Neural network

December 22, 2022

[1]:
```python
# We import some useful libraries
import pandas as pd
import numpy as np
import warnings
import sklearn

# See https://www.youtube.com/watch?v=z-ZR_8BZ1wQ&ab_channel=codebasics
import tensorflow as tf
from tensorflow.keras import initializers
from keras.models import Sequential
from keras.layers import Activation, Dense
from tensorflow.keras import regularizers
```

0.0.1 Train-test split

We drop out some outliers with the interquartile range and standarize the data based on the train transformation

[2]:
```python
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler

data = pd.read_csv("train.csv")
data['Lead'].replace({'Male':1, 'Female':0}, inplace = True)

X = data.drop(columns = ["Lead"])
y = data['Lead']

X_train, X_test, y_train, y_test = train_test_split(X, y, random_state = 4045)

columns = X_train.columns.tolist()

names = ['Number words male', "Year", "Gross"]

for name in names:
    q3 = X_train[name].quantile(0.75)
    q1 = X_train[name].quantile(0.25)
    iqr = q3 - q1
    upper_limit = q3 +2.5*iqr
    # Let's impute the values now
    X_train.loc[X_train[name] > upper_limit, name] = np.median(X_train[name])

PredictorScaler = StandardScaler()

# Storing the fit object
PredictorScalerFit = PredictorScaler.fit(X_train)

# Generating the standardized values
X_train = PredictorScalerFit.transform(X_train)

# Generating the standardized values
X_test = PredictorScalerFit.transform(X_test)
```

[23]:
```
X_train.shape, X_test.shape
```

[23]:
```
((779, 13), (260, 13))
```

[24]:
```
# Oversampling

# from imblearn.over_sampling import RandomOverSampler
# # ros = RandomOverSampler(sampling_strategy=1) # Float
# ros = RandomOverSampler(sampling_strategy="not majority") # String
# X_train, y_train = ros.fit_resample(X_train, y_train)

# ax = y_train.value_counts().plot.pie(autopct='%.2f')
# _ = ax.set_title("Over-sampling")
```

[25]:
```
import numpy as np
import tensorflow as tf
from sklearn.model_selection import GridSearchCV
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense
from scikeras.wrappers import KerasClassifier
from tensorflow.keras.layers import Dropout
from tensorflow.keras.constraints import MaxNorm
```

[32]:
```
# Function to create model, required for KerasClassifier
def create_model(neurons1):

    model = Sequential()

    # Defining the input layer and FIRST hidden layer, both are same! Relu␣
 ↪means Rectifier linear unit function
    model.add(Dense(neurons1, input_dim = 13, activation='relu'))
    # Defining the Output layer. Sigmoid means sigmoid activation function. For␣
↪Multiclass classification,
    # the activation ='softmax'. And output_dim will be equal to the number of␣
↪different classes
    model.add(Dense(1, activation='sigmoid'))

    # Optimizer == the algorithm of SGG to keep updating weights
    # loss == the loss function to measure the accuracy
    # metrics == the way we will compare the accuracy after each step of SGD
    model.compile(loss = 'binary_crossentropy', metrics = ['accuracy'])
    return model

# Create model
model = KerasClassifier(model = create_model, epochs = 150, verbose = 0)

# Define the grid search parameters
neurons1 = [13, 12, 11, 10, 9, 8, 7, 6, 5, 4]

param_grid = dict(model__neurons1 = neurons1)

# By default batch_sizeit = 1, which is the one that has proven to work best␣
↪with the combination of the other hyperparameters
grid = GridSearchCV(estimator = model, param_grid = param_grid, n_jobs = -1, cv␣
↪= 5)
grid_result = grid.fit(X_train, y_train)

# Summarize the results
print("Best: %f using %s" % (grid_result.best_score_, grid_result.best_params_))

means = grid_result.cv_results_['mean_test_score']
stds = grid_result.cv_results_['std_test_score']
params = grid_result.cv_results_['params']

for mean, stdev, param in zip(means, stds, params):
    print("%f (%f) with: %r" % (mean, stdev, param))
```

```
Best: 0.883176 using {'model__neurons1': 7}
0.879330 (0.026718) with: {'model__neurons1': 13}
0.881919 (0.027603) with: {'model__neurons1': 12}
0.875476 (0.013858) with: {'model__neurons1': 11}
0.870298 (0.028777) with: {'model__neurons1': 10}
0.880604 (0.022495) with: {'model__neurons1': 9}
0.876791 (0.028110) with: {'model__neurons1': 8}
0.883176 (0.020474) with: {'model__neurons1': 7}
0.862630 (0.019378) with: {'model__neurons1': 6}
0.876791 (0.038263) with: {'model__neurons1': 5}
0.879347 (0.018670) with: {'model__neurons1': 4}
```

