DERİN ÖĞRENME YÖNTEMLERİ İLE VERİ SETİNİN EN İYİ PERFORMANSINI BULAN MODELLER

DENİZ CAN TOŞUR

BURDUR,2021

ÖZET

Bu rapor çalışmasında, Multivariable(Çok değişkenli) girişli 2006-2011 arasında bir bölgenin kullandığı elektrik tüketimini ‘Jupyter Notebook’ ile verileri ekrana gösterildi. Öncelikle Parti Boyutu ve Dönem Sayısı hesaplama modeli yapıldı. Daha sonra Eğitim Optimizasyon Algoritmasını Ayarlama modeli yapıldı. Üçüncü olarak Öğrenme Hızını ve Momentum Ayarlama modeli yapıldı. Dördüncü olarak Ağ Ağırlığı Başlatma ayarlama modeli yapıldı. Beşinci olarak Nöron Aktivasyon Fonksiyonu ayarlama modeli yapıldı. Altıncı olarak Bırakma Düzenlemesi ayarlama modeli yapıldı. Son olarak Gizli Katmandaki Nöron Sayısı hesaplama modeli yapıldı.

Anahtar Kelimeler: KerasClassifier, GridSearchCV,Sequential, Dense, Dropout

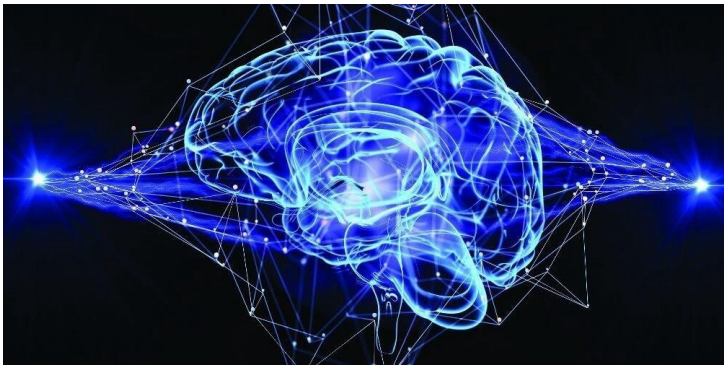
ABSTRACT

This report report is a school user 2006-2011 with Multivariable entry. Childhood and Number of Periods calculation model was made. Next, the Tuning Training Optimization Algorithm model was made. Third, the Learning Rate and Momentum Adjustment model was made. Fourth, the Network Weight Initialization model was made. Fifth, the Neuron Activation Function tuning model. Sixth, the Release Arrangement tuning pattern was made. Finally, the number of neurons in the hidden layer calculation model was made.

Keywords: KerasClassifier, GridSearchCV,Sequential, Dense, Dropout

1.GİRİŞ

Elektrik tüketimi son yıllarda yaşanan sıkıntılardan birisidir. Elektrik tüketimi hızla arttığından dağılımın doğru bir şekilde bazı aşamalardan geçip tahmin edilmesi gerekmektedir. Elektrik tüketiminde keskin yani doğru bir sonuç elde edebilmemiz için elektrik kullanımını takip etmemiz veya izlememiz gerekmektedir. Gözlemler sonucunda yapacağımız doğru bir tahmin bizi ileriki plansız elektrik tüketimlerini göz önüne alarak gereksiz elektrik dağıtımı önlenebilir. Ancak elektrik tüketimini etkileyen faktörlerin kullanılması sonucunda öngörülemeyen karmaşık bir tahmin modeli oluşabilir. Elektrik tüketimi zamana bağlı bir yapıdadır. Bu nedenle Elektrik tüketiminin tahmin modelini oluşturmak için belli başlı zaman serilerini kullanan yaklaşımlar vardır. Geçmişte kullanılan veriler, zaman serisi analizine dayalı çözümler ile zamana bağlı değişiklikleri ekrana yansıtır. Elektrik tüketimi tahminleri kısa vadeli(saatlik ile 1 hafta arası), orta vadeli (bir hafta ile bir yıl arası), uzun vadeli (bir yıldan fazla) olarak üç modelden oluşmaktadır.



Bu modelleri oluşturma aşamasında yardımımıza Derin Öğrenme yöntemleri ve Makine Öğrenimi Yöntemleri karşımıza çıkıyor. Derin Öğrenme ve Makine Öğrenimi yöntemlerinin çoğu zaman tahmine dayalı modelleme problemleri için bir çözüm anahtarı olduğu bilinmekte. Derin Öğrenme, bilgisayarlara insanların doğal olarak gelen bir şeyi yaptırmayı öğreten bir makine öğrenimi tekniğidir.

2. GENEL BİLGİLER

2.1 Veri Seti

Elektrik tüketimi konusunda tahmin modellerini yapabilmemiz için öncelikle veri setlerine ihtiyacımız bulunmaktadır. Kullanacağımız veri seti 2006-2011 yılları arasında enerji tüketimini gösteren veri seti ele alınmıştır. Veri seti hazırlanması Derin Öğrenme yönteminin ilk başlarında gelir. Veri seti ne kadar düzenli ise yapılacak işlemler o kadar daha kesin veya daha doğru sonuç elde edilmeye olanak sağlar.

Çok değişkenli veri setinin ilk 5 satırının görseli aşağıdaki gibidir.

	A	B	C	D	E	F	G	H	I	J	K	L	M	N
1	,Global_active_power,Global_reactive_power,Voltage,Global_intensity,Sub_metering_1,Sub_metering_2,Sub_metering_3,Sub_metering_4,Result													
2	0,1209,34,93552,5180,0,546,4926,14680,0													
3	1,3390,226,345725,14398,2033,4187,13341,36946,1													
4	2,2203,161,347373,9247,1063,2621,14018,19028,0													
5	3,1666,150,348479,7094,839,7602,6197,13131,1													

Veriler aşağıya doğru devam etmektedir.

3. MATERYAL VE YÖNTEM

3.1 Kütüphaneler ve Veri seti İşlemleri

```
In [1]:
1 import numpy
2 import pandas as pd
```

```
In [2]:
1 df = pd.read_csv("Son-Veri-Seti.csv",header=0,infer_datetime_format=True)
2 df.shape
```

```
Out[2]: (1442, 10)
```

- Öncelikle gerekli kütüphaneleri import ediyoruz ve as ile kısaltıyoruz. İmport ettiğimiz kütüphaneler sırasıyla; numpy, pandas ileriki kısımlarda 7 modeli anlatırken tekrardan kütüphaneleri tanımlayıp, anlatımı yapılacaktır.
- Daha sonra veri setimizi read_csv metodu ile çağırdık ve df'ye eşitledik.
- Shape metodu kullanarak veri setimizde kaç satır ve kaç sütun olduğunu ekrana yansıttık.

```
In [3]: 1 df.head()
```

Out[3]:

	Unnamed: 0	Global_active_power	Global_reactive_power	Voltage	Global_intensity	Sub_metering_1	Sub_metering_2	Sub_metering_3	Sub_metering_4	Result
0	0	1209	34	93552	5180	0	546	4926	14680	0
1	1	3390	226	345725	14398	2033	4187	13341	36946	1
2	2	2203	161	347373	9247	1063	2621	14018	19028	0
3	3	1666	150	348479	7094	839	7602	6197	13131	1
4	4	2225	160	348923	9312	0	2648	14063	20384	0

- Head() metodu ile veri setindeki ilk 5 satırı ekrana yansıttık.

```
In [4]: 1 df = df.iloc[:,1:10]
        2 df.head()
```

Out[4]:

	Global_active_power	Global_reactive_power	Voltage	Global_intensity	Sub_metering_1	Sub_metering_2	Sub_metering_3	Sub_metering_4	Result
0	1209	34	93552	5180	0	546	4926	14680	0
1	3390	226	345725	14398	2033	4187	13341	36946	1
2	2203	161	347373	9247	1063	2621	14018	19028	0
3	1666	150	348479	7094	839	7602	6197	13131	1
4	2225	160	348923	9312	0	2648	14063	20384	0

- Daha sonra 0. Sütunu ele almamak için iloc ile ayırma işlemi yaptık ve 1 ile 10 arasındaki sütunları seçtik.
- Daha sonra df.head() ile yeni oluşan veri setini ekrana yansıttık.

```
In [5]: 1 df.tail()
```

