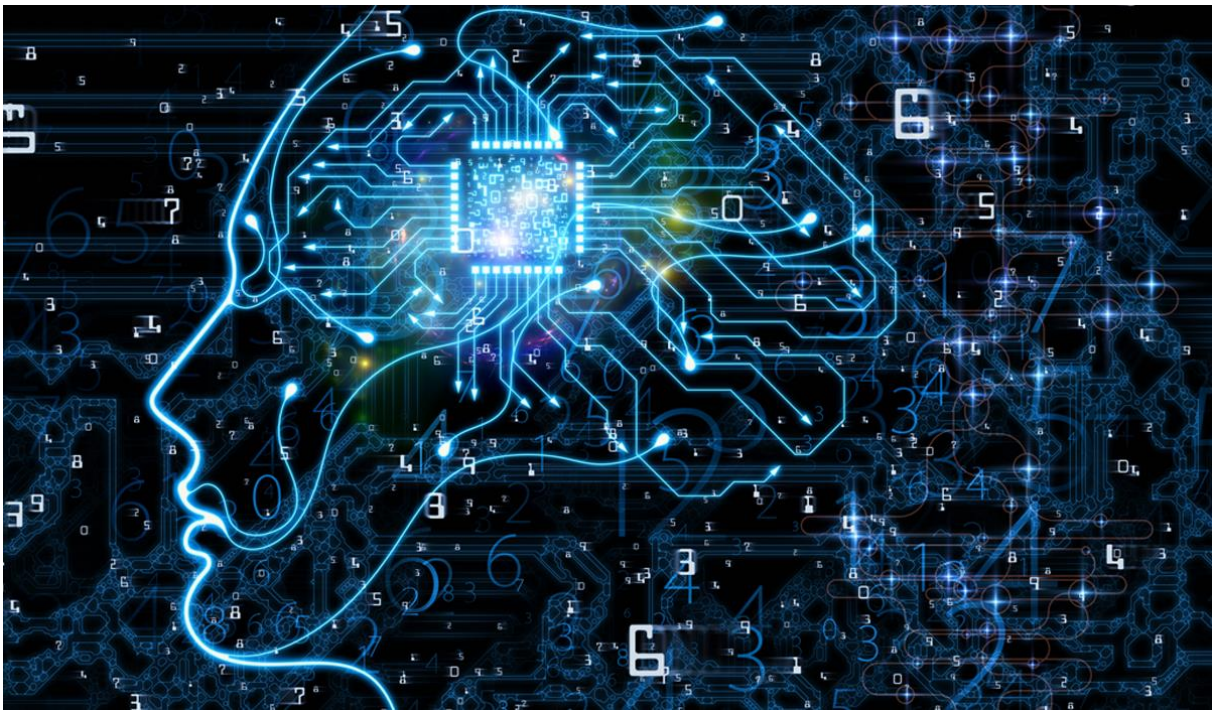
Biyomedikal Mühendisliğinde Yapay Sinir Ağı Uygulamaları Final Proje Raporu



Öğrenci No: 1928142021

Öğrenci Ad/Soyad : Muhammet VARLI

Dersin Sorumlusu: Dr.Öğr.Üyesi Hakan YILMAZ

GRİD SEARCH CROSS VALİDATİON

Model oluştururken, hangi parametrelerin ve hiper parametrelerin belirli bir modelin daha iyi tahminler yapmasını sağlayacağını bilmek genellikle zordur. Grid Search, belirli bir model için en uygun değerleri belirlemek için hiper parametre ayarlaması yapma işlemidir. Tüm modelin performansı belirtilen hiper parametre değerlerine bağlı olduğu için bunu uygulamak önemlidir. Grid Search sayesinde model için gerekli olan hiper parametrelerden model için denenmesini istediğimiz değerleri bir sözlük ya da sözlük listesi yapıları vasıtası ile model için deneyebilirirz. Grid Search girilen tüm parametreleri belirlenen katlı deneyerek bize en iyi sonuçları veren hiper parametreler kümesini vererek, modelimizi en optimum parametreler ile oluşturmamızı sağlamaktadır. Aşağıda Grid Search yapısının bazı parametreleri hakkında bilgiler verilmiştir.

estimator=Tahminci,yani Grid Search uygulanacak model

n_jobs= Paralel olarak çalıştırılacak iş sayısı(-1: Tüm işlemcilerin kullanılması anlamına gelir)

verbose= Ayrıntı düzeyini kontrol eder: daha yüksek, daha fazla mesaj anlamına gelir integer olarak değer alır.

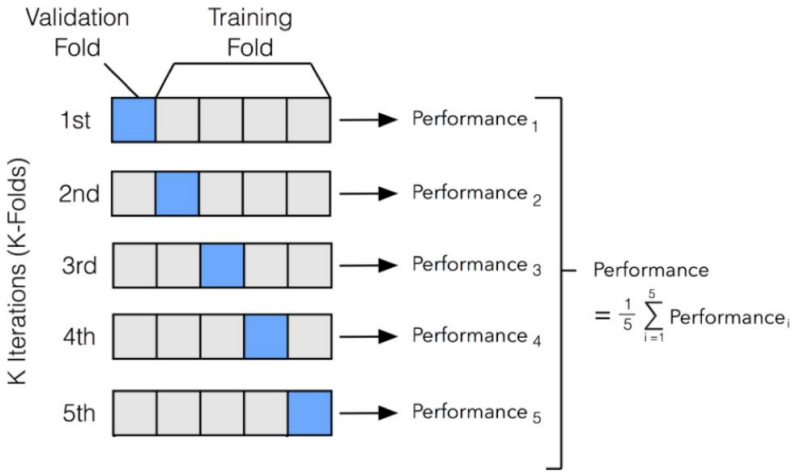
cv=İnteger olarak modele kaç katlı cross validation(çapraz doğrulama) yapılma parametresi

scoring: Test kümesindeki tahminleri değerlendirmek için kullanılacak skorlama.

param_grid= Model için denenmesi istenilen hiper parametreler.

K-FOLD CROSS VALİDATİON

Makine öğrenimi modelinizin istikrarını her zaman doğrulamaya ihtiyaç vardır. Yani modeli eğitim verilerinize sığdıramazsınız ve daha önce hiç görmediği gerçek veriler için doğru bir şekilde çalışacağından emin olunmalıdır. Modelinizi eğitmek için asla yeterli veri olmadığından, doğrulama için bir kısmının kaldırılması underfitting(yetersiz öğrenme) problemi doğurur. Eğitim verilerini azaltarak, veri setindeki önemli paternleri / eğilimleri kaybetme riskiyle karşı karşıya kalırız. Yani, ihtiyacımız olan şey, modeli eğitmek için geniş veri sağlayan ve aynı zamanda doğrulama için geniş veri bırakan bir yöntemdir. K-Katlı çapraz doğrulaması tam olarak bunu yapar. K-Folds Cross Validation'da verilerimizi k farklı alt kümeye böleriz. Verilerimizi eğitmek ve son alt kümeyi test verisi olarak bırakmak için k-1 adet alt kümeyi kullanırız. k adet deney sonucunda ortaya çıkan ortalama hata değeri modelimizin geçerliliğini belirtir.



K değeri genellikle 3 ya da 5 olarak seçilmektedir. Bu değer 10 ya da 15 de seçilebilir ancak bu işlem hacmini arttıracağı için hesaplama uzun sürer bu da zaman kaybına sebep olacaktır.

K-fold Cross Validation için bazı parametreler.

n_splits :int, varsayılan = 5. Kat sayısı yani modeli kaç alt kümeye böleceğimizi belirtir. En az 2 olmalıdır.

Shuffle: boolen, default = False. Toplu olarak bölmeden önce verilerin karıştırılıp karıştırılmayacağı. Her bölmedeki örneklerin karıştırılmayacağını belirtmek için kullanılır.

Cross Validation işleminin yapılabilmesi için Sklearn kütüphanesinden cross_val_score metotunu yüklememiz gereklidir. cross_val_score metodunun bazı parametreleri ise şu şekildedir:

estimator=Tahminci,yani K-fold uygulanacak model

X= Cross Validation işlemi yapılacak bağımsız değişken. Validasyon işlemi test seti üzerinden yapacağımız için bu parametre X_test olacaktır.

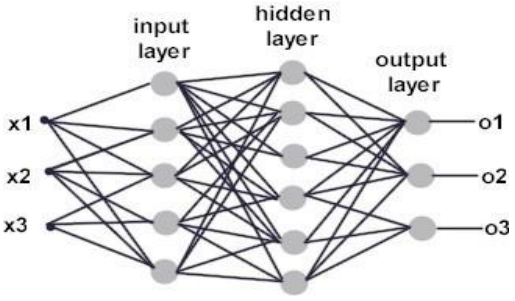
y= Cross Validation işlemi yapılacak bağımlı değişken. Validasyon işlemi test seti üzerinden yapacağımız için bu parametre y_test olacaktır.

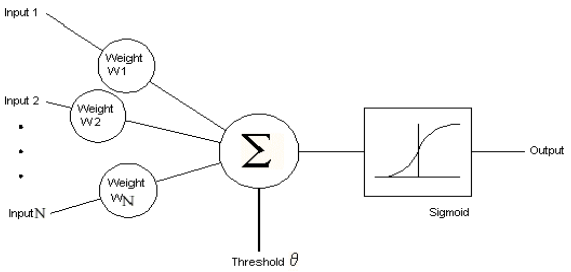
cv= İnteger olarak modele kaç katlı cross validation(çapraz doğrulama) yapılma parametresi

scoring: Test kümesindeki tahminleri değerlendirmek için kullanılacak skorlama

MULTI LAYER PERCEPTRON

Algılayıcı doğrusal bir sınıflandırıcıdır; yani, iki kategoriyi düz bir çizgi ile ayırarak girdiyi sınıflandıran bir algoritmadır. Algılayıcıda girdi tipik olarak bir özellik vektörü ağırlıklar ile çarpılır ve bir bias değeri eklenir. y = w * x + b. Bir algılayıcı, giriş ağırlıklarını kullanarak doğrusal bir kombinasyon oluşturarak (ve bazen çıktıyı doğrusal olmayan bir aktivasyon fonksiyonundan geçirerek) birkaç gerçek değerli girdiye dayanan tek bir çıktı üretir. Ancak tek katmanlı algılayıcılar XOR işlevi gibi doğrusal olmayan sınıflandırma yapmasını engelleyen sığ bir sinir ağıydı. Girdi bir özellik veya başka bir özellik gösterdiğinde 1 ancak her ikisini birden 1 olmama şartı olduğu için tek katmanlı algılayıcılar bu problemin sınıflandırmasında yetersiz kalmaktadır. Çok katmanlı algılayıcılar tam da bu problem üzerine ortaya çıkmıştır. Birçok giriş için bir nöron yeterli olmayabilir. Paralel işlem yapan birden fazla nörona ihtiyaç duyulduğunda katman kavramı devreye girer. Çok katmanlı algılayıcıların tek katmanlı algılayıcılardan farklı olarak arada gizli(hidden) katman bulunmaktadır. Giriş katmanı gelen verileri alarak ara katmana gönderir. Gelen bilgiler bir sonraki katmana aktarılırlar. Modeldeki ara katman sayısı probleme göre değişmektedir. Katmanlar ard arda yerleştiği için önceki katman sonraki katmanın girişi olmaktadır. Böylece çıkışa ulaşılmaktadır.Önceki katmandaki her bir nöron bir sonraki katmandaki bütün nöronlara bağlıdır. Ara katman sayısında olduğu gibi katmanlardaki nöron sayılarıda probleme göre değişmektedir . Çıkış katmanı önceki katmanlardan gelen verileri işleyerek ağın çıkışını belirler. Sistemin çıkış sayısı çıkış katmanında bulunan nöron sayısına eşittir.





Her bir girdi ağırlıklar ile çarpılarak toplanır ve net girdi hesaplanır. Daha sonra aktivasyon fonksiyonu uygulanır ve çıktı elde edilir. Buna İleri Yayılım(Forward Propagation) denir. Amaç gerçek değer ile tahmin edilen değer arasındaki farkı minimize etmektir. Bunun için en sık kullanılan algoritmalardan birisi Geri Yayılım(Backward Propagation) algoritmasıdır. Elde edilen hataların düşürülmesi için geriye yayılım ile çeşitli işlemler yapılır. Bu işlemler iteratif şekilde devam eder. Her iterasyonda girdilerin ağırlıkları güncellenerek en iyi sonuç veren ağırlıklar belirlenir. Buna Delta Öğrenme Kuralı denir.

1. Kardiyotokografi Veri Seti

Dataset Kaynağı: https://archive.ics.uci.edu/ml/datasets/Cardiotocography **Dataset Gözlem Sayısı:** 2129

Dataset Tanıtımı:

Dataset verileri, 2126 fetal kardiyotokogram (CTG) otomatik olarak işlenip ve ilgili tanı özellikleri ölçüldü. CTG'ler ayrıca üç uzman kadın doğum uzmanı ve her birine atanmış bir uzlaşma sınıflandırma etiketi ile sınıflandırılmıştır. Sınıflandırma hem morfolojik örüntü (A, B, C. ...) hem de fetal duruma (N, S, P) göre idi. Bu nedenle, veri kümesi 10-sınıf veya 3-sınıf deneyler için kullanılabilir. Modellerde 3 sınıflı olarak kullanılacaktır.

Değişken Adı		**Değişken Açıklaması**
FileName		CTG muayenesi
Date		Tarih
b		Başlangıç
e		Bitiş
LBE		Temel Değer(Tıp Uzmanı)
LB		FHR Taban Çizgisi (dakikadaki vuruş sayısı)
AC		Saniye Başına İvme Sayısı
FM		Saniye Başına Fetal Hareket Sayısı
UC		Saniye Başına Uterus Kasılma Sayısı
ASTV		Anormal Kısa Süreli Değişkenlik Gösteren Zaman Yüzdesi
mSTV		Kısa Vadeli Değişkenliğin Ortalama Değeri
ALTV		Anormal Uzun Dönem Değişkenlik Zaman Yüzdesi
mLTV		Uzun Dönem Değişkenliğin Ortalama Değeri
DL		Saniye Başına Hafif Yavaşlama Sayısı
DS		Saniye Başına Şiddetli Yavaşlama Sayısı
DP		Saniye Başına Uzatılmış Yavaşlama Sayısı
DR		Tekrarlayan Yavaşlama Sayısı
Width		FHR Histogramının Genişliği
Min		FHR Histogramının En Düşük Frekansı
Max		FHR Histogramının En Yüksek Frekansı
Nmax		Histogram Piklerinin Sayısı
NzeroS		Histogram Sıfırlarının Sayısı
ModE		Histogram Modu
Mean		Histogram Ortalaması
Median		Histogram Medyanı
Variance		Histogram Varyansı
Tendency		Histogram Eğilimi: -1 = sol asimetrik; 0 = simetrik; 1 = sağ asimetrik
CLASS(10 Sınıf) A'dan SUSP'ye kadar sınıflar için sınıf kodu (1 ‘den 10’a kadar)	A	Sakin Uyku
	B	REM Uykusu
	C	Sakin Uyanıklık
	D	Aktif Uyanıklık
	E	Kaydırma Deseni (A veya vardiyalı askıya alma)
	AD	Hızlanma / Yavaşlama Paterni (stres durumu)
	DE	Yavaşlama Düzeni (vagal stimülasyon)
	LD	Büyük Ölçüde Yavaşlayan Model
	FS	Düz Sinüzoidal Desen (patolojik durum)
	SUSP	Şüpheli Desen
NSP(3 Sınıf)	Normal = 1 Şüpheli = 2 Patolojik = 3	

Veri Ön İşleme ve Veri Ölçekleme:

Öncelikle verisetindeki gereksiz olan değişkenler ("FileName","Date","SegFile","b","e","A", "B","C", "D" ,"E", "AD", "DE" ,"LD", "FS", "SUSP") drop edildi. Daha sonra eksik gözlem sayılarına bakılarak bununla ilgili işlem yapıldı. Eksik gözlem sayısı en fazla 3 olduğu için bu gözlemlerin silinerek devam edilmesi en mantıklı olanıdır. Bu yüzden eksik olan gözlemler silindi. Veriseti 2 farklı sınıflandırma problemi için kullanılabilir. Yani hem 3 outputlu NSP bağımlı değişkeni hemde 10 outputlu CLASS bağımlı değişkeni için kullanılabilmektedir. Modellerde 3 outputlu olan NSP bağımlı değişkeni üzerinden modeller kurulacaktır. Bu yüzden CLASS değişkeni de verisetinden drop edildi. Daha sonra 3 outputlu yapı için label encoder işlemi yapılarak bağımlı değişkeni 0,1,2 şeklinde değerlere dönüştürüldü. Daha sonra np_utils.to_categorical ile bağımlı değişkenin formatı 0 ve 1'lerden oluşan 3 boyutlu yapıya dönüştürüldü.

Bağımlı değişken y'nin son hali → $\begin{matrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{matrix}$

Son olarak bağımsız değişkenlerimize (X) ölçekleme işlemi yapılmalıdır. Ölçekleme yöntemi olarak en uygun yöntem olarak sklearn.preprocessing içindeki StandardScaler ölçekleme yöntemi seçildi. Bağımsız değişkenlerimiz ölçeklendikten sonra model için hazır hale gelmiştir.

1.1. MLP Classifier Modeli

Multi Layer Perceptron (MLP) Classifier sklearn kütüphanesinde bulunan bir çok katmanlı algılayıcıdır. İlk olarak MLP ile model nesnesi oluşturuldu. Daha sonra model nesnesi train seti üzerinden fit edildi. Öncelikle valide edilmemiş(doğrulanmamış,ilkel) test skorlarını belirlemek istedim. Bunun için model tahmin işlemi yapılarak, "accuracy" ve "f1_weighted" skorlarını hesapladım. Burada f1 yerine f1_weigted kullanmamızın sebebi hedef değişkeninin(bağımlı değişkenin) multiclass olmasından dolayı f1 score hesaplarken average argümanına [None, 'micro', 'macro', 'weighted'] bunlardan bir tanesini kullanmak gerekmektedir aksi takdirde hata vermektedir. Bu bölümde model için hiçbir parametre girmeden yani model tune(ayar) edilmeden "accuracy" ve "f1_weighted" skorları hesaplandı. Bu skorlar nispeten tune edilmiş modele göre düşük olacaktır. Ayrıca model hedef(bağımlı) değişkeni multi-class(çok sınıflı) olduğu için model tahmin ve skorlama işlemlerinde bağımlı değişkene ait kısımlarda numpy methodu olan np.argmax ile eski haline getirilerek işlemler yapılmıştır. Model için herhangi bir katman bilgisi ya da diğer parametrelerden hiçbirisi girilmeden model oluşturuldu. Her defasında farklı sonuçlar vermemesi açısından tune edilmemiş model için de tune edilmiş model içinde random_state aynı ayarlandı. Bu bölümden sonra Grid Search yapısı ile MLP modelimizin parametre optimizasyonu için çeşitli parametreler modele denenecek ve en uygun parametreler ile model oluşturulacaktır.

Grid search ile bir sözlük yapısı vasıtası ile modele optimizasyon için gönderilen parametreler:

alpha : float, default=0.0001 L2 penalty (regularization term) parameter. (ceza parametresi)

Alpha, ağırlıkların boyutunu kısıtlayarak overfitting(aşırı öğrenme) ile mücadele eden normalleştirme terimi, yani ceza terimi için bir parametredir. Artan alfa, daha küçük ağırlıkları teşvik ederek yüksek varyans (overfitting işareti) düzeltebilir ve bu da daha az eğrilikle ortaya çıkan bir karar sınır grafiğine yol açabilir. Benzer şekilde, azalan alfa, daha büyük ağırlıkları teşvik ederek yüksek bias(yanlılık) (underfitting belirtisi) düzeltebilir ve bu da daha karmaşık bir karar sınırına yol açabilir.

hidden_layer_sizes : tuple, length = n_layers - 2, default=(100,) gizli katmandaki nöron sayısını temsil eder. Burada gizli katman ekleme için içerisine nöron sayısı girerek ifade ediyoruz. Örneğin; "hidden_layer_sizes": (100,50,10) 3 katmanlı 1. katman 100, 2. katman 50, 3. katman 10 hücreden. Kullanılacak gizli katman sayısı, katmanlardaki nöron sayısı probleme göre yani datasete göre değişiklik göstermektedir.

solver : Çözücü, {'lbfgs', 'sgd', 'adam'}, default='adam'. Çözücüler ağırlık optimizasyonu içindir.

'lbfgs': yarı Newton metotları ailesinde bir optimize edicidir. Küçük veri setleri için kullanılır.

'sgd' :stokastik gradyan iniş anlamına gelir.

sdg: Bazı özel parametreleri doğru şekilde ayarlarsanız, çoğu problemin üstesinden gelir. LBFGS, küçük veri setleri için, büyük olanlar için Adam çalışır ve parametrelerini doğru ayarlarsanız SDG çoğu problemi çözebilir. SDG'nin parametreleri öğrenme hızını yansıtabilen learning_rate ve sinir ağının daha az yararlı çözümlerden kaçınmasına yardımcı olan bir değer olan momentum (veya Nesterov'un momentumu). Öğrenme oranını belirlerken, başlangıç değerini (genellikle 0.001 civarında, ancak daha az olabilir) ve eğitim sırasında hızın nasıl değiştiğini (learning_rate_init: 'constant','invscaling', veya 'adaptive' olarak tanımlamanız gerekir.)

'adam': Kingma, Diederik ve Jimmy Ba tarafından önerilen stokastik gradyan tabanlı bir optimizer anlamına gelir. ‘adam' , çok komplex ve büyük verisetleri için iyi bir optimize edicidir.

activation : {'identity', 'logistic', 'tanh', 'relu'}, default='relu'. Gizli katman için aktivasyon fonksiyonunu ifade eder.

