FİNAL PROJE ##### Şevval Şereflican

1.B.SINIFLANDIRMA PROBLEMİ

```
#Gerekli paketleri yükleyelim.
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler, LabelEncoder
from sklearn.metrics import accuracy_score, precision_score,
recall_score, f1_score, roc_auc_score
from sklearn.linear_model import LogisticRegression
from sklearn.neighbors import KNeighborsClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier, VotingClassifier
from sklearn.svm import SVC

# Veriyi yükleyelim.
df = pd.read_csv('bank-additional-full.csv',sep=";")
df
```

```
       age          job  marital            education  default housing
loan  \
0       56    housemaid  married              basic.4y       no      no
no
1       57     services  married          high.school  unknown      no
no
2       37     services  married          high.school       no     yes
no
3       40       admin.  married              basic.6y       no      no
no
4       56     services  married          high.school       no      no
yes
...    ...          ...      ...                  ...      ...     ...
...
41183   73      retired  married  professional.course       no     yes
no
41184   46  blue-collar  married  professional.course       no      no
no
41185   56      retired  married    university.degree       no     yes
no
41186   44   technician  married  professional.course       no      no
no
41187   74      retired  married  professional.course       no     yes
no

         contact month day_of_week  ...  campaign  pdays  previous  \
0      telephone   may         mon  ...         1    999         0
1      telephone   may         mon  ...         1    999         0
```

```
2      telephone   may          mon  ...          1    999          0
3      telephone   may          mon  ...          1    999          0
4      telephone   may          mon  ...          1    999          0
...          ...   ...          ...  ...        ...    ...        ...
41183   cellular   nov          fri  ...          1    999          0
41184   cellular   nov          fri  ...          1    999          0
41185   cellular   nov          fri  ...          2    999          0
41186   cellular   nov          fri  ...          1    999          0
41187   cellular   nov          fri  ...          3    999          1

           poutcome emp.var.rate  cons.price.idx  cons.conf.idx
euribor3m  \
0      nonexistent          1.1          93.994          -36.4
4.857
1      nonexistent          1.1          93.994          -36.4
4.857
2      nonexistent          1.1          93.994          -36.4
4.857
3      nonexistent          1.1          93.994          -36.4
4.857
4      nonexistent          1.1          93.994          -36.4
4.857
...            ...          ...             ...            ...
...
41183  nonexistent         -1.1          94.767          -50.8
1.028
41184  nonexistent         -1.1          94.767          -50.8
1.028
41185  nonexistent         -1.1          94.767          -50.8
1.028
41186  nonexistent         -1.1          94.767          -50.8
1.028
41187      failure         -1.1          94.767          -50.8
1.028

       nr.employed    y
0           5191.0   no
1           5191.0   no
2           5191.0   no
3           5191.0   no
4           5191.0   no
...            ...  ...
41183       4963.6  yes
41184       4963.6   no
41185       4963.6   no
41186       4963.6  yes
41187       4963.6   no

[41188 rows x 21 columns]
```

```
# Hedef ve özellikler
X = df.drop('y', axis=1)  # 'y' hedef sütun ismi (evet/hayır)
y = df['y']

# Kategorik değişkenleri label encoding yapabiliriz
for col in X.select_dtypes(include=['object']).columns:
    le = LabelEncoder()
    X[col] = le.fit_transform(X[col])

# Hedef değişkeni de binary yapmak için:
y = y.map({'yes':1, 'no':0})

# Veriyi train-test olarak ayıralım
X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.2, random_state=42)

# Özellik ölçeklendirme
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)

logreg = LogisticRegression(random_state=42, max_iter=500)
logreg.fit(X_train, y_train)

y_pred_logreg = logreg.predict(X_test)
y_prob_logreg = logreg.predict_proba(X_test)[:, 1]

print("Lojistik Regresyon Performansı:")
print("Accuracy:", accuracy_score(y_test, y_pred_logreg))
print("Precision:", precision_score(y_test, y_pred_logreg))
print("Recall:", recall_score(y_test, y_pred_logreg))
