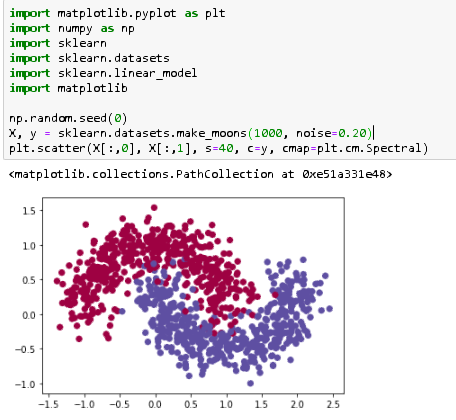
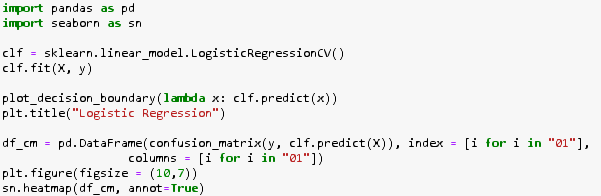
# Öncelikle Pythonda üzerinde Logical Regression ve Neural Network metodlarını eğitebilmek ve test edebilmek için bir dataset oluşturuyoruz. Resimde gördüğümüz gibi datasetimiz iç içe geçmiş aylardan oluşuyor.



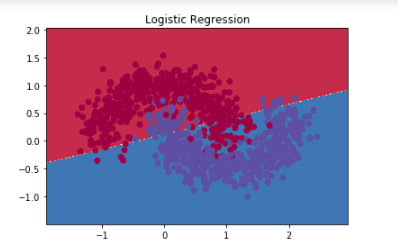
# Logistic Regression

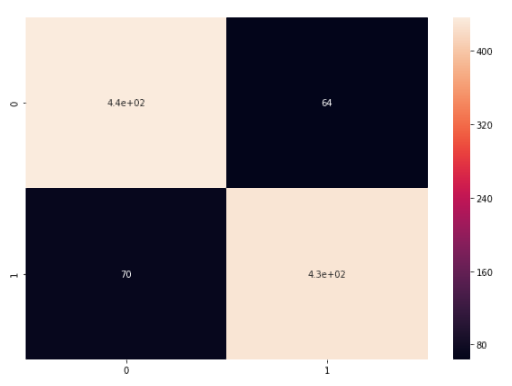
## Python’un scikit-learn kütüphanesinden Logistik Regresyon nasıl kullanılır? Giriş ve çıkış değerleri nedir? Kod örneği ile kısaca açıklayınız.



Sklearn kütüphanesinden linear\_model.LogisticRegressionCV() methodunu çağırarak logistik regresyon modülü oluşturabiliyoruz. .fit fonksiyonuna verdiğimiz training inputları ile de modulumuzu eğitiyoruz.Modelimizi ve modelimizin confusion matrixini çıkaralım.Çıkış değerlerimiz ise .predict(x) fonkiyonumuzu çizdirerek görüyoruz ki elinden geldiğince en yüksek isabet oranıyla verileri ikiye ayırmaya çalışmış.

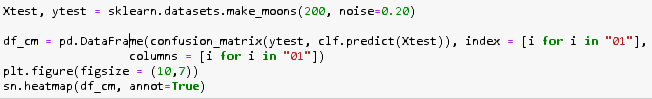
## Logistik Regresyon ile elde edilen çıktıyı raporunuzu koyunuz ve çıktıyı yorumlayınız.

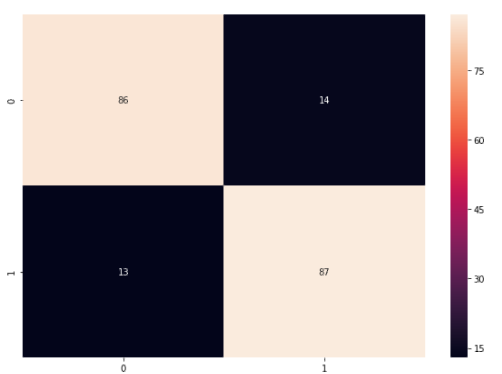


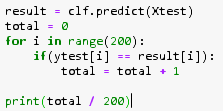


Modelimizi test etmek için test verileri oluşturuyoruz ve oluşturduğumuz modeli bu yeni oluşturduğumuz modelin görmediği veri değerleriyle deniyoruz.Confusion Matriximizden eğitim başarı oranımızı görebiliyoruz fakat bu yanıltıcı olabilir. İleride modelimizin görmediği verilerle denememiz bu oranı belirlemek için daha sağlıklı bir sonuç vericektir.Logistic Regression modelimiz elimizdeki veriyi bölebileceği en iyi şekilde bölmeye çalışmış fakat bu Logistic Regression modelimizin datayı bölmesinden görüyoruz ki bu modelimiz bu data için pek de kullanışlı olmayabilir.Çünkü iç içe hilal şeklindeki verileri araştırmada pek başarılı olamamış.Düz bir çizgi bu tür bir sınıflandırma için yeterli olmayabilir.

## Python koduna scikit-learn kütüphanesinden Confusion Matrix oluşturmak için gereken ek satırları yazarak bu matrisi oluşturun. Çıktıyı raporunuza koyarak kısaca açıklayınız. Önce 100’er tane mavi ve kırmızı test versini üretiniz ve bu test verilerinden sınıflandırma performansını bulunuz.

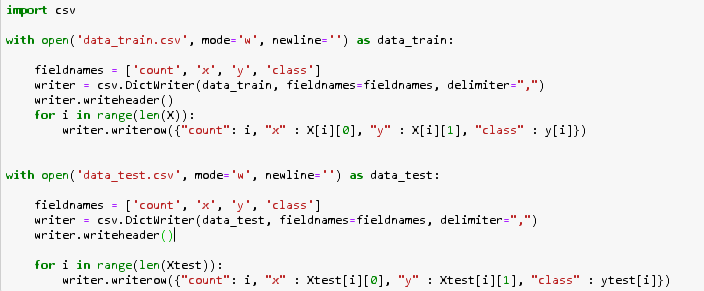




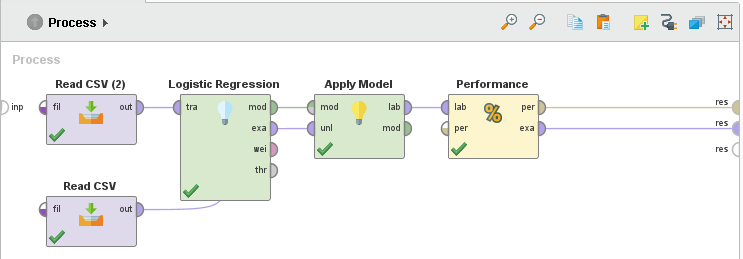


Buradan modelimizin görmediği veriler üzerinde başarı oranını görebiliriz.Confusion Matrixden gördüğümüz kadarıyla modelimizin 173/200 başarı oranı olduğunu söyleyebiliriz.

## Verilerimizi dosyamıza yazıyoruz bu sayede RapidMiner üzerinde test edebileceğiz.



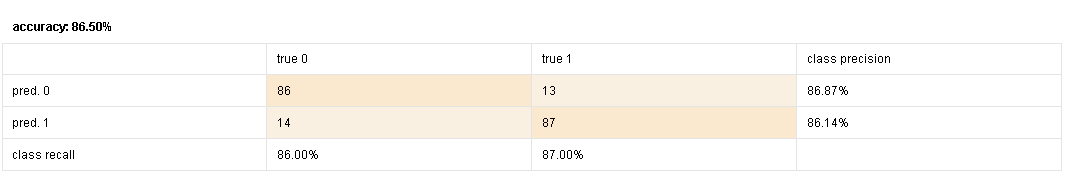
Kullandığımız eğitim ve test verilerini bir .csv uzantılı dosyaya kaydediyoruz.Şimdi bu verileri RapidMiner programında kullanarak elde ettiğimiz verileri karşılaştırıcaz.