- 'identity', f (x) = x değerini döndürür

- 'logistic', 'logistic sigmoid fonksiyonudur f (x) = 1 / (1 + exp (-x)) değerini döndürür.

- 'tanh', hiperbolik tan fonksiyonu, f (x) = tanh (x) değerini döndürür.

- 'relu', düzeltilmiş doğrusal birim fonksiyonu, f(x) = max(0, x) değerini döndürür.

Yukarıda Grid Search yapısına gönderilen, MLP Classifier’ın bazı parametreleri ile ilgili özet bilgiler verildi. Aşağıda denenmesi istenilen parametreler sözlük yapısı şeklinde gösterilmiştir. Ne kadar çok parametre gönderilirse o kadar işlem hacmi büyümektedir. Çünkü gönderilen her bir parametre bir birleri ile tek tek denenerek modeller oluşturulmaktadır.

```
mlpc_params = {"alpha": [0.1, 0.01, 0.0001],
      "hidden_layer_sizes": [(10,10,10), (100,100,100), (100,100)],
       "solver" : ["lbfgs","adam","sgd"],    "activation": ["relu","logistic"]}
```

Daha sonra model için bu parametreler Grid Search’te 5 katlı cross validation işlemi uygulandı.

```
mlpc_cv_model = GridSearchCV(mlpc, mlpc_params,
                             cv = 5, # 5 Katlı CV yapılması için
                             n_jobs = -1, # işlemciyi tam performansta kullanıma olanak sağlar
                             verbose = 2) # işlemciyi tam performansta kullanıma olanak sağlar
```

Model parametreler ile tek tek train seti üzerinden fit edildikten sonra. Grid Search’ün bir fonksiyonu olan best_params_ ile denenen parametrelerden en iyi olan parametreler gözlemlenebilmektedir. Yine bir başka fonksiyon olan best_estimator_ ise en iyi modeli vermektedir. best_estimator_ , en iyi parametrelerle kurulan en iyi sonucu veren modeli ifade eder. Grid Search’e gönderilen parametreler ile 54 parametre deneme işlemi her biri için 5 katlı cross validation yapılarak, toplamda 270 fit gerçekleştirilerek modelimiz için en iyi parametreler şu şekilde oluşmuştur:

En iyi parametreler: {'activation': 'relu', 'alpha': 0.01, 'hidden_layer_sizes': (100, 100), 'solver': 'adam'}

Bir sonraki aşamada K-fold işlemi yapılarak cross validation işlemi hem accuracy hemde f1_weighted skoru için yapılır. K-fold işlemi Grid Search’te belirlenen en iyi model üzerine uygulanmaktadır. K-fold ile veri seti 5 farklı alt kümeye bölünerek cross validation işlemi yapılır. Verisetinde bazı kısımlar modeli çok iyi ifade ederken bazı kısımlar tam olarak ifade etmez. Yani bazı kısımlarda modelin doğruluk oranı yüksek olabilirken bazı kısımlarda ise modelin doğruluk oranı düşük olabilmektedir. Amacımız modelimizin data setinin her yerinde benzer sonuçlar verip vermediğini tespit ederek, modelin tüm veri setinde düzgün çalışıp çalışmadığını kontrol etmektir. Yani dışarıdan modelin hiç görmediği bir veri seti geldiğinde modelin iyi çalışmasını sağlamaktır. K-fold ile modeli farklı bölümlere ayırarak model için farklı bölümlerdeki değerlendirme skorları elde edilerek modelin doğrulanması sağlanmış olur. Skorlama için kullanılan metrik olarak f1_weighted ve accuracy kullanılmıştır. Test

seti üzerinden 5 farklı f1_weighted ve accuracy skoru elde edilmiştir. Daha sonra bu skorların ortalaması alınarak modelin valide edilmiş(doğrulanmış) skorlar elde edilmiş olur.

Daha sonra Grid Search ile belirlenen en iyi model ile model tahmin işlemi yapılır. f1_weighted,accuracy skorları tune edilmiş model için hesaplanır. Sonuçlardan da görüleceği gibi tune(ayarlanmış) edilmiş model ile alınan skorlar tune edilmemiş model ile alınan skorlara göre daha iyi olduğu görülmektedir. Aşağıdaki tabloda tune edilmiş, tune edilmemiş ve K-fold ile alınan skorlar gösterilmiştir.

K-fold Cross Validation f1_weigted Sonuçları: [0.86830451 0.88097292 0.90014789 0.89164953 0.9158371]

K-fold Cross Validation f1_weigted Sonuçlarının Ortalaması: 0.8913823894736403

K-fold Cross Validation accuracy Sonuçları: [0.88372093 0.85882353 0.88235294 0.83529412 0.90588235]

K-fold Cross Validation accuracy Sonuçlarının Ortalaması: 0.8732147742818057

	f1_weighted	accuracy
Tune Edilmemiş(İlkel Model)	0.915629126221463	0.9084507042253521
K-fold Cross Validation Ortalaması	0.8913823894736403	0.8732147742818057
Tune Edilmiş(En İyi Modelle)	0.9118176847496193	0.9131455399061033

Modele K-fold Cross Validation uygulanmış hali ile elde edilen skorlarda bize gösteriyor ki model veri seti farklı alt kümeye bölünmesi ile farklı alt kümelerde farklı sonuçlar vermektedir. K-fold'un bize daha güvenilir sonuçlar verdiği açık bir şekilde gözlemlenmiş oldu. Model veri setinin bazı alt kümelerinde diğerlerine göre nispeten daha az tahmin başarısı elde etmiştir.

Sonraki aşamada tune edilmiş model için classification report ve confusion matrix hesaplamaları yapıldı. Classification report çeşitli skorlama türlerinin sınıflara göre değerlerini bir arada veren bir raporlama şeklidir. Aşağıda classification report ile alınan değer gösterilmiştir. Classification Report'ta elde edilen sonuçlar ileride confusion matrix ile hesaplamalı bir şekilde ayrıntılı olarak anlatılacaktır.

	precision	**recall**	**f1-score**	**support**
0	0.94	0.96	0.95	326
1	0.75	0.71	0.73	58
2	0.92	0.81	0.86	42
accuracy			0.91	426
macro avg	0.87	0.83	0.85	426
weighted avg	0.91	0.91	0.91	426

Confusion matrix ise sınıflandırma problemine ilişkin tahmin sonuçlarının bir özetidir. Doğru ve yanlış tahminlerin sayısı sayım değerleri ile özetlenir ve her sınıf tarafından ayrılır. Confusion matrixi sınıflandırma modelinizin tahminlerde ne zaman karıştırıldığını gösterir. Bize sadece bir sınıflandırıcı tarafından yapılan hatalar hakkında değil, daha da önemlisi yapılan hata türleri hakkında bilgi verir.

	Class 1 Predicted	*Class 2 Predicted*
Class 1 Actual	**TP**	**FN**
Class 2 Actual	**FP**	**TN**

Basit 2 sınıflı bir confusion matrixinin neyi ifade ettiği hakkında terimlerin açıklamaları ile aşağıda belirtilmiştir.

Sınıf 1: Pozitif

Sınıf 2: Negatif

Pozitif (P): Gözlem pozitiftir (örneğin: sağlıklı).

Negatif (N): Gözlem olumlu değildir (örneğin: sağlıklı değil).

Gerçek Pozitif (TP): Gerçekte pozitif olan gözlem pozitif olarak tahmin edilmiştir.

Yanlış Negatif (FN): Gerçekte pozitif olan gözlem negatif olarak tahmin edilmiştir.

Gerçek Negatif (TN): Gerçekte negatif olan gözlem negatif olarak tahmin edilmiştir.

Yanlış Pozitif (FP): Gerçekte negatif olan gözlem pozitif olarak tahmin edilmiştir.

Bu terimler kullanılarak çeşitli metrikler hesaplanabilir;

Accuracy = TP+TN/TP+FP+FN+TN **Precision** = TP/TP+FP **Recall** = TP/TP+FN

F1 Score = 2*(Recall * Precision) / (Recall + Precision)

Şimdi modelimiz için oluşan confusion matrixini yorumlayalım:

	Class 0 Predicted	**Class 1 Predicted**	**Class 2 Predicted**	**Toplam Support**
Class 0 Actual	314	10	2	326
Class 1 Actual	16	41	1	58
Class 2 Actual	4	4	34	42

Class 0: Normal **Class 1**: Suspect **Class 3**:Pathological

314 hasta test datasında gerçekte Normal, tahmin sonucunda da Normal (Doğru Sınıflandırma)

41 hasta test datasında gerçekte Suspect, tahmin sonucunda da Suspect (Doğru Sınıflandırma)

34 hasta test datasında gerçekte Pathological, tahmin sonucunda da Pathological (Doğru Sınıflandırma)

10 hasta test datasında gerçekte Normal, tahmin sonucunda da Suspect (Yanlış Sınıflandırma)

2 hasta test datasında gerçekte Normal, tahmin sonucunda da Pathological (Yanlış Sınıflandırma)

16 hasta test datasında gerçekte Suspect, tahmin sonucunda da Normal (Yanlış Sınıflandırma)

1 hasta test datasında gerçekte Suspect, tahmin sonucunda da Pathological (Yanlış Sınıflandırma)

4 hasta test datasında gerçekte Pathological, tahmin sonucunda da Normal (Yanlış Sınıflandırma)

4 hasta test datasında gerçekte Pathological, tahmin sonucunda da Suspect (Yanlış Sınıflandırma)

** Modelin yanlış sınıflandırmalarından en yüksek olanı 16 ile gerçekte Suspect olduğu halde modelimiz bunu Normal olarak tahmin etmiştir. Bu da classification reporttaki 1. Sınıfa ait skorların düşüklüğünü açıklamaktadır. Model sınıfları dengesiz bir şekilde dağıldığı için sınıflara göre oluşan skorlarda sayısı az olan sınıfların skorlarının düşük olmasına sebep olmaktadır. Datasetteki 0. Sınıf sayısı çok olduğu için diğer sınıflara göre başarı oranı bir hayli yüksek çıkmaktadır. Bu problemin önüne geçebilmek için class_weight ile sınıflara ağırlık verilebilir ancak MLP Classifier'ın böyle bir seçeneği mevcut olmadığı için kullanamadık[HCI in Business, Government, and Organizations: Information Systems, Sayfa 47]. Sınıflara ağırlık verme yaklaşımı bu problem için Keras Modelinde uygulanacaktır.

Şimdi de modelin confusion matrixine göre classification reporttaki değerleri hesaplayalım:

Toplam Doğru Sınıflandırma Sayısı:(314+41+34)=389

Toplam Popülasyon Sayısı: (314+10+2+16+41+1+4+4+34)=426

Accuracy=389/426=0.9131≅ 0.91 yani modelimiz %91 accuracy değerine sahip.

Sınıf 0 için: **Sınıf 1 için:**

Precision= 314/(314+16+4)=0.9401≅0.94 Precision= 41/(41+10+4)=0.7454≅0.75

Recall=314/(314+10+2)=0.9631 ≅0.96 Recall=41/(41+16+1)=0.7068 ≅0.71

f1=2*(0.96*0.94)/(0.96+0.94)=0.9498 ≅0.95 f1=2*(0.71*0.75)/(0.71+0.75)=0.7294≅0.73

Sınıf 2 için:

Precision= 34/(34+2+1)=0.9189≅0.92

Recall=34/(34+4+4)=0.8095 ≅0.81

f1=2*(0.81*0.92)/(0.81+0.92)=0.8615≅0.86

Son olarak precision,recall,f1 skorları için macro ve weighted şeklinde hesaplamalar yapılacaktır. Bu skorların macro değerleri hesaplanırken tüm sınıflara göre değerleri toplanarak toplam sınıf sayısına bölünerek bulunur. Weighted değerleri ise her sınıfın skor değeri her bir sınıfın gerçek toplam sayıları ile çarpılarak yani support değerleri ile çarpılarak toplanır ve toplam popülasyon sayısına(toplam nüfus) bölünerek bulunur.

f1_weighted=(0.95*326+0.73*58+0.86*42)/426=0.9111≅0.91

f1_macro=(0.95+0.73+0.86)/3=0.8466≅0.85

precision_weighted=(0.94*326+0.75*58+0.92*42)/ 426=0.9121≅0.91

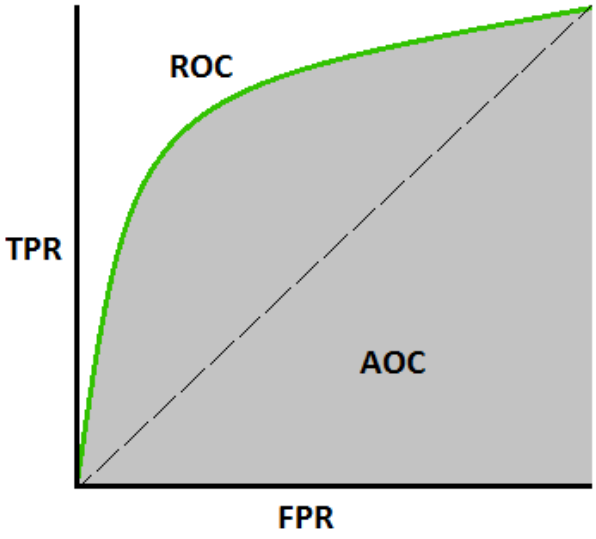
precision_macro=(0.94+0.75+0.92)/3=0.87

recall_weighted=(0.96*326+0.71*58+0.81*42)/ 426=0.9111≅0.91

recall_macro=(0.96+0.71+0.81)/3=0.8266≅0.83

Confusion matrixine göre yapılan tüm hesaplamalar görüleceği gibi classification reporttaki değerler ile bire bir örtüşmektedir.

Son olarak model için ReceiverOperating Characteristic Curve (Alıcı çalışma karakteristik eğrisi) ve Area Under the Curve(Eğri Altındaki Alan) uygulaması yapılmıştır. Çok sınıflı sınıflandırma probleminin performansını kontrol etmemiz veya görselleştirmemiz gerektiğinde, AUC (Eğrinin Altındaki Alan) ROC (Alıcı Çalışma Özellikleri) eğrisini kullanıyoruz. Herhangi bir sınıflandırma modelinin performansını kontrol etmek için en önemli değerlendirme metriklerinden biridir. AUC - ROC eğrisi, çeşitli eşik ayarlarında sınıflandırma problemi için bir performans ölçümüdür. ROC bir olasılık eğrisidir ve AUC ayrılabilirliğin derecesini veya ölçüsünü temsil eder.Modelleri sınıflar arasında ne kadar ayırt edebildiğini anlatır.AUC yükseldikçe modelin tahmin etme başarısı da artar. AUC ne kadar yüksekse, model sınıfları ayırt etmekte o kadar iyidir. ROC eğrisi, TPR'nin y ekseninde ve FPR'nin x ekseninde olduğu FPR'ye karşı TPR ile çizilerek oluşturulur.



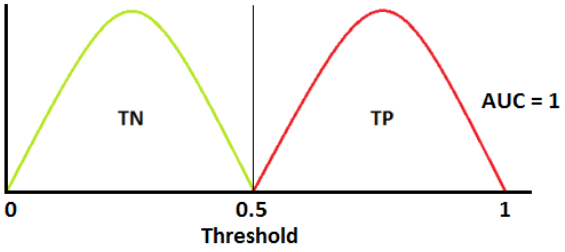
ROC Eğrisinde Kullanılan terimler:

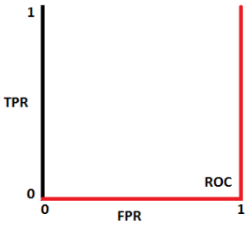
TPR (Gerçek Pozitif Hız) = Recall = Hassasiyet **TPR** =TP/(TP+FN)

Specificity(Özgünlük)=TN/(TN+FP), FPR=1-Specificity, **FPR**=FP/(TN+FP)

Mükemmel bir model, 1'e yakın AUC'ye sahiptir, bu da iyi bir ayrılabilirlik ölçüsüne sahip olduğu anlamına gelir. Zayıf bir model 0'a yakın AUC'ye sahiptir, bu da en kötü ayrılabilirlik ölçüsüne sahip olduğu anlamına gelir. Aslında bu sonucu tersine çeviriyor demektir. 0'ları 1s ve 1'leri 0s olarak tahmin ediyor. Ve AUC 0.5 olduğunda,

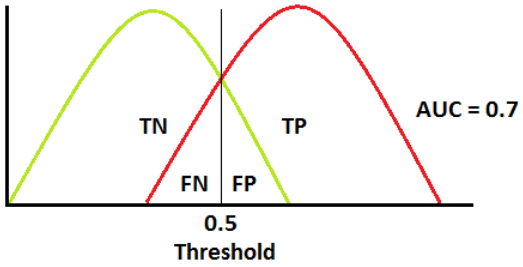
modelin hiçbir sınıf ayırma kapasitesi olmadığı anlamına gelir. ROC eğrisini ve AUC skorunu 2 sınıflı bir örnek için ele alacak olursak:

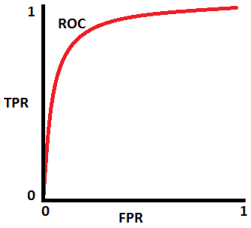




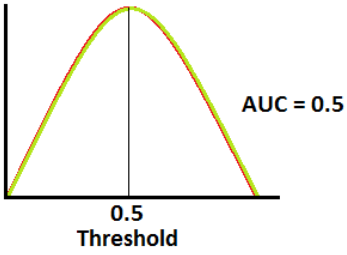
Not:Kırmızı dağılım eğrisi pozitif sınıftadır ve yeşil dağılım eğrisi negatif sınıflıdır.

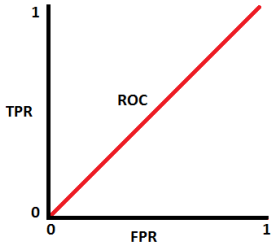
Bu ideal bir durum. İki eğri hiçbir şekilde örtüşmediğinde, model ideal bir ayrılabilirlik ölçüsüne sahiptir. Pozitif sınıf ile negatif sınıf arasında mükemmel bir ayrım yapabilir.



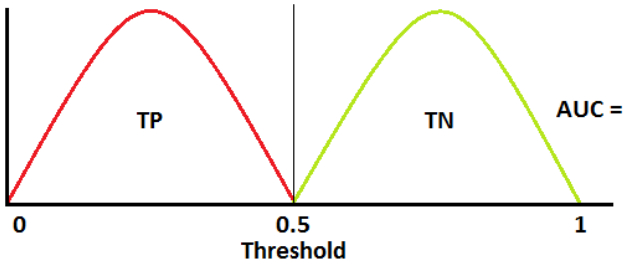


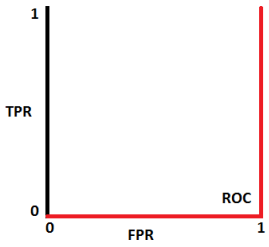
İki dağılım çakıştığında, tip 1 ve tip 2 hatasını ortaya koyarız. Eşiğe bağlı olarak, bunları en aza indirebilir veya en üst düzeye çıkarabiliriz. AUC 0.7 olduğunda, modelin pozitif sınıf ile negatif sınıf arasında ayrım yapabilme şansının% 70 olduğu anlamına gelir.



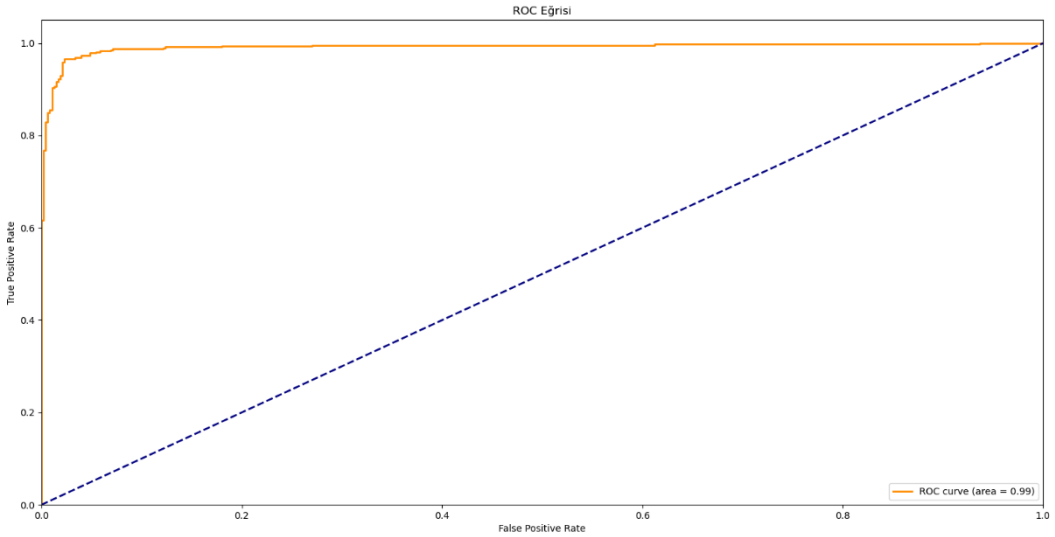


Bu en kötü durumdur. AUC yaklaşık 0.5 olduğunda, modelin pozitif sınıf ile negatif sınıf arasında ayrım yapma kapasitesi yoktur.