Out[5]:

	Global_active_power	Global_reactive_power	Voltage	Global_intensity	Sub_metering_1	Sub_metering_2	Sub_metering_3	Sub_metering_4	Result
1437	2041	142	345883	8660	4855	2110	10136	16924	0
1438	1577	137	346428	6731	1871	458	7611	16352	0
1439	1796	132	345644	7559	1096	2848	12224	13769	0
1440	1431	116	347812	6004	1076	426	5072	17278	0
1441	1488	120	303487	6259	1080	385	9989	13347	0

- Tail metodu ile veri setimizdeki son 5 veriyi ekrana yansıttık.

```
In [6]:
1 dataset=numpy.array(df)
2 dataset
```

```
Out[6]: array([[  1209,     34,  93552, ...,   4926,  14680,      0],
               [  3390,    226, 345725, ...,  13341,  36946,      1],
               [  2203,    161, 347373, ...,  14018,  19028,      0],
               ...,
               [  1796,    132, 345644, ...,  12224,  13769,      0],
               [  1431,    116, 347812, ...,   5072,  17278,      0],
               [  1488,    120, 303487, ...,   9989,  13347,      0]], dtype=int64)
```

- Df olan veri setimizi numpy.array metodu ile diziye çevirdik ve bunu dataset'e eşitledik.
- Dataset yazarak oluşan veri seti dizimizi ekrana yansıtmış olduk.

3.2 Modelleri Oluşturma

1- Parti Boyutunu ve Dönem Sayısı Nasıl Ayarlanır

```
In [7]:
1 # Epochs ve batch size'ı ızgarada aramak için scikit-learn'i kullanalım.
2 from sklearn.model_selection import GridSearchCV
3 from keras.models import Sequential
4 from keras.layers import Dense
5 from keras.wrappers.scikit_learn import KerasClassifier
```

- 1. olarak parti boyutunu ve dönem sayısını ayarlamayı yapacağız.
- Öncelikle gerekli kütüphanelerimizi import etmemiz gerekiyor.
- Bizim kullanacağımız kütüphaneler keras kütüphanesi içerisinde Sequential, Dense ve KerasClassifier.
- Sklearn kütüphanesi içerisinde ise GridSearchCv'yi kullanacağız.

In [8]:

```python
# KerasClassifier için gerekli olan modeli oluşturma
def create_model():
    # Model oluşturma
    model = Sequential()
    model.add(Dense(12, input_dim=8, activation='relu'))
    model.add(Dense(1, activation='sigmoid'))
    # Modeli derleme
    model.compile(loss='binary_crossentropy', optimizer='adam', metrics=['accuracy'])
    return model
# Tekrarlanabilirlik için rastgele tohumu ayarladık.
seed = 7
numpy.random.seed(seed)
# Giriş (X) ve çıkış (Y) değişkenlerine böldük
X = dataset[:,0:8]
Y = dataset[:,8:]
# model oluşturma
model = KerasClassifier(build_fn=create_model, verbose=0)
# Izgara arama parametrelerini tanımladık.
batch_size = [10, 20, 40, 60, 80, 100]
epochs = [10, 50, 100]
param_grid = dict(batch_size=batch_size, epochs=epochs)
grid = GridSearchCV(estimator=model, param_grid=param_grid, n_jobs=-1, cv=3)
grid_result = grid.fit(X, Y)
# Sonuçları özetleyelim.
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))
```

- Öncelikle def ile create_model tanımlayıp modelimizi oluşturmaya başlıyoruz.
- Daha sonra 3 katmanlı model oluşturuyoruz.
- İnput_dim ile giriş sayısını 8 yapıyoruz yani 8 girişli model olacak.
- Modeli derlemek için de model.compile metodunu kullanıyoruz.
- Compile'nin içerisinde loss değerini ikili çapraz entropli(binary_crossentropy)'i kullanıyoruz.
- Optimizer'değerini 'adam' yapıyoruz ve metrics değerinide doğruluk olan 'accuracy' yapıyoruz.
- Tekrarlanabilirlik için rastgele tohum kullanacağız bunun için seed değerini 7 yapıyoruz.
- Numpy.random.seed ile Numpy'nin rastgele işlemler için sözde rastgele sayılar üretmesini sağlayan temel bir girdi sağladık.
- Giriş ve Çıkış değerleri için veri setinde ayırma işlemi yaptık.
- X veri seti için dataset içerisinde 0-8 arası sütunları seçtik ve bunu X'e eşitledik.
- Y veri seti için dataset içerisinde 8 den sonraki kısım yani Result kısmını seçtik ve bunu Y'ye eşitledik.

- Daha sonra Keras sınıflandırıcısı(KerasClassifier) oluşturduk ve bunu ‘model’a eşitledik.
- KerasClassifier içerisinde verbose değerini 0 yaptık bunu istersek değiştirebiliriz.
- Build_fn yani inşa etme fonksiyonunuda create_modele eşitledik.
- Daha sonra ızgara arama parametrelerini tanımlama işlemine geçtik.
- Öncelikle batch_size değerleri için bir dizi oluşturduk.
- Model her birini sırayla deneyecektir ve bize en iyi sonucu verecektir.
- Batch_size değerlerimiz sırasıyla “10,20,40,60,80,100”dür.
- Diğer parametremiz epochs.
- Bunun için de yine dizi oluşturduk ve değerlerimiz sırasıyla “10,50,100”dür.
- Daha sonra dict() metodu ile boş bir sözlük oluşturduk ve dict içerisine batch_size ve epoch tanımlaması yaptık ve bunu da param_grid’e eşitledik.
- GridSearchCV’yi grid’e eşitliyoruz ve GridSearchCV içerisinde gerekli tanımlamaları yapıyoruz.
- GridSearchCV İşlemi yapı ve parametrelerin her kombinasyon için bir model değerlendirecektir. Çapraz doğrulama her bir modeli değerlendirmek için kullanılır ve varsayılan olarak 3-katlı çapraz doğrulama kullanıldık(cv=3).
- grid.fit() metodu ile döndürülen sonuç nesnesindeki ızgara aramasının sonucuna erişebiliriz ve grid.fit()’i grid_result’a eşitliyoruz.
- Şimdi sonuçları özetleyelim.
- Print içerisinde Best:’e karşılık gelen %f’in değeri grid_result’un en yüksek skoruna eşitledik. Using e karşılık gelen %s değeride grid_resultun en yüksek skorunun parametrelerini yazdırdık.
- grid_result.cv_results_[means_test_score] sonucu bizim her defasında doğruluk değerlerini hesaplayacaktır ve bunu means’a eşitliyoruz.
- grid_result.cv_results_['std_test_score'] ise yapılan işlemlerdeki standart sapmayı hesaplıyor ve bunuda stds’ye eşitliyoruz. - grid_result.cv_results_['params'] ise kullanılan ızgara arama parametrelerini adım adım hesaplıyor ve bunu da params’a eşitliyoruz.
- For döngüsü ile means’ı stds’yi ve params’ı kullanıyoruz. Burdaki amaç döngü oluşturarak print ile her adımı ve doğruluğu ekrana yansıtmak.

- Bu parametreleri zip olarak yazıp means, stdev ve param'a denkleştiriyoruz.
- Print ile hepsini ekrana yansıtıyoruz.

Çıktısı;

```
Best: 0.655348 using {'batch_size': 40, 'epochs': 100}
0.433356 (0.140475) with: {'batch_size': 10, 'epochs': 10}
0.486024 (0.137862) with: {'batch_size': 10, 'epochs': 50}
0.419499 (0.061743) with: {'batch_size': 10, 'epochs': 100}
0.526298 (0.088970) with: {'batch_size': 20, 'epochs': 10}
0.570109 (0.135575) with: {'batch_size': 20, 'epochs': 50}
0.511139 (0.126600) with: {'batch_size': 20, 'epochs': 100}
0.554802 (0.017965) with: {'batch_size': 40, 'epochs': 10}
0.654655 (0.016017) with: {'batch_size': 40, 'epochs': 50}
0.655348 (0.015038) with: {'batch_size': 40, 'epochs': 100}
0.589435 (0.032200) with: {'batch_size': 60, 'epochs': 10}
0.532456 (0.143396) with: {'batch_size': 60, 'epochs': 50}
0.413951 (0.084582) with: {'batch_size': 60, 'epochs': 100}
0.467313 (0.104514) with: {'batch_size': 80, 'epochs': 10}
0.445830 (0.132603) with: {'batch_size': 80, 'epochs': 50}
0.616540 (0.069913) with: {'batch_size': 80, 'epochs': 100}
0.533287 (0.046681) with: {'batch_size': 100, 'epochs': 10}
0.540288 (0.135069) with: {'batch_size': 100, 'epochs': 50}
0.592285 (0.104214) with: {'batch_size': 100, 'epochs': 100}
```

- En iyi doğruluk değeri 0.655348 ile batch_size=40 ve epochs:100 'dür.