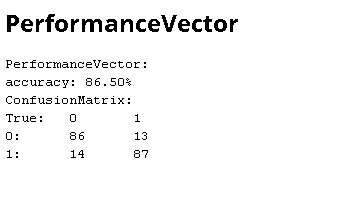


RapidMiner da Logistic Regression modelini kullanmak için kurduğum yapı.Yukarıdaki CSV dosyamız eğitimimiz, aşağıdaki CSV ise test dosyamız.Logistic Regressionun training kısmına eğitim verimizi yolluyoruz, modele de test verimizi yolluyoruz.

## RapidMiner Sonuçlarımız

Performance dan aldığımız veriler ise bunlar:



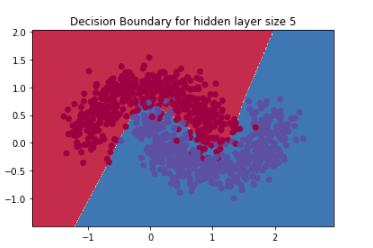


Python kodumuzdan elde ettiğimiz Confusion Matrix imizi ve başarı oranımızı RapidMiner ile karşılaştırdığımızda görüyoruz ki iki modelden de aynı başarı oranını ve aynı confusion matrixi elde etmişiz.

# Neural Networks: Python

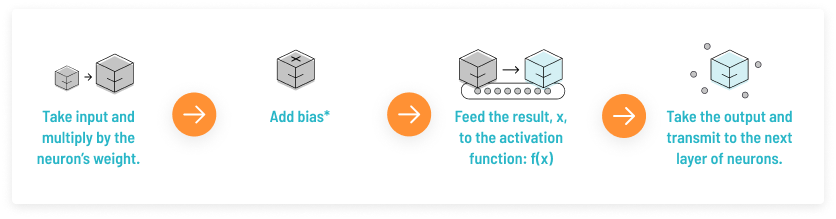
## Sinir ağının şu ayrıntılarını kısaca anlatınız: Gizli katman sayısının işlemlere ve tahminlere etkisi, Gizli katman sayısı nasıl seçilir, aktivasyon fonksiyonu nedir ve nasıl seçilir, softmax fonksiyonu nedir, ağın öğrenme parametreleri nelerdir ve nasıl bulunur, ağ nasıl öğrenir, ağ nasıl tahminde bulunur.

## 



Buradan görüyoruz ki hidden layerdaki nöron sayısını arttırmak daha başarılı bir model eğitmemiz için bir öncü olabilir fakat hidden layer sayısını çok fazla, hidden layerdaki nöron sayısını çok fazla arttırmak modelin öğrenmekten cok ezberlemesine yol açabilir bu da eğitimde başarı oranının yüksek olmasına fakat görmediği verilerde ise başarı oranının düşük olmasına yol açabilir.Genel olarak projelerde hidden layerdaki nöron sayısının seçimi çok kesin kurallara bağlı olmamakla birlikte genelde bir önceki giriş nöronlarıyla çıkış nöronlarının ortalamasını seçmek araştırmalara göre sinir ağlarında başlangıç için güzel bir seçim olabilir.Tabi en uygun seçim bizim de ileride deniyeceğimiz gibi farklı parametrelerle denemek ve uygun noktayı ezberleme yapmadan seçmek gerekir.Aktivasyon fonksiyonu ise Aktivasyon fonksiyonu, mevcut nöronu besleyen giriş ile bir sonraki katmana giden çıkışı arasındaki matematiksel bir "geçit" tir.Kullandığımız matematiksel fonksiyona göre çıktımız değişebilir.

Aşağıdaki resim activation fonksiyonunu çalışmasını açıklamaktadır.Bu fonksiyonlar sayesinde sinir ağımız daha karmaşık olan problemleri de öğrenebilir, daha iyi eğitilebilir ve daha iyi tahmin etme becerisine sahip olabilir.



Softmax fonksiyonu ise girdi olarak aldığı vektör girişini sayıların büyüklüğüyle doğru orantılı olarak olasık dağılımına dönüştüren bir fonksiyondur.Genelde 2 den fazla sınıf ile uğraşıyorsak output kısmını softmax fonksiyonuna bağlayarak olasılık elde edebiliriz bu olasıklardan da en büyük olasılığı seçerek sinir ağımızın tahmin ettiği değeri olasılığı en büyük değer olarak kabul edebiliriz.3 sınıf ile sinir ağı kurduğumuz bölümde softmax fonksiyonunu kullanıcaz.Sinir ağlarında öğrenme parametreleri genel olarak hidden layer sayısı,hidden layerdaki nöron sayısı, kullanılırsa dropout(rastgele bazı nöronların sıfırlanması) , activasyon fonksiyonu(matematiksel geçiş), nöron ağırlıklarının ilişkilendirilmesi(biz random atadık) , öğrenme oranı, epoch sayısı olabilir.Sinir ağlarımız yapı itibariyle eğitim aşamasında her epoch adındaki adımda bir adet ileri yayılma gerçekleştirir bu kısımda nöronlarımıza gelen inputlar ile nöron ağırlıklarımızı çarpıp bias ekleyip outputa kadar böyle ilerliyoruz, orda bulduğumuz sonuç ile gerçekte olması gereken arasında kayıp fonksiyonuyla aralarındakı farkı buluyoruz ve bu bulduğumuz değere göre az önce çarparak geldiğimiz ileri yayım yerine bu sefer de türevini alarak geri geldiğimiz geri yayılım kısmını uyguluyoruz, ilerlerken geçtiğimiz nöronların ağırlıklarını güncelliyoruz ve bu döngüyü bize verilen epoch sayısı kadar gerçekleştiriyoruz. Test etme aşamasında ise sadece ileri yayılım kısmını gerçekleştirip bir çıktı elde ediyoruz ve bu çıktı bizim gerçek çıktıyla doğru mu diye karşılaştırıyoruz böylece de modelimizin başarını oranını buluyoruz.

## Gradient Decent nedir nasıl çalışır? Ne gibi çeşitleri (batch gradient descent, stochastic, gradient descent veya minibatch gradient descent gibi) vardır ve özellikleri nedlerdir?

Gradient Descent, bir fonksiyonun minimumunu bulmamızı sağlayan bir optimizasyon algoritmasıdır.Gradient Descent ,her veri seti yinelemesinden sonra ağırlıkları küçük artışlarla değiştirerek çalışır.

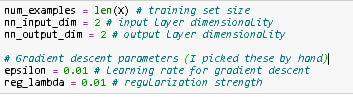
Batch Gradient Descent ise tek bir adım atmak için tüm eğitim verileri dikkate alınır.Tüm eğitim örneklerinin degradelerinin ortalamasını alıp parametrelerimizi güncellemek için bu ortalama gradyanı kullanıyoruz.Yani bu bir epochdaki sadece bir adım.

Stochastic Gradient Descent : Batch Gradient Descent in her adımı için tüm örnekleri ele alıyoruz.Fakat datasetimiz çok büyükse bu büyük veriyi daha optimal kullanarak daha iyi bir model elde edebiliriz. Stochastic Gradient Descentde her adımda tek bir örneği ele alıyoruz.

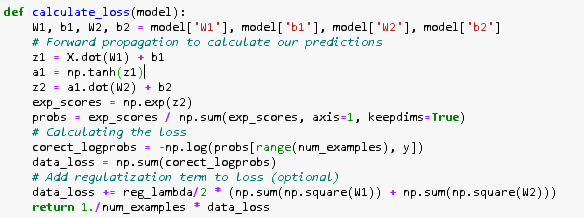
Mini Batch Gradient Descent : Stochastic Gradient Descent 'de her seferinde sadece bir örnek kullandığımızdan, üzerine vektörleştirilmiş uygulamayı uygulayamayız. Bu hesaplamaları yavaşlatabilir. Bu sorunu çözmek için Batch Gradient Descent ve SGD karışımı kullanılır. Ne tüm veri kümesini aynı anda kullanıyoruz ne de tek seferde tek örneği kullanıyoruz. Gerçek veri kümesinden daha az olan sabit sayıda eğitim örneğinden oluşan bir toplu iş kullanırız ve buna Mini Batch Gradient Descent diyoruz.

## Verilen programın build\_model, calculate\_loss ve predict gibi metotlarını ve önemli değişkenlerini kısaca açıklayınız.