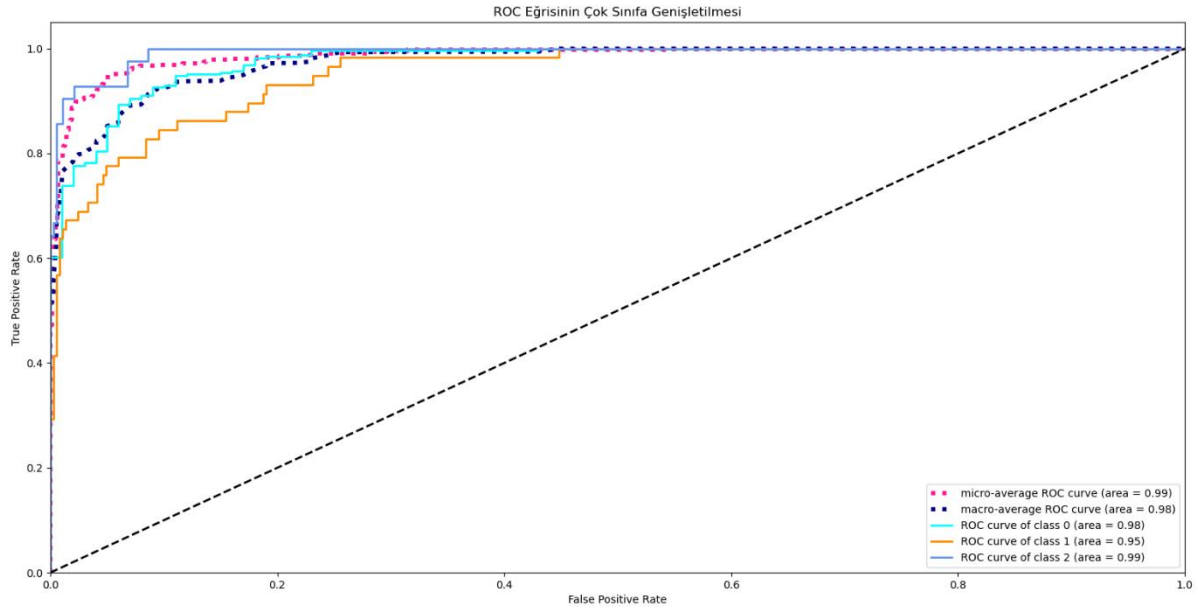




AUC yaklaşık 0 olduğunda, model aslında sınıfları terse çevirmiş olur. Bu, modelin negatif sınıfı pozitif bir sınıf olarak tahmin ettiği ya da pozitif bir sınıfı negatif bir sınıf olarak tahmin ettiği anlamına gelir. Çok sınıflı modelde, One ve ALL yöntemlerini kullanarak n sayı sınıfları için n sayıda AUC ROC eğrisi çizebiliriz. Modelimizde 0, 1 ve 2 adında üç sınıfınız var , 1 ve 2'ye karşı sınıflandırılmış 0 için bir ROC, 0 ve 3'e karşı sınıflandırılmış 2 için başka bir ROC ve 2 ve 0'a göre sınıflandırılmış 3 için üçüncü bir ROC eğrisine sahip olmaktayız. Model için ROC eğrileri ve AUC değerleri hesaplaması öncelikle gözlemlemek adına kendi seçtiğimiz herhangi bir sınıf özelinde yapıldı. Aşağıdaki grafik sınıf 2'ye ait ROC eğrisini göstermektedir.



Model sınıf 2 için AUC değeri 0.99 gibi yüksek bir orana sahip. Bunun anlamı modelin sınıf ayrımı yaparken 2 sınıf için başarısı %99 oranında olduğu anlamına gelmektedir. Daha sonra tüm sınıfların ROC eğrileri ve her bir sınıfın AUC değerleri ve macro-micro ortalama ROC eğrileri ve AUC değerleri hesaplanarak çizdirilmiştir.



Model için elde edilen AUC değerleri; micro-average ROC eğrisi için 0.99, macro-average ROC eğrisi için 0.98, class 0 ROC eğrisi için(Normal) 0.98, class 1 ROC eğrisi için(Suspect) 0.95 ve class 2 ROC eğrisi için (Pathological) 0.99 şeklinde gözlemlenmiştir. Sonuçlardan da görüleceği gibi modelimiz sınıf ayrımını yapma doğruluğu konusunda class 1 için diğerlerine göre biraz daha düşük sonuç ortaya koymuştur. Bunu zaten confusion matrisinde class 1 için olan değerlerde tespit etmiştik. Genel olarak modelimiz sınıf ayrımını yüksek bir başarı ile yapabilmektedir. Bu oranlar ne kadar yüksek olursa modelin tahminleri yaparken hata yapma olasılığı o kadar düşük olmaktadır. ROC eğrileri bu konuda bize bu önemli bilgileri vermektedir.

1.2. Keras Modeli

Multi Layer Perceptron için Keras Python kütüphanesi, modellerin bir katman dizisi olarak oluşturulmasına odaklanır. Keras ile model için her bir katman tek tek oluşturulabilir ve her bir katman özelinde ayarlamalar yapılmasına olanak sağladığı için daha esnek bir yapı ortaya koymaktadır. Model katmanları tek tek oluşturulduğu için her bir katmandaki nöron sayısı Dense fonksiyonu ile belirtilmelidir. Giriş katmanı için veri setindeki bağımsız değişken sayısı(girdi boyutu) verilmelidir. Data setimizde 23 tane bağımsız değişkenimiz olduğu için input_dim=23 olarak belirtildi. Daha sonra her bir katman için aktivasyon fonksiyonu da her bir katman özelinde ayarlanabildiği için bunun da belirtilmesi gerekmektedir. Keras için çıktı katmanın nöron sayısının da(sınıf sayısının) belirtilmesi gerekmektedir. Ayrıca Keras modelinde sınıf sayısı 2'den fazla ise çıktı katmanının aktivasyon fonksiyonu "softmax" olarak belirtilmelidir. MLPClassifier'da ise bunları belirtmemize gerek yok. Keras modelini compile(derleme) ederken loss(kayıp) fonksiyonunu belirtmemiz gerekmektedir. MLPClassifier için loss fonksiyonunu belirtmeye gerek yoktur kendisi otomatik olarak seçer.

Kayıp Fonksiyonu ve Kayıp Fonksiyonu Seçimi

Bir sinir ağı modelindeki parametre değerlerini optimize etmek için bir kayıp fonksiyonu kullanılır. Kayıp fonksiyonları, ağ için bir dizi parametre değerini, bu parametrenin ağın yapmayı amaçladığı görevi ne kadar iyi başardığını gösteren bir skaler değere eşler. Tipik olarak sinir ağları ile hatayı en aza indirmeye çalışırız. Bu nedenle, objektif fonksiyon genellikle bir maliyet fonksiyonu veya bir kayıp fonksiyonu olarak adlandırılır ve kayıp fonksiyonu tarafından hesaplanan değer basitçe " kayıp " olarak adlandırılır. Kayıp fonksiyonu seçimi doğrudan sinir ağınızın çıkış katmanında kullanılan aktivasyon fonksiyonu ile ilgilidir bu yüzden önemlidir. Sınıflandırma problemleri için kayıp fonksiyonu çapraz entropi ve regresyon problemleri için ortalama kare hata kaybı fonksiyonu kullanılır. Kerasta, 2 sınıflı sınıflandırma problemleri için hedef değişkeni {-1,1} şeklinde ise 'hinge' veya 'squared_hinge' kayıp fonksiyonları, hedef değişkeni {0,1} şeklinde binary formda ise 'binary_crossentropy' kayıp fonksiyonu seçilir. Hedef değişkeni {0,1,2..,n} şeklinde her sınıfa benzersiz bir tamsayı değerine sahip olduğu çok sınıflı sınıflandırma problemlerinde 'categorical_crossentropy' kayıp fonksiyonu kullanılır. Çok fazla sayıda etiketle sınıflandırma sorunları olan 'categorical_crossentropy' kullanıldığında olası bir hayal kırıklığı yapabilir bunun nedeni, one hot encoding işlemidir. Örneğin, bir kelime haznesindeki kelimeleri tahmin etmek, her bir etiket için bir tane olmak üzere on veya yüz binlerce kategoriye sahip olabilir. Bu, her eğitim örneğinin hedef elemanının, önemli bellek gerektiren on veya yüz binlerce sıfır değere sahip one hot encoding gerektirebileceği anlamına gelebilir. Bu durumlarda 'sparse_categorical_crossentropy' kullanılarak, hedef değişkenin eğitimden önce one hot encoding olmasını gerektirmeden aynı çapraz entropi hata hesaplamasını yaparak bunu giderir. Bir başka çok sınıflı sınıflandırma problemlerinde kullanılan kayıp fonksiyonu ise 'kullback_leibler_divergence' 'dir. Kullback Leibler Divergence, bir olasılık dağılımının bir taban çizgisi dağılımından nasıl farklı olduğunun bir ölçüsüdür. 0'lık bir KL diverjans kaybı, dağılımların aynı olduğunu gösterir. Uygulamada, KL Diverjansının davranışı çapraz entropiye çok benzer. İstenen hedef olasılık dağılımını tahmin etmek için tahmin edilen olasılık dağılımı kullanılırsa, ne kadar bilginin (bit cinsinden) kaybedileceğini hesaplar. Bu nedenle, KL ıraksama kaybı fonksiyonu, sadece çok sınıflı bir sınıflandırmadan daha karmaşık bir işlevi tahmin etmeyi öğrenen modeller kullanılırken, örneğin bir model altında yoğun bir özellik sunumunu öğrenmek için daha yaygın olarak kullanılır. Bununla birlikte, çok sınıflı sınıflandırma için kullanılabilir, bu durumda fonksiyonel olarak çok sınıflı çapraz entropiye eşdeğerdir.

Keras modelimiz 3 sınıflı hedef değişkenine sahip olduğu için modeli compile ederken loss fonksiyonu 'categorical_crossentropy' olarak seçildi. Modeli ilk başta kurarken optimizer için "adam" olarak compile ettim. Modelin metrics argümanını "accuracy" olarak belirledim. Modeli bir fonksiyon şeklinde tanımlayıp kurduktan sonra modelin loss fonksiyonunun değerinin epoch sayısına göre değişimini train ve test seti özelinde gözlemlemek için train seti ile eğittim. Burada epochs olarak 100 , batch_size olarak ise 32 olarak belirledim. Şimdi batch_size argümanı hakkında bilgiler verelim. Küme büyüklüğü (batch_size) bir seferde yapay sinir ağını eğitmek için kullanılacak örnek sayısını belirtir. Batch_size değerini belirlemede 3 temel yaklaşım vardır:

1. Batch Gradient Descent: Bu yaklaşımda batch_size, train setindeki toplam örnek sayısına ayarlanır.

2. Stochastic Gradient Descent: Bu yaklaşımda batch_size 1'e ayarlanır.

3. Minibatch Gradient Descent: Bu yaklaşımda batch_size 1'den daha büyük ve train setindeki toplam örnek sayısından daha az bir değere ayarlanır.

Derin öğrenme sinir ağlarını eğitmek için sıklıkla çok büyük veri setleri kullanıldığı göz önüne alındığında, batch_size nadiren train veri kümesinin boyutuna ayarlanır.

Daha küçük batch_size iki ana nedenden dolayı kullanılır:

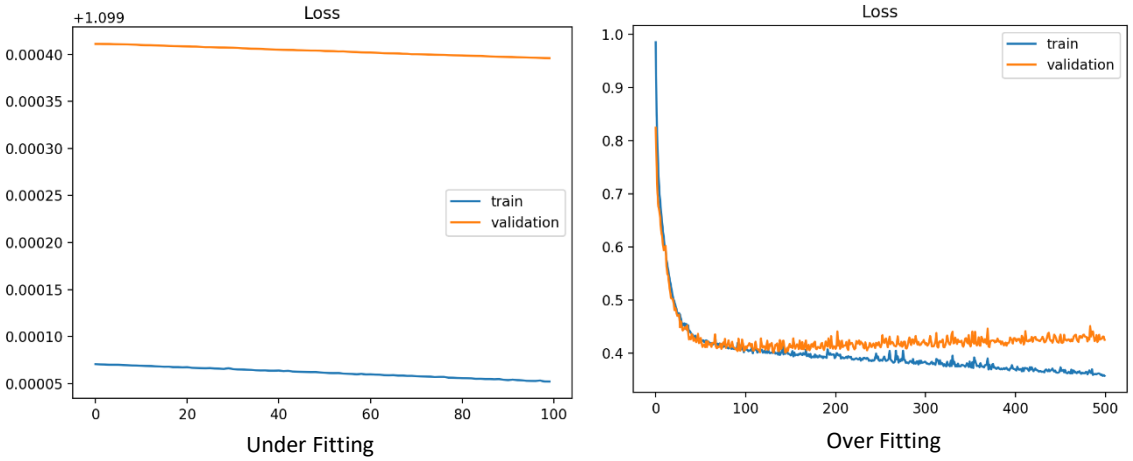
* Daha küçük batch_size'lar gürültülüdür, düzenleyici bir etki ve daha düşük genelleme hatası sunar.

* Daha küçük batch_size'lar belleğe bir parti train verisinin sığmasını kolaylaştırır (yani GPU kullanırken).

* Üçüncü neden, batch_size'ın genellikle 32 örnek gibi küçük bir şeye ayarlanmış olması ve uygulayıcı tarafından ayarlanmamış olmasıdır. 32 gibi küçük batch_size'lar genellikle iyi çalışır.

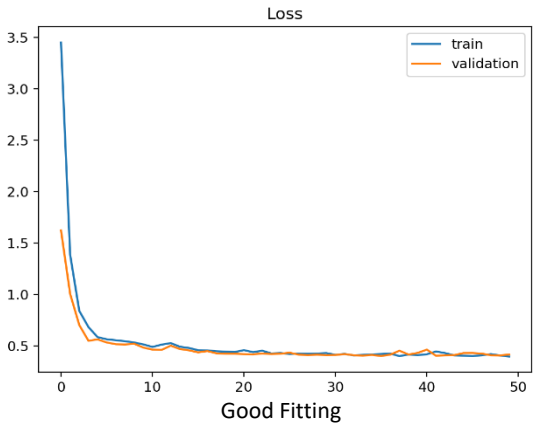
Küçük batch_size'lar genellikle hızlı öğrenmeyle sonuçlanmaktadır, ancak sınıflandırma doğruluğunda daha yüksek değişkenliğe sahip uçucu bir öğrenme süreciyle sonuçlandığını gözlemlenmektedir. Daha büyük batch_size'lar öğrenme sürecini yavaşlatır, ancak son aşamalar sınıflandırma doğruluğunda daha düşük sapma ile örneklenen daha kararlı bir modele yakınsama ile sonuçlanır. Dolayısıyla modelimi eğitirken ilk olarak batch_size'ı 32 gibi genellikle iyi sonuç veren bir değer ile eğittim. Eğitim için validation_data argümanına da X_test ve y_test girilerek eğitim işlemi yapılır. Model eğitimi sırasında train ve test kaybını gözlemlemek için eğitilen modelin history fonksiyonu ile bir grafik üzerinde gözlemlendi. Kayıp değerlerinin gözlemlenmesi modelin under_fitting(yetersiz öğrenme) ya da over_fitting(aşırı öğrenme) gibi problemlerinin belirlenmesi için büyük önem arz etmektedir. Ayrıca model performansının gözlemlenmesi açısından da önemli bir işlemdir.

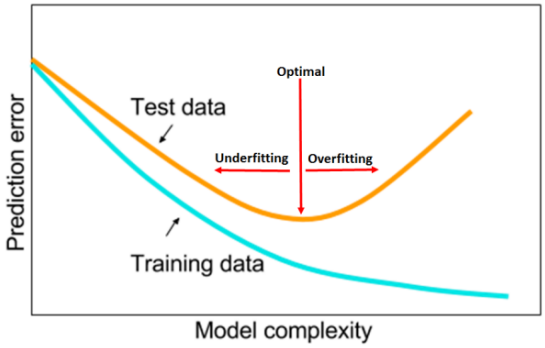
Underfit model, eğitim veri kümesinde iyi performans gösteren ve test veri kümesinde zayıf performans gösteren bir modeldir. Underfit bir model yüksek eğitim ve yüksek test hatasına sahiptir. Bir overfit modeli, eğitim setindeki performansın iyi olduğu ve gelişmeye devam ettiği modelde, validasyon(test) setindeki performans bir noktaya yükselir ve daha sonra azalmaya başlar. Bu, eğitim kaybının aşağıya doğru eğildiği ve test kaybının aşağıya doğru eğildiği, bir bükülme noktasına çarptığı ve tekrar yukarı doğru eğime başladığından teşhis edilebilir. Bir overfit modelinde son derece düşük eğitim hatası, ancak yüksek test hatası olacaktır. Overfit bir model, train veri kümesindeki istatistiksel gürültü veya rastgele dalgalanmalar dahil olmak üzere train veri kümesini çok iyi öğrenmiş bir modeli ifade eder. Goodfit(iyi uyumlu), modelin performansının hem eğitim hem de test setlerinde iyi olduğu bir durumdur. İyi bir modelde train ve test hataları düşüktür. Aşağıda underfitting,overfitting ve goodfitting örnekler ile gösterilmiştir.



Under Fitting

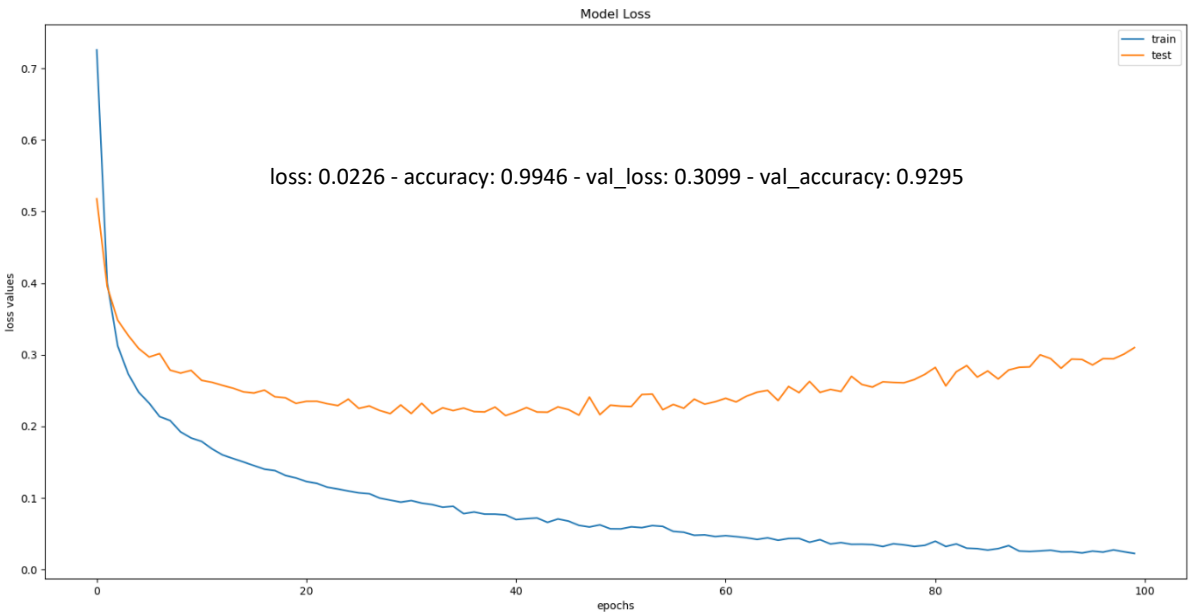
Over Fitting





İlk olarak modeli kötü bir şekilde kuralım ve train kaybı ve test kaybını gözlemleyelim.

```
def create_model(optimizer="adam"):
    model = Sequential()
    model.add(Dense(64, input_dim=23, activation='relu'))
    model.add(Dense(32, activation='relu'))
    model.add(Dense(3, activation='softmax'))
    model.compile(loss='categorical_crossentropy', optimizer=optimizer,metrics=["accuracy"])
    return model
model = create_model()
egitim=model.fit(X_train, y_train, epochs=100, batch_size=32, verbose=1,validation_data=(X_test,y_test))
```



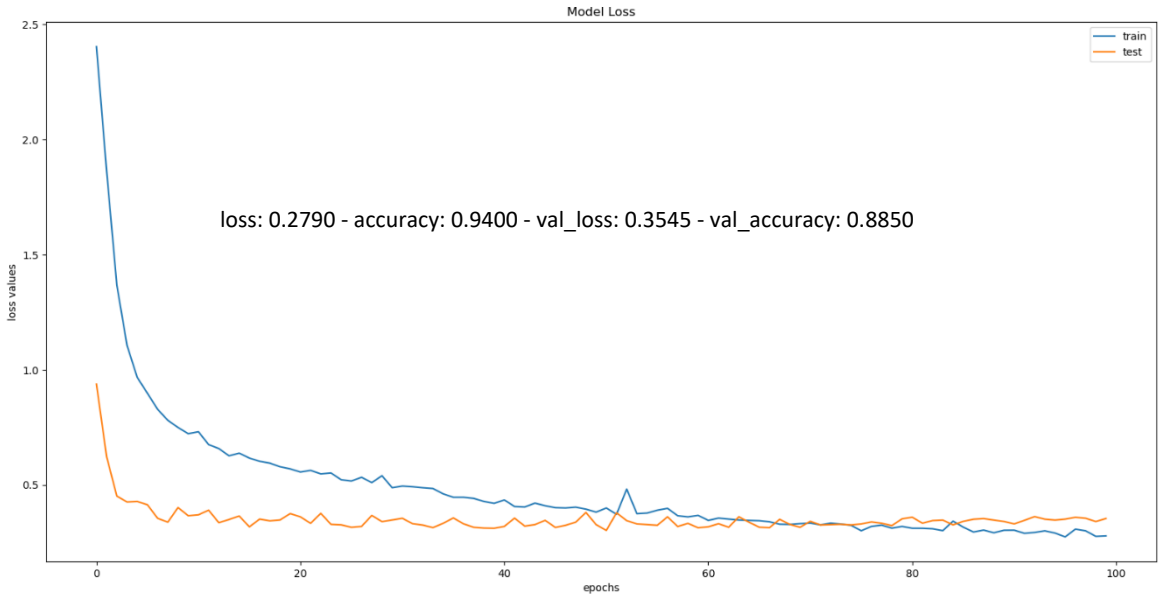
Grafikten de anlaşılacağı üzere oldukça kötü bir model. Modelin test hatası artarken train hatası düşmektedir. Bu da modelin overfittinge doğru gittiğini yani modelin train seti üzerinde yüksek doğruluğa sahip olup test seti üzerinde daha düşük doğruluğa sahip olduğunu göstermektedir. Model train setini çok başarılı bir şekilde tahmin ederken, test setini o başarıda tahmin edememiştir. Şimdi modelimizi biraz geliştirelim overfittingin önüne geçmek için modelde düzenleme yapmamız gerekiyor. Modele fazladan bir katman eklendi ve sınıf sayılarındaki dengesizlik için class_weight kullanıldı ve katmanların nöron sayılarıda düzenlendi.

