3417085              0.5081967                0.1313131    0.0177305 0.2995529
619       0.5294118         0.5628141              0.6721311                0.2424242    0.0000000 0.4202683
620       0.0000000         0.5979899              0.0000000                0.0000000    0.0000000 0.4828614
    DiabetesPedigreeFunction        Age
615               0.20452605 0.48333333
616               0.05508113 0.10000000
617               0.03373185 0.15000000
618               0.07643040 0.03333333
619               0.51409052 0.48333333
620               0.02690009 0.05000000
```

```
> nn$result.matrix
                                             [,1]
error                                4.320053e+01
reached.threshold                    9.937623e-03
steps                                4.729000e+03
Intercept.to.1layhid1                5.821279e+00
X0TimesPregnant.to.1layhid1         -9.943362e-01
PlasmaGlucoseConc.to.1layhid1       -4.388082e+00
DiastolicBloodPressure.to.1layhid1   3.742786e-01
TricepsSkinFoldThickness.to.1layhid1 1.945970e+00
SerumInsulin.to.1layhid1            -3.654856e+00
BMI.to.1layhid1                     -4.499135e+00
DiabetesPedigreeFunction.to.1layhid1 -2.223742e+00
Age.to.1layhid1                      1.636956e+00
Intercept.to.1layhid2               -7.938379e-01
X0TimesPregnant.to.1layhid2         -5.231193e-01
PlasmaGlucoseConc.to.1layhid2        3.496672e+00
DiastolicBloodPressure.to.1layhid2  -2.764459e+00
TricepsSkinFoldThickness.to.1layhid2 4.733127e+00
SerumInsulin.to.1layhid2            -9.302873e+00
BMI.to.1layhid2                      1.206430e+00
DiabetesPedigreeFunction.to.1layhid2 7.711523e-01
Age.to.1layhid2                      1.143018e+01
Intercept.to.2layhid1                3.164293e-01
1layhid1.to.2layhid1                 4.084031e+00
1layhid2.to.2layhid1                -3.608559e+00
Intercept.to.Diabetes                3.219325e+00
2layhid1.to.Diabetes                -7.708448e+00
```

```
$net.result
          [,1]
615 0.77665766
616 0.06212715
617 0.09912713
618 0.02287310
619 0.37935852
620 0.34133622
621 0.19079303
622 0.24055323
623 0.93723970
624 0.06269083
625 0.03862114
626 0.16018191
627 0.04205176
628 0.10494110
629 0.39292667
630 0.03966441
631 0.38773735
632 0.08916785
633 0.05835261
634 0.02382234
635 0.09581610
```

```
> table(actual,prediction)
      prediction
actual  0  1
     0 86 13
     1 21 34
```

```
> (86+34)/(86+34+21+13)
[1] 0.7792208
```

As you see that neural network classification has the highest accuracy.

Assignment#8:

Pre-trained models in Keras

- Use the pre-trained models VGG16 and VGG19 in Keras to classify images.
- Test on other images than the examples. Are they classified correctly?

Solution:

Convolutional neural networks are now capable of outperforming humans on some computer vision tasks, such as classifying images. That is, given a photograph of an object, answer the question as to which of 1,000 specific objects the photograph shows. A competition-winning model for this task is the VGG model by researchers at Oxford. What is important about this model, besides its capability of classifying objects in photographs, is that the model weights are freely available and can be loaded and used in your own models and applications.

VGG16

I used water bottle, car ,signboard images for testing VGG16 algorithm.

In [1]:
```
from tensorflow.keras.applications.vgg16 import VGG16
model = VGG16()
```

In [2]:
```
print(model.summary())
```

```
Model: "vgg16"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_1 (InputLayer)         [(None, 224, 224, 3)]     0
_________________________________________________________________
block1_conv1 (Conv2D)        (None, 224, 224, 64)      1792
_________________________________________________________________
block1_conv2 (Conv2D)        (None, 224, 224, 64)      36928
_________________________________________________________________
block1_pool (MaxPooling2D)   (None, 112, 112, 64)      0
_________________________________________________________________
block2_conv1 (Conv2D)        (None, 112, 112, 128)     73856
_________________________________________________________________
block2_conv2 (Conv2D)        (None, 112, 112, 128)     147584
_________________________________________________________________
block2_pool (MaxPooling2D)   (None, 56, 56, 128)       0
_________________________________________________________________
block3_conv1 (Conv2D)        (None, 56, 56, 256)       295168
_________________________________________________________________
block3_conv2 (Conv2D)        (None, 56, 56, 256)       590080
_________________________________________________________________
block3_conv3 (Conv2D)        (None, 56, 56, 256)       590080
_________________________________________________________________
block3_pool (MaxPooling2D)   (None, 28, 28, 256)       0
```