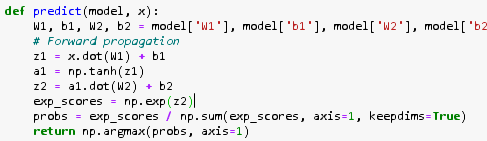
Burada Sinir Ağımız için kullanıcağımız değişkenleri tanımlıyoruz. Num\_examples bizim verimizin sayısı, input\_dim verdiğimiz verimizin boyut sayısı, output dim ise alıcağımız çıktının boyut sayısı.Ek olarak epsilon değerimiz bizim öğrenme oranımız, öğrenme oranımızın çok olması modelin hızlıca eğitilmesini, en minimum kayıpla elde edilebilecek modelin hızlı adım atılmasıyla geçilmesine yol açabilir, öğrenme oranının çok az olması ise modelimizin minimum kayıba ulaşması için gereken zamanın çok uzun sürmesine yol açabilir.reg\_lambda değerimiz ise aşırı öğrenmeyi azaltmak için parametre değerlerinin büyüklüğünü artırmak için bir ceza uyguluyor.



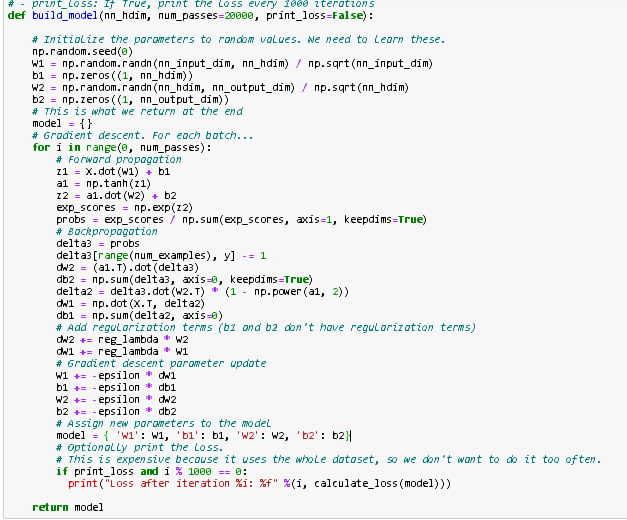
Modelimizin ne kadar iyi sonuç verdiğini, tasarlanan modelin hata oranını aynı zamanda başarımını ölçen fonksiyondur. Derin ağların son katmanı loss fonksiyonun tanımlandığı katmandır.



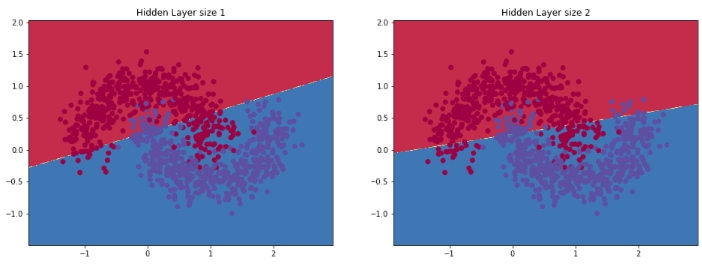
Modelimizin çıktısından en yüksek olasılığa sahip sınıfı seçmesine yarayan fonksiyonumuz ise predict.Gelen outputdan en yüksek olasılığa sahip sınıfı seçiyor.

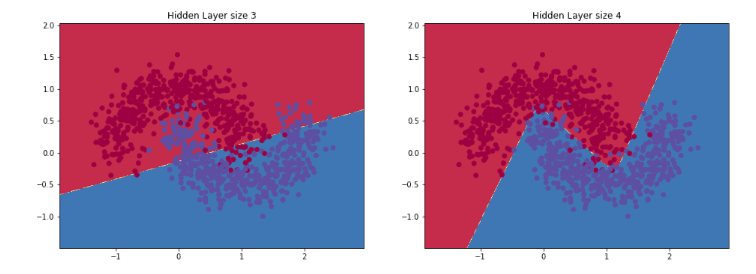


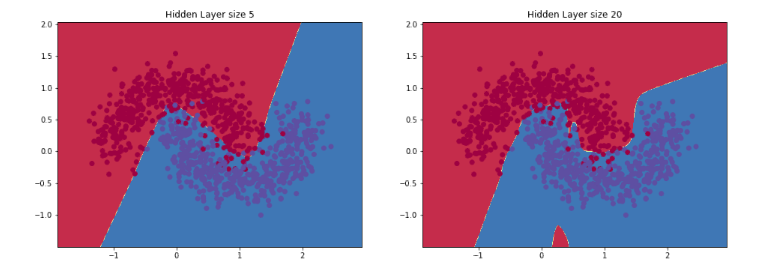
Modelimizi kurmak ve eğitmek için oluşturduğumuz fonksiyonumuz, bulunmuş türevleri kullanarak batch gradient descent kullanarak ileri yayılım, geri yayılım, parametre güncelleştirme işlemlerini gerçekletiriyor.num\_passes sayımız kaç kere eğitmemizi sağlayan içerideki döngüyü çalıştırıyor, a bizim nöronlarımızın ağırlığı, b bias değerimiz bu değerleri her seferinde ileri yayılımla hesaplayıp bu hesapladığımız değere göre geri yayılımda güncelliyoruz ve num\_pass sonucunda eğitebildiğimiz modeli eğitiyoruz.

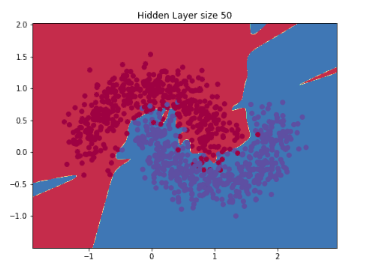


## d) Gizli katman sayısı arttırıldıkça ne oluyor? Bu ikili sınıflandırma için iyi midir, yoksa kötü müdür? Tam uygunluk (Overfitting) ile ilişkisi nedir? Raporunuza çıktılar koyarak bu soruları açıklayınız.





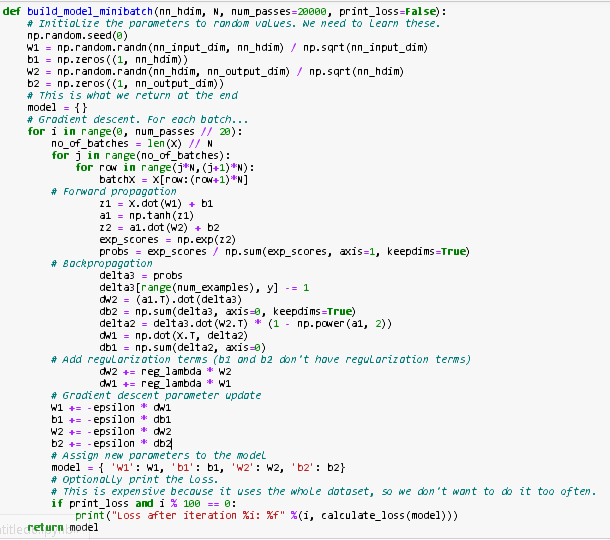




5 farklı gizli katman sayılı sinie ağı çalıştırmamızdan sonra görüyoruz ki belli bir sayıdan sonra sinir ağımız o classa ait olma olasılığı az da olsa bazı özel bölgeleri özel classlara atıyor.50 hidden layerda görüyoruz ki bazı bölgeler overfittingden dolayı diğer bölgeye ait işaretlenmiş.Burdan görüyoruz ki doğru layer sayısını seçmek öğrenme aşaması için çok önemli.

## Programda kullanılan Batch Gradient Decent yerine, ağı eğitmek için minibatchGradient Decent nasıl kullanılır. Kod örneği ile raporunuzda açıklayınız. Bu yöntem genellikle pratikte daha iyi performans gösterir.

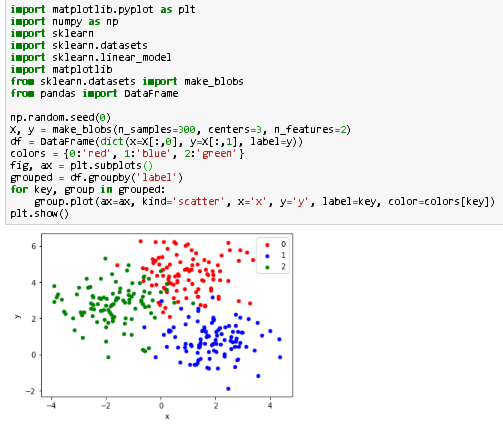
Kodumuzda küçük değişiklik yaparak verimizi tüm seferde vermek yerine belirlediğimiz oranda parça parça vererek daha fazla başarı oranı edebiliyoruz. Fakat bu modelimiz diğer modelimize oranla çok daha yavaş çalışıyor. Yani performanstan kazanırken zamandan feda etmiş oluyoruz.