0. Sınıf: 1655 1. Sınıf: 295 2. Sınıf: 176 görüldüğü gibi sınıfların dağılımı dengesiz bu test ve train tahminindeki farklılığa etki etmektedir. Bu yüzden sınıflara ağırlıklar verilerek bu dengesizliğin etkisini biraz olsun giderildi. Sınıf ağırlıkları 1655'e diğer sınıfları eşitleyecek şekilde verilmiştir.

```
class_weight = {0: 1, 1: 5.74, 2: 9.4}

def create_model(optimizer="adam"):
    model = Sequential()
    model.add(Dense(20, input_dim=23, activation='relu'))
    model.add(Dense(40, activation='sigmoid'))
    model.add(Dense(60, activation='relu'))
    model.add(Dense(3, activation='softmax'))
    model.compile(loss='categorical_crossentropy', optimizer=optimizer,metrics=["accuracy"])
    return model
model = create_model()

egitim=model.fit(X_train, y_train, epochs=100, batch_size=32,class_weight=class_weight, verbose=1,validation_data=(X_test,y_test))
```



Grafikte de görüldüğü gibi modelin test ve train kayıpları birbirine yaklaşmıştır. Modelin train setindeki tahmin başarısı ve test setindeki tahmin başarısı da birbirine yakın değerler göstermektedir.

Öncelikle valide edilmemiş(doğrulanmamış,ilkel) test skorlarını belirlemek istedim. Bunun için model tahmin işlemi yapılarak, "accuracy" ve "f1_weighted" skorlarını hesapladım. Burada f1 yerine f1_weigted kullanmamızın sebebi hedef değişkeninin(bağımlı değişkenin) multiclass olmasından dolayı f1 score hesaplarken average argümanına [None, 'micro', 'macro', 'weighted'] bunlardan bir tanesini kullanmak gerekmektedir aksi takdirde hata vermektedir. Bu bölümde model için hiçbir parametre girmeden yani model tune(ayar) edilmeden "accuracy" ve "f1_weighted" skorları hesaplandı. Bu skorlar nispeten tune edilmiş modele göre düşük olacaktır. Ayrıca model hedef(bağımlı) değişkeni multi-class(çok sınıflı) olduğu için model tahmin ve skorlama işlemlerinde bağımlı değişkene ait kısımlarda numpy methodu olan np.argmax ile eski haline getirilerek işlemler yapılmıştır. Bu skorlar nispeten tune edilmiş modele göre düşük olacaktır. Bu bölümden sonra Grid Search yapısı ile Keras modelimizin parametre optimizasyonu için çeşitli parametreler modele denenecek ve en uygun parametreler ile model oluşturulacaktır.

Grid search ile bir sözlük yapısı vasıtası ile modele optimizasyon için gönderilen parametreler:

epochs: Bir Epoch, tüm veri kümesinin sinir ağı üzerinden sadece bir kez ileri ve geri aktarılmasıdır. Başlangıçta, tüm veri kümesinin bir sinir ağı üzerinden geçirilmesi yeterli değildir. Ve tam veri kümesini aynı sinir ağına birçok kez geçirmemiz gerekiyor. Ancak, sınırlı bir veri kümesi kullandığımızı ve yinelemeli bir süreç olan Gradient Descent kullandığımız öğrenmeyi ve grafiği optimize etmek için kullandığımızı unutulmamalıdır. Bu nedenle, ağırlıkların tek geçiş veya bir dönemle güncellenmesi yeterli değildir.

batch_size: Küme büyüklüğü (batch_size) bir seferde yapay sinir ağını eğitmek için kullanılacak örnek sayısını belirtir.

optimizer : Optimize edici, { 'RMSprop', 'Adam','SGD', Adadelta, Adagrad, Adamax, Nadam, Ftrl}, Optimizerler ağırlık optimizasyonu içindir.

activation : { sigmoid, relu, softmax, softplus,tanh, softsign,selu,elu, exponential ve keras.layers.advanced_activations ile PreLU, LeakyReLU}, Gizli katman için aktivasyon fonksiyonunu ifade eder.

Yukarıda Grid Search yapısına gönderilen, Keras modelinin bazı parametreleri verilmiştir. Aktivasyon fonksiyonlarından en önemlileri relu genellikle gizli katmanlarda yani çıkış değil ara katmanlarda kullanılır. Sigmoid ise genelde sınıflandırıcılarda iyi çalışmakta ve iki çıkışlı sınıflandırma problemi için genelde sigmoid tercih edilir. Softmax ise genelde çok sınıflı problemlerde çıkış katmanı olarak kullanılır. Relu ile modelde ölü nöronlara rastlanırsa LeakyReLU daha iyi bir seçim olabilir.

Aşağıda denenmesi istenilen parametreler sözlük yapısı şeklinde gösterilmiştir. Ne kadar çok parametre gönderilirse o kadar işlem hacmi büyümektedir. Çünkü gönderilen her bir parametre bir birleri ile tek tek denenerek modeller oluşturulmaktadır.

```
param_grid = {   'epochs': [50,100,150], 'batch_size':[32,50,100],  'optimizer':['RMSprop', 'Adam','SGD'], }
```

Daha sonra model KerasClassifier ile oluşturularak. Grid Search Cross Validation yapısına gönderilir. Gönderilen parametreler ile Grid Search'te 5 katlı cross validation işlemi uygulandı.

```
grid = GridSearchCV(estimator=model_cv, n_jobs=-1, verbose=1, cv=5, param_grid=param_grid)
```

Model parametreler ile tek tek train seti üzerinden fit edildikten sonra. Grid Search'ün bir fonksiyonu olan best_params_ ile denenen parametrelerden en iyi olan parametreler gözlemlenebilmektedir. Yine bir başka fonksiyon olan best_estimator_ ise en iyi modeli vermektedir. best_estimator_ , en iyi parametrelerle kurulan en iyi sonucu veren modeli ifade eder. Grid Search'e gönderilen parametreler ile 27 parametre deneme işlemi her biri için 5 katlı cross validation yapılarak, toplamda 135 fit gerçekleştirilerek modelimiz için en iyi parametreler şu şekilde oluşmuştur:

En iyi parametreler: 'batch_size': 50, 'epochs': 150, 'optimizer': 'Adam'

Bir sonraki aşamada K-fold işlemi yapılarak cross validation işlemi hem accuracy hemde f1_weighted skoru için yapılır. K-fold işlemi Grid Search'te belirlenen en iyi model üzerine uygulanmaktadır. K-fold ile veri seti 5 farklı alt kümeye bölünerek cross validation işlemi yapılır. Verisetinde bazı kısımlar modeli çok iyi ifade ederken bazı kısımlar tam olarak ifade etmez. Yani bazı kısımlarda modelin doğruluk oranı yüksek olabilirken bazı kısımlarda ise modelin doğruluk oranı düşük olabilmektedir. Amacımız modelimizin data setinin her yerinde benzer sonuçlar verip vermediğini tespit ederek, modelin tüm veri setinde düzgün çalışıp çalışmadığını kontrol etmektir. Yani dışarıdan modelin hiç görmediği bir veri seti geldiğinde modelin iyi çalışmasını sağlamaktır. K-fold ile modeli farklı bölümlere ayırarak model için farklı bölümlerdeki değerlendirme skorları elde edilerek modelin doğrulanması sağlanmış olur. Skorlama için kullanılan metrik olarak f1_weighted ve accuracy kullanılmıştır. Test seti üzerinden 5 farklı f1_weighted ve accuracy skoru elde edilmiştir. Daha sonra bu skorların ortalaması alınarak modelin valide edilmiş(doğrulanmış) skorlar elde edilmiş olur.

Daha sonra Grid Search ile belirlenen en iyi model ile model tahmin işlemi yapılır. f1_weighted,accuracy skorları tune edilmiş model için hesaplanır. Sonuçlardan da görüleceği gibi tune(ayarlanmış) edilmiş model ile alınan

skorlar tune edilmemiş model ile alınan skorlara göre daha iyi olduğu görülmektedir. Aşağıdaki tabloda tune edilmiş, tune edilmemiş ve K-fold ile alınan skorlar gösterilmiştir.

K-fold Cross Validation Accuracy Sonuçları: [0.88372093 0.90588235 0.87058824 0.89411765 0.81176471]

K-fold Cross Validation Accuracy Sonuçlarının Ortalaması: 0.8732147742818057

K-fold Cross Validation f1_weighted Sonuçları: [0.88235709 0.94141427 0.8982699 0.87373411 0.82106618]

K-fold Cross Validation f1_weighted Sonuçlarının Ortalaması: 0.8833683094688147

	f1_weighted	accuracy
Tune Edilmemiş(İlkel Model)	0.8898234437932919	0.8849765258215962
K-fold Cross Validation Ortalaması	0.8833683094688147	0.8732147742818057
Tune Edilmiş(En İyi Modelle)	0.9225352112676056	0.9230346636719856

Modele K-fold Cross Validation uygulanmış hali ile elde edilen skorlarda bize gösteriyor ki model veri seti farklı alt kümeye bölünmesi ile farklı alt kümelerde farklı sonuçlar vermektedir. K-fold'un bize daha güvenilir sonuçlar verdiği açık bir şekilde gözlemlenmiş oldu. Model veri setinin bazı alt kümelerinde diğerlerine göre nispeten daha az tahmin başarısı elde etmiştir.

Sonraki aşamada tune edilmiş model için classification report ve confusion matrix hesaplamaları yapıldı. Classification report çeşitli skorlama türlerinin sınıflara göre değerlerini bir arada veren bir raporlama şeklidir. Aşağıda classification report ile alınan değer gösterilmiştir. Classification Report'ta elde edilen sonuçlar ileride confusion matrix ile hesaplamalı bir şekilde ayrıntılı olarak anlatılacaktır.

	precision	**recall**	**f1-score**	**support**
0	0.96	0.96	0.96	326
1	0.72	0.81	0.76	58
2	0.94	0.76	0.84	42
accuracy			0.92	426
macro avg	0.87	0.84	0.85	426
weighted avg	0.93	0.92	0.92	426

Accuracy = TP+TN/TP+FP+FN+TN **Precision** = TP/TP+FP **Recall** = TP/TP+FN

F1 Score = 2*(Recall * Precision) / (Recall + Precision)

Şimdi modelimiz için oluşan confusion matrixini yorumlayalım:

	Class 0 Predicted	**Class 1 Predicted**	**Class 2 Predicted**	**Toplam Support**
Class 0 Actual	314	10	2	326
Class 1 Actual	11	47	0	58
Class 2 Actual	2	8	32	42

Class 0: Normal **Class 1**: Suspect **Class 3**:Pathological

314 hasta test datasında gerçekte Normal, tahmin sonucunda da Normal (Doğru Sınıflandırma)

47 hasta test datasında gerçekte Suspect, tahmin sonucunda da Suspect (Doğru Sınıflandırma)

32 hasta test datasında gerçekte Pathological, tahmin sonucunda da Pathological (Doğru Sınıflandırma)

10 hasta test datasında gerçekte Normal, tahmin sonucunda da Suspect (Yanlış Sınıflandırma)

2 hasta test datasında gerçekte Normal, tahmin sonucunda da Pathological (Yanlış Sınıflandırma)

11 hasta test datasında gerçekte Suspect, tahmin sonucunda da Normal (Yanlış Sınıflandırma)

2 hasta test datasında gerçekte Pathological, tahmin sonucunda da Normal (Yanlış Sınıflandırma)

8 hasta test datasında gerçekte Pathological, tahmin sonucunda da Suspect (Yanlış Sınıflandırma)

** Modelin yanlış sınıflandırmalarından en yüksek olanı 11 ile gerçekte Suspect olduğu halde modelimiz bunu Normal olarak tahmin etmiştir. Bu da classification reporttaki 1. Sınıfa ait skorların düşüklüğünü açıklamaktadır. Model sınıfları dengesiz bir şekilde dağıldığı için sınıflara göre oluşan skorlarda sayısı az olan sınıfların skorlarının düşük olmasına sebep olmaktadır. Datasetteki 0. Sınıf sayısı çok olduğu için diğer sınıflara göre başarı oranı bir hayli yüksek çıkmaktadır. Bu problemin önüne geçmek için her ne kadar sınıflara ağırlık vermiş olsamda yine de bazı skorlar diğerlerine göre düşük olmuştur.

Şimdi de modelin confusion matrixine göre classification reporttaki değerleri hesaplayalım:

Toplam Doğru Sınıflandırma Sayısı:(314+47+32)=393

Toplam Popülasyon Sayısı: (314+10+2+11+47+0+2+8+32)=426

Accuracy=393/426=0.9225≅ 0.92 yani modelimiz %82 accuracy değerine sahip.

Sınıf 0 için:

Precision= 314/(314+11+2)=0.9602≅0.96

Recall=314/(314+10+2)=0.9631 ≅0.96

f1=2*(0.96*0.96)/(0.96+0.96)=0.96

Sınıf 1 için:

Precision= 47/(47+10+8)=0.7230≅0.72

Recall=47/(47+11+0)=0.8103 ≅0.81

f1=2*(0.72*0.81)/(0.72+0.81)=0.7623≅0.76

Sınıf 2 için:

Precision= 32/(32+2+0)=0.9411≅0.94

Recall=32/(32+2+8)=0.7619 ≅0.76

f1=2*(0.76*0.94)/(0.76+0.94)=0.8404≅0.84

Son olarak precision,recall,f1 skorları için macro ve weighted şeklinde hesaplamalar yapılacaktır. Bu skorların macro değerleri hesaplanırken tüm sınıflara göre değerleri toplanarak toplam sınıf sayısına bölünerek bulunur. Weighted değerleri ise her sınıfın skor değeri her bir sınıfın gerçek toplam sayıları ile çarpılarak yani support değerleri ile çarpılarak toplanır ve toplam popülasyon sayısına(toplam nüfus) bölünerek bulunur.

f1_weighted=(0.96*326+0.76*58+0.84*42)/426=0.9209≅0.92

f1_macro=(0.96+0.76+0.84)/3=0.8533≅0.85

precision_weighted=(0.96*326+0.72*58+0.94*42)/ 426=0.9253≅0.93

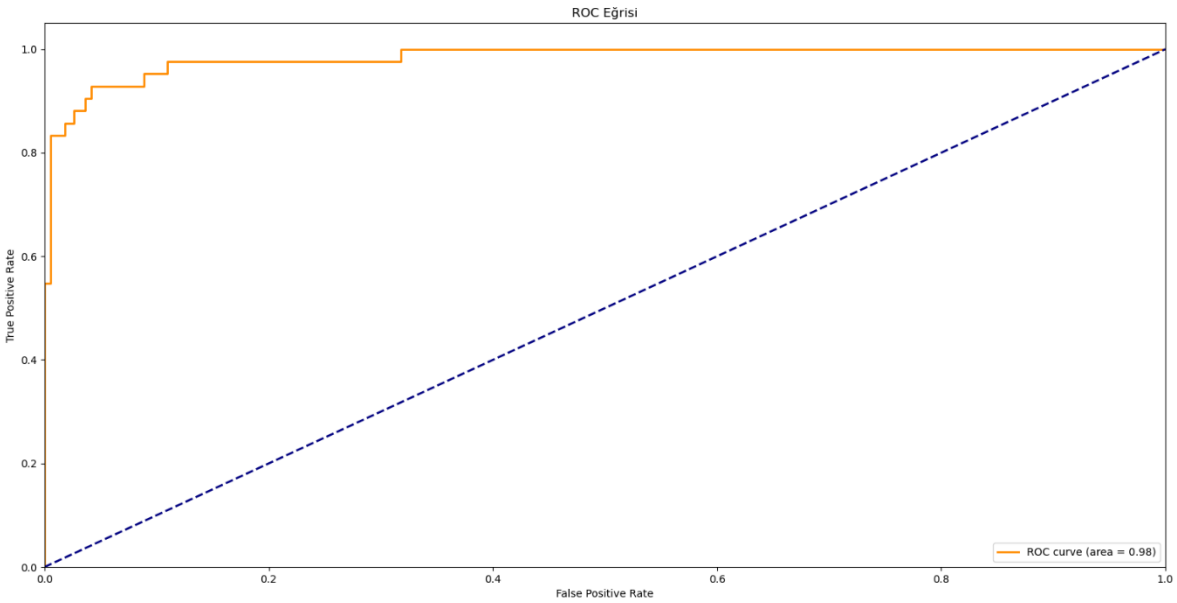
precision_macro=(0.96+0.72+0.94)/3=0.8733≅0.87

recall_weighted=(0.96*326+0.81*58+0.76*42)/ 426=0.9198≅0.92

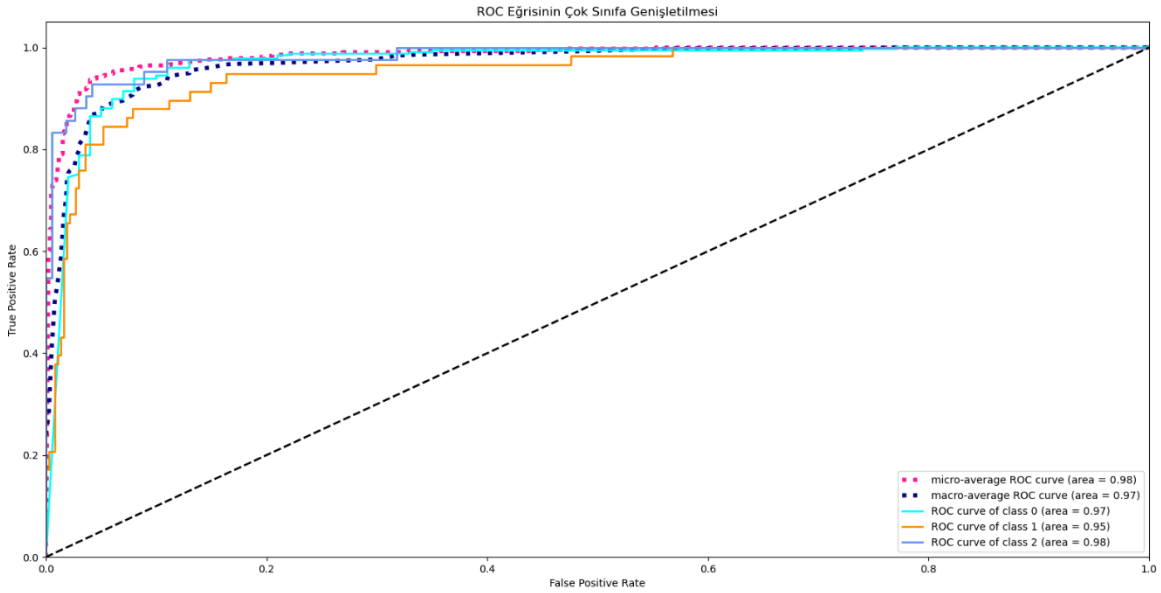
recall_macro=(0.96+0.81+0.76)/3=0.8433≅0.84

Confusion matrixine göre yapılan tüm hesaplamalar görüleceği gibi classification reporttaki değerler ile bire bir örtüşmektedir. Özellikle sınıf sayılarında dengesizlik olan sınıflandırma problemlerinde sınıflar özelindeki skorlardan ziyade weighted(ağırlıklı) skorlara bakmak daha mantıklı olacaktır. Çünkü sınıflar özelindeki skorlar sınıf dengesizliğinden dolayı düşük olabilmektedir.

Son olarak model için ReceiverOperating Characteristic Curve (Alıcı çalışma karakteristik eğrisi) ve Area Under the Curve(Eğri Altındaki Alan) uygulaması yapılmıştır. Modelimizde 0, 1 ve 2 adında üç sınıfınız var , 1 ve 2'ye karşı sınıflandırılmış 0 için bir ROC, 0 ve 3'e karşı sınıflandırılmış 2 için başka bir ROC ve 2 ve 0'a göre sınıflandırılmış 3 için üçüncü bir ROC eğrisine sahip olmaktayız. Model için ROC eğrileri ve AUC değerleri hesaplaması öncelikle gözlemlemek adına kendi seçtiğimiz herhangi bir sınıf özelinde yapıldı. Aşağıdaki grafik sınıf 2'ye ait ROC eğrisini göstermektedir.



Model sınıf 2 için AUC değeri 0.98 gibi yüksek bir orana sahip. Bunun anlamı modelin sınıf ayrımı yaparken 2 sınıf için başarısı %98 oranında olduğu anlamına gelmektedir. Daha sonra tüm sınıfların ROC eğrileri ve her bir sınıfın AUC değerleri ve macro-micro ortalama ROC eğrileri ve AUC değerleri hesaplanarak çizdirilmiştir.



