```
import os
import logging

import pandas as pd
import tensorflow.keras as keras

from tensorflow.keras.callbacks import ModelCheckpoint
from tensorflow.keras.utils import plot_model

# Log setting
logging.basicConfig(format="%(asctime)s %(levelname)s %(message)s", datefmt="%H:%M:%S", level

# Change display.max_rows to show all features.
pd.set_option("display.max_rows", 85)


df_train = pd.read_csv(('/content/drive/MyDrive/datasets/train_MachineLearningCVE.csv'), skip
logging.info("Class distribution\n{}".format(df_train.Label.value_counts()))
df_test = pd.read_csv(('/content/drive/MyDrive/datasets/test_MachineLearningCVE.csv'), skipin
logging.info("Class distribution\n{}".format(df_test.Label.value_counts()))


df_train.Label.unique()
```

```
array([ 0, 10,  4,  7,  3,  5,  6, 11,  1, 12, 14,  9,  8, 13,  2])
```

```
df.head()
```

	Destination Port	Flow Duration	Total Fwd Packets	Total Backward Packets	Total Length of Fwd Packets	Total Length of Bwd Packets	Fwd Packet Length Max	Fwd Packet Length Min	Fw Packe Lengt Mea
0	80	6018089.0	5	3	177	994	159	0	35.40000
1	443	323049.0	8	6	531	3208	194	0	66.37500
2	80	39270118.0	9	10	898	3944	431	0	99.77777
3	4848	43.0	1	1	2	6	2	2	2.00000
4	80	5754816.0	3	1	12	0	6	0	4.00000

5 rows × 79 columns

```
df_test.Label.unique()
```

```
array([ 0,  4, 10,  3,  7,  6,  5, 11, 12,  1, 14,  9,  8, 13,  2])
```

```
df = pd.concat([df_train, df_test], axis=0, copy=True)

df.Label.unique()
```

```
array([ 0, 10,  4,  7,  3,  5,  6, 11,  1, 12, 14,  9,  8, 13,  2])
```

```
import numpy as np
import tensorflow as tf
import matplotlib.pyplot as plt

from sklearn.metrics import classification_report
from sklearn.preprocessing import MinMaxScaler

def preprocessing(df: pd.DataFrame) -> (np.ndarray, np.ndarray):
    # Shuffle the dataset
    df = df.sample(frac=1)

    # Split features and labels
    x = df.iloc[:, df.columns != 'Label']
    y = df[['Label']].to_numpy()

    # Scale the features between 0 ~ 1
    scaler = MinMaxScaler()
    x = scaler.fit_transform(x)

    return x, y

x, y = preprocessing(df)


from sklearn.model_selection import train_test_split
X_train,X_test,y_train,y_test = train_test_split(x, y, train_size=0.7, random_state=42)


print(X_train.shape)
print(X_test.shape)
print(y_train.shape)
print(y_test.shape)
```

```
(1981520, 78)
(849223, 78)
(1981520, 1)
(849223, 1)
```

```
logging.info("Class distribution\n{}".format(df.Label.value_counts()))


print(y_train)
```

```
[[0]
 [4]
 [0]
```

```
...
[2]
[0]
[0]]
```

Training a perceptron via scikit-learn

```
from sklearn.preprocessing import StandardScaler

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

```
from sklearn.linear_model import Perceptron

ppn = Perceptron(eta0=0.1, random_state=1)
ppn.fit(X_train_std, y_train)
```

```
/usr/local/lib/python3.7/dist-packages/sklearn/utils/validation.py:993: DataConversionWa
  y = column_or_1d(y, warn=True)
Perceptron(eta0=0.1, random_state=1)
```

```
y_pred = ppn.predict(X_test_std)
# print('Misclassified examples: %d' % (y_test != y_pred).sum())
```

```
from sklearn.metrics import accuracy_score

print('Accuracy: %.3f' % accuracy_score(y_test, y_pred))
```

```
Accuracy: 0.972
```

```
print('Accuracy: %.3f' % ppn.score(X_test_std, y_test))
```

```
Accuracy: 0.972
```

```
from matplotlib.colors import ListedColormap
import matplotlib.pyplot as plt


def plot_decision_regions(X, y, classifier, test_idx=None, resolution=0.02):

    # setup marker generator and color map
    markers = ('s', 'x', 'o', '^', 'v')
    colors = ('red', 'blue', 'lightgreen', 'gray', 'cyan')
```

```
    cmap = ListedColormap(colors[:len(np.unique(y))])

    # plot the decision surface
    x1_min, x1_max = X[:, 0].min() - 1, X[:, 0].max() + 1
    x2_min, x2_max = X[:, 1].min() - 1, X[:, 1].max() + 1
    xx1, xx2 = np.meshgrid(np.arange(x1_min, x1_max, resolution),
                           np.arange(x2_min, x2_max, resolution))
    Z = classifier.predict(np.array([xx1.ravel(), xx2.ravel()]).T)
    Z = Z.reshape(xx1.shape)
    plt.contourf(xx1, xx2, Z, alpha=0.3, cmap=cmap)
    plt.xlim(xx1.min(), xx1.max())
    plt.ylim(xx2.min(), xx2.max())

    for idx, cl in enumerate(np.unique(y)):
        plt.scatter(x=X[y == cl, 0],
                    y=X[y == cl, 1],
                    alpha=0.8,
                    c=colors[idx],
                    marker=markers[idx],
                    label=cl,
                    edgecolor='black')

    # highlight test examples
    if test_idx:
        # plot all examples
        X_test, y_test = X[test_idx, :], y[test_idx]

        plt.scatter(X_test[:, 0],
                    X_test[:, 1],
                    c='',
                    edgecolor='black',
                    alpha=1.0,
                    linewidth=1,
                    marker='o',
                    s=100,
                    label='test set')


print(X_train_std.shape)
X_train_std = X_train_std.reshape(154558560,1)
X_train_std = X_train_std[1:66239395]
print(X_train_std.shape)
X_test_std = X_test_std.reshape(66239394,1)
print(X_test_std.shape)
```

```
 (1981520, 78)
 (66239394, 1)
 (66239394, 1)
```

```
y_train.shape
```

```
(1981520, 1)
```

```
y_test.shape
```

```
(849223, 1)
```

```
X_combined_std = np.vstack((X_train_std, X_test_std))
y_combined = np.hstack((y_train, y_test))

plot_decision_regions(X=X_combined_std, y=y_combined,
                      classifier=ppn, test_idx=range(105, 150))
plt.xlabel('petal length [standardized]')
plt.ylabel('petal width [standardized]')
plt.legend(loc='upper left')

plt.tight_layout()
#plt.savefig('images/03_01.png', dpi=300)
plt.show()
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
<ipython-input-20-89c32ac889b8> in <module>
      1 X_combined_std = np.vstack((X_train_std, X_test_std))
----> 2 y_combined = np.hstack((y_train, y_test))
      3
      4 plot_decision_regions(X=X_combined_std, y=y_combined,
      5                       classifier=ppn, test_idx=range(105, 150))

<__array_function__ internals> in hstack(*args, **kwargs)

/usr/local/lib/python3.7/dist-packages/numpy/core/shape_base.py in hstack(tup)
    343         return _nx.concatenate(arrs, 0)
    344     else:
--> 345         return _nx.concatenate(arrs, 1)
    346
    347

<__array_function__ internals> in concatenate(*args, **kwargs)

ValueError: all the input array dimensions for the concatenation axis must match exactly
but along dimension 0, the array at index 0 has size 1981520 and the array at index 1 ha
size 849223
```

Colab paid products - Cancel contracts here