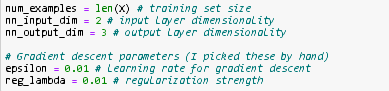
## f) Yukarıda verilen örnekteki verileri bu sefer 3 sınıfa (mesela aşağıdaki kırmızı, mavi ve

## yeşil örnekler gibi) ayıracak şekilde genişletin. Bunun için uygun yeni bir veri kümesi oluşturmanız gerekir. Kodda gereken değişiklikleri yaparak benzer sonuçlar için bulduğunuz sonuçları açıklayın.

Bunun için bize verilen veri oluştur fonksiyonunu oluşturuyoruz.Verimiz aşağıda:

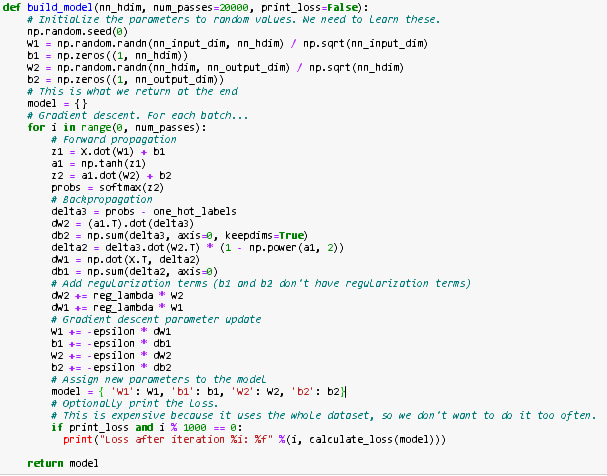


Önceki 2 sınıflı modelimize bu değişiklikleri yapıp 3 sınıf için çalışır hale getirmemiz lazım bunun için:

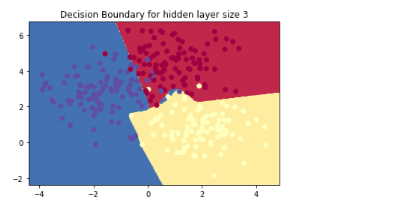


İnput boyutumuz aynı fakat outputumuz bu sefer 3 farklı sınıf için o sınıflara ait olasılık yüzdesi vericek.Fakat bunun için de softmax() fonksiyonunu kodumuza eklememiz gerekiyor.





Burada probs = softmax(z2) kısmında yaptığımız değişiklikten sonra modelimiz birden fazla sınıflar için de çalışır durumda.

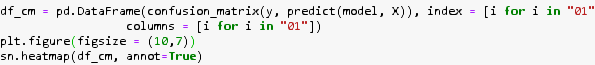


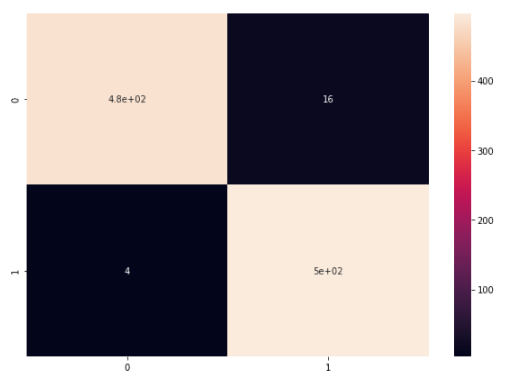
Gördüğümüz gibi sinir ağımız 3 sınıf da olsa gayet güzel bir biçimde çalışır durumda.

# Neural Networks: Python ve Rapid Miner Studio (RMS) Sonuçları

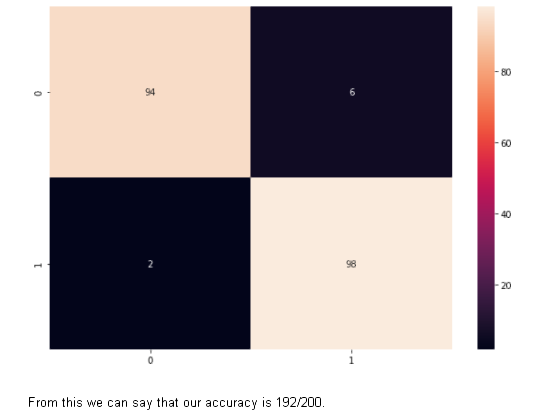
## Yukarıda verilen Python kodunda önce 500 kırmızı ve 500 mavi eğitim verileri kullanarak ikili sınıflandırma için bir sinir ağı modeli oluşturun. Sonra scikit-learn’in desteğiyle Confusion Matrix oluşturmak için gereken ek satırları yazarak bu matrisi oluşturun. Kodunuzu ve çıktıyı raporunuza koyarak kısaca açıklayınız. Önce 100’er tane mavi ve kırmızı test versini üretiniz ve bu test verilerinden sınıflandırma performansını bulunuz.

Eğitim verimizde confusion matriximizi oluşturuyoruz.Bunun için y ve predict(model,X) verileri bize yeterli olucaktır.





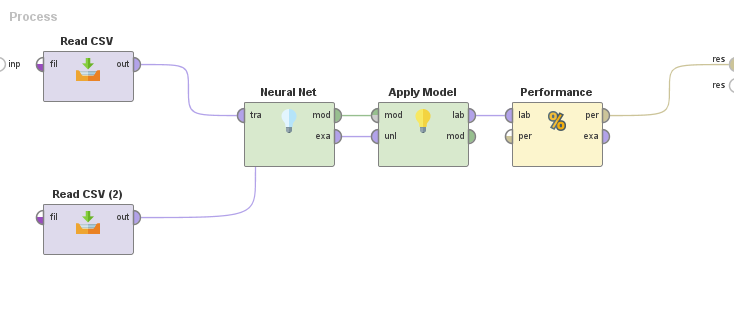
Bir de test verimizle confusion matriximizi test edelim.



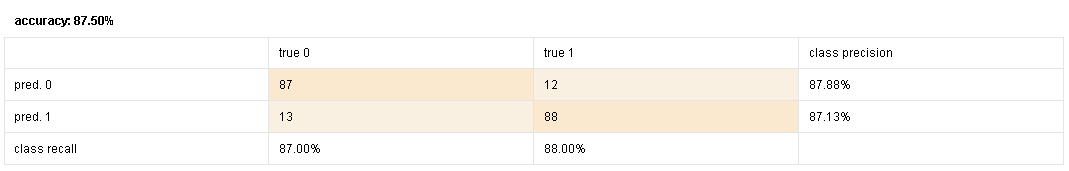
Modelimiz görmediği veri setinde 192/200 ile %96 başarı oranı elde etti.Burdan görebiliyoruz ki bu tür sınıflandırma problemlerinde doğru parametreler ile seçilmiş Sinir Ağları bize güzel sonuçlar verebiliyor.

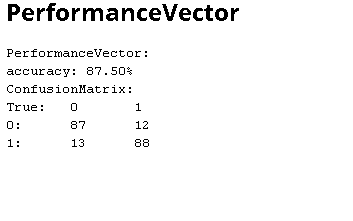
## Yukarıdak programdaki 500 kırmızı ve 500 mavi eğitim verileri ve RMS’da sinir ağı operatorü kullanarak bir sinir ağı modeli oluşturun. Daha sonra bu modelin 100 kırmızı ve 100 mavi test verileri üzerindeki performansını bulup yukarıda Python koduyla elde edilen sonuçlarla raporunuzda karşılaştırınız.

