YILDIZ TEKNİK ÜNİVERSİTESİ

ELEKTRİK ELEKTRONİK FAKÜLTESİ

BİLGİSAYAR MÜHENDİSLİĞİ

BLM 4510 YAPAY ZEKA DERSİ

ÖDEV 2 RAPORU

20011023 – Mehmet Alperen Ölçer
19011038 – Şevval Bulburu

alperen.olcer@std.yildiz.edu.tr
sevval.bulburu@std.yildiz.edu.tr

Verilen proje kapsamında Mackolik web sitesinden Süper Lig futbolcularıyla ilgili bilgileri web scraping yöntemiyle elde edilecektir. Eski versiyon Mackolik web sitesi farklı yıllar için futbolcuların piyasa değerlerini içerirken, yeni versiyon her yıl ve her futbolcu için istatistiksel verileri içermektedir. Futbolcuların piyasa değerlerinin bulunduğu ardışık iki yılı belirleyecek ve bu yıllar arasındaki istatistiksel verilerden (pozisyon, boy, kilo, top çalma oranı, oynama süresi, gol sayısı vb.) ilgili bilgileri çıkaracağız. Her bir futbolcu için yaklaşık olarak 10 ila 15 adet bilgi kullanılacak ve bunlar son bir tabloya dönüştürülecektir.

Ana hipotez ve araştırma soruları:

Her yıl, futbolcuların piyasa değerleri performanslarına bağlı olarak değişmektedir. Bir sezon içinde, futbolcu istatistikleri her maçtan sonra değişmektedir. Bunun arkasındaki fikir, bir futbolcunun sezon boyunca aynı performans seviyesini sürdürdüğünde potansiyel piyasa değerindeki artış veya azalış hakkında oyuncuların veya menajerlerin varsayımlar yapabilmesini sağlamaktır.

Yöntemler:

Veri seti bir tablo olarak oluşturulduktan sonra, her bir futbolcuya fiyatlarının sezonlar boyunca artıp azalmış olup olmadığını belirten bir etiket atanacaktır. Çeşitli denetimli öğrenme yöntemleri uygulanacak ve performans metrikleri analiz edilecektir. Tahminleyici algoritmaları olarak Naïve Bayes, Karar Ağaçları, Random Forest, , K-En Yakın Komşular (KNN) ve Gradient Boosting algoritmalarının kullanılması kararlaştırılmıştır. Bu yöntemleri uygulayarak, performans istatistiklerine dayanarak futbolcuların gelecekteki piyasa değerlerinin trendlerini tahmin etmeyi ve her bir modelin etkinliğini değerlendirmesi amaçlanmaktadır.

DATASET PREPARATION

Verisetini oluşturma adımları:

- **İnternetten ham veri elde etme.**
  - **Maçkoliğin eski sitesinden süperlig'te oynamış ve oynayan oyuncuların piyasa değerlerini web scrapping ile kaydetme.**
    - player_market_value adında bir fonksiyon tanımlanır. Bu fonksiyon, futbolcuların piyasa değerlerini çeker.
    - url değişkeni, futbolcuların piyasa değeri verilerini içeren web sayfasının URL'sini temsil eder.
    - Bir döngü başlatılır ve futbolcuların piyasa değerlerini içeren web sayfasına istekler gönderilir. Her sayfa için gerekli işlemler gerçekleştirilir.
    - Web sayfasının HTML içeriği çekilir ve BeautifulSoup kullanılarak bu içerik parse edilir.
    - Tablo verisi bulunur ve tablonun satırları alınır.
    - Her satır için futbolcunun detaylarını içeren sayfaya istek gönderilir ve gerekli işlemler gerçekleştirilir.
    - Futbolcunun adı ve piyasa değerlerine ilişkin veriler bir sözlükte toplanır.
    - İşlem tamamlandığında, elde edilen veriler result adlı bir sözlükte saklanır ve döndürülür.
    - player_market_value fonksiyonu çağrılır ve elde edilen sonuç r değişkenine atanır.
    - Elde edilen veriler player_price_data.csv adlı bir CSV dosyasına yazılır. Önce başlık (header) bilgileri tanımlanır, ardından csv.DictWriter kullanılarak veriler satır satır yazılır.
  - **Maçkoliğin yeni sitesinden yıllara göre oyuncuların istatistiklerini içeren sayfaların url'lerini tutma.**
    - HTML içeriği çekilir ve BeautifulSoup kullanılarak bu içerik parse edilir. İstenilen url'ler filtreleme yapılarak elde edilir.
  - **Kaydedilmiş url'ler açıldığında verinin geldiği network ağı incelenerek verinin geldi API özelliklerini kaydetme.**
    - liste_U22_23, liste_U21_22, liste_U20_21, liste_U19_20, liste_U18_19 adlı listelerin birleşimi olan temp oluşturulur.