[33]:
```python
def create_model(neurons1, neurons2):

    model = Sequential()

    model.add(Dense(neurons1, input_dim = 13, activation='relu'))

    # Defining the SECOND hidden layer, here we have not defined input because␣
 ↪it is
    # it will get input as the output of first hidden layer
    model.add(Dense(neurons2, activation='relu'))

    model.add(Dense(1, activation='sigmoid'))

    model.compile(loss='binary_crossentropy', metrics=['accuracy'])
    return model

model = KerasClassifier(model=create_model, epochs=150, verbose=0)

neurons1 = [10, 9, 8, 7]
neurons2 = [6, 5, 4, 3]

param_grid = dict(model__neurons1 = neurons1, model__neurons2 = neurons2)

grid = GridSearchCV(estimator=model, param_grid=param_grid, n_jobs=-1, cv=5)
grid_result = grid.fit(X_train, y_train)

print("Best: %f using %s" % (grid_result.best_score_, grid_result.best_params_))

means = grid_result.cv_results_['mean_test_score']
stds = grid_result.cv_results_['std_test_score']
params = grid_result.cv_results_['params']

for mean, stdev, param in zip(means, stds, params):
    print("%f (%f) with: %r" % (mean, stdev, param))
```

```
Best: 0.885757 using {'model__neurons1': 10, 'model__neurons2': 6}
0.885757 (0.031747) with: {'model__neurons1': 10, 'model__neurons2': 6}
0.871646 (0.019347) with: {'model__neurons1': 10, 'model__neurons2': 5}
0.880596 (0.023962) with: {'model__neurons1': 10, 'model__neurons2': 4}
0.876782 (0.025693) with: {'model__neurons1': 10, 'model__neurons2': 3}
0.861332 (0.019927) with: {'model__neurons1': 9, 'model__neurons2': 6}
0.881902 (0.023133) with: {'model__neurons1': 9, 'model__neurons2': 5}
0.876749 (0.016544) with: {'model__neurons1': 9, 'model__neurons2': 4}
0.869024 (0.026071) with: {'model__neurons1': 9, 'model__neurons2': 3}
0.870339 (0.020479) with: {'model__neurons1': 8, 'model__neurons2': 6}
0.884442 (0.015917) with: {'model__neurons1': 8, 'model__neurons2': 5}
0.874160 (0.018707) with: {'model__neurons1': 8, 'model__neurons2': 4}
0.871671 (0.025487) with: {'model__neurons1': 8, 'model__neurons2': 3}
```

```
0.876774 (0.023383) with: {'model__neurons1': 7, 'model__neurons2': 6}
0.856203 (0.024335) with: {'model__neurons1': 7, 'model__neurons2': 5}
0.866402 (0.046708) with: {'model__neurons1': 7, 'model__neurons2': 4}
0.869041 (0.019859) with: {'model__neurons1': 7, 'model__neurons2': 3}
```