Model için elde edilen AUC değerleri; micro-average ROC eğrisi için 0.98, macro-average ROC eğrisi için 0.97, class 0 ROC eğrisi için(Normal) 0.97, class 1 ROC eğrisi için(Suspect) 0.95 ve class 2 ROC eğrisi için (Pathological) 0.98 şeklinde gözlemlenmiştir. Sonuçlardan da görüleceği gibi modelimiz sınıf ayrımını yapma doğruluğu konusunda class 1 için diğerlerine göre biraz daha düşük sonuç ortaya koymuştur. Bunu zaten confusion matrisinde class 1 için olan değerlerde tespit etmiştik. Genel olarak modelimiz sınıf ayrımını yüksek bir başarı ile yapabilmektedir. Bu oranlar ne kadar yüksek olursa modelin tahminleri yaparken hata yapma olasılığı o kadar düşük olmaktadır. ROC eğrileri bu konuda bize bu önemli bilgileri vermektedir.

2. Moleküler Biyoloji (Splice-junction Gene Sequences) Dataseti

Dataset Kaynağı: https://www.openml.org/d/40670 **Dataset Gözlem Sayısı:** 3186

Dataset Tanıtımı:

Ekleme birleşimleri, daha yüksek organizmalarda protein oluşturma işlemi sırasında `` gereksiz '' DNA'nın çıkarıldığı bir DNA dizisi üzerindeki noktalardır. Bu veri kümesinde ortaya çıkan sorun, bir DNA dizisi verildiğinde, eksonlar (DNA dizisinin eklendikten sonra tutulan kısımları) ve intronlar (DNA dizisinin eklenmiş kısımları) arasındaki sınırları tanımaktır. Bu sorun iki alt görevden oluşur: ekson / intron sınırlarını tanıma (EI yerleri olarak adlandırılır) ve intron / ekson sınırlarını (IE yerleri) tanımlama. (Biyolojik toplulukta, IE sınırları ``alıcı '', EI sınırları `` bağışçı '' olarak adlandırılır). Veri noktaları 180 gösterge ikili değişkeni ile tanımlanır ve sorun 3 sınıfı (EI, IE, Neither) tanımak, yani eksonlar (DNA dizisinin eklendikten sonra tutulan kısımları) ve intronlar (eklenmiş DNA dizisi).

StatLog DNA dataseti, Irvine veritabanının işlenmiş bir sürümüdür. Temel fark, nükleotitleri temsil eden sembolik değişkenlerin (sadece A, G, T, C) 3 binary gösterge değişkeniyle değiştirilmesidir. Böylece orijinal 60 sembolik özellik 180 ikili özellik olarak değiştirildi. Örneklerin isimleri kaldırıldı. Belirsizliği olan örnekler kaldırıldı (çok azı, 4 tanesi). Bu veri kümesinin StatLog sürümü, Strathclyde Üniversitesi'nde Ross King tarafından üretildi. Orijinal ayrıntılar için Irvine veritabanı belgelerine bakılmalıdır.

A, C, G, T nükleotidlerine aşağıdaki şekilde gösterge değerleri verildi:

A -> 1 0 0 C -> 0 1 0 G -> 0 0 1 T -> 0 0 0 şeklinde 3'lü binary kombinasyonlarını göstermektedir.

Orijinal datasetteki 60 değişken yani 60 nükleotit 3'lü binary kombinasyonları ile 180 değişken elde edilmiştir.

Bağımlı değişkeni 3 sınıflı bir yapıdadır.

EI:Ekson Intron : EI sınırları "bağışçır" olarak datasette :1

IE:Intron Ekson: IE sınırları "alıcı" olarak datasette:2

Neither: Hiçbiri olarak datasette:3

0. İndex için A0,A1,A2:C A3,A4,A5:T

6. İndex için A0,A1,A2:G A3,A4,A5:A

10. İndex için A0,A1,A2:C A3,A4,A5:A şeklinde daha iyi anlaşılması için 3 sınıf içinde çeşitli indexlerde örnekle açıklandı. Tablo olarak görünümü ise aşağıdaki gibidir.

İndex	**A0**	**A1**	**A2**	**A3**	**A4**	**A5**	**…. A179**	**class**
0.	0	1	0	0	0	0	0	3:Neither
6.	0	0	1	1	0	0	1	1:EI
10.	0	1	0	1	0	0	0	2:IE

Veri Ön İşleme:

Öncelikle verisetindeki eksik gözlem olup olmadığı kontrol edildi. Kontrol sonucunda verisetinde herhangi bir eksik gözlem olmadığı görüldü. Veri setindeki değişkenlerin tipleri kontrol edilerek bu teğit edildi. Çünkü bazen eksik gözlem olmasına rağmen eksik gözlem .nan şeklinde olmadığı için(örneğin: '?' şeklinde) bu eksik gözlem olarak algılanmaz. Bunu değişken tiplerine bakarak anlayabiliriz. Eğer sayısal bir değişken olması gerekirken object tipinde görünürse burada eksik gözlem ya da olmaması gereken karakterler olduğu ve bu gözlemlerin düzenlenmesi gerektiğini gösterir. Veri seti 3 outputlu bir yapıya sahiptir(1,2,3 şeklinde). Öncelikle 3 outputlu yapı için label encoder işlemi yapılarak bağımlı değişkeni 0,1,2 şeklinde değerlere dönüştürüldü. Daha sonra np_utils.to_categorical ile bağımlı değişkenin formatı 0 ve 1'lerden oluşan 3 boyutlu yapıya dönüştürüldü.

Bağımlı değişken y'nin son hali → $\begin{matrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{matrix}$

Verisetindeki bağımsız değişkenler zaten 0,1 şeklinde binary formda olduğu için bunlar için herhangibir veri ölçeklemeye ihtiyaç duyulmamaktadır. Artık verimiz modellerin kurulması için hazır bir haldedir.

2.1. MLP Classifier Modeli

Multi Layer Perceptron (MLP) Classifier sklearn kütüphanesinde bulunan bir çok katmanlı algılayıcıdır. İlk olarak MLP ile model nesnesi oluşturuldu. Daha sonra model nesnesi train seti üzerinden fit edildi. Öncelikle valide edilmemiş(doğrulanmamış,ilkel) test skorlarını belirlemek istedim. Bunun için model tahmin işlemi yapılarak, "accuracy" ve "f1_weighted" skorlarını hesapladım. Burada f1 yerine f1_weigted kullanmamızın sebebi hedef değişkeninin(bağımlı değişkenin) multiclass olmasından dolayı f1 score hesaplarken average argümanına [None, 'micro', 'macro', 'weighted'] bunlardan bir tanesini kullanmak gerekmektedir aksi takdirde hata vermektedir. Bu bölümde model için hiçbir parametre girmeden yani model tune(ayar) edilmeden "accuracy" ve "f1_weighted" skorları hesaplandı. Bu skorlar nispeten tune edilmiş modele göre düşük olacaktır. Ayrıca model hedef(bağımlı) değişkeni multi-class(çok sınıflı) olduğu için model tahmin ve skorlama işlemlerinde bağımlı değişkene ait kısımlarda numpy methodu olan np.argmax ile eski haline getirilerek işlemler yapılmıştır. Model için herhangi bir katman bilgisi ya da diğer parametrelerden hiçbirisi girilmeden model oluşturuldu. Her defasında farklı sonuçlar vermemesi açısından tune edilmemiş model için de tune edilmiş model içinde random_state aynı ayarlandı. Bu bölümden sonra Grid Search yapısı ile MLP modelimizin parametre optimizasyonu için çeşitli parametreler modele denenecek ve en uygun parametreler ile model oluşturulacaktır.

Grid search ile bir sözlük yapısı vasıtası ile modele optimizasyon için gönderilen parametreler ile açıklamalar daha önceki bölümde yapıldığı için bu bölümde bu parametreler hakkında bilgi verilmeyecektir.

Aşağıda denenmesi istenilen parametreler sözlük yapısı şeklinde gösterilmiştir. Ne kadar çok parametre gönderilirse o kadar işlem hacmi büyümektedir. Çünkü gönderilen her bir parametre bir birleri ile tek tek denenerek modeller oluşturulmaktadır.

```
mlpc_params = {"alpha": [0.1, 0.01, 0.0001],

    "hidden_layer_sizes": [(10,10,10), (100,100,100), (100,100)],

     "solver" : ["lbfgs","adam","sgd"],    "activation": ["relu","logistic"]}
```

Daha sonra model için bu parametreler Grid Search'te 5 katlı cross validation işlemi uygulandı.

```
mlpc_cv_model = GridSearchCV(mlpc, mlpc_params, cv = 5,  n_jobs = -1, verbose = 2)
```

Model parametreler ile tek tek train seti üzerinden fit edildikten sonra. Grid Search'ün bir fonksiyonu olan best_params_ ile denenen parametrelerden en iyi olan parametreler gözlemlenebilmektedir. Yine bir başka fonksiyon olan best_estimator_ ise en iyi modeli vermektedir. best_estimator_ , en iyi parametrelerle kurulan en iyi sonucu veren modeli ifade eder. Grid Search'e gönderilen parametreler ile 54 parametre deneme işlemi her biri için 5 katlı cross validation yapılarak, toplamda 270 fit gerçekleştirilerek modelimiz için en iyi parametreler şu şekilde oluşmuştur:

```
En iyi parametreler: {'activation': 'logistic', 'alpha': 0.1, 'hidden_layer_sizes': (10, 10, 10), 'solver': 'lbfgs'}
```

Bir sonraki aşamada K-fold işlemi yapılarak cross validation işlemi hem accuracy hemde f1_weighted skoru için yapılır. K-fold işlemi Grid Search'te belirlenen en iyi model üzerine uygulanmaktadır. K-fold ile veri seti 5 farklı alt kümeye bölünerek cross validation işlemi yapılır. Verisetinde bazı kısımlar modeli çok iyi ifade ederken bazı kısımlar tam olarak ifade etmez. Yani bazı kısımlarda modelin doğruluk oranı yüksek olabilirken bazı kısımlarda ise modelin doğruluk oranı düşük olabilmektedir. Amacımız modelimizin data setinin her yerinde benzer sonuçlar verip vermediğini tespit ederek, modelin tüm veri setinde düzgün çalışıp çalışmadığını kontrol etmektir. Yani dışarıdan modelin hiç görmediği bir veri seti geldiğinde modelin iyi çalışmasını sağlamaktır. K-fold ile modeli farklı bölümlere ayırarak model için farklı bölümlerdeki değerlendirme skorları elde edilerek modelin doğrulanması sağlanmış olur. Skorlama için kullanılan metrik olarak f1_weighted ve accuracy kullanılmıştır. Test seti üzerinden 5 farklı f1_weighted ve accuracy skoru elde edilmiştir. Daha sonra bu skorların ortalaması alınarak modelin valide edilmiş(doğrulanmış) skorlar elde edilmiş olur.

Daha sonra Grid Search ile belirlenen en iyi model ile model tahmin işlemi yapılır. f1_weighted,accuracy skorları tune edilmiş model için hesaplanır. Sonuçlardan da görüleceği gibi tune(ayarlanmış) edilmiş model ile alınan skorlar tune edilmemiş model ile alınan skorlara göre daha iyi olduğu görülmektedir. Aşağıdaki tabloda tune edilmiş, tune edilmemiş ve K-fold ile alınan skorlar gösterilmiştir.

K-fold Cross Validation f1_weigted Sonuçları: [0.89644986 0.93728441 0.92150848 0.93271709 0.9272748]

K-fold Cross Validation f1_weigted Sonuçlarının Ortalaması: 0.9230469279533569

K-fold Cross Validation accuracy Sonuçları: [0.91666667 0.91623037 0.93193717 0.92146597 0.91623037]

K-fold Cross Validation accuracy Sonuçlarının Ortalaması: 0.9205061082024433

	f1_weighted	accuracy
Tune Edilmemiş(İlkel Model)	0.9389470932760243	0.9184100418410042
K-fold Cross Validation Ortalaması	0.9230469279533569	0.9205061082024433
Tune Edilmiş(En İyi Modelle)	0.9446167860293517	0.944560669456067

Yukarıdaki değerlerde görüldüğü gibi modelin tune edilmemiş skorları tune edilmiş yani Grid Search ile belirlenen en iyi model ile elde edilmiş skorlara göre düşüktür. Nitekim bunun böyle olmasını bekliyorduk. Modele K-fold Cross Validation uygulanmış hali ile elde edilen skorlarda bize gösteriyor ki model veri seti farklı alt kümeye bölünmesi ile farklı alt kümelerde farklı sonuçlar vermektedir. K-fold'un bize daha güvenilir sonuçlar verdiği açık bir şekilde gözlemlenmiş oldu. Model veri setinin bazı alt kümelerinde diğerlerine göre nispeten daha az tahmin başarısı elde etmiştir.

Sonraki aşamada tune edilmiş model için classification report ve confusion matrix hesaplamaları yapıldı. Classification report çeşitli skorlama türlerinin sınıflara göre değerlerini bir arada veren bir raporlama şeklidir.

Aşağıda classification report ile alınan değer gösterilmiştir.

	precision	**recall**	**f1-score**	**support**
0	0.94	0.94	0.94	248
1	0.90	0.91	0.91	229
2	0.96	0.96	0.96	479
accuracy			0.94	956
macro avg	0.94	0.94	0.94	956
weighted avg	0.94	0.94	0.94	956

Confusion matrix ise sınıflandırma problemine ilişkin tahmin sonuçlarının bir özetidir. Doğru ve yanlış tahminlerin sayısı sayım değerleri ile özetlenir ve her sınıf tarafından ayrılır. Confusion matrixi sınıflandırma modelinizin tahminlerde ne zaman karıştırıldığını gösterir. Bize sadece bir sınıflandırıcı tarafından yapılan hatalar hakkında değil, daha da önemlisi yapılan hata türleri hakkında bilgi verir.

Accuracy = TP+TN/TP+FP+FN+TN **Precision** = TP/TP+FP **Recall** = TP/TP+FN

F1 Score = 2*(Recall * Precision) / (Recall + Precision)

Şimdi modelimiz için oluşan confusion matrixini yorumlayalım:

	Class 0 Predicted	**Class 1 Predicted**	**Class 2 Predicted**	**Toplam Support**
Class 0 Actual	234	6	8	248
Class 1 Actual	11	209	9	229
Class 2 Actual	3	16	460	479

Class 0: EI **Class 1**: IE **Class 2**:Neither

234 DNA dizisi test datasında gerçekte EI, tahmin sonucunda da EI (Doğru Sınıflandırma)

209 DNA dizisi test datasında gerçekte IE, tahmin sonucunda da IE (Doğru Sınıflandırma)

460 DNA dizisi test datasında gerçekte Neither, tahmin sonucunda da Neither (Doğru Sınıflandırma)

6 DNA dizisi test datasında gerçekte EI, tahmin sonucunda da IE (Yanlış Sınıflandırma)

8 DNA dizisi test datasında gerçekte EI, tahmin sonucunda da Neither (Yanlış Sınıflandırma)

11 DNA dizisi test datasında gerçekte IE, tahmin sonucunda da EI (Yanlış Sınıflandırma)

9 DNA dizisi test datasında gerçekte IE, tahmin sonucunda da Neither (Yanlış Sınıflandırma)

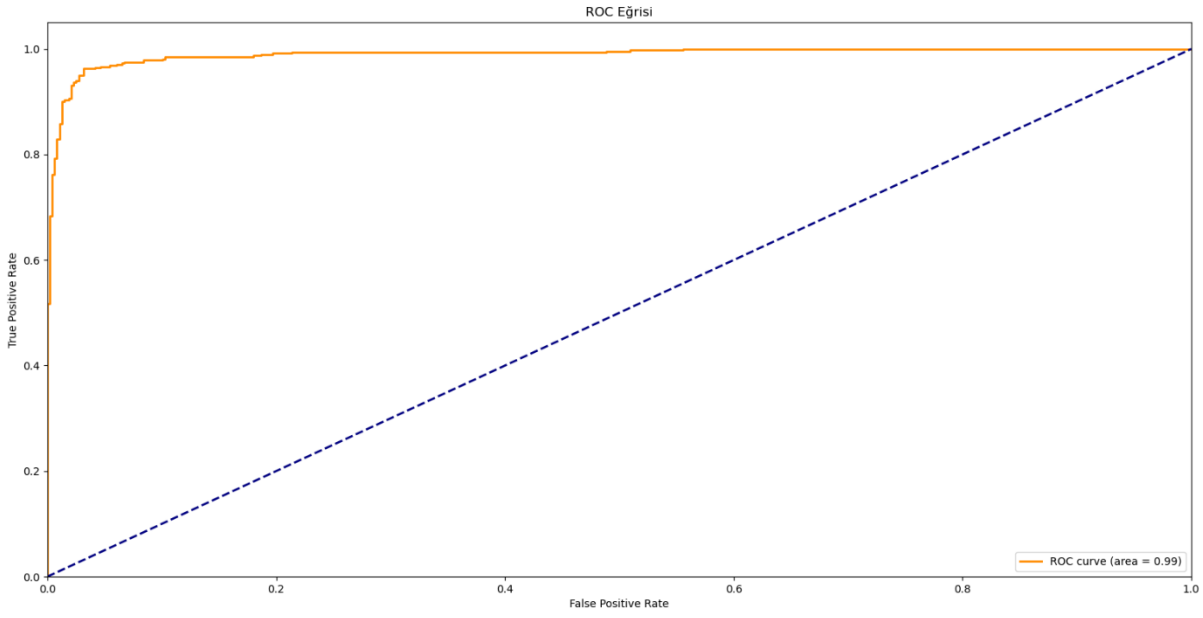
3 DNA dizisi test datasında gerçekte Neither, tahmin sonucunda da EI (Yanlış Sınıflandırma)

16 DNA dizisi test datasında gerçekte Neither, tahmin sonucunda da IE (Yanlış Sınıflandırma)

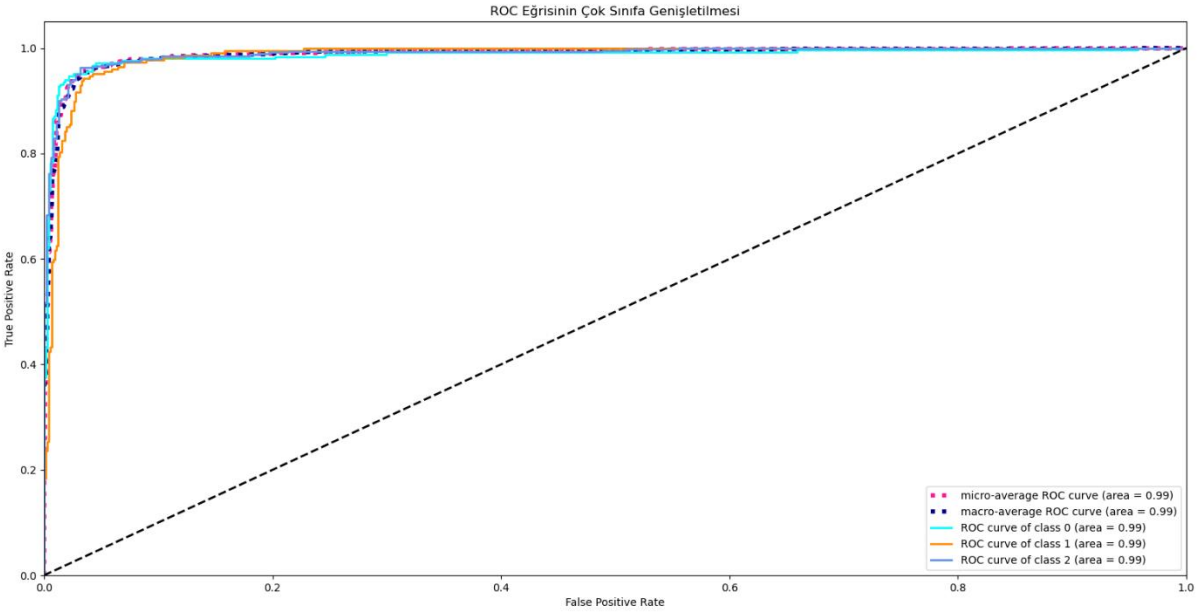
** Modelin yanlış sınıflandırmalarından en yüksek olanı 16 ile gerçekte Neither olduğu halde modelimiz bunu IE olarak tahmin etmiştir.

Modelin confusion matrixine göre classification reporttaki skorların hesaplanması daha önceden anlatıldığı gibi yapılmaktadır o yüzden burada tekrardan bir hesaplama yapılmayacaktır. Hesaplamalar yapıldığında classification reporttaki skorlar ile aynı olduğu gözlemlenir. Genel olarak modelimizin sınıf tahmininde oldukça başarılı olduğu saptanmıştır. Bu sonucun ROC eğrilerinde ve AUC değerlerinde de gözlemlenmesi beklenmektedir.

Son olarak model için ReceiverOperating Characteristic Curve (Alıcı çalışma karakteristik eğrisi) ve Area Under the Curve(Eğri Altındaki Alan) uygulaması yapılmıştır.



Model sınıf 2 için AUC değeri 0.99 gibi yüksek bir orana sahip. Bunun anlamı modelin sınıf ayrımı yaparken 2 sınıf için başarısı %99 oranında olduğu anlamına gelmektedir. Daha sonra tüm sınıfların ROC eğrileri ve her bir sınıfın AUC değerleri ve macro-micro ortalama ROC eğrileri ve AUC değerleri hesaplanarak çizdirilmiştir.



Model için elde edilen AUC değerleri; micro-average ROC eğrisi için 0.99, macro-average ROC eğrisi için 0.99, class 0 ROC eğrisi için(EI) 0.99, class 1 ROC eğrisi için(IE) 0.99 ve class 2 ROC eğrisi için(Neither) 0.99 şeklinde gözlemlenmiştir. Sonuçlardan da görüleceği gibi modelimiz sınıf ayrımını çok yüksek bir başarı ile yapabilmektedir. Bu oranlar ne kadar yüksek olursa modelin tahminleri yaparken hata yapma olasılığı o kadar düşük olmaktadır. ROC eğrileri bu konuda bize bu önemli bilgileri vermektedir.