RapidMiner kullanarak yukarıda yaptıklarımızı yapalım.Rapidminer process diagramımız şöyle:



Yukarıdaki CSV dosyamız eğitim için, alttaki ise test için. Eğitimi Neural Nete bağlayıp testi ise Apply Modele bağlıyoruz bu çıktıyı da Performance’ye bağlıyoruz.Elde ettiğimiz sonuçlar:



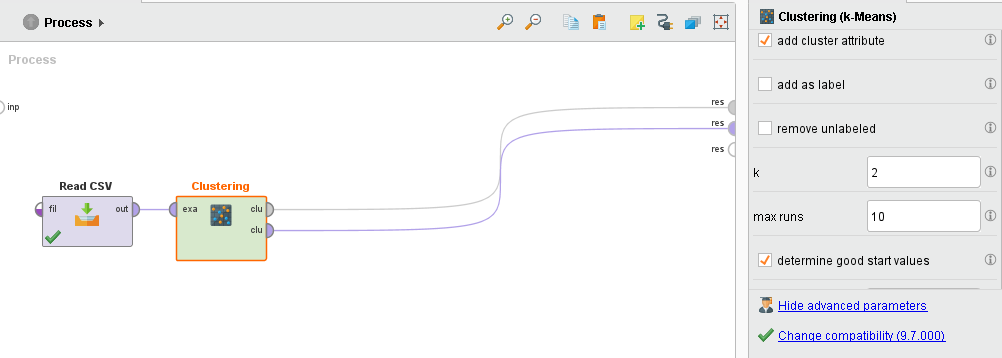


Python koduyla elde edilen performance vector, confusion matrix ile RapidMinerdan elde ettiğimiz performance vektor, confusion matrixi karşılaştırdığımızda görüyoruz ki pythonda elde ettiğimiz model ile daha iyi bir başarım elde etmişiz.Bunun sebebi Pythonda kullandığımız parametreler.Pythonda %96 başarı elde ederken RapidMinerda %87.5 başarım oranı elde etmişiz.Burdan Pythonda yukarıda bahsettiğimiz parametrelerin ne kadar önemli olduğunu ve tüm projelerde dikkat edilerek seçilmesi gerektiğinin, eğer dikkat edilirse çok daha iyi modeller elde edilebileceğini , dikkat edilmezse ise modelimiz kendi sınırına çıkamayacağını görmüş olduk.Bu ve bundan sonraki tüm projelerde dikkat etmemiz gerektiğini anladık.

# Clustering / Outlier Analysis

## RMS’da Clustering (k-Means) operatörü kullanarak yukarıda 2 farklı veri örneklerinden oluşmuş veri setini 2 farklı kümeye ayıran, kırmızı ve mavi noktalar olarak scatter çizim (plot) ile gösteren çıktıyı üretiniz. Proses diyagramını raporunuza koyarak kullandığınız parametrelerle beraber kısaca açıklayınız.

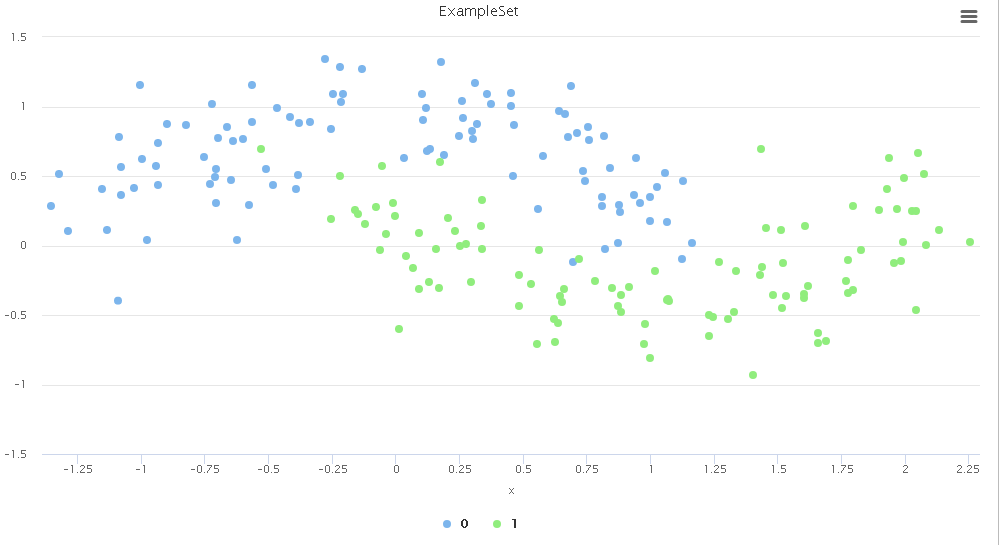
Processimiz:



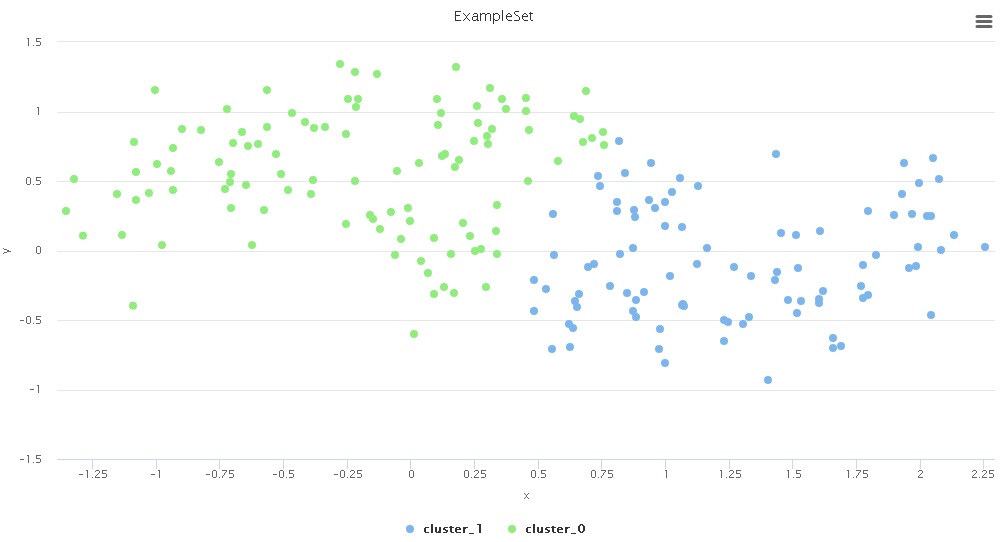
Parametreler :

* K = 2(2 ayrı cluster istiyoruz 2 farklı sınıf var bu yüzden k = 2)
* Measure types = BregmanDivergences(default)
* Max optimization steps = 100(default)
* Divengerce = Squared Euclidean Distance(uzaklığa göre gruplandırma)

Original n = 200



Clustered k = 2

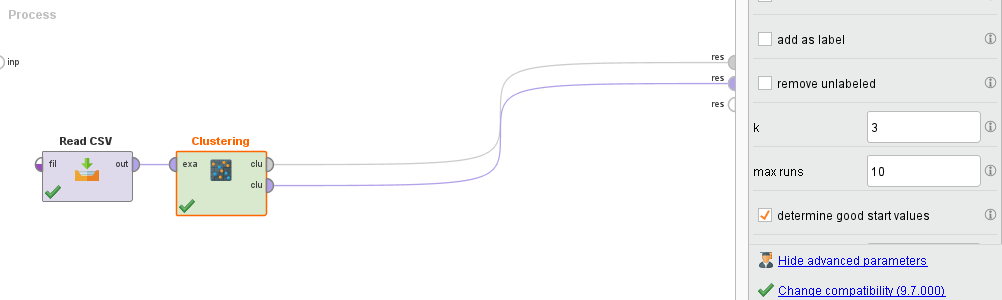


Gördüğümüz üzere bazı noktalar normal durumumuzda farklı bir sınıftayken clusterdan sonra diğer sınıfa ait şekilde clusterlanmış.Bunun sebebi bu gruplandırmayı uzaklığa bağlı bir şekilde yapıyor olmamız.