[14]:
```
def create_model():

    model = Sequential()
    model.add(Dense(7, input_dim=13, activation='relu'))
    model.add(Dense(1, activation='sigmoid'))

    model.compile(loss='binary_crossentropy', metrics=['accuracy'])
    return model


model = KerasClassifier(model=create_model, verbose=0)

epochs = [10, 25 ,50, 100, 150, 175, 200, 220, 250]
param_grid = dict(epochs = epochs)

grid = GridSearchCV(estimator = model, param_grid = param_grid, n_jobs = -1, cv␣
 ↪= 5)
grid_result = grid.fit(X_train, y_train)

print("Best: %f using %s" % (grid_result.best_score_, grid_result.best_params_))

means = grid_result.cv_results_['mean_test_score']
stds = grid_result.cv_results_['std_test_score']
params = grid_result.cv_results_['params']

for mean, stdev, param in zip(means, stds, params):
    print("%f (%f) with: %r" % (mean, stdev, param))
```

```
Best: 0.885749 using {'epochs': 150}
0.753548 (0.012862) with: {'epochs': 10}
0.792060 (0.018645) with: {'epochs': 25}
0.844665 (0.009542) with: {'epochs': 50}
0.840827 (0.014777) with: {'epochs': 100}
0.885749 (0.022728) with: {'epochs': 150}
0.863954 (0.028962) with: {'epochs': 175}
0.858842 (0.025764) with: {'epochs': 200}
0.879297 (0.030884) with: {'epochs': 220}
0.865219 (0.030306) with: {'epochs': 250}
```

We will select 150, since it is the threshold where the performance continuously increases and suffers a decrease, continuing to modify the parameters of the model by iterating more epochs no longer makes sense and can cause the overfitting to increase.

[15]:
```
def create_model():

    model = Sequential()
    model.add(Dense(7, input_dim = 13, activation='relu'))
    model.add(Dense(1, activation='sigmoid'))

    return model

model = KerasClassifier(model = create_model, loss="binary_crossentropy",␣
 ↪epochs=150, verbose=0)

optimizer = ['SGD', 'RMSprop', 'Adagrad', 'Adadelta', 'Adam', 'Adamax', 'Nadam']
param_grid = dict(optimizer = optimizer)

grid = GridSearchCV(estimator = model, param_grid = param_grid, n_jobs = -1, cv␣
 ↪= 5)
grid_result = grid.fit(X_train, y_train)

print("Best: %f using %s" % (grid_result.best_score_, grid_result.best_params_))

means = grid_result.cv_results_['mean_test_score']
stds = grid_result.cv_results_['std_test_score']
params = grid_result.cv_results_['params']

for mean, stdev, param in zip(means, stds, params):
    print("%f (%f) with: %r" % (mean, stdev, param))
```

```
Best: 0.884433 using {'optimizer': 'Adam'}
0.835666 (0.021850) with: {'optimizer': 'SGD'}
0.875467 (0.023227) with: {'optimizer': 'RMSprop'}
0.687866 (0.077832) with: {'optimizer': 'Adagrad'}
0.586873 (0.129930) with: {'optimizer': 'Adadelta'}
0.884433 (0.024852) with: {'optimizer': 'Adam'}
0.833085 (0.016110) with: {'optimizer': 'Adamax'}
0.875451 (0.037409) with: {'optimizer': 'Nadam'}
```