2- Eğitim Optimizasyon Algoritmasını Ayarlama

In [9]:
```
import numpy
from sklearn.model_selection import GridSearchCV
from keras.models import Sequential
from keras.layers import Dense
from keras.wrappers.scikit_learn import KerasClassifier
```

- 2. olarak Eğitim Optimizasyon Algorimasını ayarlamayı yapacağız.
- Öncelikle gerekli kütüphanelerimizi import etmemiz gerekiyor.

- Bizim kullanacağımız kütüphaneler keras kütüphanesi içerisinde Sequential, Dense ve KerasClassifier.
- Sklearn kütüphanesi içerisinde ise GridSearchCv'yi kullanacağız.

In [10]:

```
# KerasClassifier için gerekli olan modeli oluşturma
def create_model(optimizer='adam'):
    # Model oluşturma
    model = Sequential()
    model.add(Dense(12, input_dim=8, activation='relu'))
    model.add(Dense(1, activation='sigmoid'))
    # Modeli derleme
    model.compile(loss='binary_crossentropy', optimizer=optimizer, metrics=['accuracy'])
    return model
# Tekrarlanabilirlik için rastgele tohumu ayarladık.
seed = 7
numpy.random.seed(seed)
# Giriş (X) ve çıkış (Y) değişkenlerine böldük
X = dataset[:,0:8]
Y = dataset[:,8:]
# Model oluşturma
model = KerasClassifier(build_fn=create_model, epochs=100, batch_size=10, verbose=0)
# Izgara arama parametrelerini tanımladık.
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=3)
grid_result = grid.fit(X, Y)
# Sonuçları özetleyelim.
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))
```

- Öncelikle def ile create_model tanımlayıp modelimizi oluşturmaya başlıyoruz. Ve optimizer değerini çaba kaybını azaltmak için yazıyoruz ve değerimiz 'adam' olarak ayarlıyoruz.
- Daha sonra 3 katmanlı model oluşturuyoruz.
- İnput_dim ile giriş sayısını 8 yapıyoruz yani 8 girişli model olacak.
- Modeli derlemek için de model.compile metodunu kullanıyoruz.
- Compile'nin içerisinde loss değerini ikili çapraz entropli(binary_crossentropy)'i kullanıyoruz.

- Optimizer'değerini optimizer'e eşitliyoruz ve metrics değerinide doğruluk olan 'accuracy' yapıyoruz.
- Tekrarlanabilirlik için rastgele tohum kullanacağız bunun için seed değerini 7 yapıyoruz.
- Numpy.random.seed ile Numpy'nin rastgele işlemler için sözde rastgele sayılar üretmesini sağlayan temel bir girdi sağladık.
- Giriş ve Çıkış değerleri için veri setinde ayırma işlemi yaptık.
- X veri seti için dataset içerisinde 0-8 arası sütunları seçtik ve bunu X'e eşitledik.
- Y veri seti için dataset içerisinde 8 den sonraki kısım yani Result kısmını seçtik ve bunu Y'ye eşitledik.
- Daha sonra Keras sınıflandırıcısı(KerasClassifier) oluşturduk ve bunu 'model'a eşitledik.
- KerasClassifier içerisinde verbose değerini 0 yaptık bunu istersek değiştirebiliriz.
- Build_fn yani inşa etme fonksiyonunuda create_modele eşitledik.
- Epochs değerini 100 ve batch_size değerini de 10 yaptık.
- Daha sonra ızgara arama parametrelerini tanımlama işlemine geçtik.
- Burda optimizer dizisi oluşturacağız ve dizi elemanlarımız sırasıyla "SGD, RMSprop, Adagrad, Adadelta, Adam, Adamax, Nadam" olarak ayarlandı.
- Daha sonra dict() metodu ile boş bir sözlük oluşturduk ve dict içerisine optimizer tanımlaması yaptık ve bunu da param_grid'e eşitledik.
- GridSearchCV'yi grid'e eşitliyoruz ve GridSearchCV içerisinde gerekli tanımlamaları yapıyoruz.
- GridSearchCV İşlemi yapı ve parametrelerin her kombinasyon için bir model değerlendirecektir. Çapraz doğrulama her bir modeli değerlendirmek için kullanılır ve varsayılan olarak 3-katlı çapraz doğrulama kullanıldık(cv=3).
- grid.fit() metodu ile döndürülen sonuç nesnesindeki ızgara aramasının sonucuna erişebiliriz ve grid.fit()'i grid_result'a eşitliyoruz.
- Şimdi sonuçları özetleyelim.
- Print içerisinde Best:'e karşılık gelen %f'in değeri grid_result'un en yüksek skoruna eşitledik. Using e karşılık gelen %s değeride grid_resultun en yüksek skorunun parametrelerini yazdırdık.

- grid_result.cv_results_[means_test_score] sonucu bizim her defasında doğruluk değerlerini hesaplayacaktır ve bunu means'a eşitliyoruz.
- grid_result.cv_results_['std_test_score'] ise yapılan işlemlerdeki standart sapmayı hesaplıyor ve bunuda stds'ye eşitliyoruz. - grid_result.cv_results_['params'] ise kullanılan ızgara arama parametrelerini adım adım hesaplıyor ve bunu da params'a eşitliyoruz.
- For döngüsü ile means'ı stds'yi ve params'ı kullanıyoruz. Burdaki amaç döngü oluşturarak print ile her adımı ve doğruluğu ekrana yansıtmak.
- Bu parametreleri zip olarak yazıp means, stdev ve param'a denkleştiriyoruz.
- Print ile hepsini ekrana yansıtıyoruz.

Çıktısı;

```
Best: 0.654655 using {'optimizer': 'SGD'}
0.654655 (0.016017) with: {'optimizer': 'SGD'}
0.654655 (0.016017) with: {'optimizer': 'RMSprop'}
0.529120 (0.043681) with: {'optimizer': 'Adagrad'}
0.443112 (0.078186) with: {'optimizer': 'Adadelta'}
0.439605 (0.136162) with: {'optimizer': 'Adam'}
0.559703 (0.137086) with: {'optimizer': 'Adamax'}
0.426426 (0.130675) with: {'optimizer': 'Nadam'}
```

- En iyi doğruluk değeri 0.654655 ile Optimizer:SGD'dir.

3- Öğrenme Hızını ve Momentum Ayarlama

In [11]:

```
# Öğrenme hızını ve momentumu ızgarada aramak için scikit-learn'i kullanalım
import numpy
from sklearn.model_selection import GridSearchCV
from keras.models import Sequential
from keras.layers import Dense
from keras.wrappers.scikit_learn import KerasClassifier
from keras.optimizers import SGD
```

- 3. olarak Öğrenme Hızını ve Momentum ayarlamayı yapacağız.

- Öncelikle gerekli kütüphanelerimizi import etmemiz gerekiyor.
- Bizim kullanacağımız kütüphaneler keras kütüphanesi içerisinde Sequential, Dense, KerasClassifier ve SGD.
- Numpy kütüphanesi.
- Sklearn kütüphanesi içerisinde ise GridSearchCv'yi kullanacağız.