## RMS’da Clustering (k-Means) operatörü kullanarak yukarıda 3 farklı veri örneklerinden oluşmuş veri setini 3 farklı kümeye ayıran, kırmızı, mavi ve yeşil noktalar olarak scatter çizim (plot) ile gösteren çıktıyı üretiniz. Proses diyagramını raporunuza koyarak kullandığınız parametrelerle beraber kısaca

## açıklayınız.

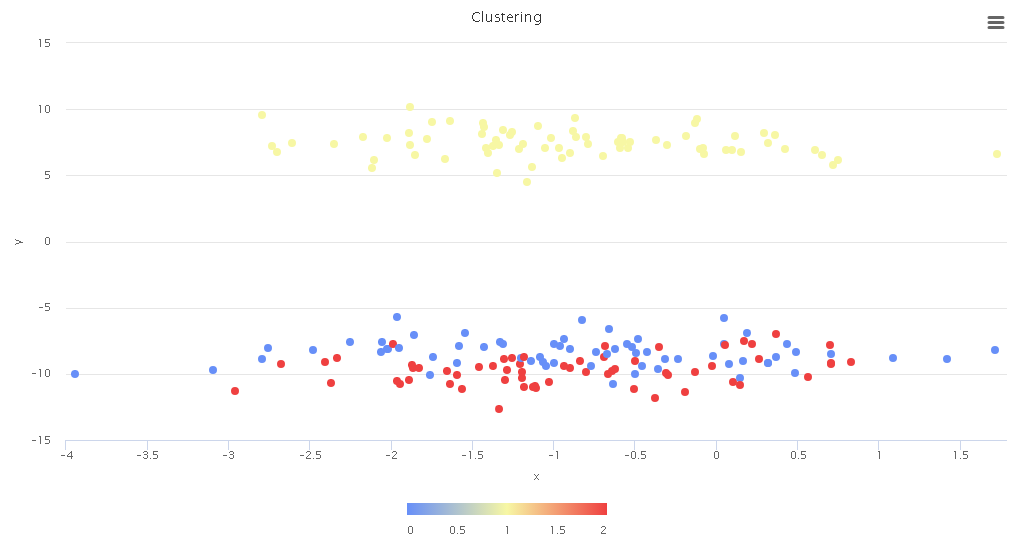
Processimiz :



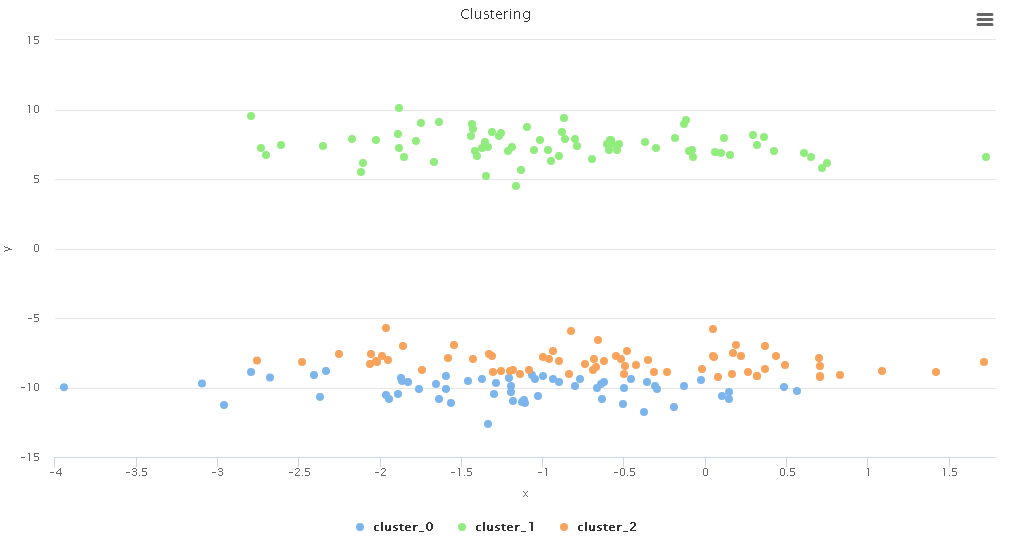
Parametrelerimiz:

* K = 3(3 ayrı cluster istiyoruz 3 ayrı sınıf var bu yüzden k = 3)
* Measure types = BregmanDivergences(default)
* Max optimization steps = 100(default)
* Divengerce = Squared Euclidean Distance(uzaklığa göre gruplandırma)

Original n = 300



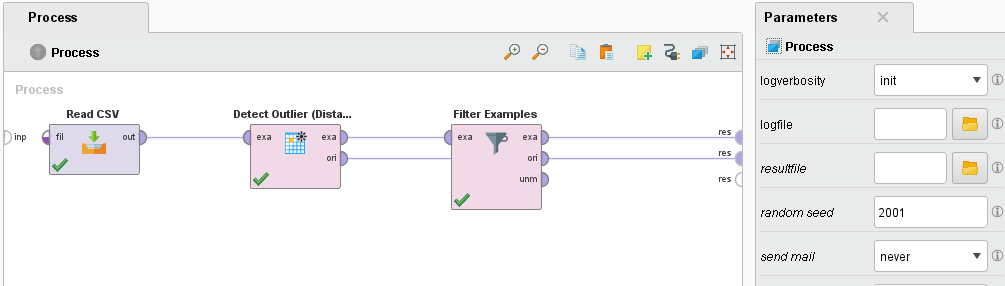
After cluster n = 300



Gördüğümüz üzere bu örnekte özellikle 2 sınıf çok bitişik olduğu için onlar arasında birbirlerinin bölgelerinde olan noktalar diğer cluster’a dahil edilmiş fakat üstteki sınıf bundan etkilenmemiş çünkü programımız bu cluster işlemini uzaklığa bağlı olarak yapıyor.

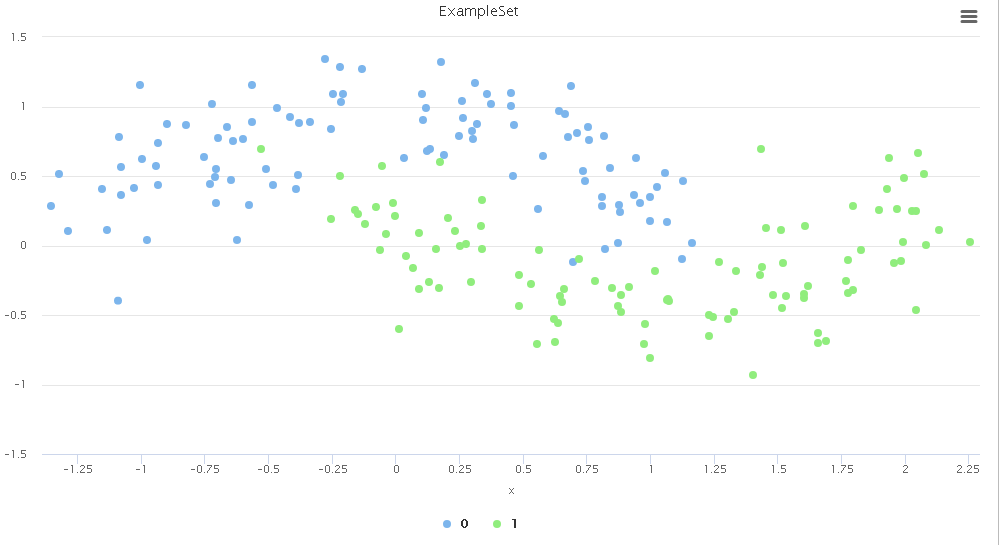
## Yukarıda 2 farklı veri örneklerinden oluşmuş veri setinindeki aykırı (outliers) verileri RMS’da ilgili operatörleri kullanarak tespit edin ve temizleyiniz. Proses diyagramını raporunuza koyarak kullandığınız parametrelerle beraber kısaca açıklayınız.