[16]:
```
def create_model(init_mode = 'glorot_normal'):

    model = Sequential()
    model.add(Dense(7, input_dim = 13, kernel_initializer=init_mode, activation␣
 ↪='relu'))
    model.add(Dense(1, kernel_initializer=init_mode, activation='sigmoid'))

    model.compile(loss='binary_crossentropy', optimizer='Adam',␣
 ↪metrics=['accuracy'])
    return model
```

```
model = KerasClassifier(model=create_model, epochs = 150, verbose = 0)

init_mode = [ "RandomNormal", "RandomUniform", "TruncatedNormal", "Zeros",␣
↪"Ones", "GlorotNormal", "GlorotUniform", "HeUniform", "Identity", ␣
↪"Orthogonal", "Constant", "VarianceScaling"]
param_grid = dict(model__init_mode = init_mode)

grid = GridSearchCV(estimator = model, param_grid = param_grid, n_jobs = -1, cv␣
↪= 5)
grid_result = grid.fit(X_train, y_train)

print("Best: %f using %s" % (grid_result.best_score_, grid_result.best_params_))

means = grid_result.cv_results_['mean_test_score']
stds = grid_result.cv_results_['std_test_score']
params = grid_result.cv_results_['params']

for mean, stdev, param in zip(means, stds, params):
    print("%f (%f) with: %r" % (mean, stdev, param))
```

```
Best: 0.872895 using {'model__init_mode': 'GlorotUniform'}
0.871613 (0.023038) with: {'model__init_mode': 'RandomNormal'}
0.866476 (0.025876) with: {'model__init_mode': 'RandomUniform'}
0.870331 (0.020935) with: {'model__init_mode': 'TruncatedNormal'}
0.757386 (0.001952) with: {'model__init_mode': 'Zeros'}
0.757386 (0.001952) with: {'model__init_mode': 'Ones'}
0.869057 (0.033108) with: {'model__init_mode': 'GlorotNormal'}
0.872895 (0.028298) with: {'model__init_mode': 'GlorotUniform'}
0.863904 (0.024940) with: {'model__init_mode': 'HeUniform'}
0.835691 (0.010347) with: {'model__init_mode': 'Identity'}
0.869065 (0.023832) with: {'model__init_mode': 'Orthogonal'}
0.757386 (0.001952) with: {'model__init_mode': 'Constant'}
0.866468 (0.013388) with: {'model__init_mode': 'VarianceScaling'}
```

[18]:
```
init = tf.keras.initializers.GlorotUniform(seed=123)

def create_model():

    model = Sequential()
    model.add(Dense(7, input_dim = 13, kernel_initializer = init,␣
↪activation='relu'))
    model.add(Dense(1, kernel_initializer = init, activation='sigmoid'))
    return model

model = KerasClassifier(model=create_model, loss="binary_crossentropy",␣
↪optimizer="Adam", epochs=150, verbose=0)
```

```
learn_rate = [0.001, 0.01, 0.1, 0.2, 0.3]

param_grid = dict(optimizer__learning_rate=learn_rate)

grid = GridSearchCV(estimator=model, param_grid=param_grid, n_jobs=-1, cv=5)
grid_result = grid.fit(X_train, y_train)


print("Best: %f using %s" % (grid_result.best_score_, grid_result.best_params_))

means = grid_result.cv_results_['mean_test_score']
stds = grid_result.cv_results_['std_test_score']
params = grid_result.cv_results_['params']

for mean, stdev, param in zip(means, stds, params):
    print("%f (%f) with: %r" % (mean, stdev, param))
```

```
Best: 0.907552 using {'optimizer__learning_rate': 0.01}
0.880604 (0.025573) with: {'optimizer__learning_rate': 0.001}
0.907552 (0.010642) with: {'optimizer__learning_rate': 0.01}
0.890852 (0.019252) with: {'optimizer__learning_rate': 0.1}
0.897246 (0.028393) with: {'optimizer__learning_rate': 0.2}
0.903681 (0.023166) with: {'optimizer__learning_rate': 0.3}
```

[21]:
```
init = tf.keras.initializers.GlorotUniform(seed=123)

def create_model(activation='relu'):

    model = Sequential()
    model.add(Dense(7, input_dim= 13, kernel_initializer = init, activation =␣
 ↪activation))
    model.add(Dense(1, kernel_initializer = init, activation='sigmoid'))

    model.compile(loss='binary_crossentropy', optimizer="Adam",␣
 ↪metrics=['accuracy'])
    return model

optimizer = tf.keras.optimizers.SGD(learning_rate=0.1)

model = KerasClassifier(model=create_model, epochs=150, verbose=0)

activation = ['softmax', 'softplus', 'softsign', 'relu', 'tanh', 'sigmoid',␣
 ↪'hard_sigmoid', 'linear']
param_grid = dict(model__activation=activation)

grid = GridSearchCV(estimator=model, param_grid=param_grid, n_jobs=-1, cv=5)
grid_result = grid.fit(X_train, y_train)
```

```
print("Best: %f using %s" % (grid_result.best_score_, grid_result.best_params_))

means = grid_result.cv_results_['mean_test_score']
stds = grid_result.cv_results_['std_test_score']
params = grid_result.cv_results_['params']

for mean, stdev, param in zip(means, stds, params):
    print("%f (%f) with: %r" % (mean, stdev, param))
```

```
Best: 0.883168 using {'model__activation': 'relu'}
0.857486 (0.016147) with: {'model__activation': 'softmax'}
0.861307 (0.033620) with: {'model__activation': 'softplus'}
0.876758 (0.029657) with: {'model__activation': 'softsign'}
0.883168 (0.022441) with: {'model__activation': 'relu'}
0.881894 (0.029979) with: {'model__activation': 'tanh'}
0.840794 (0.018127) with: {'model__activation': 'sigmoid'}
0.843366 (0.020297) with: {'model__activation': 'hard_sigmoid'}
0.854946 (0.013151) with: {'model__activation': 'linear'}
```