        - 'players_url_api_all_save.csv' adlı bir CSV dosyasından veri okunur ve url sütunundaki değerler readed_urls listesine atanır.
        - player_page_urls listesi, temp listesindeki değerler arasında readed_urls listesinde bulunmayan URL'leri içerir.
        - Browsermob-proxy sunucusu başlatılır ve bir proxy oluşturulur.
        - Chrome sürücüsü proxy ile yapılandırılır.
        - Her bir URL için aşağıdaki adımlar gerçekleştirilir:
            - URL'nin bölümlerine ayrılır ve son bölüm çıkarılarak modified_url oluşturulur.
            - BeautifulSoup kullanılarak HTML içeriği çekilir ve parse edilir.
            - Futbolcu adı alınır.
            - Ağ trafiğini yakalamak için proxy ayarları yapılır.
            - Sayfa yüklenir.
            - Yakalanan ağ trafiği elde edilir ve en son API URL'si (url_api) bulunur.
            - Bulunan veriler result sözlüğüne eklenir.
        - players_url_api_all.csv adlı bir CSV dosyası oluşturulur ve elde edilen veriler satır satır yazılır.
    - **Birleşmiş oyuncu-ful-adı, istatistik sayfası url'i ve url'e has API bilgisinin kayıtlı olduğu tablodan faydalanarak elde edilebilen tüm verilerin elde edilmesi.**
        - İlk olarak, gerekli kütüphaneler (csv, json, BeautifulSoup, requests, pandas, urllib) import edilir.
        - add_to_dict_from_response fonksiyonu bir HTTP yanıtından verileri çıkarır ve bir sözlüğe ekler.
        - get_player_id fonksiyonu, bir futbolcunun bilgilerini ve istatistiklerini bir sözlüğe ekler.
        - get_player_statics fonksiyonu, futbolcuların istatistiklerini çekmek için get_player_id fonksiyonunu kullanır.
        - read_player_url_apis fonksiyonu, 'players_url_api.csv' adlı bir CSV dosyasından futbolcu URL'leri ve API URL'lerini okur ve bir sözlük olarak döndürür.
        - read_player_url_apis fonksiyonu kullanılarak url_apis değişkeni, futbolcu URL'leri ve API URL'lerini içeren bir sözlüğe atanır.
        - get_player_statics fonksiyonu kullanılarak r değişkeni, futbolcuların istatistiklerini içeren bir sözlüğe atanır.
        - 'player_statistics.csv' adlı bir CSV dosyası oluşturulur ve başlık satırı yazılır.
        - Futbolcu isimleri üzerinde döngü oluşturulur ve her futbolcu için istatistik değerleri sözlüğe eklenir.
    - **Son ham veri tablosunda piyasa değeri bulunmayanların el ile araştırılarak doldurulması.**
- **Verileri tek bir tabloda düzenlemek:**
    - **247 oyuncu ve 125 adet özellik içeren ilk tablo üzerinden bilgisi çok az olan 15 oyuncunun çıkarılması.**
    - **125 özelliğin incelenmesi ve indirgemelerin yapılması:**

- Alaka düzeyine göre bazıları elendi.
    - Örneğin oyuncuların nationality, backward passes gibi özellikleri çıkarıldı.
- Çok eksik veri içerenler silindi.
- Alakalı olup bazı eksikleri olanlar internetten bakılıp dolduruldu.
- Alakalı olup bazı eksikleri olanlar mod, medyan veya ortalamaya bakılarak dolduruldu.
- Birbiri ile ilişkili olanlar işleme sokulup tek ayırıcı özelliğe dönüştürüldü.
    - Toplam pas sayısı, başarılı pas sayısı, başarısız pas sayısı vb. ilişkili özellikler işleme sokularak başarılı pas oranı ve toplam başarılı pas sayısı olmak üzere iki özelliğe indirgendi.
    - Duel sayısı, kazanılan ve kaybedilen duel sayıları ile ilişkili özellikler işleme sokularak duel başarı oranı ve duel sayısı olmak üzere iki özelliğe indirgendi.