```
block4_conv1 (Conv2D)        (None, 28, 28, 512)       1180160
_________________________________________________________________
block4_conv2 (Conv2D)        (None, 28, 28, 512)       2359808
_________________________________________________________________
block4_conv3 (Conv2D)        (None, 28, 28, 512)       2359808
_________________________________________________________________
block4_pool (MaxPooling2D)   (None, 14, 14, 512)       0
_________________________________________________________________
block5_conv1 (Conv2D)        (None, 14, 14, 512)       2359808
_________________________________________________________________
block5_conv2 (Conv2D)        (None, 14, 14, 512)       2359808
_________________________________________________________________
block5_conv3 (Conv2D)        (None, 14, 14, 512)       2359808
_________________________________________________________________
block5_pool (MaxPooling2D)   (None, 7, 7, 512)         0
_________________________________________________________________
flatten (Flatten)            (None, 25088)             0
_________________________________________________________________
fc1 (Dense)                  (None, 4096)              102764544
_________________________________________________________________
fc2 (Dense)                  (None, 4096)              16781312
_________________________________________________________________
predictions (Dense)          (None, 1000)              4097000
=================================================================
Total params: 138,357,544
Trainable params: 138,357,544
Non-trainable params: 0
```

In [3]:
```
from keras.preprocessing.image import load_img
# load an image from file
image = load_img('araba.jpg', target_size=(224, 224))
```
```
Using TensorFlow backend.
```

In [4]:
```
from keras.preprocessing.image import img_to_array
# convert the image pixels to a numpy array
image = img_to_array(image)
```

In [5]:
```
# reshape data for the model
image = image.reshape((1, image.shape[0], image.shape[1], image.shape[2]))
```

In [6]:
```
from keras.applications.vgg16 import preprocess_input
# prepare the image for the VGG model
image = preprocess_input(image)
```

In [7]:
```
# predict the probability across all output classes
prob = model.predict(image)
```

In [8]:
```
from keras.applications.vgg16 import decode_predictions
# convert the probabilities to class labels
label = decode_predictions(prob)
# retrieve the most likely result, e.g. highest probability
label = label[0][0]
# print the classification
print('%s (%.2f%%)' % (label[1], label[2]*100))
```

OUTPUTS:

racer (61.10%)

water_bottle (99.59%)


```
street_sign (90.77%)
```

VGG19

I used water bottle, car, signboard, monkey, dog, cat, jeep images for testing VGG19 algorithm.

In [1]:
```
from tensorflow.keras.applications.vgg19 import VGG19
model = VGG19()
```

In [2]:
```
print(model.summary())
```

```
Model: "vgg19"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_1 (InputLayer)         [(None, 224, 224, 3)]     0
_________________________________________________________________
block1_conv1 (Conv2D)        (None, 224, 224, 64)      1792
_________________________________________________________________
block1_conv2 (Conv2D)        (None, 224, 224, 64)      36928
_________________________________________________________________
block1_pool (MaxPooling2D)   (None, 112, 112, 64)      0
_________________________________________________________________
block2_conv1 (Conv2D)        (None, 112, 112, 128)     73856
_________________________________________________________________
block2_conv2 (Conv2D)        (None, 112, 112, 128)     147584
_________________________________________________________________
block2_pool (MaxPooling2D)   (None, 56, 56, 128)       0
_________________________________________________________________
block3_conv1 (Conv2D)        (None, 56, 56, 256)       295168
_________________________________________________________________
block3_conv2 (Conv2D)        (None, 56, 56, 256)       590080
_________________________________________________________________
block3_conv3 (Conv2D)        (None, 56, 56, 256)       590080
_________________________________________________________________
block3_conv4 (Conv2D)        (None, 56, 56, 256)       590080
_________________________________________________________________
block3_pool (MaxPooling2D)   (None, 28, 28, 256)       0
_________________________________________________________________
block4_conv1 (Conv2D)        (None, 28, 28, 512)       1180160
_________________________________________________________________
block4_conv2 (Conv2D)        (None, 28, 28, 512)       2359808
_________________________________________________________________
block4_conv3 (Conv2D)        (None, 28, 28, 512)       2359808
_________________________________________________________________
block4_conv4 (Conv2D)        (None, 28, 28, 512)       2359808
_________________________________________________________________
block4_pool (MaxPooling2D)   (None, 14, 14, 512)       0
_________________________________________________________________
block5_conv1 (Conv2D)        (None, 14, 14, 512)       2359808
_________________________________________________________________
block5_conv2 (Conv2D)        (None, 14, 14, 512)       2359808
_________________________________________________________________
block5_conv3 (Conv2D)        (None, 14, 14, 512)       2359808
_________________________________________________________________
block5_conv4 (Conv2D)        (None, 14, 14, 512)       2359808
_________________________________________________________________
block5_pool (MaxPooling2D)   (None, 7, 7, 512)         0
_________________________________________________________________
flatten (Flatten)            (None, 25088)             0
_________________________________________________________________
fc1 (Dense)                  (None, 4096)              102764544
_________________________________________________________________
fc2 (Dense)                  (None, 4096)              16781312
_________________________________________________________________
predictions (Dense)          (None, 1000)              4097000
=================================================================
Total params: 143,667,240
Trainable params: 143,667,240
Non-trainable params: 0
```

```
In [3]: from keras.preprocessing.image import load_img
        # load an image from file
        image = load_img('signboard.jpg', target_size=(224, 224))

        Using TensorFlow backend.

In [4]: from keras.preprocessing.image import img_to_array
        # convert the image pixels to a numpy array
        image = img_to_array(image)

In [5]: # reshape data for the model
        image = image.reshape((1, image.shape[0], image.shape[1], image.shape[2]))

In [6]: from keras.applications.vgg16 import preprocess_input
        # prepare the image for the VGG model
        image = preprocess_input(image)

In [7]: # predict the probability across all output classes
        prob = model.predict(image)

In [8]: from keras.applications.vgg16 import decode_predictions
        # convert the probabilities to class labels
        label = decode_predictions(prob)
        # retrieve the most likely result, e.g. highest probability
        label = label[0][0]
        # print the classification
        print('%s (%.2f%%)' % (label[1], label[2]*100))
```

street_sign (91.03%)

sports_car (52.97%)

water_bottle (98.88%)

tabby (51.24%)

Labrador_retriever (36.40%)

patas (94.09%)

jeep (29.14%)

As you see, they are correctly classified, the algorithm even can understand type of cars and different species of animals.