[22]:
```
init = tf.keras.initializers.GlorotUniform(seed=123)

def create_model(dropout_rate, weight_constraint):

    model = Sequential()
    model.add(Dense(7, input_dim = 13, kernel_initializer=init,␣
 ↪activation='relu', kernel_constraint = MaxNorm(weight_constraint)))
    model.add(Dense(1, kernel_initializer = init, activation='sigmoid'))

    model.compile(loss='binary_crossentropy', optimizer='Adam',␣
 ↪metrics=['accuracy'])
    return model

optimizer = tf.keras.optimizers.SGD(learning_rate=0.01)

model = KerasClassifier(model=create_model, epochs=150, verbose=0)

weight_constraint = [0.0, 1.0, 2.0, 3.0, 4.0, 5.0]
dropout_rate = [0.0, 0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9]
param_grid = dict(model__dropout_rate=dropout_rate,␣
 ↪model__weight_constraint=weight_constraint)


grid = GridSearchCV(estimator=model, param_grid=param_grid, n_jobs=-1, cv=5)
grid_result = grid.fit(X_train, y_train)

print("Best: %f using %s" % (grid_result.best_score_, grid_result.best_params_))
```

```python
means = grid_result.cv_results_['mean_test_score']
stds = grid_result.cv_results_['std_test_score']
params = grid_result.cv_results_['params']

for mean, stdev, param in zip(means, stds, params):
    print("%f (%f) with: %r" % (mean, stdev, param))
```

```
Best: 0.888313 using {'model__dropout_rate': 0.1, 'model__weight_constraint':
4.0}
0.757386 (0.001952) with: {'model__dropout_rate': 0.0,
'model__weight_constraint': 0.0}
0.885732 (0.023180) with: {'model__dropout_rate': 0.0,
'model__weight_constraint': 1.0}
0.881902 (0.024846) with: {'model__dropout_rate': 0.0,
'model__weight_constraint': 2.0}
0.879313 (0.024895) with: {'model__dropout_rate': 0.0,
'model__weight_constraint': 3.0}
0.884467 (0.023292) with: {'model__dropout_rate': 0.0,
'model__weight_constraint': 4.0}
0.883168 (0.024540) with: {'model__dropout_rate': 0.0,
'model__weight_constraint': 5.0}
0.757386 (0.001952) with: {'model__dropout_rate': 0.1,
'model__weight_constraint': 0.0}
0.885732 (0.023180) with: {'model__dropout_rate': 0.1,
'model__weight_constraint': 1.0}
0.880620 (0.025492) with: {'model__dropout_rate': 0.1,
'model__weight_constraint': 2.0}
0.878023 (0.027025) with: {'model__dropout_rate': 0.1,
'model__weight_constraint': 3.0}
0.888313 (0.021328) with: {'model__dropout_rate': 0.1,
'model__weight_constraint': 4.0}
0.884467 (0.025643) with: {'model__dropout_rate': 0.1,
'model__weight_constraint': 5.0}
0.757386 (0.001952) with: {'model__dropout_rate': 0.2,
'model__weight_constraint': 0.0}
0.879322 (0.017931) with: {'model__dropout_rate': 0.2,
'model__weight_constraint': 1.0}
0.881902 (0.022412) with: {'model__dropout_rate': 0.2,
'model__weight_constraint': 2.0}
0.884467 (0.025643) with: {'model__dropout_rate': 0.2,
'model__weight_constraint': 3.0}
0.881886 (0.023921) with: {'model__dropout_rate': 0.2,
'model__weight_constraint': 4.0}
0.880604 (0.024253) with: {'model__dropout_rate': 0.2,
'model__weight_constraint': 5.0}
0.757386 (0.001952) with: {'model__dropout_rate': 0.3,
'model__weight_constraint': 0.0}
```

```
0.884450 (0.024076) with: {'model__dropout_rate': 0.3,
'model__weight_constraint': 1.0}
0.881902 (0.016988) with: {'model__dropout_rate': 0.3,
'model__weight_constraint': 2.0}
0.884467 (0.023292) with: {'model__dropout_rate': 0.3,
'model__weight_constraint': 3.0}