print("F1-Score:", f1_score(y_test, y_pred_logreg))
print("ROC-AUC:", roc_auc_score(y_test, y_prob_logreg))
```

```
Lojistik Regresyon Performansı:
Accuracy: 0.9102937606215101
Precision: 0.6672354948805461
Recall: 0.41818181818181815
F1-Score: 0.5141354372123603
ROC-AUC: 0.9317328385302064
```

```
knn = KNeighborsClassifier(n_neighbors=5)
knn.fit(X_train, y_train)

y_pred_knn = knn.predict(X_test)
y_prob_knn = knn.predict_proba(X_test)[:, 1]

print("KNN Performansı:")
print("Accuracy:", accuracy_score(y_test, y_pred_knn))
print("Precision:", precision_score(y_test, y_pred_knn))
```

```
print("Recall:", recall_score(y_test, y_pred_knn))
print("F1-Score:", f1_score(y_test, y_pred_knn))
print("ROC-AUC:", roc_auc_score(y_test, y_prob_knn))
```

```
KNN Performansı:
Accuracy: 0.8998543335761107
Precision: 0.5867507886435331
Recall: 0.39786096256684494
F1-Score: 0.47418738049713194
ROC-AUC: 0.8570310640781277
```

```
dt = DecisionTreeClassifier(random_state=42)
dt.fit(X_train, y_train)

y_pred_dt = dt.predict(X_test)
y_prob_dt = dt.predict_proba(X_test)[:, 1]

print("Decision Tree Performansı:")
print("Accuracy:", accuracy_score(y_test, y_pred_dt))
print("Precision:", precision_score(y_test, y_pred_dt))
print("Recall:", recall_score(y_test, y_pred_dt))
print("F1-Score:", f1_score(y_test, y_pred_dt))
print("ROC-AUC:", roc_auc_score(y_test, y_prob_dt))
```

```
Decision Tree Performansı:
Accuracy: 0.8889293517844137
Precision: 0.5107066381156317
Recall: 0.5101604278074866
F1-Score: 0.5104333868378812
ROC-AUC: 0.7237917023331559
```

```
rf = RandomForestClassifier(random_state=42, n_jobs=-1)
rf.fit(X_train, y_train)

y_pred_rf = rf.predict(X_test)
y_prob_rf = rf.predict_proba(X_test)[:, 1]

print("Random Forest Performansı:")
print("Accuracy:", accuracy_score(y_test, y_pred_rf))
print("Precision:", precision_score(y_test, y_pred_rf))
print("Recall:", recall_score(y_test, y_pred_rf))
print("F1-Score:", f1_score(y_test, y_pred_rf))
print("ROC-AUC:", roc_auc_score(y_test, y_prob_rf))
```

```
Random Forest Performansı:
Accuracy: 0.9127215343529983
Precision: 0.6447721179624665
Recall: 0.5144385026737968
F1-Score: 0.5722784057108864
ROC-AUC: 0.9432266133396209
```

```python
from sklearn.svm import LinearSVC
from sklearn.calibration import CalibratedClassifierCV

linear_svc = LinearSVC(random_state=42, max_iter=10000)
calibrated_svc = CalibratedClassifierCV(linear_svc)  # ROC-AUC için probability verecek

calibrated_svc.fit(X_train, y_train)
y_pred_svm = calibrated_svc.predict(X_test)
y_prob_svm = calibrated_svc.predict_proba(X_test)[:, 1]

print("SVM (LinearSVC) Performansı:")
print("Accuracy:", accuracy_score(y_test, y_pred_svm))
print("Precision:", precision_score(y_test, y_pred_svm))
print("Recall:", recall_score(y_test, y_pred_svm))
print("F1-Score:", f1_score(y_test, y_pred_svm))
print("ROC-AUC:", roc_auc_score(y_test, y_prob_svm))
```

```
SVM (LinearSVC) Performansı:
Accuracy: 0.9068948773974266
Precision: 0.6468531468531469
Recall: 0.39572192513368987
F1-Score: 0.49104180491041804
ROC-AUC: 0.9305176028311566
```

```python
from sklearn.ensemble import VotingClassifier
from sklearn.calibration import CalibratedClassifierCV

# LinearSVC modelini calibrate et
svc = LinearSVC(random_state=42)
calibrated_svc = CalibratedClassifierCV(svc)

# VotingClassifier - soft voting
voting_clf = VotingClassifier(
    estimators=[
        ('lr', LogisticRegression(max_iter=1000)),
        ('rf', RandomForestClassifier(random_state=42)),
        ('svc', calibrated_svc)
    ],
    voting='soft'
)

# Eğitimi
voting_clf.fit(X_train, y_train)

# Tahmin ve olasılıklar
y_pred_vote = voting_clf.predict(X_test)
y_prob_vote = voting_clf.predict_proba(X_test)[:, 1]

# Performans metrikleri
```

```
accuracy_vote = accuracy_score(y_test, y_pred_vote)
precision_vote = precision_score(y_test, y_pred_vote)
recall_vote = recall_score(y_test, y_pred_vote)
f1_vote = f1_score(y_test, y_pred_vote)
roc_auc_vote = roc_auc_score(y_test, y_prob_vote)

print("Voting Classifier Performansı:")
print(f"Accuracy: {accuracy_vote}")
print(f"Precision: {precision_vote}")
print(f"Recall: {recall_vote}")
print(f"F1-Score: {f1_vote}")
print(f"ROC-AUC: {roc_auc_vote}")
```

```
Voting Classifier Performansı:
Accuracy: 0.9102937606215101
Precision: 0.6617161716171617
Recall: 0.4288770053475936
F1-Score: 0.5204412719013628
ROC-AUC: 0.9437409430305178
```