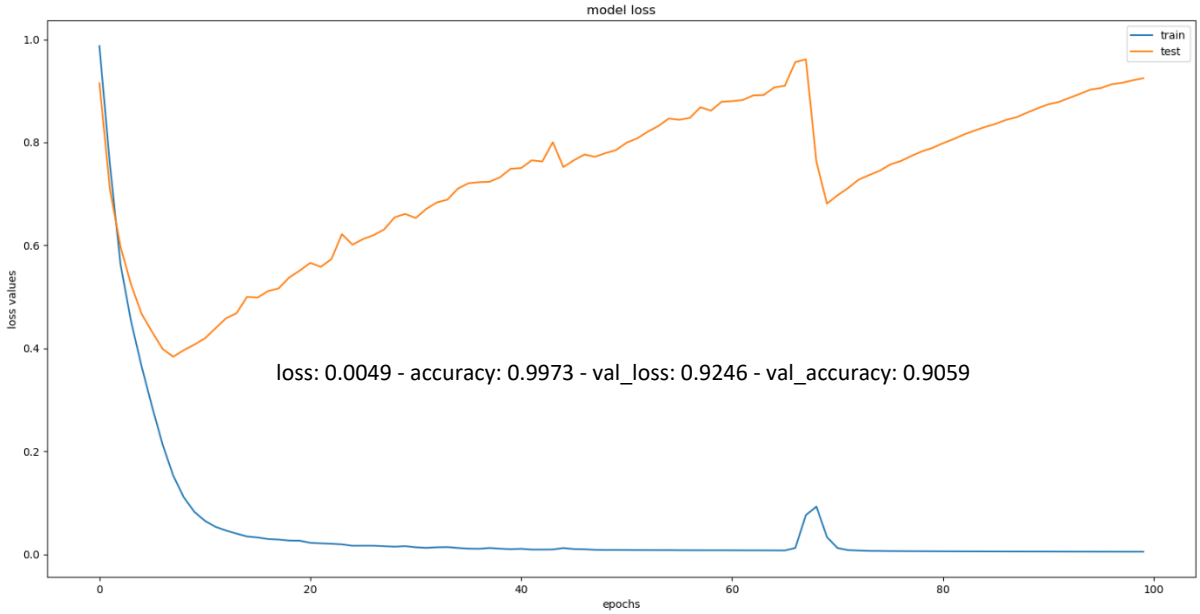
2.2. Keras Modeli

Multi Layer Perceptron için Keras Python kütüphanesi, modellerin bir katman dizisi olarak oluşturulmasına odaklanır. Keras ile model için her bir katman tek tek oluşturulabilir ve her bir katman özelinde ayarlamalar yapılmasına olanak sağladığı için daha esnek bir yapı ortaya koymaktadır. Model katmanları tek tek oluşturulduğu için her bir katmandaki nöron sayısı Dense fonksiyonu ile belirtilmelidir. Giriş katmanı için veri setindeki bağımsız değişken sayısı(girdi boyutu) verilmelidir. Data setimizde 180 tane bağımsız değişkenimiz olduğu için input_dim=180 olarak belirtildi. Daha sonra her bir katman için aktivasyon fonksiyonu da her bir katman özelinde ayarlanabildiği için bunun da belirtilmesi gerekmektedir. Keras için çıktı katmanın nöron sayısının da(sınıf sayısının) belirtilmesi gerekmektedir. Ayrıca Keras modelinde sınıf sayısı 2'den fazla ise çıktı katmanının aktivasyon fonksiyonu "softmax" olarak belirtilmelidir. MLPClassifier'da ise bunları belirtmemize gerek yok. Keras modelini compile(derleme) ederken loss(kayıp) fonksiyonunu belirtmemiz gerekmektedir. MLPClassifier için loss fonksiyonunu belirtmeye gerek yoktur kendisi otomatik olarak seçer.Hedef değişkeni {0,1,2..,n} şeklinde her sınıfa benzersiz bir tamsayı değerine sahip olduğu çok sınıflı sınıflandırma problemlerinde 'categorical_crossentropy' kayıp fonksiyonu kullanılır. Keras modelimiz 3 sınıflı hedef değişkenine sahip olduğu için modeli compile ederken loss fonksiyonu 'categorical_crossentropy' olarak seçildi. Modeli ilk başta kurarken optimizer için "adam" olarak compile ettim. Modelin metrics argümanını "accuracy" olarak belirledim. Modeli bir fonksiyon şeklinde tanımlayıp kurduktan sonra modelin loss fonksiyonunun değerinin epoch sayısına göre değişimini train ve test seti özelinde gözlemlemek için train seti ile eğittim. Burada epochs olarak 100 , batch_size olarak ise genel olarak uygulamalarda kullanılan bir değer olan 32 olarak belirledim. Daha önceki bölümde batch_size argümanı için detaylı bir şekilde açıklamalar ve mevcut yaklaşımlar hakkında bilgiler verildiği için burada anlatılmadı.

İlk olarak modeli kötü bir şekilde kuralım ve train kaybı ve test kaybını gözlemleyelim.

```
def create_model(optimizer="adam"):
    model = Sequential()
    model.add(Dense(7, input_dim=180, activation='relu'))
    model.add(Dense(27, activation='relu'))
```

```
    model.add(Dense(25, activation='relu'))
    model.add(Dense(5, activation='relu'))
    model.add(Dense(3, activation='softmax'))
    model.compile(loss='categorical_crossentropy', optimizer=optimizer,metrics=["accuracy"])
    return model
model = create_model() # tune edilmemiş model nesnesini oluşturma
egitim=model.fit(X_train, y_train, epochs=100, batch_size=32, verbose=1,validation_data=(X_test,y_test))
```



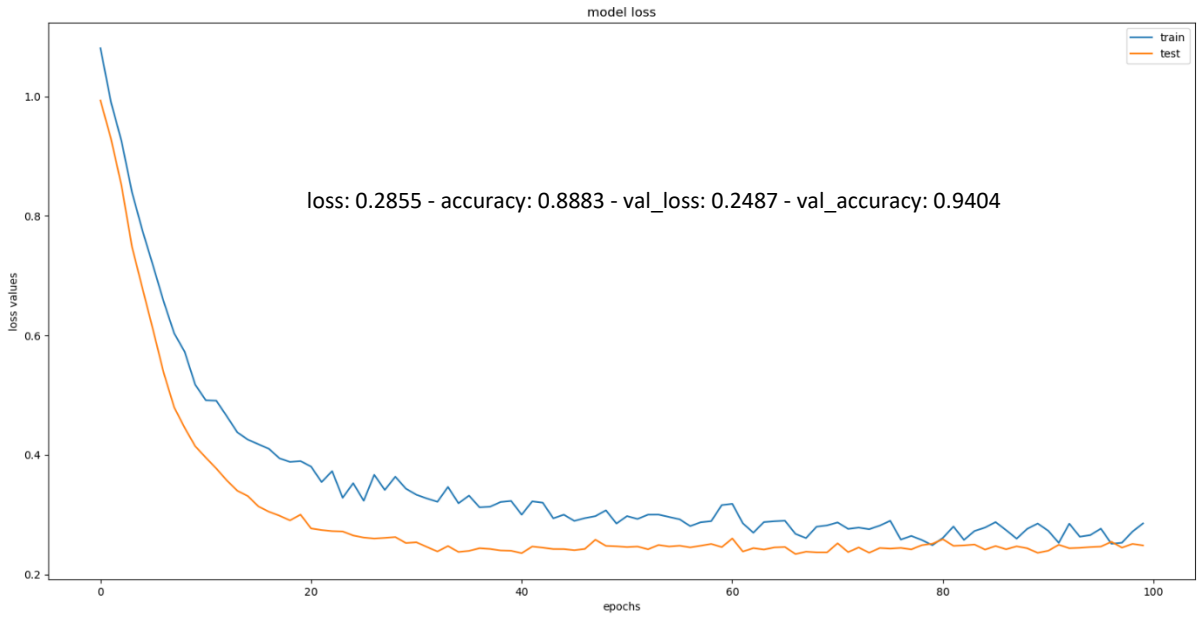
Grafikten de anlaşılacağı üzere oldukça kötü bir model. Modelin test hatası artarken train hatası düşmektedir. Bu da modelin overfittinge doğru gittiğini yani modelin train seti üzerinde yüksek doğruluğa sahip olup test seti üzerinde daha düşük doğruluğa sahip olduğunu göstermektedir. Model train setini çok başarılı bir şekilde tahmin ederken, test setini o başarıda tahmin edememiştir. Şimdi modelimizi biraz geliştirelim overfittingin önüne geçmek için modele Droupout(bırakma) katmanları eklendi. Bırakma katmanı ekleme overfittingi kolay ve etkili bir yolu olabilir . Bir bırakma katmanı, katmanlar arasındaki bazı bağlantıları rastgele bırakır. Bu, aşırı öğrenmeyi önlemeye yardımcı olur. Model katman aktivasyon fonksiyonlarında değişiklik yapıldı ve katmanların nöron sayılarıda düzenlendi.

```
def create_model(optimizer="adam"):
    model = Sequential()
    model.add(Dense(7, input_dim=180, activation='relu'))
    model.add(Dropout(0.3))
    model.add(Dense(41, activation='relu'))
    model.add(Dropout(0.4))
    model.add(Dense(32, activation='relu'))
    model.add(Dropout(0.2))
    model.add(Dense(5, activation='sigmoid'))
    model.add(Dropout(0.3))
    model.add(Dense(3, activation='softmax'))
    model.compile(loss='categorical_crossentropy', optimizer=optimizer,metrics=["accuracy"])
```

```
    return model

model = create_model()

egitim=model.fit(X_train, y_train, epochs=100, batch_size=32, verbose=1,validation_data=(X_test,y_test))
```



Grafikte de görüldüğü gibi modelin test ve train kayıpları birbirine çok yaklaşmıştır. Modelin train setindeki tahmin başarısı ve test setindeki tahmin başarısı da birbirine yakın değerler göstermektedir.

Öncelikle valide edilmemiş(doğrulanmamış,ilkel) test skorlarını belirlemek istedim. Bunun için model tahmin işlemi yapılarak, “accuracy” ve “f1_weighted” skorlarını hesapladım. Burada f1 yerine f1_weigted kullanmamızın sebebi hedef değişkeninin(bağımlı değişkenin) multiclass olmasından dolayı f1 score hesaplarken average argümanına [None, 'micro', 'macro', 'weighted'] bunlardan bir tanesini kullanmak gerekmektedir aksi takdirde hata vermektedir. Bu bölümde model için hiçbir parametre girmeden yani model tune(ayar) edilmeden “accuracy” ve “f1_weighted” skorları hesaplandı. Ayrıca model hedef(bağımlı) değişkeni multi-class(çok sınıflı) olduğu için model tahmin ve skorlama işlemlerinde bağımlı değişkene ait kısımlarda numpy methodu olan np.argmax ile eski haline getirilerek işlemler yapılmıştır. Bu skorlar nispeten tune edilmiş modele göre düşük olacaktır. Bu bölümden sonra Grid Search yapısı ile Keras modelimizin parametre optimizasyonu için çeşitli parametreler modele denenecek ve en uygun parametreler ile model oluşturulacaktır.

Grid search ile bir sözlük yapısı vasıtası ile modele optimizasyon için gönderilen parametreler hakkında daha önceden detaylı bilgi verilmiştir. O yüzden bu bölümde bu parametreler hakkında bilgilere değinilmeyecektir.

Aşağıda denenmesi istenilen parametreler sözlük yapısı şeklinde gösterilmiştir. Ne kadar çok parametre gönderilirse o kadar işlem hacmi büyümektedir. Çünkü gönderilen her bir parametre bir birleri ile tek tek denenerek modeller oluşturulmaktadır.

```
param_grid = {   'epochs': [100,150,200],

   'batch_size':[50,100],  'optimizer':['RMSprop', 'Adam','SGD'], }
```

Daha sonra model KerasClassifier ile oluşturularak. Grid Search Cross Validation yapısına gönderilir. Gönderilen parametreler ile Grid Search’te 5 katlı cross validation işlemi uygulandı.

```
grid = GridSearchCV(estimator=model_cv, n_jobs=-1, verbose=1, cv=5, param_grid=param_grid)
```

Model parametreler ile tek tek train seti üzerinden fit edildikten sonra. Grid Search’ün bir fonksiyonu olan best_params_ ile denenen parametrelerden en iyi olan parametreler gözlemlenebilmektedir. Yine bir başka fonksiyon olan best_estimator_ ise en iyi modeli vermektedir. best_estimator_ , en iyi parametrelerle kurulan en iyi sonucu veren modeli ifade eder. Grid Search’e gönderilen parametreler ile 18 parametre deneme işlemi

her biri için 5 katlı cross validation yapılarak, toplamda 90 fit gerçekleştirilerek modelimiz için en iyi parametreler şu şekilde oluşmuştur:

En iyi parametreler: 'batch_size': 50, 'epochs': 100, 'optimizer': 'Adam'

Bir sonraki aşamada K-fold işlemi yapılarak cross validation işlemi hem accuracy hemde f1_weighted skoru için yapılır. K-fold işlemi Grid Search'te belirlenen en iyi model üzerine uygulanmaktadır. K-fold ile veri seti 5 farklı alt kümeye bölünerek cross validation işlemi yapılır. Verisetinde bazı kısımlar modeli çok iyi ifade ederken bazı kısımlar tam olarak ifade etmez. Yani bazı kısımlarda modelin doğruluk oranı yüksek olabilirken bazı kısımlarda ise modelin doğruluk oranı düşük olabilmektedir. Amacımız modelimizin data setinin her yerinde benzer sonuçlar verip vermediğini tespit ederek, modelin tüm veri setinde düzgün çalışıp çalışmadığını kontrol etmektir. Yani dışarıdan modelin hiç görmediği bir veri seti geldiğinde modelin iyi çalışmasını sağlamaktır. K-fold ile modeli farklı bölümlere ayırarak model için farklı bölümlerdeki değerlendirme skorları elde edilerek modelin doğrulanması sağlanmış olur. Skorlama için kullanılan metrik olarak f1_weighted ve accuracy kullanılmıştır. Test seti üzerinden 5 farklı f1_weighted ve accuracy skoru elde edilmiştir. Daha sonra bu skorların ortalaması alınarak modelin valide edilmiş(doğrulanmış) skorlar elde edilmiş olur.

Daha sonra Grid Search ile belirlenen en iyi model ile model tahmin işlemi yapılır. f1_weighted,accuracy skorları tune edilmiş model için hesaplanır. Sonuçlardan da görüleceği gibi tune(ayarlanmış) edilmiş model ile alınan skorlar tune edilmemiş model ile alınan skorlara göre daha iyi olduğu görülmektedir. Aşağıdaki tabloda tune edilmiş, tune edilmemiş ve K-fold ile alınan skorlar gösterilmiştir.

K-fold Cross Validation f1_weighted Sonuçları: [0.89506414 0.89430731 0.87896309 0.91669499 0.87071713]

K-fold Cross Validation f1_weighted Sonuçlarının Ortalaması: 0.8911493325769275

K-fold Cross Validation Accuracy Sonuçları: [0.82291667 0.90052356 0.84816754 0.88481675 0.76963351]

K-fold Cross Validation Accuracy Sonuçlarının Ortalaması: 0.8452116055846421

	f1_weighted	accuracy
Tune Edilmemiş(İlkel Model)	0.9405680579186984	0.9403765690376569
K-fold Cross Validation Ortalaması	0.8911493325769275	0.8452116055846421
Tune Edilmiş(En İyi Modelle)	0.9414342787958342	0.9414225941422594

Modele K-fold Cross Validation uygulanmış hali ile elde edilen skorlarda bize gösteriyor ki model veri seti farklı alt kümeye bölünmesi ile farklı alt kümelerde farklı sonuçlar vermektedir. K-fold'un bize daha güvenilir sonuçlar verdiği açık bir şekilde gözlemlenmiş oldu. Model veri setinin bazı alt kümelerinde diğerlerine göre nispeten daha az tahmin başarısı elde etmiştir.

Sonraki aşamada tune edilmiş model için classification report ve confusion matrix hesaplamaları yapıldı. Classification report çeşitli skorlama türlerinin sınıflara göre değerlerini bir arada veren bir raporlama şeklidir. Aşağıda classification report ile alınan değer gösterilmiştir.

	precision	**recall**	**f1-score**	**support**
0	0.94	0.94	0.94	248
1	0.91	0.92	0.92	229
2	0.96	0.96	0.96	479
accuracy			0.94	956
macro avg	0.94	0.94	0.94	956
weighted avg	0.94	0.94	0.94	956

Confusion matrix ise sınıflandırma problemine ilişkin tahmin sonuçlarının bir özetidir. Doğru ve yanlış tahminlerin sayısı sayım değerleri ile özetlenir ve her sınıf tarafından ayrılır. Confusion matrixi sınıflandırma modelinizin tahminlerde ne zaman karıştırıldığını gösterir. Bize sadece bir sınıflandırıcı tarafından yapılan hatalar hakkında değil, daha da önemlisi yapılan hata türleri hakkında bilgi verir.

Accuracy = TP+TN/TP+FP+FN+TN **Precision** = TP/TP+FP **Recall** = TP/TP+FN

F1 Score = 2*(Recall * Precision) / (Recall + Precision)

Şimdi modelimiz için oluşan confusion matrixini yorumlayalım:

	Class 0 Predicted	**Class 1 Predicted**	**Class 2 Predicted**	**Toplam Support**
Class 0 Actual	232	4	12	248
Class 1 Actual	10	210	9	229
Class 2 Actual	5	16	458	479
	Class 0: EI	**Class 1**: IE	**Class 3**:Neither	

232 DNA dizisi test datasında gerçekte EI, tahmin sonucunda da EI (Doğru Sınıflandırma)

210 DNA dizisi test datasında gerçekte IE, tahmin sonucunda da IE (Doğru Sınıflandırma)

458 DNA dizisi test datasında gerçekte Neither, tahmin sonucunda da Neither (Doğru Sınıflandırma)

4 DNA dizisi test datasında gerçekte EI, tahmin sonucunda da IE (Yanlış Sınıflandırma)

12 DNA dizisi test datasında gerçekte EI, tahmin sonucunda da Neither (Yanlış Sınıflandırma)

10 DNA dizisi test datasında gerçekte IE, tahmin sonucunda da EI (Yanlış Sınıflandırma)

9 DNA dizisi test datasında gerçekte IE, tahmin sonucunda da Neither (Yanlış Sınıflandırma)

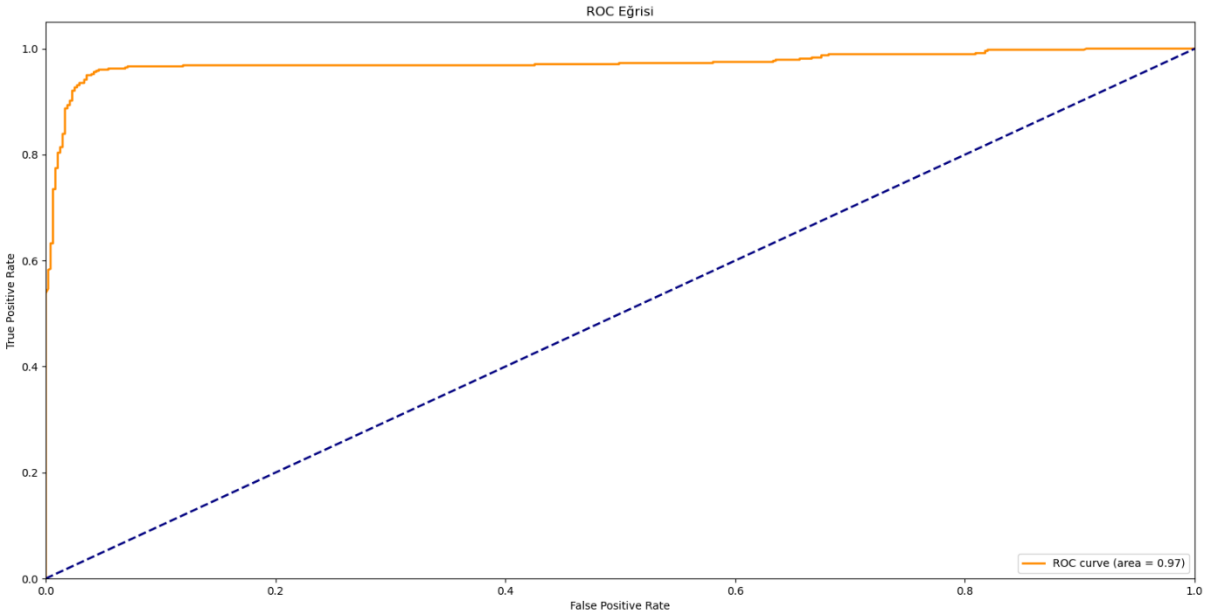
5 DNA dizisi test datasında gerçekte Neither, tahmin sonucunda da EI (Yanlış Sınıflandırma)

16 DNA dizisi test datasında gerçekte Neither, tahmin sonucunda da IE (Yanlış Sınıflandırma)

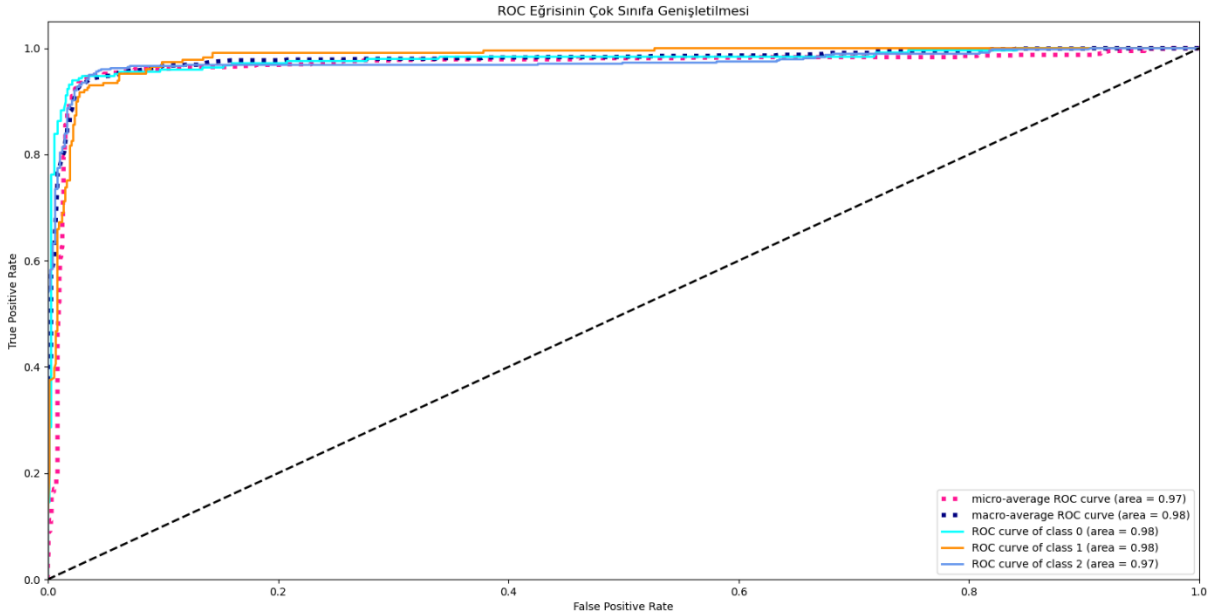
** Modelin yanlış sınıflandırmalarından en yüksek olanı 16 ile gerçekte Neither olduğu halde modelimiz bunu IE olarak tahmin etmiştir.