In [12]:

```
# KerasClassifier için gerekli olan modeli oluşturma
def create_model(learn_rate=0.01, momentum=0):
    # Model oluşturma
    model = Sequential()
    model.add(Dense(12, input_dim=8, activation='relu'))
    model.add(Dense(1, activation='sigmoid'))
    # Modeli derleme
    optimizer = SGD(lr=learn_rate, momentum=momentum)
    model.compile(loss='binary_crossentropy', optimizer=optimizer, metrics=['accuracy'])
    return model
# Tekrarlanabilirlik için rastgele tohumu ayarladık.
seed = 7
numpy.random.seed(seed)
# Giriş (X) ve çıkış (Y) değişkenlerine böldük
X = dataset[:,0:8]
Y = dataset[:,8:]
# Model oluşturma
model = KerasClassifier(build_fn=create_model, epochs=100, batch_size=10, verbose=0)
# Izgara arama parametrelerini tanımladık.
learn_rate = [0.001, 0.01, 0.1, 0.2, 0.3]
momentum = [0.0, 0.2, 0.4, 0.6, 0.8, 0.9]
param_grid = dict(learn_rate=learn_rate, momentum=momentum)
grid = GridSearchCV(estimator=model, param_grid=param_grid, n_jobs=-1, cv=3)
grid_result = grid.fit(X, Y)
# Sonuçları özetleyelim.
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))
```

- Öncelikle def ile create_model tanımlayıp modelimizi oluşturmaya başlıyoruz. Ve Öğrenme oranını 0.01 olarak ayarlayıp momentum değerinide 0 yapıyoruz.
- Daha sonra 3 katmanlı model oluşturuyoruz.

- İnput_dim ile giriş sayısını 8 yapıyoruz yani 8 girişli model olacak.
- SGD içerisine learn_rate yani öğrenme oranını tanımlayıp lr'ye eşitliyoruz. Ve momentumuda tanımlıyoruz.
- SGD'yi optimize etmek için kullanıyoruz.
- Oluşturduğumuz SGD'yi de optimizer'e eşitliyoruz.
- Modeli derlemek için de model.compile metodunu kullanıyoruz.
- Compile'nin içerisinde loss değerini ikili çapraz entropli(binary_crossentropy)'i kullanıyoruz.
- Optimizer'değerini optimizer'e eşitliyoruz ve metrics değerinide doğruluk olan 'accuracy' yapıyoruz.
- Tekrarlanabilirlik için rastgele tohum kullanacağız bunun için seed değerini 7 yapıyoruz.
- Numpy.random.seed ile Numpy'nin rastgele işlemler için sözde rastgele sayılar üretmesini sağlayan temel bir girdi sağladık.
- Giriş ve Çıkış değerleri için veri setinde ayırma işlemi yaptık.
- X veri seti için dataset içerisinde 0-8 arası sütunları seçtik ve bunu X'e eşitledik.
- Y veri seti için dataset içerisinde 8 den sonraki kısım yani Result kısmını seçtik ve bunu Y'ye eşitledik.
- Daha sonra Keras sınıflandırıcısı(KerasClassifier) oluşturduk ve bunu 'model'a eşitledik.
- KerasClassifier içerisinde verbose değerini 0 yaptık bunu istersek değiştirebiliriz.
- Build_fn yani inşa etme fonksiyonunuda create_modele eşitledik.
- Epochs değerini 100 ve batch_size değerini de 10 yaptık.
- Daha sonra ızgara arama parametrelerini tanımlama işlemine geçtik.
- Öncelikle learn_rate dizisi oluşturduk ve içerisine değerlerimizi yazdık. Bu değerler "0.001, 0.01, 0.1, 0.2, 0.3" şeklindedir.
- Daha sonra momentum dizisi oluşturduk ve içerisine değerlerimizi yazdık. Bu değerler "0.0, 0.2, 0.4, 0.6, 0.8, 0.9" şeklindedir.
- Daha sonra dict() metodu ile boş bir sözlük oluşturduk ve dict içerisine learn_rate ve momentum tanımlaması yaptık ve bunu da param_grid'e eşitledik.

- GridSearchCV'yi grid'e eşitliyoruz ve GridSearchCV içerisinde gerekli tanımlamaları yapıyoruz.
- GridSearchCV İşlemi yapı ve parametrelerin her kombinasyon için bir model değerlendirecektir. Çapraz doğrulama her bir modeli değerlendirmek için kullanılır ve varsayılan olarak 3-katlı çapraz doğrulama kullanıldık(cv=3).
- grid.fit() metodu ile döndürülen sonuç nesnesindeki ızgara aramasının sonucuna erişebiliriz ve grid.fit()'i grid_result'a eşitliyoruz.
- Şimdi sonuçları özetleyelim.
- Print içerisinde Best:'e karşılık gelen %f'in değeri grid_result'un en yüksek skoruna eşitledik. Using e karşılık gelen %s değeride grid_resultun en yüksek skorunun parametrelerini yazdırdık.
- grid_result.cv_results_[means_test_score] sonucu bizim her defasında doğruluk değerlerini hesaplayacaktır ve bunu means'a eşitliyoruz.
- grid_result.cv_results_['std_test_score'] ise yapılan işlemlerdeki standart sapmayı hesaplıyor ve bunuda stds'ye eşitliyoruz. - grid_result.cv_results_['params'] ise kullanılan ızgara arama parametrelerini adım adım hesaplıyor ve bunu da params'a eşitliyoruz.
- For döngüsü ile means'ı stds'yi ve params'ı kullanıyoruz. Burdaki amaç döngü oluşturarak print ile her adımı ve doğruluğu ekrana yansıtmak.
- Bu parametreleri zip olarak yazıp means, stdev ve param'a denkleştiriyoruz.
- Print ile hepsini ekrana yansıtıyoruz.

Çıktısı;

```
Best: 0.656041 using {'learn_rate': 0.2, 'momentum': 0.9}
0.654655 (0.016017) with: {'learn_rate': 0.001, 'momentum': 0.0}
0.654655 (0.016017) with: {'learn_rate': 0.001, 'momentum': 0.2}
0.654655 (0.016017) with: {'learn_rate': 0.001, 'momentum': 0.4}
0.654655 (0.016017) with: {'learn_rate': 0.001, 'momentum': 0.6}
0.654655 (0.016017) with: {'learn_rate': 0.001, 'momentum': 0.8}
0.654655 (0.016017) with: {'learn_rate': 0.001, 'momentum': 0.9}
0.653269 (0.017977) with: {'learn_rate': 0.01, 'momentum': 0.0}
0.654655 (0.016017) with: {'learn_rate': 0.01, 'momentum': 0.2}
0.654655 (0.016017) with: {'learn_rate': 0.01, 'momentum': 0.4}
0.654655 (0.016017) with: {'learn_rate': 0.01, 'momentum': 0.6}
0.654655 (0.016017) with: {'learn_rate': 0.01, 'momentum': 0.8}
0.654655 (0.016017) with: {'learn_rate': 0.01, 'momentum': 0.9}
0.654655 (0.016017) with: {'learn_rate': 0.1, 'momentum': 0.0}
0.654655 (0.016017) with: {'learn_rate': 0.1, 'momentum': 0.2}
0.654655 (0.016017) with: {'learn_rate': 0.1, 'momentum': 0.4}
0.654655 (0.016017) with: {'learn_rate': 0.1, 'momentum': 0.6}
0.654655 (0.016017) with: {'learn_rate': 0.1, 'momentum': 0.8}
0.654655 (0.016017) with: {'learn_rate': 0.1, 'momentum': 0.9}
0.654655 (0.016017) with: {'learn_rate': 0.2, 'momentum': 0.0}
0.654655 (0.016017) with: {'learn_rate': 0.2, 'momentum': 0.2}
0.654655 (0.016017) with: {'learn_rate': 0.2, 'momentum': 0.4}
0.654655 (0.016017) with: {'learn_rate': 0.2, 'momentum': 0.6}
0.654655 (0.016017) with: {'learn_rate': 0.2, 'momentum': 0.8}
0.656041 (0.014059) with: {'learn_rate': 0.2, 'momentum': 0.9}
0.654655 (0.016017) with: {'learn_rate': 0.3, 'momentum': 0.0}
0.654655 (0.016017) with: {'learn_rate': 0.3, 'momentum': 0.2}
0.654655 (0.016017) with: {'learn_rate': 0.3, 'momentum': 0.4}
0.654655 (0.016017) with: {'learn_rate': 0.3, 'momentum': 0.6}
0.654655 (0.016017) with: {'learn_rate': 0.3, 'momentum': 0.8}
0.654655 (0.016017) with: {'learn_rate': 0.3, 'momentum': 0.9}
```

- En iyi doğruluk değeri 0.656041 ile learn_rate=0.2 ve momentum=0.9'dur.