1. Logistic Regression Accuracy (Doğruluk): 0.91

Precision: 0.667

Recall: 0.418

F1-Score: 0.514

ROC-AUC: 0.932

Yorum: Lojistik Regresyon, ROC-AUC açısından oldukça güçlü bir modeldir ve doğruluğu da yüksektir. Ancak duyarlılık (recall) değeri göreceli olarak düşüktür, bu da modelin "evet" diyen müşterileri saptamada sınırlı kaldığını gösterir.

1. KNN Accuracy: 0.899

Precision: 0.587

Recall: 0.398

F1-Score: 0.474

ROC-AUC: 0.857

Yorum: KNN modeli, diğer modellere kıyasla en düşük ROC-AUC skorlarından birine sahiptir. Duyarlılık ve F1 skoru da düşüktür. Karmaşık sınırlarda zorlanan, mesafeye duyarlı bir algoritma olduğu için bu veri setinde başarılı olamamıştır.

1. Decision Tree Accuracy: 0.889

Precision: 0.511

Recall: 0.510

F1-Score: 0.510

ROC-AUC: 0.724

Yorum: Karar ağacı modeli dengeleyici bir performans sunar; precision ve recall birbirine yakındır. Ancak ROC-AUC değeri oldukça düşüktür. Bu model, veriyi ezberlemeye meyilli olduğu için genelleme yeteneği sınırlıdır.

1. Random Forest Accuracy: 0.913

Precision: 0.645

Recall: 0.514

F1-Score: 0.572

ROC-AUC: 0.943

Yorum: Random Forest, dengeli ve güçlü bir modeldir. Tüm metriklerde yüksek performans göstermiştir. ROC-AUC değeri en yüksek seviyelerdedir, bu da sınıflar arasında iyi bir ayrım yaptığını gösterir. Önerilen modeller arasında öne çıkar.

1. LinearSVC (Calibrated) Accuracy: 0.907

Precision: 0.647

Recall: 0.396

F1-Score: 0.491

ROC-AUC: 0.931

Yorum: LinearSVC, doğruluğu ve ROC-AUC'si açısından iyidir ancak düşük recall değeri "evet" yanıtlarını kaçırmasına neden olur. Bu da pazarlama başarısını azaltabilir. ROC-AUC değeri yüksek olmasına rağmen duyarlılık sorunu sınırlayıcıdır.