Modelin confusion matrixine göre classification reporttaki skorların hesaplanması daha önceden anlatıldığı gibi yapılmaktadır o yüzden burada tekrardan bir hesaplama yapılmayacaktır. Hesaplamalar yapıldığında classification reporttaki skorlar ile aynı olduğu gözlemlenir. Genel olarak modelimizin sınıf tahmininde oldukça başarılı olduğu saptanmıştır. Bu sonucun ROC eğrilerinde ve AUC değerlerinde de gözlemlenmesi beklenmektedir.

Son olarak model için ReceiverOperating Characteristic Curve (Alıcı çalışma karakteristik eğrisi) ve Area Under the Curve(Eğri Altındaki Alan) uygulaması yapılmıştır.



Model sınıf 2 için AUC değeri 0.97 gibi yüksek bir orana sahip. Bunun anlamı modelin sınıf ayrımı yaparken 2 sınıf için başarısı %97 oranında olduğu anlamına gelmektedir. Daha sonra tüm sınıfların ROC eğrileri ve her bir sınıfın AUC değerleri ve macro-micro ortalama ROC eğrileri ve AUC değerleri hesaplanarak çizdirilmiştir.



Model için elde edilen AUC değerleri; micro-average ROC eğrisi için 0.97, macro-average ROC eğrisi için 0.98, class 0 ROC eğrisi için(EI) 0.98, class 1 ROC eğrisi için(IE) 0.98 ve class 2 ROC eğrisi için(Neither) 0.97 şeklinde gözlemlenmiştir. Sonuçlardan da görüleceği gibi modelimiz sınıf ayrımını yapma doğruluğu konusunda class 2 için diğerlerine göre biraz daha düşük sonuç ortaya koymuştur. Bunu zaten confusion matrisinde class 2 için olan değerlerde tespit etmiştik. Genel olarak modelimiz sınıf ayrımını yüksek bir başarı ile yapabilmektedir. Bu oranlar ne kadar yüksek olursa modelin tahminleri yaparken hata yapma olasılığı o kadar düşük olmaktadır. ROC eğrileri bu konuda bize bu önemli bilgileri vermektedir.

3. EEG GÖZ DURUMU Dataseti

Dataset Kaynağı: https://www.openml.org/d/40670 **Dataset Gözlem Sayısı:** 14980

Dataset Tanıtımı:

Tüm veriler, Emotiv EEG Neuroheadset ile sürekli bir EEG ölçümüdür. Ölçüm süresi 117 saniye idi. EEG ölçümü sırasında göz durumu bir kamera aracılığıyla tespit edildi ve daha sonra video kareleri analiz edildikten sonra dosyaya manuel olarak eklendi. 'Closed' gözün kapalı olduğunu ve 'Open' gözün açık durumunu gösterir. Tüm değerler kronolojik sıradadır ve ilk ölçülen değer verinin üstündedir.

Bağımsız değişkenler, orijinal olarak AF3, F7, F3, FC5, T7, P7, O1, O2, P8, T8, FC6, F4, F8, AF4 olarak etiketlenmiş olup Emotiv EEG Neuroheadset'ten(EEG sensör/elektrot) gelen 14 EEG ölçümüne karşılık gelir. Bağımsız değişkenlerin hepsi 'float64' tipinde sürekli değişkenlerdir.

Bağımlı değişken ise eye olarak etiketlenmiş olup. Kategorik tiptedir. Yani dtype'ı 'object' 'tir.

Veri Ön İşleme ve Ölçekleme:

Öncelikle datasette eksik veri olup olmadığı kontrol edildi. Herhangi bir eksik veri olmadığı gözlemlendi. Daha sonra bağımlı değişken ('eye') kategorik tipten 0,1 dönüşümü yapılarak numerik veriye dönüştürüldü. Open:1,Closed:0 şeklinde kategorik yapı sayısal hale getirildi. Bağımlı değişken dataframe'den bir değişkene atandı. Bağımsız değişkenler de dataframe'den 'eye' değişkeni hariç olacak şekilde bir değişkene atandı. Veri ön işleme kısmında yapılacak olan işlemler bunlardan ibarettir.

Daha sonra bağımsız değişkenleri ölçekleme işlemine geçildi. Bağımsız değişkenlere StandardScaler() uygulanarak veriler standartlaştırıldı. Böylece model başarı puanları ölçekleme öncesine göre artmış oldu.

3.1. MLP Classifier Modeli

Multi Layer Perceptron (MLP) Classifier sklearn kütüphanesinde bulunan bir çok katmanlı algılayıcıdır. İlk olarak MLP ile model nesnesi oluşturuldu. Daha sonra model nesnesi train seti üzerinden fit edildi. Öncelikle valide edilmemiş(doğrulanmamış,ilkel) test skorlarını belirlemek istedim. Bunun için model tahmin işlemi yapılarak, "accuracy" ve "f1" skorlarını hesapladım. Bu bölümde model için hiçbir parametre girmeden yani model tune(ayar) edilmeden "accuracy" ve "f1" skorları hesaplandı. Bu skorlar nispeten tune edilmiş modele göre düşük olacaktır. Model için herhangi bir katman bilgisi ya da diğer parametrelerden hiçbirisi girilmeden model oluşturuldu. Her defasında farklı sonuçlar vermemesi açısından tune edilmemiş model için de tune edilmiş model içinde random_state aynı ayarlandı. Bu bölümden sonra Grid Search yapısı ile MLP modelimizin parametre optimizasyonu için çeşitli parametreler modele denenecek ve en uygun parametreler ile model oluşturulacaktır.

Grid search ile bir sözlük yapısı vasıtası ile modele optimizasyon için gönderilen parametreler ile açıklamalar daha önceki bölümde yapıldığı için bu bölümde bu parametreler hakkında bilgi verilmeyecektir.

Aşağıda denenmesi istenilen parametreler sözlük yapısı şeklinde gösterilmiştir. Ne kadar çok parametre gönderilirse o kadar işlem hacmi büyümektedir. Çünkü gönderilen her bir parametre bir birleri ile tek tek denenerek modeller oluşturulmaktadır.

```
mlpc_params = {"alpha": [0.1, 0.01, 0.001],
        "hidden_layer_sizes": [(100,100), (100,100,100)],
          "solver" : ["adam","sgd"],  "activation": ["relu","logistic"]}
```

Daha sonra model için bu parametreler Grid Search'te 5 katlı cross validation işlemi uygulandı.

```
mlpc_cv_model = GridSearchCV(mlpc, mlpc_params, cv = 5,  n_jobs = -1, verbose = 2)
```

Model parametreler ile tek tek train seti üzerinden fit edildikten sonra. Grid Search'ün bir fonksiyonu olan best_params_ ile denenen parametrelerden en iyi olan parametreler gözlemlenebilmektedir. Yine bir başka

fonksiyon olan best_estimator_ ise en iyi modeli vermektedir. best_estimator_ , en iyi parametrelerle kurulan en iyi sonucu veren modeli ifade eder. Grid Search'e gönderilen parametreler ile 72 parametre deneme işlemi her biri için 5 katlı cross validation yapılarak, toplamda 360 fit gerçekleştirilerek modelimiz için en iyi parametreler şu şekilde oluşmuştur:

En iyi parametreler: {'activation': 'relu', 'alpha': 0.001, 'hidden_layer_sizes': (100, 100, 100), 'solver': 'adam'}

Bir sonraki aşamada K-fold işlemi yapılarak cross validation işlemi hem accuracy hemde f1 skoru için yapılır. K-fold işlemi Grid Search'te belirlenen en iyi model üzerine uygulanmaktadır. K-fold ile veri seti 5 farklı alt kümeye bölünerek cross validation işlemi yapılır. Verisetinde bazı kısımlar modeli çok iyi ifade ederken bazı kısımlar tam olarak ifade etmez. Yani bazı kısımlarda modelin doğruluk oranı yüksek olabilirken bazı kısımlarda ise modelin doğruluk oranı düşük olabilmektedir. Amacımız modelimizin data setinin her yerinde benzer sonuçlar verip vermediğini tespit ederek, modelin tüm veri setinde düzgün çalışıp çalışmadığını kontrol etmektir. Yani dışarıdan modelin hiç görmediği bir veri seti geldiğinde modelin iyi çalışmasını sağlamaktır. K-fold ile modeli farklı bölümlere ayırarak model için farklı bölümlerdeki değerlendirme skorları elde edilerek modelin doğrulanması sağlanmış olur. Skorlama için kullanılan metrik olarak f1 ve accuracy kullanılmıştır. Test seti üzerinden 5 farklı f1 ve accuracy skoru elde edilmiştir. Daha sonra bu skorların ortalaması alınarak modelin valide edilmiş(doğrulanmış) skorlar elde edilmiş olur.

Daha sonra Grid Search ile belirlenen en iyi model ile model tahmin işlemi yapılır. f1,accuracy skorları tune edilmiş model için hesaplanır. Sonuçlardan da görüleceği gibi tune(ayarlanmış) edilmiş model ile alınan skorlar tune edilmemiş model ile alınan skorlara göre daha iyi olduğu görülmektedir. Aşağıdaki tabloda tune edilmiş, tune edilmemiş ve K-fold ile alınan skorlar gösterilmiştir.

K-fold Cross Validation f1 Sonuçları: [0.84412955 0.8691796 0.84793555 0.82425488 0.82153539]

K-fold Cross Validation f1 Sonuçlarının Ortalaması: 0.8414069960024129

K-fold Cross Validation Accuracy Sonuçları: [0.80756396 0.81979978 0.84538376 0.80645161 0.85634744]

K-fold Cross Validation Accuracy Sonuçlarının Ortalaması: 0.8271093097750285

	f1	accuracy
Tune Edilmemiş(İlkel Model)	0.8932079414838036	0.8862928348909658
K-fold Cross Validation Ortalaması	0.7996845958884821	0.8650420168067227
Tune Edilmiş(En İyi Modelle)	0.823943661971831	0.8815165876777251

Yukarıdaki değerlerde görüldüğü gibi modelin tune edilmemiş skorları tune edilmiş yani Grid Search ile belirlenen en iyi model ile elde edilmiş skorlara göre düşüktür. Nitekim bunun böyle olmasını bekliyorduk. Modele K-fold Cross Validation uygulanmış hali ile elde edilen skorlarda bize gösteriyor ki model veri seti farklı alt kümeye bölünmesi ile farklı alt kümelerde farklı sonuçlar vermektedir. K-fold'un bize daha güvenilir sonuçlar verdiği açık bir şekilde gözlemlenmiş oldu. Model veri setinin bazı alt kümelerinde diğerlerine göre nispeten daha az tahmin başarısı elde etmiştir.

Sonraki aşamada tune edilmiş model için classification report ve confusion matrix hesaplamaları yapıldı. Classification report çeşitli skorlama türlerinin sınıflara göre değerlerini bir arada veren bir raporlama şeklidir.

Aşağıda classification report ile alınan değer gösterilmiştir.

	precision	**recall**	**f1-score**	**support**
0	0.87	0.89	0.88	2073
1	0.90	0.88	0.89	2421
accuracy			0.88	4494
macro avg	0.89	0.89	0.89	4494
weighted avg	0.89	0.89	0.89	4494

Confusion matrix ise sınıflandırma problemine ilişkin tahmin sonuçlarının bir özetidir. Doğru ve yanlış tahminlerin sayısı sayım değerleri ile özetlenir ve her sınıf tarafından ayrılır. Confusion matrixi sınıflandırma

modelinizin tahminlerde ne zaman karıştırıldığını gösterir. Bize sadece bir sınıflandırıcı tarafından yapılan hatalar hakkında değil, daha da önemlisi yapılan hata türleri hakkında bilgi verir.

Accuracy = TP+TN/TP+FP+FN+TN **Precision** = TP/TP+FP **Recall** = TP/TP+FN

F1 Score = 2*(Recall * Precision) / (Recall + Precision)

Şimdi modelimiz için oluşan confusion matrixini yorumlayalım:

	Class 0 Predicted	**Class 1 Predicted**	**Toplam Support**
Class 0 Actual	1846	227	2073
Class 1 Actual	284	2137	2421

Class 0: Göz Kapalı (Close) **Class 1**: Göz Açık(Open)

1846 örnekte test datasında gerçekte göz kapalı, tahmin sonucunda da göz kapalı (Doğru Sınıflandırma)

2137 örnekte test datasında gerçekte göz açık, tahmin sonucunda göz açık (Doğru Sınıflandırma)

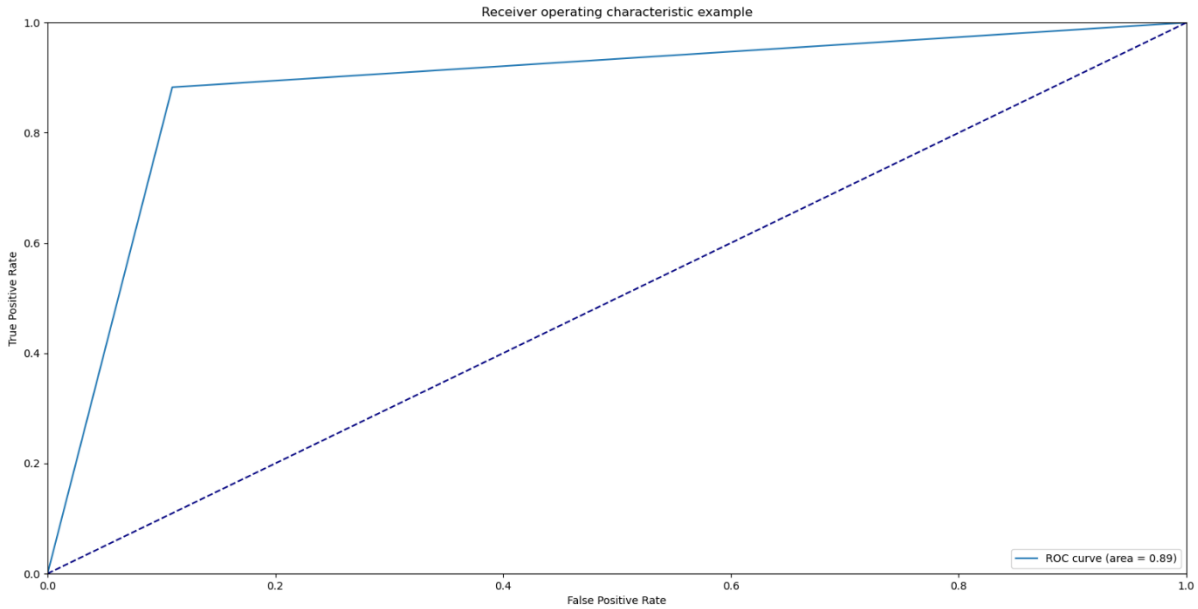
227 örnekte test datasında gerçekte göz kapalı, tahmin sonucunda göz açık (Yanlış Sınıflandırma)

284 örnekte test datasında gerçekte göz açık, tahmin sonucunda da göz kapalı (Yanlış Sınıflandırma)

** Modelin yanlış sınıflandırmalarından en yüksek olanı 284 ile gerçekte göz açık durumda olduğu halde modelimiz göz kapalı olarak tahmin etmiştir.

Modelin confusion matrixine göre classification reporttaki skorların hesaplanması daha önceden anlatıldığı gibi yapılmaktadır o yüzden burada tekrardan bir hesaplama yapılmayacaktır. Hesaplamalar yapıldığında classification reporttaki skorlar ile aynı olduğu gözlemlenir. Genel olarak modelimizin sınıf tahmininde oldukça başarılı olduğu saptanmıştır. Bu sonucun ROC eğrilerinde ve AUC değerlerinde de gözlemlenmesi beklenmektedir.

Son olarak model için ReceiverOperating Characteristic Curve (Alıcı çalışma karakteristik eğrisi) ve Area Under the Curve(Eğri Altındaki Alan) uygulaması yapılmıştır.



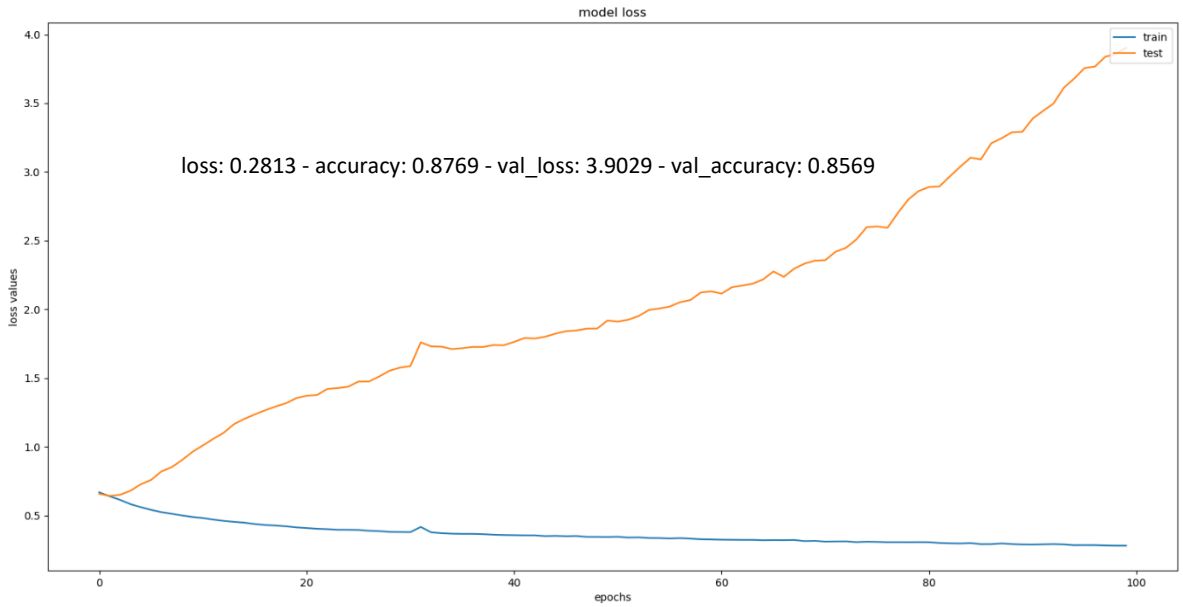
Modelimizin AUC değeri 0.89 gibi bir orana sahip. Bunun anlamı modelin sınıf ayrımı yaparken gözün kapalı olması ve gözün açık olması sınıflarını ayırt etme başarısı %89 oranında olduğu anlamına gelmektedir.

3.2. Keras Modeli

Multi Layer Perceptron için Keras Python kütüphanesi, modellerin bir katman dizisi olarak oluşturulmasına odaklanır. Keras ile model için her bir katman tek tek oluşturulabilir ve her bir katman özelinde ayarlamalar yapılmasına olanak sağladığı için daha esnek bir yapı ortaya koymaktadır. Model katmanları tek tek oluşturulduğu için her bir katmandaki nöron sayısı Dense fonksiyonu ile belirtilmelidir. Giriş katmanı için veri setindeki bağımsız değişken sayısı(girdi boyutu) verilmelidir. Data setimizde 14 tane bağımsız değişkenimiz olduğu için input_dim=14 olarak belirtildi. Daha sonra her bir katman için aktivasyon fonksiyonu da her bir katman özelinde ayarlanabildiği için bunun da belirtilmesi gerekmektedir. Keras için çıktı katmanın nöron sayısının da(sınıf sayısının) belirtilmesi gerekmektedir. Ayrıca Keras modelinde sınıf sayısı 2 outputlu problemlerde çıktı katmanında "sigmoid" fonksiyonu kullanılır. Daha önceki multioutput problemlerinde ise "softmax" kullanılmıştı. MLPClassifier'da ise bunları belirtmemize gerek yok. Keras modelini compile(derleme) ederken loss(kayıp) fonksiyonunu belirtmemiz gerekmektedir. MLPClassifier için loss fonksiyonunu belirtmeye gerek yoktur kendisi otomatik olarak seçer.Hedef değişkeni 2 outputlu ve 0,1 'den oluştuğu için loss fonksiyonu olarak 'binary_crossentropy' seçildi. Önceki bölümde hangi loss fonksiyonunun neden seçildiği hakkında bilgiler verilmişti. Modeli ilk başta kurarken optimizer için "adam" olarak compile ettim. "Adam" optimizer genellikle yüksek boyuttaki ve komplex datasetler için iyi sonuçlar vermektedir. Modelin metrics argümanını "accuracy" olarak belirledim. Modeli bir fonksiyon şeklinde tanımlayıp kurduktan sonra modelin loss fonksiyonunun değerinin epoch sayısına göre değişimini train ve test seti özelinde gözlemlemek için train seti ile eğittim. Burada epochs olarak 100 , batch_size olarak ise 50 olarak belirledim. Daha önceki bölümde batch_size argümanı için detaylı bir şekilde açıklamalar ve mevcut yaklaşımlar hakkında bilgiler verildiği için burada anlatılmadı.