- **String'ler sayısal değerlere dönüştürüldü.**
    - Label, posision ve foot özelliklerinin string değerleri integer değerlerle temsil edilecek şekilde düzenlendi.

TAHMİNLEYİCİLERİN OLUŞTURULMASI:

10 katlı çapraz geçerleme sonuçları elde edilmesi için cross validation kullanılmıştır.

Projede tahminleyici olarak aşağıdaki algoritmalar kullanılmıştır:

- Naive Bayesian Algorithm
- KNN
- Decision Tree
- Random Forests
- Gradient Boosting

Ilk önce normalizasyon ve özellik dönüşümü uygulanmadan veriseti bu makine öğrenmesi algoritmalarına sokulmuştur. Elde edilen skorlar aşağıdaki gibidir:

```
# Naive Bayes classifier

# Create a Naive Bayes classifier
nb_classifier = GaussianNB()

# Perform cross-validation
NB_cv_scores = cross_val_score(nb_classifier, X, y, cv=10, scoring='accuracy')

# Print the cross-validation scores
print("Cross-validation Scores:")
print(NB_cv_scores)
print("max:",NB_cv_scores.max(), '\nmin:',NB_cv_scores.min(), '\nmean:',NB_cv_scores.mean())
```

```
Cross-validation Scores:
[0.75       0.69565217 0.65217391 0.65217391 0.73913043 0.69565217
 0.73913043 0.60869565 0.60869565 0.65217391]
max: 0.75
min: 0.6086956521739131
mean: 0.6793478260869565
```

```
# Create a KNN classifier
knn_classifier = KNeighborsClassifier()

# Perform cross-validation
KNN_cv_scores = cross_val_score(knn_classifier, X, y, cv=10, scoring='accuracy')

# Print the cross-validation scores
print("Cross-validation Scores:")
print(KNN_cv_scores)
print("max:", KNN_cv_scores.max(), "\nmin:", KNN_cv_scores.min(), "\nmean:", KNN_cv_scores.mean())
```

```
Cross-validation Scores:
[0.41666667 0.43478261 0.60869565 0.47826087 0.56521739 0.65217391
 0.73913043 0.60869565 0.69565217 0.52173913]
max: 0.7391304347826086
min: 0.4166666666666667
mean: 0.5721014492753622
```

```
# Create a Random Forest classifier
rf_classifier = RandomForestClassifier()

# Perform cross-validation
RF_cv_scores = cross_val_score(rf_classifier, X, y, cv=10, scoring='accuracy')

# Print the cross-validation scores
print("Cross-validation Scores:")
print(RF_cv_scores)
print("max:", RF_cv_scores.max(), "\nmin:", RF_cv_scores.min(), "\nmean:", RF_cv_scores.mean())
```

```
Cross-validation Scores:
[0.66666667 0.7826087  0.56521739 0.60869565 0.73913043 0.65217391
 0.30434783 0.65217391 0.60869565 0.65217391]
max: 0.782608695652174
min: 0.30434782608695654
mean: 0.6231884057971014
```

```
# Create a Gradient Boosting classifier
gbm_classifier = GradientBoostingClassifier()

# Perform cross-validation
gbm_cv_scores = cross_val_score(gbm_classifier, X, y, cv=10, scoring='accuracy')

# Print the cross-validation scores
print("Cross-validation Scores:")
print(gbm_cv_scores)
print("max:", gbm_cv_scores.max(), "\nmin:", gbm_cv_scores.min(), "\nmean:", gbm_cv_scores.mean())
```

```
Cross-validation Scores:
[0.66666667 0.73913043 0.69565217 0.56521739 0.73913043 0.56521739
 0.39130435 0.56521739 0.56521739 0.60869565]
max: 0.7391304347826086
min: 0.391304347826087
mean: 0.610144927536232
```

```
# Create a Decision Tree classifier
dt_classifier = DecisionTreeClassifier()

# Perform cross-validation
dt_cv_scores = cross_val_score(dt_classifier, X, y, cv=10, scoring='accuracy')

# Print the cross-validation scores
print("Cross-validation Scores:")
print(dt_cv_scores)
print("max:", dt_cv_scores.max(), "\nmin:", dt_cv_scores.min(), "\nmean:", dt_cv_scores.mean())
```

```
Cross-validation Scores:
[0.58333333 0.69565217 0.52173913 0.60869565 0.7826087  0.60869565
 0.39130435 0.65217391 0.65217391 0.52173913]
max: 0.782608695652174
min: 0.391304347826087
mean: 0.6018115942028985
```

Normalizasyon ve Özellik Dönüşümü Uygulanması

Normalizasyon yapmak için min-max scale tekniği kullanılmıştır. Standard scale tekniği de bazı özellikler için denenmiş fakat tercih edilmemiştir. Özellik dönüşümü için ise PCA kullanılmıştır. PCA'nin component sayısı denemeler sonrası en iyi sonuçla belirlenmiştir.