4- Ağ Ağırlığı Başlatma Nasıl Ayarlanır

In [13]:
```python
# Ağırlık başlatmayı ızgarada aramak için scikit-learn'i kullanalım.
import numpy
from sklearn.model_selection import GridSearchCV
from keras.models import Sequential
from keras.layers import Dense
from keras.wrappers.scikit_learn import KerasClassifier
```

- 4. olarak Ağ Ağırlığı Başlatma ayarlamayı yapacağız.
- Öncelikle gerekli kütüphanelerimizi import etmemiz gerekiyor.
- Bizim kullanacağımız kütüphaneler keras kütüphanesi içerisinde Sequential, Dense, KerasClassifier.
- Numpy kütüphanesi.
- Sklearn kütüphanesi içerisinde ise GridSearchCv'yi kullanacağız.

In [14]:
```python
# KerasClassifier için gerekli olan modeli oluşturma
def create_model(init_mode='uniform'):
    # Model oluşturma
    model = Sequential()
    model.add(Dense(12, input_dim=8, kernel_initializer=init_mode, activation='relu'))
    model.add(Dense(1, kernel_initializer=init_mode, activation='sigmoid'))
    # Modeli derleme
    model.compile(loss='binary_crossentropy', optimizer='adam', metrics=['accuracy'])
    return model
# Tekrarlanabilirlik için rastgele tohumu ayarladık.
seed = 7
numpy.random.seed(seed)
# Giriş (X) ve çıkış (Y) değişkenlerine böldük
X = dataset[:,0:8]
Y = dataset[:,8]
# Model oluşturma
model = KerasClassifier(build_fn=create_model, epochs=100, batch_size=10, verbose=0)
# Izgara arama parametrelerini tanımladık.
init_mode = ['uniform', 'lecun_uniform', 'normal', 'zero', 'glorot_normal', 'glorot_uniform', 'he_normal', 'he_uniform']
param_grid = dict(init_mode=init_mode)
grid = GridSearchCV(estimator=model, param_grid=param_grid, n_jobs=-1, cv=3)
grid_result = grid.fit(X, Y)
# Sonuçları özetleyelim.
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))
```

- Öncelikle def ile create_model tanımlayıp modelimizi oluşturmaya başlıyoruz. İnit_mode'u yani başlatma modunu uniform yapıyoruz.
- Daha sonra 3 katmanlı model oluşturuyoruz.
- İnput_dim ile giriş sayısını 8 yapıyoruz yani 8 girişli model olacak.

- Modeli derlemek için de model.compile metodunu kullanıyoruz.
- Compile'nin içerisinde loss değerini ikili çapraz entropli(binary_crossentropy)'i kullanıyoruz.
- Optimizer'değerini 'adam' yapıyoruz ve metrics değerinide doğruluk olan 'accuracy' yapıyoruz.
- Tekrarlanabilirlik için rastgele tohum kullanacağız bunun için seed değerini 7 yapıyoruz.
- Numpy.random.seed ile Numpy'nin rastgele işlemler için sözde rastgele sayılar üretmesini sağlayan temel bir girdi sağladık.
- Giriş ve Çıkış değerleri için veri setinde ayırma işlemi yaptık.
- X veri seti için dataset içerisinde 0-8 arası sütunları seçtik ve bunu X'e eşitledik.
- Y veri seti için dataset içerisinde 8 den sonraki kısım yani Result kısmını seçtik ve bunu Y'ye eşitledik.
- Daha sonra Keras sınıflandırıcısı(KerasClassifier) oluşturduk ve bunu 'model'a eşitledik.
- KerasClassifier içerisinde verbose değerini 0 yaptık bunu istersek değiştirebiliriz.
- Build_fn yani inşa etme fonksiyonunuda create_modele eşitledik.
- Epochs değerini 100 ve batch_size değerini de 10 yaptık.
- Daha sonra ızgara arama parametrelerini tanımlama işlemine geçtik.
- Öncelikle init_mode dizisi oluşturduk ve içerisine değerlerimizi yazdık. Bu değerler "uniform, lecun_uniform, normal, zero, glorot_normal, glorot_uniform, he_normal, he_uniform" şeklindedir.
- Daha sonra dict() metodu ile boş bir sözlük oluşturduk ve dict içerisine init_mode tanımlaması yaptık ve bunu da param_grid'e eşitledik.
- GridSearchCV'yi grid'e eşitliyoruz ve GridSearchCV içerisinde gerekli tanımlamaları yapıyoruz.
- GridSearchCV İşlemi yapı ve parametrelerin her kombinasyon için bir model değerlendirecektir. Çapraz doğrulama her bir modeli değerlendirmek için kullanılır ve varsayılan olarak 3-katlı çapraz doğrulama kullanıldık(cv=3).
- grid.fit() metodu ile döndürülen sonuç nesnesindeki ızgara aramasının sonucuna erişebiliriz ve grid.fit()'i grid_result'a eşitliyoruz.

- Şimdi sonuçları özetleyelim.
- Print içerisinde Best:'e karşılık gelen %f'in değeri grid_result'un en yüksek skoruna eşitledik. Using e karşılık gelen %s değeride grid_resultun en yüksek skorunun parametrelerini yazdırdık.
- grid_result.cv_results_[means_test_score] sonucu bizim her defasında doğruluk değerlerini hesaplayacaktır ve bunu means'a eşitliyoruz.
- grid_result.cv_results_['std_test_score'] ise yapılan işlemlerdeki standart sapmayı hesaplıyor ve bunuda stds'ye eşitliyoruz. - grid_result.cv_results_['params'] ise kullanılan ızgara arama parametrelerini adım adım hesaplıyor ve bunu da params'a eşitliyoruz.
- For döngüsü ile means'ı stds'yi ve params'ı kullanıyoruz. Burdaki amaç döngü oluşturarak print ile her adımı ve doğruluğu ekrana yansıtmak.
- Bu parametreleri zip olarak yazıp means, stdev ve param'a denkleştiriyoruz.
- Print ile hepsini ekrana yansıtıyoruz. Çıktısı;

```
Best: 0.654655 using {'init_mode': 'uniform'}
0.654655 (0.016017) with: {'init_mode': 'uniform'}
0.436128 (0.138571) with: {'init_mode': 'lecun_uniform'}
0.554655 (0.133619) with: {'init_mode': 'normal'}
0.654655 (0.016017) with: {'init_mode': 'zero'}
0.544237 (0.149674) with: {'init_mode': 'glorot_normal'}
0.440979 (0.128514) with: {'init_mode': 'glorot_uniform'}
0.651884 (0.021446) with: {'init_mode': 'he_normal'}
0.543544 (0.149260) with: {'init_mode': 'he_uniform'}
```

- En iyi doğruluk değeri 0.654655 ile init_mode=uniform'dur.

5- Nöron Aktivasyon Fonksiyonu Nasıl Ayarlanır

In [15]:
```
# Aktivasyon işlevini ızgarada aramak için scikit-learn'i kullanın
import numpy
from sklearn.model_selection import GridSearchCV
from keras.models import Sequential
from keras.layers import Dense
from keras.wrappers.scikit_learn import KerasClassifier
```

- 5. olarak Nöron Aktivasyon Fonksiyonu ayarlamayı yapacağız.
- Öncelikle gerekli kütüphanelerimizi import etmemiz gerekiyor.

- Bizim kullanacağımız kütüphaneler keras kütüphanesi içerisinde Sequential, Dense, KerasClassifier.
- Numpy kütüphanesi.
- Sklearn kütüphanesi içerisinde ise GridSearchCv'yi kullanacağız.