Process for 2 class:

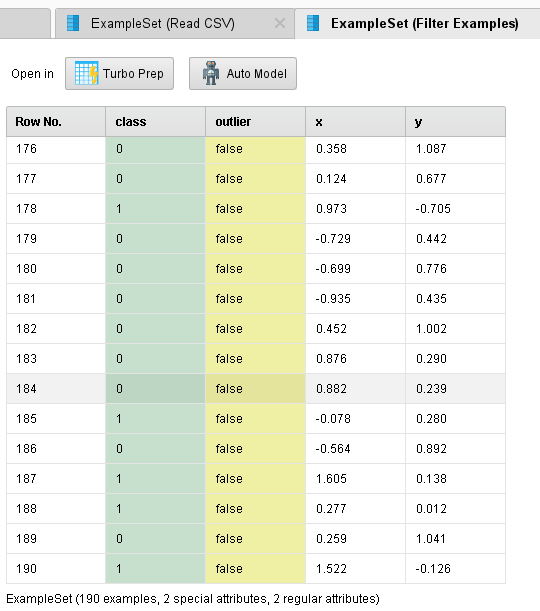


Aykırı değerleri tespit edebilmek için Parameters:

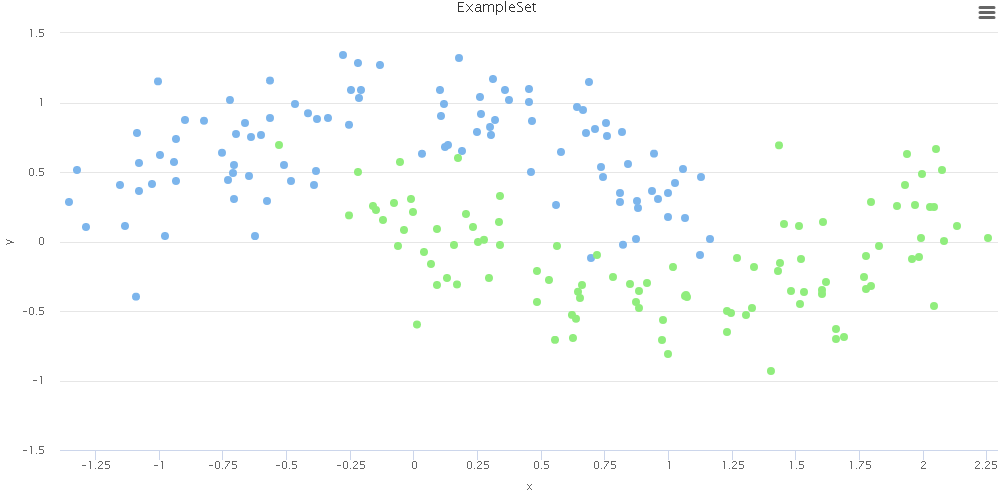
* Number of neighbors = 10(bir noktanın 10 adet komşusu)
* Number of outliers = 10
* Distance function = Euclidian distance(mesafe hesabı için)
* Condition class = attribute value filter(outlierları tespit için)
* Parameter string = outlier = True(outlierları tespit için)
* İnvert filter(outlierları tespit için)

Our original dataset

After finding and removing outliers



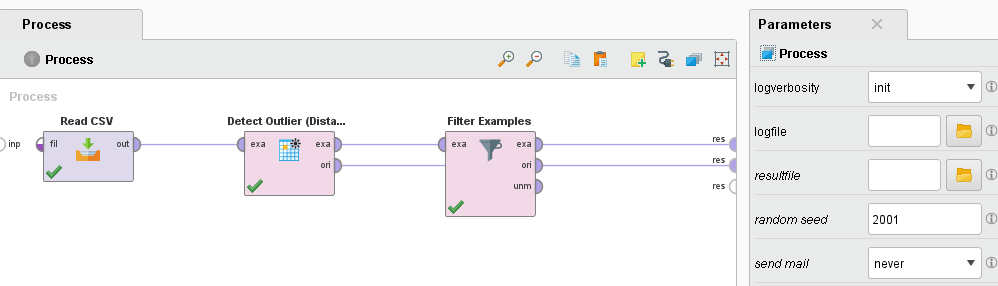
After removing the outliers



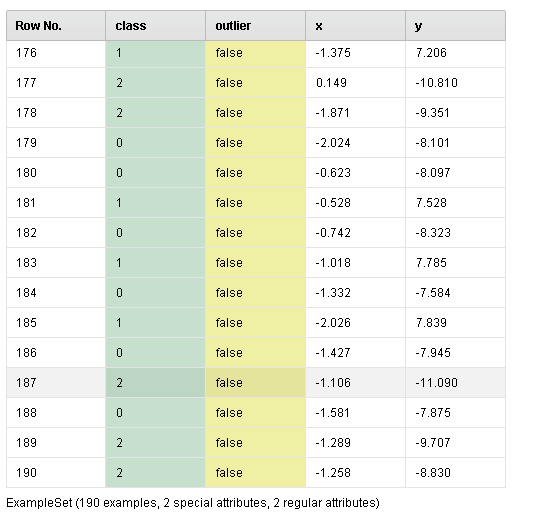
## Yukarıda 3 farklı veri örneklerinden oluşmuş veri setinindeki aykırı (outliers) verileri RMS’da ilgili operatörleri kullanarak tespit edin ve temizleyiniz. Proses diyagramını raporunuza koyarak kullandığınız parametrelerle beraber kısaca açıklayınız.

Process for 3 class:

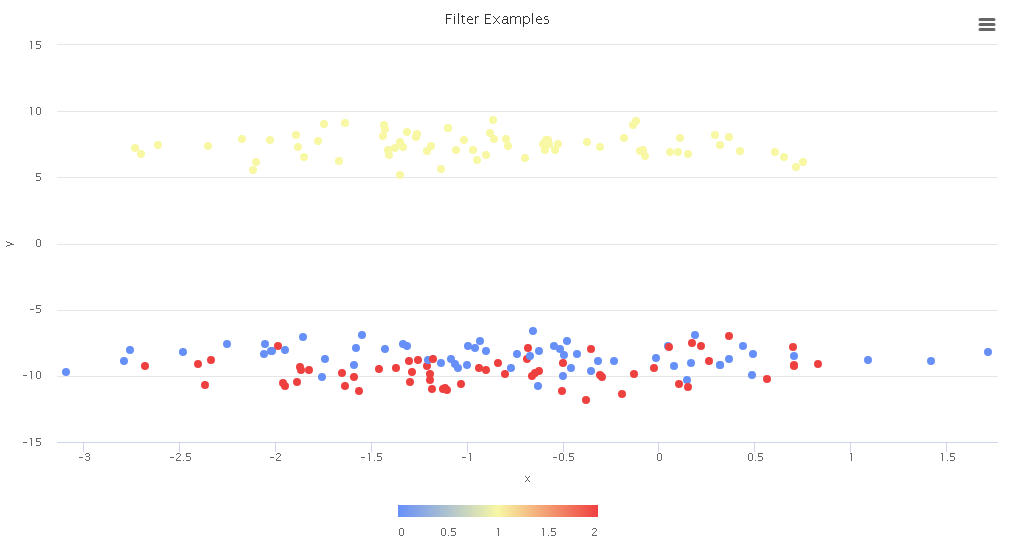
* Number of neighbors = 10(bir noktanın 10 adet komşusu)
* Number of outliers = 10
* Distance function = Euclidian distance(mesafe hesabı için)
* Condition class = attribute value filter(outlierları tespit için)
* Parameter string = outlier = True(outlierları tespit için)
* İnvert filter(outlierları tespit için)



Outliers removed



After removing the outliers



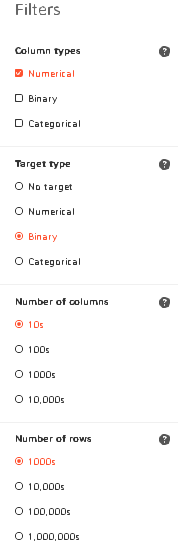
# Makine Öğrenmesi Algoritmaları

## Yukarıdaki 2 farklı veri örneklerinden oluşmuş veri seti için en başarılı modeli üreten algoritmayı nasıl bulursunuz? Yukarıda Logistic Regression kullanıldı. Bu algoritmayla bulunan sonucu Neural Network kullanılarak bulunan sonuçla karşılaştırınız. Linear Regression kullanılsaydı ne olurdu?

En başarılı algoritmayı bulmak için iki algoritmanın da daha önce görmedikleri veri setinde verdiği sonuçları göz önünde bulundurarak karar vermek en mantıklısı olur. Bu durumda %87.5 başarı oranına sahip Logistic Regression , %96 başarı oranı veren Neural Network’ün gerisinde kalır.Genel olarak eğitim başarısını değil de Test başarısını referans almalıyız çünkü eğitim başarısı overfitting durumunda n dolayı yanıltıcı olabilir. Bu tür iç içe geçmiş hilallerden oluşan bir veri setinde linear regression başarılı olamazdı ve büyük ihtimalle başarı sıralamasında 3. Sıraya yerleşirdi.Bunun sebebi ise Linear Regressionun genel olarak düz bir çizgi çizmesi ve bu bizim data setimiz için hiç de uygun bir yol değil.