Normalizasyon öncesi:

	priceA	height	body_mass_index	foot	age	position	Successfull Pass Rate	Total Successful Passes	Duels Rate	Duels	...	Appearances
0	200000000	1.76	22.598	1	28	1	0.822034	97.0	0.428571	21.0	...	5
1	4000000	1.76	23.889	1	33	1	0.849638	469.0	0.492754	69.0	...	14
2	225000	1.79	22.471	1	32	1	0.709677	22.0	0.600000	10.0	...	4
3	1600000	1.78	23.040	1	28	1	0.801136	141.0	0.444444	36.0	...	13
4	1500000	1.86	23.124	1	30	2	0.773389	372.0	0.438095	105.0	...	8
...	...	...	...	...	...	...	...	...	...	...	...	...
226	150000	1.94	22.585	2	24	1	0.862761	1075.0	0.651316	152.0	...	22
227	750000	1.76	19.693	2	26	2	0.794224	220.0	0.477273	132.0	...	23
228	600000	1.89	22.676	3	29	1	0.795888	1084.0	0.657895	228.0	...	30
229	1750000	1.78	22.409	1	31	2	0.856955	653.0	0.522727	88.0	...	21
230	10000000	1.96	23.688	2	35	3	0.529703	107.0	0.437500	112.0	...	9

231 rows × 22 columns

Normalizasyon sonrası:

	priceA	height	body_mass_index	foot	age	position	Successfull Pass Rate	Total Successful Passes	Duels Rate	Duels	...	Appe
0	1.000000	0.354839	0.548668	0.0	0.428571	0.0	0.133025	0.037561	0.229286	0.036437	...	
1	0.019755	0.354839	0.714011	0.0	0.666667	0.0	0.145586	0.187803	0.269239	0.133603	...	
2	0.000875	0.451613	0.532403	0.0	0.619048	0.0	0.081897	0.007270	0.336000	0.014170	...	
3	0.007752	0.419355	0.605277	0.0	0.428571	0.0	0.123515	0.055331	0.239167	0.066802	...	
4	0.007252	0.677419	0.616035	0.0	0.523810	0.5	0.110889	0.148627	0.235214	0.206478	...	
...	...	...	...	...	...	...	...	...	...	...	...	
226	0.000500	0.935484	0.547003	0.5	0.238095	0.0	0.151557	0.432553	0.367944	0.301619	...	
227	0.003501	0.354839	0.176614	0.5	0.333333	0.5	0.120370	0.087237	0.259602	0.261134	...	
228	0.002751	0.774194	0.558658	1.0	0.476190	0.0	0.121127	0.436187	0.372039	0.455466	...	
229	0.008502	0.419355	0.524462	0.0	0.571429	0.5	0.148916	0.262116	0.287898	0.172065	...	
230	0.049762	1.000000	0.688268	0.5	0.761905	1.0	0.000000	0.041599	0.234844	0.220648	...	

231 rows × 22 columns

PCA (Principal Component Analysis)

```
[ ]  pca = PCA(n_components = 5)
     X2 = pca.fit_transform(X)
```