In [16]:

```
# KerasClassifier için gerekli olan modeli oluşturma
def create_model(activation='relu'):
    # Model oluşturma
    model = Sequential()
    model.add(Dense(12, input_dim=8, kernel_initializer='uniform', activation=activation))
    model.add(Dense(1, kernel_initializer='uniform', activation='sigmoid'))
    # Modeli derleme
    model.compile(loss='binary_crossentropy', optimizer='adam', metrics=['accuracy'])
    return model
# Tekrarlanabilirlik için rastgele tohumu ayarladık.
seed = 7
numpy.random.seed(seed)
# Giriş (X) ve çıkış (Y) değişkenlerine böldük
X = dataset[:,0:8]
Y = dataset[:,8]
# Model oluşturma
model = KerasClassifier(build_fn=create_model, epochs=100, batch_size=10, verbose=0)
# Izgara arama parametrelerini tanımladık.
activation = ['softmax', 'softplus', 'softsign', 'relu', 'tanh', 'sigmoid', 'hard_sigmoid', 'linear']
param_grid = dict(activation=activation)
grid = GridSearchCV(estimator=model, param_grid=param_grid, n_jobs=-1, cv=3)
grid_result = grid.fit(X, Y)
# Sonuçları özetleyelim.
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))
```

- Öncelikle def ile create_model tanımlıyoruz.
- Daha sonra 3 katmanlı model oluşturuyoruz.
- İnput_dim ile giriş sayısını 8 yapıyoruz yani 8 girişli model olacak.
- Modeli derlemek için de model.compile metodunu kullanıyoruz.
- Compile'nin içerisinde loss değerini ikili çapraz entropli(binary_crossentropy)'i kullanıyoruz.
- Optimizer'değerini 'adam' yapıyoruz ve metrics değerinide doğruluk olan 'accuracy' yapıyoruz.

- Tekrarlanabilirlik için rastgele tohum kullanacağız bunun için seed değerini 7 yapıyoruz.
- Numpy.random.seed ile Numpy'nin rastgele işlemler için sözde rastgele sayılar üretmesini sağlayan temel bir girdi sağladık.
- Giriş ve Çıkış değerleri için veri setinde ayırma işlemi yaptık.
- X veri seti için dataset içerisinde 0-8 arası sütunları seçtik ve bunu X'e eşitledik.
- Y veri seti için dataset içerisinde 8 den sonraki kısım yani Result kısmını seçtik ve bunu Y'ye eşitledik.
- Daha sonra Keras sınıflandırıcısı(KerasClassifier) oluşturduk ve bunu 'model'a eşitledik.
- KerasClassifier içerisinde verbose değerini 0 yaptık bunu istersek değiştirebiliriz.
- Build_fn yani inşa etme fonksiyonunuda create_modele eşitledik.
- Epochs değerini 100 ve batch_size değerini de 10 yaptık.
- Daha sonra ızgara arama parametrelerini tanımlama işlemine geçtik.
- Öncelikle activation dizisi oluşturduk ve içerisine değerlerimizi yazdık. Bu değerler "softmax, softplus, softsign, relu, tanh, sigmoid, hard_sigmoid, linear" şeklindedir.
- Daha sonra dict() metodu ile boş bir sözlük oluşturduk ve dict içerisine activation tanımlaması yaptık ve bunu da param_grid'e eşitledik.
- GridSearchCV'yi grid'e eşitliyoruz ve GridSearchCV içerisinde gerekli tanımlamaları yapıyoruz.
- GridSearchCV İşlemi yapı ve parametrelerin her kombinasyon için bir model değerlendirecektir. Çapraz doğrulama her bir modeli değerlendirmek için kullanılır ve varsayılan olarak 3-katlı çapraz doğrulama kullanıldık(cv=3).
- grid.fit() metodu ile döndürülen sonuç nesnesindeki ızgara aramasının sonucuna erişebiliriz ve grid.fit()'i grid_result'a eşitliyoruz.
- Şimdi sonuçları özetleyelim.
- Print içerisinde Best:'e karşılık gelen %f'in değeri grid_result'un en yüksek skoruna eşitledik. Using e karşılık gelen %s değeride grid_resultun en yüksek skorunun parametrelerini yazdırdık.
- grid_result.cv_results_[means_test_score] sonucu bizim her defasında doğruluk değerlerini hesaplayacaktır ve bunu means'a eşitliyoruz.

- grid_result.cv_results_['std_test_score'] ise yapılan işlemlerdeki standart sapmayı hesaplıyor ve bunuda stds'ye eşitliyoruz. - grid_result.cv_results_['params'] ise kullanılan ızgara arama parametrelerini adım adım hesaplıyor ve bunu da params'a eşitliyoruz.
- For döngüsü ile means'ı stds'yi ve params'ı kullanıyoruz. Burdaki amaç döngü oluşturarak print ile her adımı ve doğruluğu ekrana yansıtmak.
- Bu parametreleri zip olarak yazıp means, stdev ve param'a denkleştiriyoruz.
- Print ile hepsini ekrana yansıtıyoruz.

Çıktısı;

```
Best: 0.655348 using {'activation': 'softplus'}
0.654655 (0.016017) with: {'activation': 'softmax'}
0.655348 (0.015038) with: {'activation': 'softplus'}
0.654655 (0.016017) with: {'activation': 'softsign'}
0.654655 (0.016017) with: {'activation': 'relu'}
0.654655 (0.016017) with: {'activation': 'tanh'}
0.654655 (0.016017) with: {'activation': 'sigmoid'}
0.654655 (0.016017) with: {'activation': 'hard_sigmoid'}
0.552752 (0.128993) with: {'activation': 'linear'}
```

- En iyi doğruluk değeri 0.655348 ile activation=softplus'dır.

6- Bırakma Düzenlemesini Ayarlama

In [17]:

```
# Bırakma oranını ızgarada aramak için scikit-learn'i kullanalım.
import numpy
from sklearn.model_selection import GridSearchCV
from keras.models import Sequential
from keras.layers import Dense
from keras.layers import Dropout
from keras.wrappers.scikit_learn import KerasClassifier
from keras.constraints import maxnorm
```

- 6. olarak Bırakma Düzenlemesini ayarlamayı yapacağız.
- Öncelikle gerekli kütüphanelerimizi import etmemiz gerekiyor.
- Bizim kullanacağımız kütüphaneler keras kütüphanesi içerisinde Sequential, Dense, Dropout, maxnorm, KerasClassifier.
- Numpy kütüphanesi.
- Sklearn kütüphanesi içerisinde ise GridSearchCv'yi kullanacağız.

In [18]:

```
# KerasClassifier için gerekli olan modeli oluşturma
def create_model(dropout_rate=0.0, weight_constraint=0):
    # Model oluşturma
    model = Sequential()
    model.add(Dense(12, input_dim=8, kernel_initializer='uniform', activation='linear', kernel_constraint=maxnorm(weight_constraint)))
    model.add(Dropout(dropout_rate))
    model.add(Dense(1, kernel_initializer='uniform', activation='sigmoid'))
    # Modeli derleme
    model.compile(loss='binary_crossentropy', optimizer='adam', metrics=['accuracy'])
    return model
# Tekrarlanabilirlik için rastgele tohumu ayarladık.
seed = 7
numpy.random.seed(seed)
# Giriş (X) ve çıkış (Y) değişkenlerine böldük
X = dataset[:,0:8]
Y = dataset[:,8]
# Model oluşturma
model = KerasClassifier(build_fn=create_model, epochs=100, batch_size=10, verbose=0)
#Izgara arama parametrelerini tanımladık.
weight_constraint = [1, 2, 3, 4, 5]
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(dropout_rate=dropout_rate, weight_constraint=weight_constraint)
grid = GridSearchCV(estimator=model, param_grid=param_grid, n_jobs=-1, cv=3)
grid_result = grid.fit(X, Y)
# Sonuçları özetleyelim.
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))
```

- Öncelikle def ile create_model tanımlıyoruz. Ve dropout_rate yani bırakma oranını 0.0 yapıyoruz ve weight_constraint(kısıtlama) değerini 0 yapıyoruz.