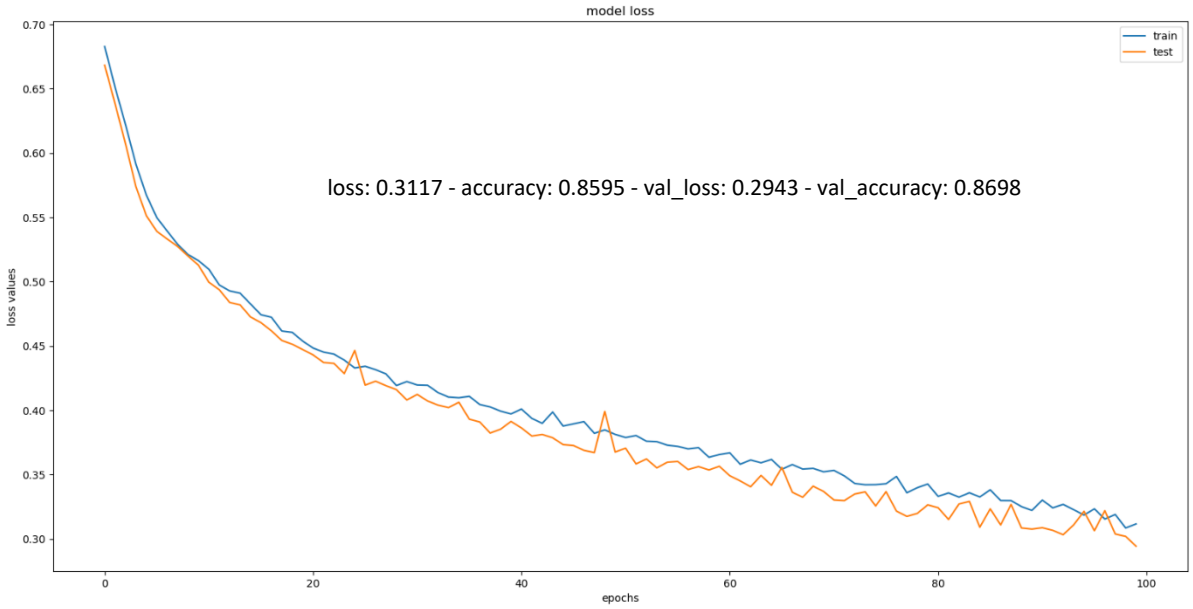
İlk olarak modeli kötü bir şekilde kuralım ve train kaybı ve test kaybını gözlemleyelim.

```
def create_model(optimizer="adam"):
    model = Sequential()
    model.add(Dense(32, input_dim=14, activation='relu'))
    model.add(Dense(16, activation='relu'))
    model.add(Dense(8, activation='relu'))
    model.add(Dense(1, activation='sigmoid'))
    model.compile(loss='binary_crossentropy', optimizer=optimizer,metrics=["accuracy"])
    return model
model = create_model()
egitim=model.fit(X_train, y_train, epochs=100, batch_size=50, verbose=1,validation_data=(X_test,y_test))
```



Grafikten de anlaşılacağı üzere oldukça kötü bir model. Modelin test hatası aşırı artarken train hatası düşmektedir. Bu da modelin overfittinge doğru gittiğini yani modelin train seti üzerinde yüksek doğruluğa sahip olup test seti üzerinde daha düşük doğruluğa sahip olduğunu göstermektedir. Model train setini çok başarılı bir şekilde tahmin ederken, test setini o başarıda tahmin edememiştir. Şimdi modelimizi biraz geliştirelim overfittingin önüne geçmek için modele Droupout(bırakma) katmanı eklendi. Bırakma katmanı ekleme overfittingi kolay ve etkili bir yolu olabilir . Bir bırakma katmanı, katmanlar arasındaki bazı bağlantıları rastgele bırakır. Bu, aşırı öğrenmeyi önlemeye yardımcı olur. Droupout katmanı ilk katmanda eklenmesinin sebebi denemeler ile daha güzel sonucun alınmasından dolayıdır. Diğer katmanlara da Droupout eklendiğinde model "accuracy" değeri bir hayli düşük olduğu için model veri setini yeteri kadar öğrenememektedir. Bunlara ek olarak modele yeni katmanlar eklendi ve ara katmanlardaki aktivasyon fonksiyonlarında da değişiklik yapıldı.

```
def create_model(optimizer="adam"):
    model = Sequential()
    model.add(Dense(32, input_dim=14, activation='relu'))
    model.add(Dropout(0.1))
    model.add(Dense(20, activation='relu'))
    model.add(Dense(16, activation='relu'))
    model.add(Dense(8, activation='sigmoid'))
    model.add(Dense(4, activation='relu'))
    model.add(Dense(1, activation='sigmoid'))
    model.compile(loss='binary_crossentropy', optimizer=optimizer,metrics=["accuracy"])
    return model
model = create_model()
egitim=model.fit(X_train, y_train, epochs=100, batch_size=50, verbose=1,validation_data=(X_test,y_test))
```



Grafikte de görüldüğü gibi modelin test ve train kayıpları birbirine çok yaklaşmıştır. Modelin train setindeki tahmin başarısı ve test setindeki tahmin başarısı da birbirine yakın değerler göstermektedir. Hatta modelimiz test setinde train setinden daha yüksek bir başarı elde etmektedir. Dolayısıyla modelimiz yapılan düzenlemeler neticesinde daha uygun bir hale geldi.

Öncelikle valide edilmemiş(doğrulanmamış,ilkel) test skorlarını belirlemek istedim. Bunun için model tahmin işlemi yapılarak, “accuracy” ve “f1” skorlarını hesapladım. Bu bölümde model için hiçbir parametre girmeden yani model tune(ayar) edilmeden “accuracy” ve “f1” skorları hesaplandı. Ayrıca model hedef(bağımlı) değişkeni multi-class(çok sınıflı) olduğu için model tahmin ve skorlama işlemlerinde bağımlı değişkene ait kısımlarda numpy methodu olan np.argmax ile eski haline getirilerek işlemler yapılmıştır. Bu skorlar nispeten tune edilmiş modele göre düşük olacaktır. Bu bölümden sonra Grid Search yapısı ile Keras modelimizin parametre optimizasyonu için çeşitli parametreler modele denenecek ve en uygun parametreler ile model oluşturulacaktır.

Grid search ile bir sözlük yapısı vasıtası ile modele optimizasyon için gönderilen parametreler hakkında daha önceden detaylı bilgi verilmiştir. O yüzden bu bölümde bu parametreler hakkında bilgilere değinilmeyecektir.

Aşağıda denenmesi istenilen parametreler sözlük yapısı şeklinde gösterilmiştir. Ne kadar çok parametre gönderilirse o kadar işlem hacmi büyümektedir. Çünkü gönderilen her bir parametre bir birleri ile tek tek denenerek modeller oluşturulmaktadır. Özellikle bu datasetin boyutu çok büyük olduğu için parametre sayısı diğer modellere göre daha az girilmiştir.

```
param_grid = {   'epochs': [100,150],

   'batch_size':[32,50],  'optimizer':[ 'Adam','SGD'], }
```

Daha sonra model KerasClassifier ile oluşturularak. Grid Search Cross Validation yapısına gönderilir. Gönderilen parametreler ile Grid Search’te 5 katlı cross validation işlemi uygulandı.

```
grid = GridSearchCV(estimator=model_cv, n_jobs=-1, verbose=1, cv=5, param_grid=param_grid)
```

Model parametreler ile tek tek train seti üzerinden fit edildikten sonra. Grid Search’ün bir fonksiyonu olan best_params_ ile denenen parametrelerden en iyi olan parametreler gözlemlenebilmektedir. Yine bir başka fonksiyon olan best_estimator_ ise en iyi modeli vermektedir. best_estimator_ , en iyi parametrelerle kurulan en iyi sonucu veren modeli ifade eder. Grid Search’e gönderilen parametreler ile 8 parametre deneme işlemi her biri için 5 katlı cross validation yapılarak, toplamda 40 fit gerçekleştirilerek modelimiz için en iyi parametreler şu şekilde oluşmuştur:

En iyi parametreler: 'batch_size': 32, 'epochs': 150, 'optimizer': 'Adam'

Bir sonraki aşamada K-fold işlemi yapılarak cross validation işlemi hem accuracy hemde f1 skoru için yapılır. K-fold işlemi Grid Search'te belirlenen en iyi model üzerine uygulanmaktadır. K-fold ile veri seti 5 farklı alt kümeye bölünerek cross validation işlemi yapılır. Verisetinde bazı kısımlar modeli çok iyi ifade ederken bazı kısımlar tam olarak ifade etmez. Yani bazı kısımlarda modelin doğruluk oranı yüksek olabilirken bazı kısımlarda ise modelin doğruluk oranı düşük olabilmektedir. Amacımız modelimizin data setinin her yerinde benzer sonuçlar verip vermediğini tespit ederek, modelin tüm veri setinde düzgün çalışıp çalışmadığını kontrol etmektir. Yani dışarıdan modelin hiç görmediği bir veri seti geldiğinde modelin iyi çalışmasını sağlamaktır. K-fold ile modeli farklı bölümlere ayırarak model için farklı bölümlerdeki değerlendirme skorları elde edilerek modelin doğrulanması sağlanmış olur. Skorlama için kullanılan metrik olarak f1 ve accuracy kullanılmıştır. Test seti üzerinden 5 farklı f1 ve accuracy skoru elde edilmiştir. Daha sonra bu skorların ortalaması alınarak modelin valide edilmiş(doğrulanmış) skorlar elde edilmiş olur.

Daha sonra Grid Search ile belirlenen en iyi model ile model tahmin işlemi yapılır. f1, accuracy skorları tune edilmiş model için hesaplanır. Sonuçlardan da görüleceği gibi tune(ayarlanmış) edilmiş model ile alınan skorlar tune edilmemiş model ile alınan skorlara göre daha iyi olduğu görülmektedir. Aşağıdaki tabloda tune edilmiş, tune edilmemiş ve K-fold ile alınan skorlar gösterilmiştir.

K-fold Cross Validation f1 Sonuçları: [0.84200196 0.83864542 0.85093781 0.83516484 0.83282051]

K-fold Cross Validation f1 Sonuçlarının Ortalaması: 0.8399141075020428

K-fold Cross Validation Accuracy Sonuçları: [0.80978865 0.83648498 0.80200222 0.80533927 0.80178174]

K-fold Cross Validation Accuracy Sonuçlarının Ortalaması: 0.811079373022735

	f1_weighted	accuracy
Tune Edilmemiş(İlkel Model)	0.8810250152532032	0.869826435246996
K-fold Cross Validation Ortalaması	0.8399141075020428	0.811079373022735
Tune Edilmiş(En İyi Modelle)	0.910964550700742	0.9038718291054739

Yukarıdaki değerlerde görüldüğü gibi modelin tune edilmemiş skorları tune edilmiş yani Grid Search ile belirlenen en iyi model ile elde edilmiş skorlara göre düşüktür. Nitekim bunun böyle olmasını bekliyorduk. Modele K-fold Cross Validation uygulanmış hali ile elde edilen skorlarda bize gösteriyor ki model veri seti farklı alt kümeye bölünmesi ile farklı alt kümelerde farklı sonuçlar vermektedir. K-fold'un bize daha güvenilir sonuçlar verdiği açık bir şekilde gözlemlenmiş oldu. Model veri setinin bazı alt kümelerinde diğerlerine göre nispeten daha az tahmin başarısı elde etmiştir.

Sonraki aşamada tune edilmiş model için classification report ve confusion matrix hesaplamaları yapıldı. Classification report çeşitli skorlama türlerinin sınıflara göre değerlerini bir arada veren bir raporlama şeklidir.

Aşağıda classification report ile alınan değer gösterilmiştir.

	precision	**recall**	**f1-score**	**support**
0	0.90	0.89	0.90	2073
1	0.91	0.91	0.91	2421
accuracy			0.90	4494
macro avg	0.90	0.90	0.90	4494
weighted avg	0.90	0.90	0.90	4494

Confusion matrix ise sınıflandırma problemine ilişkin tahmin sonuçlarının bir özetidir. Doğru ve yanlış tahminlerin sayısı sayım değerleri ile özetlenir ve her sınıf tarafından ayrılır. Confusion matrixi sınıflandırma modelinizin tahminlerde ne zaman karıştırıldığını gösterir. Bize sadece bir sınıflandırıcı tarafından yapılan hatalar hakkında değil, daha da önemlisi yapılan hata türleri hakkında bilgi verir.

Accuracy = TP+TN/TP+FP+FN+TN **Precision** = TP/TP+FP **Recall** = TP/TP+FN

F1 Score = 2*(Recall * Precision) / (Recall + Precision)

Şimdi modelimiz için oluşan confusion matrixini yorumlayalım:

	Class 0 Predicted	Class 1 Predicted	Toplam Support
Class 0 Actual	1852	221	2073
Class 1 Actual	211	2210	2421

Class 0: Göz Kapalı (Close) **Class 1**: Göz Açık(Open)

1852 örnekte test datasında gerçekte göz kapalı, tahmin sonucunda da göz kapalı (Doğru Sınıflandırma)

2210 örnekte test datasında gerçekte göz açık, tahmin sonucunda göz açık (Doğru Sınıflandırma)

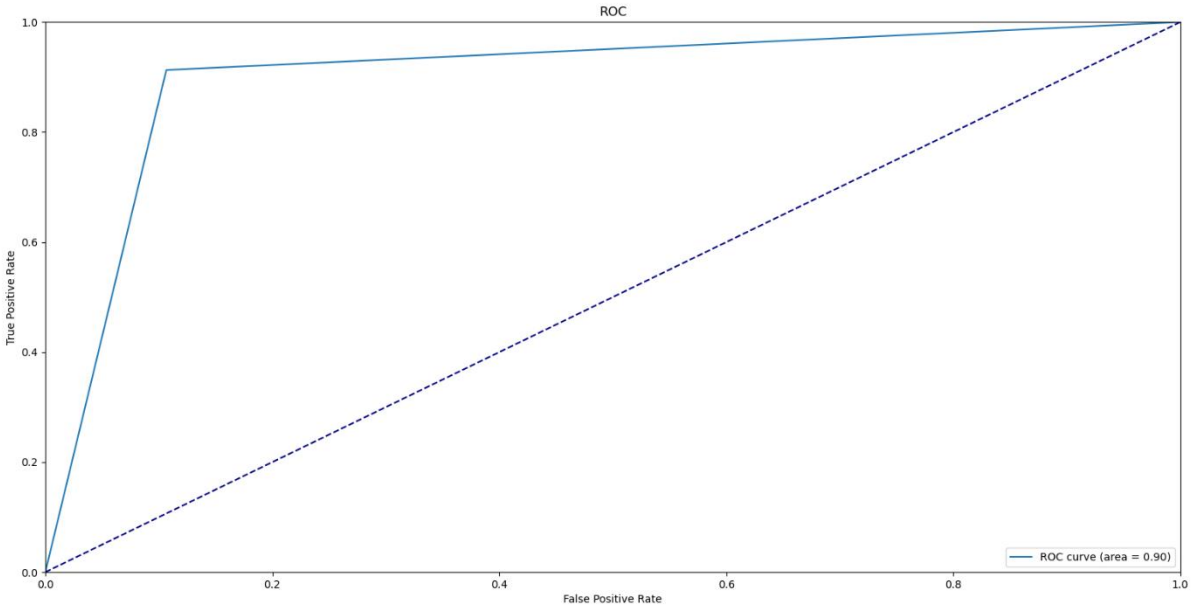
221 örnekte test datasında gerçekte göz kapalı, tahmin sonucunda göz açık (Yanlış Sınıflandırma)

211 örnekte test datasında gerçekte göz açık, tahmin sonucunda da göz kapalı (Yanlış Sınıflandırma)

** Modelin yanlış sınıflandırmalarından en yüksek olanı 221 ile gerçekte göz kapalı durumda olduğu halde modelimiz göz açık olarak tahmin etmiştir.

Modelin confusion matrixine göre classification reporttaki skorların hesaplanması daha önceden anlatıldığı gibi yapılmaktadır o yüzden burada tekrardan bir hesaplama yapılmayacaktır. Hesaplamalar yapıldığında classification reporttaki skorlar ile aynı olduğu gözlemlenir. Genel olarak modelimizin sınıf tahmininde oldukça başarılı olduğu saptanmıştır. Bu sonucun ROC eğrilerinde ve AUC değerlerinde de gözlemlenmesi beklenmektedir.

Son olarak model için ReceiverOperating Characteristic Curve (Alıcı çalışma karakteristik eğrisi) ve Area Under the Curve(Eğri Altındaki Alan) uygulaması yapılmıştır.



Modelimizin AUC değeri 0.90 gibi bir orana sahip. Bunun anlamı modelin sınıf ayrımı yaparken gözün kapalı olması ve gözün açık olması sınıflarını ayırt etme başarısı %90 oranında olduğu anlamına gelmektedir.

Kaynaklar

1.https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.GridSearchCV.html

2. https://medium.com/@gulcanogundur/model-se%C3%A7imi-k-fold-cross-validation-4635b61f143c

3. https://towardsdatascience.com/multi-class-metrics-made-simple-part-ii-the-f1-score-ebe8b2c2ca1

4. https://medium.com/greyatom/lets-learn-about-auc-roc-curve-4a94b4d88152

5. https://scikit-learn.org/stable/auto_examples/model_selection/plot_roc.html

6. https://scikit-learn.org/stable/modules/generated/sklearn.metrics.roc_curve.html

7. https://machinelearningmastery.com/how-to-choose-loss-functions-when-training-deep-learning-neural-networks/

8. https://machinelearningmastery.com/how-to-control-the-speed-and-stability-of-training-neural-networks-with-gradient-descent-batch-size/

9. https://towardsdatascience.com/dont-overfit-how-to-prevent-overfitting-in-your-deep-learning-models-63274e552323

10. https://machinelearningmastery.com/cost-sensitive-neural-network-for-imbalanced-classification/

11.https://books.google.com.tr/books?id=OgUqDwAAQBAJ&pg=PA347&lpg=PA347&dq=mlp+classifier+class+weight&source=bl&ots=MpQhj3AhZc&sig=ACfU3U2pQI3VuXrxNtaYKmUiVoKEVCsoxQ&hl=tr&sa=X&ved=2ahUKEwjd_vnv3M_pAhXvxIsKHWuqAOoQ6AEwB3oECAkQAQ#v=onepage&q=mlp%20classifier%20class%20weight&f=false (Sayfa 347 MLPClassifier class_weight bulunmuyor)

12. https://machinelearningmastery.com/grid-search-hyperparameters-deep-learning-models-python-keras/

13. https://medium.com/@sayedathar11/multi-layered-perceptron-models-on-mnist-dataset-say-no-to-overfitting-3efa128a019c

14. https://towardsdatascience.com/overfitting-vs-underfitting-a-complete-example-d05dd7e19765

15. https://stackoverflow.com/questions/41494625/issues-using-keras-np-utils-to-categorical

16. https://machinelearningmastery.com/how-to-one-hot-encode-sequence-data-in-python/

17. https://numpy.org/doc/stable/reference/generated/numpy.argmax.html

18. https://www.geeksforgeeks.org/numpy-argmax-python/

19. https://stackoverflow.com/questions/46732881/how-to-calculate-f1-score-for-multilabel-classification

20. https://scikit-learn.org/stable/modules/cross_validation.html

21. https://www.kaggle.com/learn-forum/59719

22. https://energie.labs.fhv.at/~kr/ds/10_Model_Evaluation.html

23. https://scikit-learn.org/stable/modules/model_evaluation.html

24. https://scikit-learn.org/stable/modules/generated/sklearn.metrics.classification_report.html#sklearn.metrics.classification_report

25. https://scikit-learn.org/stable/modules/generated/sklearn.metrics.confusion_matrix.html#sklearn.metrics.confusion_matrix

26. https://machinelearningmastery.com/dropout-regularization-deep-learning-models-keras/

27. https://keras.io/

28. https://machinelearningmastery.com/adam-optimization-algorithm-for-deep-learning/

29. https://machinelearningmastery.com/5-step-life-cycle-neural-network-models-keras/

30. https://keras.io/api/optimizers/

31. https://keras.io/api/losses/

32. https://keras.io/api/layers/activations/

33. https://stackoverflow.com/questions/40393629/how-to-pass-a-parameter-to-scikit-learn-keras-model-function

34. https://www.dezyre.com/recipes/find-optimal-parameters-using-gridsearchcv

35. https://stackoverflow.com/questions/53806892/i-cant-add-optimizer-parameter-in-gridsearch