## RMS için geliştirilmiş aşağıdaki adrese giderek, 500 kırmızı ve 500 mavi eğitim verilerinden oluşmuş veri setine uygun filtreleri seçerek, tavsiye edilen makine öğrenmesi algoritmalarına bakınız:<https://mod.rapidminer.com/> Tavsiye edilen ilk 5 algoritma nelerdir?

Filtreler:

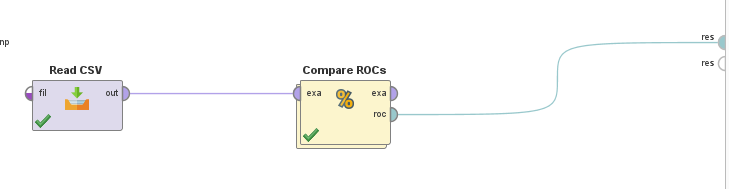
:

Tavsiye Edilen Top 5 Model:

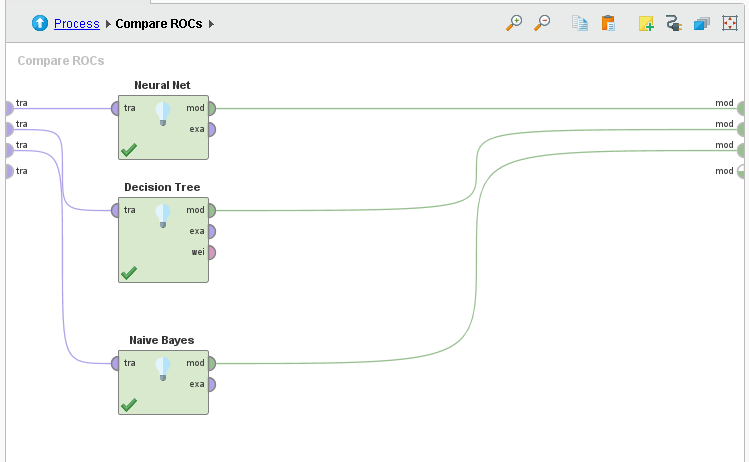


## RMS içinde Compare ROCs operatörü ve sinir ağı ve tavsiye edilen diğer 2 makine öğrenmesi algoritmasını da kullanarak, ROC karşılaştırma grafiğini elde ediniz ve raporunuza koyunuz.

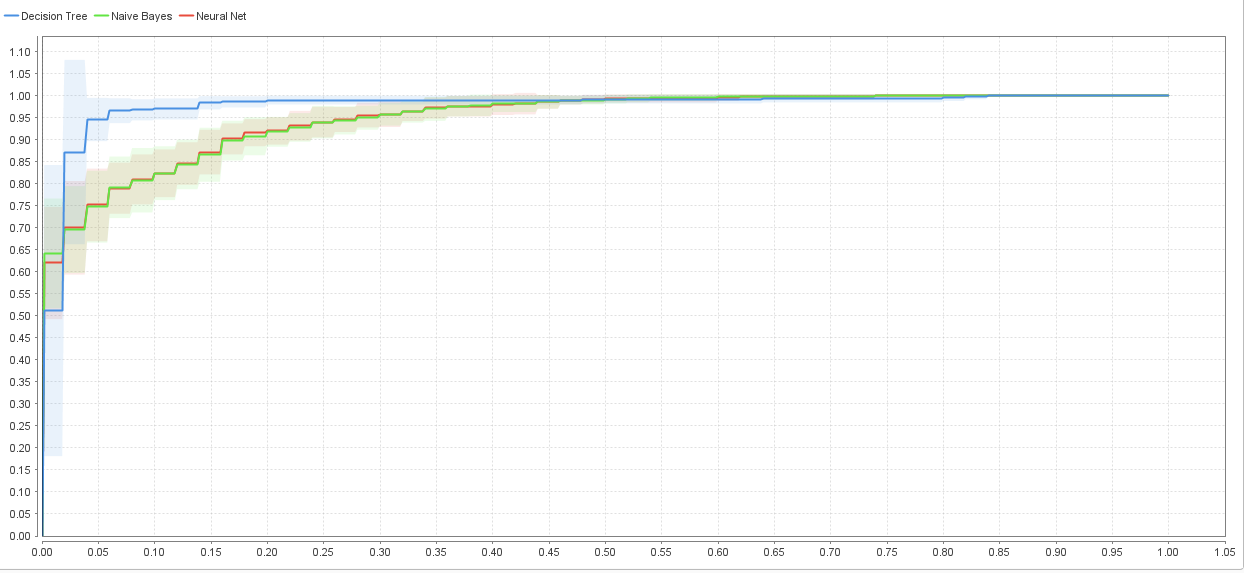
Processimiz:



Compare ROCs kısmının için:



En yüksek 3 başarı oranına sahip 3 modeli test etmek istedim bu yüzden Neural Net, Desicion Tree ve Naive Bayes algoritmalarını koydum.



## ROC grafiği ne gösterir, önemi nedir ve nasıl kullanılır? Yukarıda (c) şıkkında elde etmiş olduğunuz ROC grafiğini yorumlayınız. Bu grafik yukarıda (a) şıkkında elde ettiğiniz sonuçla uyumlu mu?

Makine öğrenmesinde performans ölçümü önemli bir konudur.Bu yüzden ROC grafiği makine öğrenmesi algoritmalarının performansını değerlendirmek için en yaygın kullanılan ölçümlerden biridir. Ve modelin tahmininde ne kadar iyi olduğunu açıklar.ROC eğrisinde farklı modellerin eğrileri genel olarak veya farklı eşikler için doğrudan karşılaştırılabilir.Çizginin altında kalan alan ne kadar büyükse modelin başarı oranı o kadar yüksek demektir. Genellikle, başarılı modeller arsanın sol üst kısmına eğilen eğriler ile temsil edilir.Burada Roc grafiğine bakarsak Desicion Tree modelimizin altında kalan alan diğerlerinden fazla, Naive Bayes ile Neural Network de bazı yerlerde Naive Bayes, bazı yerlerde Neural Network daha önde buradan ikisi hakkında yorum yapabilmek zor fakat başlangıça baktıgımızda Naive Bayes biraz daha yukarıdan başlamış fakat en sonunda 2 si de eşit bir biçimde sonuca ulaşmış.A şıkkında Neural Networkün daha iyi olduğunu bulmuştuk fakat burda görebiliyoruz ki bu problem için Desicion Tree modeli Neural Networkden daha iyi bir sonuç verebikirmiş.

## e) Derste sunulan 6 tane sınıflandırma algoritmasını karşılaştırınız. Aşağıda başlamanız için bir tabloörneği ve 4 tane önemli özellik solda verilmiştir. Önemli başka 6 tane özellik daha sol baş sütuna yazarak bu tabloyu doldurunuz. Planlanan ancak derste sunulamayan SVM ve Naïve Bayes sütünlarına birşey yazmayınız. Tablonuzu doldurdukdan sonra, kısaca cümlelerinizle de 6 algoritma için bir karşılaştırma paragrafı yazınız.

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | Karar Ağaçları | Kural Tabanlı Sınırlandırıcılar | K-NN | Doğrusal Regresyon | Logictik Regresyon | Sinir Ağları | SVM | Naive Bayes |
| Doğruluk | Yüksek | Orta | Yüksek | Orta | Orta | Yüksek | Orta | Yüksek |
| Öğrenme Hızı(özelliklerine ve örneklerin sayısına göre) | N | N | Eğitim Yok | N | N | Epoch, batche göre değişir.(N den fazla) | N | N |
| Sınıflandırma Hızı | Log(N) | Log(N) | K | 1 | 1 | 1 ileri yayılım | 1 | 1 |
| Model Üretme | Hızlı | Hızlı | Hızlı | Hızlı | Hızlı | Yavaş | Hızlı | Orta |
| Logaritmik kayıp | Az | Az | Az | Orta | Orta | Orta | Az | Az |
| Veri İhtiyacı | Orta | Orta | Fazla | Orta | Orta | Fazla | Orta | Fazla |
| Karmaşıklık | Orta | Orta | Fazla | Orta | Orta | Fazla | Fazla | Orta |
| Zorluk | Basit | Basit | Basit | Basit | Orta | Zor | Zor | Orta |