- Daha sonra 4 katmanlı model oluşturuyoruz.
- İnput_dim ile giriş sayısını 8 yapıyoruz yani 8 girişli model olacak.
- Kernel_initializer yani çekirdek başlatıcı değerini ‘uniform’ yapıyoruz ve activation yani aktivasyon değerini ‘linear’ yapıyoruz.
- Modeli derlemek için de model.compile metodunu kullanıyoruz.
- Compile’nin içerisinde loss değerini ikili çapraz entropli(binary_crossentropy)’i kullanıyoruz.
- Optimizer’değerini ‘adam’ yapıyoruz ve metrics değerinide doğruluk olan ‘accuracy’ yapıyoruz.
- Tekrarlanabilirlik için rastgele tohum kullanacağız bunun için seed değerini 7 yapıyoruz.
- Numpy.random.seed ile Numpy'nin rastgele işlemler için sözde rastgele sayılar üretmesini sağlayan temel bir girdi sağladık.
- Giriş ve Çıkış değerleri için veri setinde ayırma işlemi yaptık.
- X veri seti için dataset içerisinde 0-8 arası sütunları seçtik ve bunu X’e eşitledik.
- Y veri seti için dataset içerisinde 8 den sonraki kısım yani Result kısmını seçtik ve bunu Y’ye eşitledik.
- Daha sonra Keras sınıflandırıcısı(KerasClassifier) oluşturduk ve bunu ‘model’a eşitledik.
- KerasClassifier içerisinde verbose değerini 0 yaptık bunu istersek değiştirebiliriz.
- Build_fn yani inşa etme fonksiyonunuda create_modele eşitledik.
- Epochs değerini 100 ve batch_size değerini de 10 yaptık.
- Daha sonra ızgara arama parametrelerini tanımlama işlemine geçtik.
- Öncelikle weight_constraint dizisi oluşturduk ve içerisine değerlerimizi yazdık. Bu değerler “1, 2, 3, 4, 5” şeklindedir.
- Daha sonra dropout_rate dizisi oluşturduk ve içerisine değerlerimizi yazdık. Bu değerler “0.0, 0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9” şeklindedir.
- Daha sonra dict() metodu ile boş bir sözlük oluşturduk ve dict içerisine weight_constraint ve dropout_rate tanımlaması yaptık ve bunu da param_grid’e eşitledik.

- GridSearchCV'yi grid'e eşitliyoruz ve GridSearchCV içerisinde gerekli tanımlamaları yapıyoruz.
- GridSearchCV İşlemi yapı ve parametrelerin her kombinasyon için bir model değerlendirecektir. Çapraz doğrulama her bir modeli değerlendirmek için kullanılır ve varsayılan olarak 3-katlı çapraz doğrulama kullanıldık(cv=3).
- grid.fit() metodu ile döndürülen sonuç nesnesindeki ızgara aramasının sonucuna erişebiliriz ve grid.fit()'i grid_result'a eşitliyoruz.
- Şimdi sonuçları özetleyelim.
- Print içerisinde Best:'e karşılık gelen %f'in değeri grid_result'un en yüksek skoruna eşitledik. Using e karşılık gelen %s değeride grid_resultun en yüksek skorunun parametrelerini yazdırdık.
- grid_result.cv_results_[means_test_score] sonucu bizim her defasında doğruluk değerlerini hesaplayacaktır ve bunu means'a eşitliyoruz.
- grid_result.cv_results_['std_test_score'] ise yapılan işlemlerdeki standart sapmayı hesaplıyor ve bunuda stds'ye eşitliyoruz. - grid_result.cv_results_['params'] ise kullanılan ızgara arama parametrelerini adım adım hesaplıyor ve bunu da params'a eşitliyoruz.
- For döngüsü ile means'ı stds'yi ve params'ı kullanıyoruz. Burdaki amaç döngü oluşturarak print ile her adımı ve doğruluğu ekrana yansıtmak.
- Bu parametreleri zip olarak yazıp means, stdev ve param'a denkleştiriyoruz.
- Print ile hepsini ekrana yansıtıyoruz.

Çıktısı;

```
Best: 0.656734 using {'dropout_rate': 0.1, 'weight_constraint': 5}
0.655348 (0.016500) with: {'dropout_rate': 0.0, 'weight_constraint': 1}
0.654655 (0.016017) with: {'dropout_rate': 0.0, 'weight_constraint': 2}
0.647725 (0.025814) with: {'dropout_rate': 0.0, 'weight_constraint': 3}
0.542365 (0.132142) with: {'dropout_rate': 0.0, 'weight_constraint': 4}
0.585353 (0.112542) with: {'dropout_rate': 0.0, 'weight_constraint': 5}
0.654655 (0.016017) with: {'dropout_rate': 0.1, 'weight_constraint': 1}
0.566644 (0.140475) with: {'dropout_rate': 0.1, 'weight_constraint': 2}
0.546547 (0.150233) with: {'dropout_rate': 0.1, 'weight_constraint': 3}
0.534766 (0.142541) with: {'dropout_rate': 0.1, 'weight_constraint': 4}
0.656734 (0.013080) with: {'dropout_rate': 0.1, 'weight_constraint': 5}
0.653266 (0.015068) with: {'dropout_rate': 0.2, 'weight_constraint': 1}
0.579118 (0.122834) with: {'dropout_rate': 0.2, 'weight_constraint': 2}
0.654655 (0.016017) with: {'dropout_rate': 0.2, 'weight_constraint': 3}
0.543080 (0.148011) with: {'dropout_rate': 0.2, 'weight_constraint': 4}
0.538912 (0.144573) with: {'dropout_rate': 0.2, 'weight_constraint': 5}
0.644260 (0.016921) with: {'dropout_rate': 0.3, 'weight_constraint': 1}
0.566644 (0.140475) with: {'dropout_rate': 0.3, 'weight_constraint': 2}
0.461307 (0.146517) with: {'dropout_rate': 0.3, 'weight_constraint': 3}
0.435435 (0.143415) with: {'dropout_rate': 0.3, 'weight_constraint': 4}
0.651190 (0.020915) with: {'dropout_rate': 0.3, 'weight_constraint': 5}
0.544468 (0.148987) with: {'dropout_rate': 0.4, 'weight_constraint': 1}
0.543544 (0.149260) with: {'dropout_rate': 0.4, 'weight_constraint': 2}
0.455532 (0.148987) with: {'dropout_rate': 0.4, 'weight_constraint': 3}
0.544468 (0.148987) with: {'dropout_rate': 0.4, 'weight_constraint': 4}
0.654655 (0.016017) with: {'dropout_rate': 0.4, 'weight_constraint': 5}
0.653962 (0.016997) with: {'dropout_rate': 0.5, 'weight_constraint': 1}
0.651877 (0.014327) with: {'dropout_rate': 0.5, 'weight_constraint': 2}
0.611599 (0.054127) with: {'dropout_rate': 0.5, 'weight_constraint': 3}
0.614461 (0.051317) with: {'dropout_rate': 0.5, 'weight_constraint': 4}
0.544468 (0.148987) with: {'dropout_rate': 0.5, 'weight_constraint': 5}
0.566644 (0.140475) with: {'dropout_rate': 0.6, 'weight_constraint': 1}
0.654655 (0.016017) with: {'dropout_rate': 0.6, 'weight_constraint': 2}
0.654655 (0.016017) with: {'dropout_rate': 0.6, 'weight_constraint': 3}
0.481866 (0.137743) with: {'dropout_rate': 0.6, 'weight_constraint': 4}
0.567337 (0.139495) with: {'dropout_rate': 0.6, 'weight_constraint': 5}
0.567337 (0.139495) with: {'dropout_rate': 0.7, 'weight_constraint': 1}
0.566644 (0.140475) with: {'dropout_rate': 0.7, 'weight_constraint': 2}
0.654655 (0.016017) with: {'dropout_rate': 0.7, 'weight_constraint': 3}
0.654655 (0.016017) with: {'dropout_rate': 0.7, 'weight_constraint': 4}
0.544930 (0.149967) with: {'dropout_rate': 0.7, 'weight_constraint': 5}
0.654655 (0.016017) with: {'dropout_rate': 0.8, 'weight_constraint': 1}
0.571492 (0.130676) with: {'dropout_rate': 0.8, 'weight_constraint': 2}
0.654655 (0.016017) with: {'dropout_rate': 0.8, 'weight_constraint': 3}
0.654655 (0.016017) with: {'dropout_rate': 0.8, 'weight_constraint': 4}
0.654655 (0.016017) with: {'dropout_rate': 0.8, 'weight_constraint': 5}
0.567337 (0.139495) with: {'dropout_rate': 0.9, 'weight_constraint': 1}
0.544468 (0.148987) with: {'dropout_rate': 0.9, 'weight_constraint': 2}
0.654655 (0.016017) with: {'dropout_rate': 0.9, 'weight_constraint': 3}
0.654655 (0.016017) with: {'dropout_rate': 0.9, 'weight_constraint': 4}
0.651883 (0.019936) with: {'dropout_rate': 0.9, 'weight_constraint': 5}
```

- En iyi doğruluk değeri 0.656734 ile dropout_rate: 0.1 ve weight_constraint: 5'dir.