0.884467 (0.023292) with: {'model__dropout_rate': 0.3,
'model__weight_constraint': 4.0}
0.883176 (0.025802) with: {'model__dropout_rate': 0.3,
'model__weight_constraint': 5.0}
0.757386 (0.001952) with: {'model__dropout_rate': 0.4,
'model__weight_constraint': 0.0}
0.888296 (0.021044) with: {'model__dropout_rate': 0.4,
'model__weight_constraint': 1.0}
0.883184 (0.019180) with: {'model__dropout_rate': 0.4,
'model__weight_constraint': 2.0}
0.883176 (0.023467) with: {'model__dropout_rate': 0.4,
'model__weight_constraint': 3.0}
0.878031 (0.026671) with: {'model__dropout_rate': 0.4,
'model__weight_constraint': 4.0}
0.881878 (0.025305) with: {'model__dropout_rate': 0.4,
'model__weight_constraint': 5.0}
0.757386 (0.001952) with: {'model__dropout_rate': 0.5,
'model__weight_constraint': 0.0}
0.888304 (0.020989) with: {'model__dropout_rate': 0.5,
'model__weight_constraint': 1.0}
0.883193 (0.021173) with: {'model__dropout_rate': 0.5,
'model__weight_constraint': 2.0}
0.884467 (0.025643) with: {'model__dropout_rate': 0.5,
'model__weight_constraint': 3.0}
0.885749 (0.021616) with: {'model__dropout_rate': 0.5,
'model__weight_constraint': 4.0}
0.880596 (0.024638) with: {'model__dropout_rate': 0.5,
'model__weight_constraint': 5.0}
0.757386 (0.001952) with: {'model__dropout_rate': 0.6,
'model__weight_constraint': 0.0}
0.885732 (0.025542) with: {'model__dropout_rate': 0.6,
'model__weight_constraint': 1.0}
0.881878 (0.025305) with: {'model__dropout_rate': 0.6,
'model__weight_constraint': 2.0}
0.880596 (0.021040) with: {'model__dropout_rate': 0.6,
'model__weight_constraint': 3.0}
0.879305 (0.025597) with: {'model__dropout_rate': 0.6,
'model__weight_constraint': 4.0}
0.884467 (0.023292) with: {'model__dropout_rate': 0.6,
'model__weight_constraint': 5.0}
0.757386 (0.001952) with: {'model__dropout_rate': 0.7,
'model__weight_constraint': 0.0}
```

```
0.885724 (0.020609) with: {'model__dropout_rate': 0.7,
'model__weight_constraint': 1.0}
0.883184 (0.023071) with: {'model__dropout_rate': 0.7,
'model__weight_constraint': 2.0}
0.884467 (0.025643) with: {'model__dropout_rate': 0.7,
'model__weight_constraint': 3.0}
0.884467 (0.025643) with: {'model__dropout_rate': 0.7,
'model__weight_constraint': 4.0}
0.884467 (0.025643) with: {'model__dropout_rate': 0.7,
'model__weight_constraint': 5.0}
0.757386 (0.001952) with: {'model__dropout_rate': 0.8,
'model__weight_constraint': 0.0}
0.884442 (0.027019) with: {'model__dropout_rate': 0.8,
'model__weight_constraint': 1.0}
0.884458 (0.023335) with: {'model__dropout_rate': 0.8,
'model__weight_constraint': 2.0}
0.883184 (0.023772) with: {'model__dropout_rate': 0.8,
'model__weight_constraint': 3.0}
0.883184 (0.025117) with: {'model__dropout_rate': 0.8,
'model__weight_constraint': 4.0}
0.883184 (0.021215) with: {'model__dropout_rate': 0.8,
'model__weight_constraint': 5.0}
0.757386 (0.001952) with: {'model__dropout_rate': 0.9,
'model__weight_constraint': 0.0}
0.885724 (0.022877) with: {'model__dropout_rate': 0.9,
'model__weight_constraint': 1.0}
0.884458 (0.024029) with: {'model__dropout_rate': 0.9,
'model__weight_constraint': 2.0}
0.883184 (0.025117) with: {'model__dropout_rate': 0.9,
'model__weight_constraint': 3.0}
0.880604 (0.028026) with: {'model__dropout_rate': 0.9,
'model__weight_constraint': 4.0}
0.884467 (0.023292) with: {'model__dropout_rate': 0.9,
'model__weight_constraint': 5.0}
```