Bu değişiklikler sonrası model sonuçları aşağıdaki gibidir.

```
# Naive Bayes classifier

# Create a Naive Bayes classifier
nb_classifier = GaussianNB()

# Perform cross-validation
NB2_cv_scores = cross_val_score(nb_classifier, X2, y, cv=10, scoring='accuracy')

# Print the cross-validation scores
print("Cross-validation Scores:")
print(NB2_cv_scores)
print("max:",NB2_cv_scores.max(), '\nmin:',NB2_cv_scores.min(), '\nmean:',NB2_cv_scores.mean())
```

```
Cross-validation Scores:
[0.70833333 0.65217391 0.69565217 0.65217391 0.73913043 0.69565217
 0.60869565 0.65217391 0.60869565 0.65217391]
max: 0.7391304347826086
min: 0.6086956521739131
mean: 0.6664855072463769
```

```
# Create a KNN classifier
knn_classifier = KNeighborsClassifier()

# Perform cross-validation
KNN2_cv_scores = cross_val_score(knn_classifier, X2, y, cv=10, scoring='accuracy')

# Print the cross-validation scores
print("Cross-validation Scores:")
print(KNN2_cv_scores)
print("max:", KNN2_cv_scores.max(), "\nmin:", KNN2_cv_scores.min(), "\nmean:", KNN2_cv_scores.mean())
```

```
Cross-validation Scores:
[0.41666667 0.43478261 0.60869565 0.47826087 0.56521739 0.65217391
 0.73913043 0.60869565 0.69565217 0.52173913]
max: 0.7391304347826086
min: 0.4166666666666667
mean: 0.5721014492753622
```

```
# Create a Random Forest classifier
rf_classifier = RandomForestClassifier()

# Perform cross-validation
RF2_cv_scores = cross_val_score(rf_classifier, X2, y, cv=10, scoring='accuracy')

# Print the cross-validation scores
print("Cross-validation Scores:")
print(RF2_cv_scores)
print("max:", RF2_cv_scores.max(), "\nmin:", RF2_cv_scores.min(), "\nmean:", RF2_cv_scores.mean())
```

```
Cross-validation Scores:
[0.66666667 0.69565217 0.52173913 0.65217391 0.65217391 0.7826087
 0.26086957 0.60869565 0.65217391 0.52173913]
max: 0.782608695652174
min: 0.2608695652173913
mean: 0.6014492753623188
```

```
# Create a Gradient Boosting classifier
gbm_classifier = GradientBoostingClassifier()

# Perform cross-validation
gbm2_cv_scores = cross_val_score(gbm_classifier, X2, y, cv=10, scoring='accuracy')

# Print the cross-validation scores
print("Cross-validation Scores:")
print(gbm2_cv_scores)
print("max:", gbm2_cv_scores.max(), "\nmin:", gbm2_cv_scores.min(), "\nmean:", gbm2_cv_scores.mean())
```

```
Cross-validation Scores:
[0.54166667 0.73913043 0.56521739 0.56521739 0.43478261 0.73913043
 0.30434783 0.56521739 0.60869565 0.56521739]
max: 0.7391304347826086
min: 0.30434782608695654
mean: 0.5628623188405796
```

```
# Create a Decision Tree classifier
dt_classifier = DecisionTreeClassifier()

# Perform cross-validation
dt2_cv_scores = cross_val_score(dt_classifier, X2, y, cv=10, scoring='accuracy')

# Print the cross-validation scores
print("Cross-validation Scores:")
print(dt2_cv_scores)
print("max:", dt2_cv_scores.max(), "\nmin:", dt2_cv_scores.min(), "\nmean:", dt2_cv_scores.mean())
```

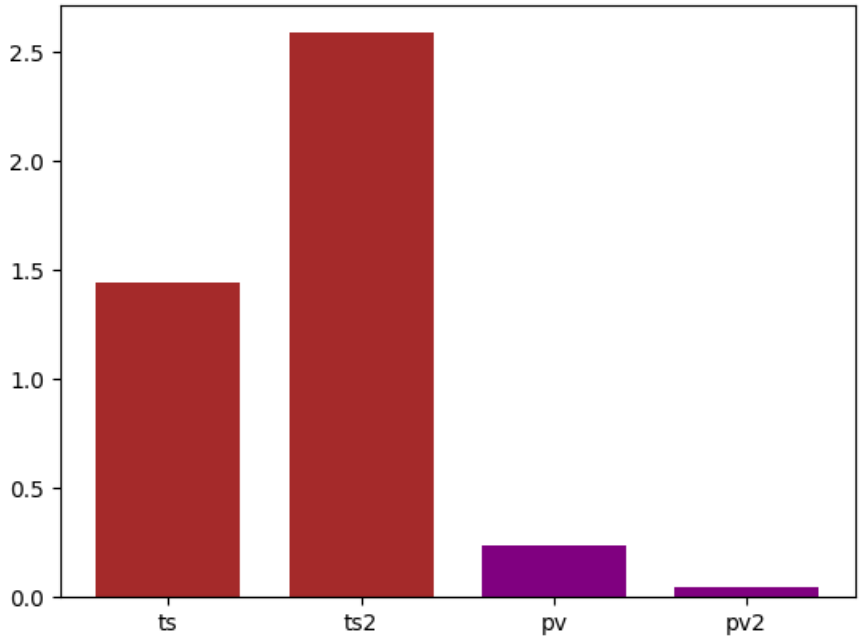
```
Cross-validation Scores:
[0.375      0.65217391 0.43478261 0.60869565 0.26086957 0.60869565
 0.34782609 0.56521739 0.60869565 0.56521739]
max: 0.6521739130434783
min: 0.2608695652173913
mean: 0.5027173913043479
```