7- Gizli Katmandaki Nöron Sayısı Nasıl Ayarlanır

In [19]:
```
# Nöron sayısını ızgarada aramak için scikit-learn'i kullanalım.
import numpy
from sklearn.model_selection import GridSearchCV
from keras.models import Sequential
from keras.layers import Dense
from keras.layers import Dropout
from keras.wrappers.scikit_learn import KerasClassifier
from keras.constraints import maxnorm
```

- 7. olarak Gizli Katmandaki Nöron Sayısı ayarlamayı yapacağız.
- Öncelikle gerekli kütüphanelerimizi import etmemiz gerekiyor.
- Bizim kullanacağımız kütüphaneler keras kütüphanesi içerisinde Sequential, Dense, Dropout, maxnorm, KerasClassifier.
- Numpy kütüphanesi.
- Sklearn kütüphanesi içerisinde ise GridSearchCv'yi kullanacağız.

In [20]:
```
# KerasClassifier için gerekli olan modeli oluşturma
def create_model(neurons=1):
    # Model oluşturma
    model = Sequential()
    model.add(Dense(neurons, input_dim=8, kernel_initializer='uniform', activation='linear', kernel_constraint=maxnorm(4)))
    model.add(Dropout(0.2))
    model.add(Dense(1, kernel_initializer='uniform', activation='sigmoid'))
    # Modeli derleme
    model.compile(loss='binary_crossentropy', optimizer='adam', metrics=['accuracy'])
    return model
# Tekrarlanabilirlik için rastgele tohumu ayarladık.
seed = 7
numpy.random.seed(seed)
# Giriş (X) ve çıkış (Y) değişkenlerine böldük
X = dataset[:,0:8]
Y = dataset[:,8]
# Model oluşturma
model = KerasClassifier(build_fn=create_model, epochs=100, batch_size=10, verbose=0)
# Izgara arama parametrelerini tanımladık.
neurons = [1, 5, 10, 15, 20, 25, 30]
param_grid = dict(neurons=neurons)
grid = GridSearchCV(estimator=model, param_grid=param_grid, n_jobs=-1, cv=3)
grid_result = grid.fit(X, Y)
# Sonuçları özetleyelim.
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))
```

- Öncelikle def ile create_model tanımlıyoruz. Ve neurons yani nöron değerini 1 yapıyoruz.
- Daha sonra 4 katmanlı model oluşturuyoruz.
- İnput_dim ile giriş sayısını 8 yapıyoruz yani 8 girişli model olacak.

- Kernel_initializer yani çekirdek başlatıcı değerini ‘uniform’ yapıyoruz ve activation yani aktivasyon değerini ‘linear’ yapıyoruz.
- Modeli derlemek için de model.compile metodunu kullanıyoruz.
- Compile’nin içerisinde loss değerini ikili çapraz entropli(binary_crossentropy)’i kullanıyoruz.
- Optimizer’değerini ‘adam’ yapıyoruz ve metrics değerinide doğruluk olan ‘accuracy’ yapıyoruz.
- Tekrarlanabilirlik için rastgele tohum kullanacağız bunun için seed değerini 7 yapıyoruz.
- Numpy.random.seed ile Numpy'nin rastgele işlemler için sözde rastgele sayılar üretmesini sağlayan temel bir girdi sağladık.
- Giriş ve Çıkış değerleri için veri setinde ayırma işlemi yaptık.
- X veri seti için dataset içerisinde 0-8 arası sütunları seçtik ve bunu X’e eşitledik.
- Y veri seti için dataset içerisinde 8 den sonraki kısım yani Result kısmını seçtik ve bunu Y’ye eşitledik.
- Daha sonra Keras sınıflandırıcısı(KerasClassifier) oluşturduk ve bunu ‘model’a eşitledik.
- KerasClassifier içerisinde verbose değerini 0 yaptık bunu istersek değiştirebiliriz.
- Build_fn yani inşa etme fonksiyonunuda create_modele eşitledik.
- Epochs değerini 100 ve batch_size değerini de 10 yaptık.
- Daha sonra ızgara arama parametrelerini tanımlama işlemine geçtik.
- Öncelikle neurons dizisi oluşturduk ve içerisine değerlerimizi yazdık. Bu değerler “1, 5, 10, 15, 20, 25, 30” şeklindedir.
- Daha sonra dict() metodu ile boş bir sözlük oluşturduk ve dict içerisine neurons tanımlaması yaptık ve bunu da param_grid’e eşitledik.
- GridSearchCV’yi grid’e eşitliyoruz ve GridSearchCV içerisinde gerekli tanımlamaları yapıyoruz.
- GridSearchCV İşlemi yapı ve parametrelerin her kombinasyon için bir model değerlendirecektir. Çapraz doğrulama her bir modeli değerlendirmek için kullanılır ve varsayılan olarak 3-katlı çapraz doğrulama kullanıldık(cv=3).

- grid.fit() metodu ile döndürülen sonuç nesnesindeki ızgara aramasının sonucuna erişebiliriz ve grid.fit()'i grid_result'a eşitliyoruz.
- Şimdi sonuçları özetleyelim.
- Print içerisinde Best:'e karşılık gelen %f'in değeri grid_result'un en yüksek skoruna eşitledik. Using e karşılık gelen %s değeride grid_resultun en yüksek skorunun parametrelerini yazdırdık.
- grid_result.cv_results_[means_test_score] sonucu bizim her defasında doğruluk değerlerini hesaplayacaktır ve bunu means'a eşitliyoruz.
- grid_result.cv_results_['std_test_score'] ise yapılan işlemlerdeki standart sapmayı hesaplıyor ve bunuda stds'ye eşitliyoruz. - grid_result.cv_results_['params'] ise kullanılan ızgara arama parametrelerini adım adım hesaplıyor ve bunu da params'a eşitliyoruz.
- For döngüsü ile means'ı stds'yi ve params'ı kullanıyoruz. Burdaki amaç döngü oluşturarak print ile her adımı ve doğruluğu ekrana yansıtmak.
- Bu parametreleri zip olarak yazıp means, stdev ve param'a denkleştiriyoruz.
- Print ile hepsini ekrana yansıtıyoruz.

Çıktısı;

```
Best: 0.654655 using {'neurons': 1}
0.654655 (0.016017) with: {'neurons': 1}
0.654655 (0.016017) with: {'neurons': 5}
0.653962 (0.016997) with: {'neurons': 10}
0.624795 (0.034685) with: {'neurons': 15}
0.654655 (0.016017) with: {'neurons': 20}
0.543544 (0.149260) with: {'neurons': 25}
0.455532 (0.148987) with: {'neurons': 30}
```

- En iyi doğruluk değeri 0.654655 ile neurons:1'dir.

4 Sonuç

Multivariable(Çok değişkenli) girişli 2006-2011 arasında bir bölgenin kullandığı elektrik tüketimini 'Jupyter Notebook' ile verileri ekrana göstererek daha sonra bu veri seti ile veri görselleştirmesi, güzelleştirmesi ve düzenlemesi yapılmıştır. Öncelikle modelde önce kütüphaneler belirlendi daha sonra veri seti ekrana yansıtılıp veri setini işledikten(düzenledikten) sonragüzelleştirme yapılıp 7 farklı model oluşturuldu. Daha sonra veri ön işleme adımları uygulanarak model geliştirildi. Öncelikle Parti Boyutu ve Dönem Sayısı hesaplama modeli yapıldı. Daha sonra Eğitim Optimizasyon Algoritmasını Ayarlama modeli yapıldı. Üçüncü olarak Öğrenme Hızını ve Momentum Ayarlama modeli yapıldı. Dördüncü olarak Ağ Ağırlığı Başlatma ayarlama modeli yapıldı. Beşinci olarak Nöron Aktivasyon Fonksiyonu ayarlama modeli yapıldı. Altıncı olarak Bırakma Düzenlemesi ayarlama modeli yapıldı. Son olarak Gizli Katmandaki Nöron Sayısı hesaplama modeli yapıldı.

KAYNAKÇA

- https://www.sharpsightlabs.com/blog/numpy-random-seed/
- https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.GridSearchCV.html
- https://machinelearningmastery.com/grid-search-hyperparameters-deep-learning-modelspython-keras/
- https://keras.io/api/layers/initializers/