[25]:
```
init = tf.keras.initializers.GlorotUniform(seed=123)

def create_model(values1, values2):
    model = Sequential()

    model.add(Dense(
    units = 7, input_dim = 13, kernel_initializer = init, activation = 'relu',␣
  ↪kernel_constraint=MaxNorm(4),
    kernel_regularizer= regularizers.L1(values1)))

    model.add(Dropout(0.1))
    model.add(Dense(units=1, kernel_initializer=init,  kernel_regularizer=␣
↪regularizers.L1(values2), activation='sigmoid'))


    model.compile(loss='binary_crossentropy', optimizer='RMSprop',␣
↪metrics=['accuracy'])


    return model

optimizer = tf.keras.optimizers.SGD(learning_rate = 0.01)

model = KerasClassifier(model = create_model, epochs = 150, verbose = 0)

# Grid search parameters
values1 = [1e-2, 1e-3, 1e-4, 1e-5]
values2 = [1e-2, 1e-3, 1e-4, 1e-5]

param_grid = dict(model__values1 = values1, model__values2 = values2)
grid = GridSearchCV(estimator=model, param_grid = param_grid, n_jobs=-1, cv=5)


grid_result = grid.fit(X_train, y_train)

print("Best: %f using %s" % (grid_result.best_score_, grid_result.best_params_))

means = grid_result.cv_results_['mean_test_score']
stds = grid_result.cv_results_['std_test_score']
params = grid_result.cv_results_['params']

for mean, stdev, param in zip(means, stds, params):
    print("%f (%f) with: %r" % (mean, stdev, param))
```

```
Best: 0.897295 using {'model__values1': 1e-05, 'model__values2': 1e-05}
0.867725 (0.026189) with: {'model__values1': 0.01, 'model__values2': 0.01}
0.876725 (0.023693) with: {'model__values1': 0.01, 'model__values2': 0.001}
0.876725 (0.021511) with: {'model__values1': 0.01, 'model__values2': 0.0001}
0.881861 (0.024423) with: {'model__values1': 0.01, 'model__values2': 1e-05}
0.881878 (0.027782) with: {'model__values1': 0.001, 'model__values2': 0.01}
0.889586 (0.018413) with: {'model__values1': 0.001, 'model__values2': 0.001}
0.893441 (0.020177) with: {'model__values1': 0.001, 'model__values2': 0.0001}
0.892167 (0.019199) with: {'model__values1': 0.001, 'model__values2': 1e-05}
0.885732 (0.027404) with: {'model__values1': 0.0001, 'model__values2': 0.01}
0.893449 (0.020929) with: {'model__values1': 0.0001, 'model__values2': 0.001}
0.894731 (0.021708) with: {'model__values1': 0.0001, 'model__values2': 0.0001}
0.894731 (0.024883) with: {'model__values1': 0.0001, 'model__values2': 1e-05}
0.887006 (0.026295) with: {'model__values1': 1e-05, 'model__values2': 0.01}
```

```
0.892159 (0.023481) with: {'model__values1': 1e-05, 'model__values2': 0.001}
0.896013 (0.023810) with: {'model__values1': 1e-05, 'model__values2': 0.0001}
0.897295 (0.022248) with: {'model__values1': 1e-05, 'model__values2': 1e-05}
```