T-Test Sonucu

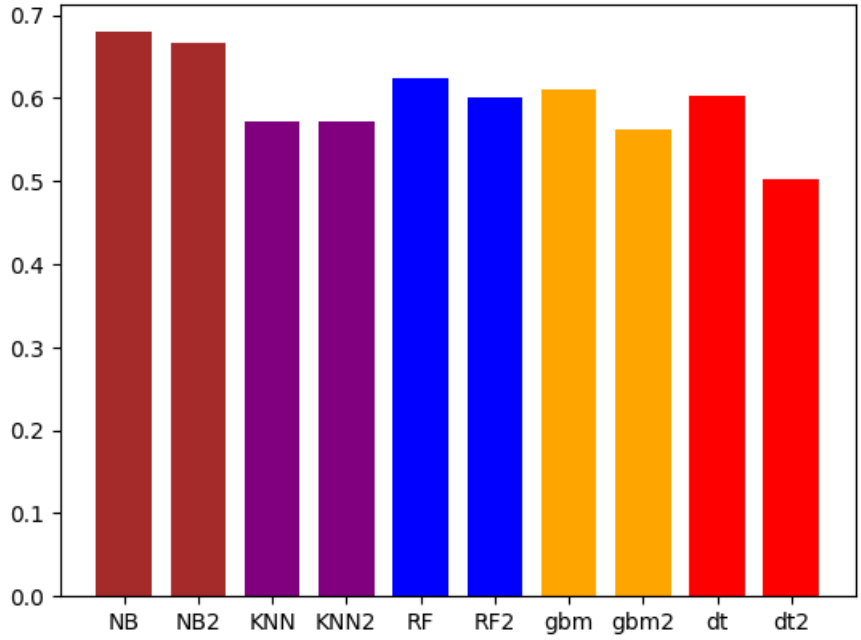
```
t_statistic, p_value = stats.f_oneway(
NB_cv_scores, KNN_cv_scores, RF_cv_scores, gbm_cv_scores, dt_cv_scores
)

new_t_statistic, new_p_value = stats.f_oneway(
NB2_cv_scores, KNN2_cv_scores, RF2_cv_scores, gbm2_cv_scores, dt2_cv_scores
)

print("T-Statistic:", t_statistic)
print("new_T-Statistic:", new_t_statistic)
print("P-Value:", p_value)
print("new_P-Value:", new_p_value)
```

```
T-Statistic: 1.4450658510196943
new_T-Statistic: 2.5857820153370885
P-Value: 0.2348202791918063
new_P-Value: 0.04951910673756055
```

T-Test Grafik



Modellerin Grafiği



Normalizasyon ve PCA öncesinde eğitilen modeller için T-Statistic sonucuna bakıldığında, gruplar arasında istatistiksel olarak anlamlı bir fark olduğunu göstermek için yeterli değildir. P değeri için genellikle anlamlılık 0.05 altında kabul edilir. İlk durumda p değeri 0.2348202791918063 olduğundan, belirlenen anlamlılık düzeyine göre istatistiksel olarak anlamlı bir fark yoktur.

Normalizasyon ve PCA uygulanması sonucunda elde edilen sonuçlarda, gruplar arasında istatistiksel olarak anlamlı bir fark olduğunu gösteren bir T istatistiği vardır. P değeri

0.04951910673756055 olduğundan, belirlenen anlamlılık düzeyine göre istatistiksel olarak anlamlı bir fark vardır.

Elde edilen veriler sonucunda, uyguladığımız özellik dönüşümü(PCA) ve normalizasyon sonucunda tekrardan oluşturulmuş modelin daha başarılı olduğunu gözlemleyebiliriz.

Elde edilen accuracy değerlerine bakıldığında normalizasyon ve PCA sonrasında doğruluk değerlerinin düştüğü gözlenmiştir. Bunun sebebi ise bazı özelliklerin diğerlerinden daha önemli olduğu için normalizasyona sokulmaması gerekliliği olabilir.