[72]:
```
init = tf.keras.initializers.GlorotUniform(seed=123)

model = Sequential()

model.add(Dense(
units = 7, input_dim = 13, kernel_initializer = init, activation = 'relu',␣
 ↪kernel_constraint = MaxNorm(4),
kernel_regularizer= regularizers.L1(1e-05)))

model.add(Dropout(0.1))

model.add(Dense(units=1, kernel_initializer=init, kernel_regularizer=␣
 ↪regularizers.L1(1e-05), activation='sigmoid'))

model.compile(loss='binary_crossentropy', optimizer='Adam',␣
 ↪metrics=['accuracy'])

optimizer = tf.keras.optimizers.SGD(learning_rate = 0.01)

# Fit the model
model.fit(X_train, y_train, epochs = 150, verbose=0)
```

[72]:
```
<keras.callbacks.History at 0x2afb38eff40>
```

[74]:
```
from sklearn.metrics import accuracy_score, precision_score, recall_score,␣
 ↪f1_score

# Defining the probability threshold
def probThreshold(inpProb):
    if (inpProb >= 0.5):
        return(1)
    else:
        return(0)

Predictions_train = model.predict(X_train)
Train_pred = pd.DataFrame(data = Predictions_train)
Train_pred[0]= Train_pred[0].apply(probThreshold)

print('Training set metrics:')
print('Accuracy:', accuracy_score(y_train, Train_pred))
print('Precision:', precision_score(y_train, Train_pred))
print('Recall:', recall_score(y_train, Train_pred))
print('F1:', f1_score(y_train, Train_pred))
```

```


Predictions_test = model.predict(X_test)

# Generating a data frame for analyzing the test data
test_pred = pd.DataFrame(data = Predictions_test)
test_pred[0]= test_pred[0].apply(probThreshold)

print('\n')
print('Test set metrics:')
print('Accuracy:', accuracy_score(y_test, test_pred))
print('Precision:', precision_score(y_test, test_pred))
print('Recall:', recall_score(y_test, test_pred))
print('F1:', f1_score(y_test, test_pred))
```

```
25/25 [==============================] - 0s 748us/step
Training set metrics:
Accuracy: 0.9165596919127086
Precision: 0.9254457050243112
Recall: 0.9677966101694915
F1: 0.9461474730737366
9/9 [==============================] - 0s 992us/step


Test set metrics:
Accuracy: 0.9
Precision: 0.9121951219512195
Recall: 0.958974358974359
F1: 0.935
```

[19]:
```
from keras.wrappers.scikit_learn import KerasClassifier, KerasRegressor
import eli5
from eli5.sklearn import PermutationImportance

warnings.filterwarnings("ignore")

init = tf.keras.initializers.GlorotUniform(seed=123)

def base_model():

    model = Sequential()

    model.add(Dense(
    units = 7, input_dim = 13, kernel_initializer = init, activation = 'relu',␣
 ↪kernel_constraint=MaxNorm(4),
    kernel_regularizer= regularizers.L1(1e-05)))

    model.add(Dropout(0.1))

    model.add(Dense(units = 1, kernel_initializer = init, kernel_regularizer =␣
  ↪regularizers.L1(1e-05), activation = 'sigmoid'))

    model.compile(loss = 'binary_crossentropy', optimizer='Adam', metrics =␣
  ↪['accuracy'])

    return model

optimizer = tf.keras.optimizers.SGD(learning_rate = 0.01)


my_model = KerasClassifier(build_fn= base_model, epochs = 150, verbose = 0)
my_model.fit(X_train, y_train)

perm = PermutationImportance(my_model, random_state=1).fit(X_train,y_train)
eli5.show_weights(perm, feature_names = columns)
```

```
[19]: <IPython.core.display.HTML object>
```