1. Voting Classifier Accuracy: 0.910

Precision: 0.662

Recall: 0.429

F1-Score: 0.520

ROC-AUC: 0.944

Yorum: Voting Classifier, birden fazla güçlü modelin birleşiminden oluşur ve en yüksek ROC-AUC skorunu yakalamıştır. Duyarlılık da Logistic Regression'dan biraz daha yüksektir. Bu model, istikrarlı ve güvenilir bir tercih olabilir.

Genel Değerlendirme -Eğer duyarlılık (recall) değerini öncelik alırsak, yani "evet" diyen müşterileri yakalamak öncelikliyse, Random Forest en iyi seçimdir.

-ROC-AUC öncelikliyse, Voting Classifier en güçlü modeldir.

-Basit, yorumlanabilir bir model istiyorsak Logistic Regression tercih edilebiliriz.

-KNN ve Decision Tree, bu veri seti için önerilmez çünkü ROC-AUC ve genel skorları düşüktür.

2.BÖLÜM

1.GBM algoritmasının mantığı, zayıf öğrenici kavramı

2.XGBoost'un farklılıkları (regularizasyon, erken durdurma, paralelleşme)

3.Uygulama: XGBRegressor veya XGBClassifier

4.Diğer modellerle karşılaştırma

```
# Hedef değişken kategorik olduğundan XGBClassifier kullanacağız.
from xgboost import XGBClassifier

# Modeli tanımla ve eğit
xgb_clf = XGBClassifier(use_label_encoder=False,
eval_metric='logloss', random_state=42)
xgb_clf.fit(X_train, y_train)
```

```
C:\Users\seref\anaconda3\Lib\site-packages\xgboost\training.py:183:
UserWarning: [08:59:36] WARNING: C:\actions-runner\_work\xgboost\
xgboost\src\learner.cc:738:
Parameters: { "use_label_encoder" } are not used.

  bst.update(dtrain, iteration=i, fobj=obj)
```

```
XGBClassifier(base_score=None, booster=None, callbacks=None,
              colsample_bylevel=None, colsample_bynode=None,
              colsample_bytree=None, device=None,
early_stopping_rounds=None,
              enable_categorical=False, eval_metric='logloss',
              feature_types=None, feature_weights=None, gamma=None,
              grow_policy=None, importance_type=None,
              interaction_constraints=None, learning_rate=None,
max_bin=None,
              max_cat_threshold=None, max_cat_to_onehot=None,
              max_delta_step=None, max_depth=None, max_leaves=None,
              min_child_weight=None, missing=nan,
monotone_constraints=None,
              multi_strategy=None, n_estimators=None, n_jobs=None,
              num_parallel_tree=None, ...)
```

```
from sklearn.metrics import accuracy_score, precision_score,
recall_score, f1_score, roc_auc_score

# Tahminler
```

```
y_pred_xgb = xgb_clf.predict(X_test)
y_prob_xgb = xgb_clf.predict_proba(X_test)[:, 1]

# Performans metrikleri
acc = accuracy_score(y_test, y_pred_xgb)
prec = precision_score(y_test, y_pred_xgb)
rec = recall_score(y_test, y_pred_xgb)
f1 = f1_score(y_test, y_pred_xgb)
roc_auc = roc_auc_score(y_test, y_prob_xgb)

print("XGBClassifier Performansı:")
print(f"Accuracy: {acc}")
print(f"Precision: {prec}")
print(f"Recall: {rec}")
print(f"F1-Score: {f1}")
print(f"ROC-AUC: {roc_auc}")
```

```
XGBClassifier Performansı:
Accuracy: 0.9155134741442098
Precision: 0.6499372647427855
Recall: 0.5540106951871657
F1-Score: 0.5981524249422633
ROC-AUC: 0.9451377757730507
```

Accuracy: 0.9155 Tüm sınıflar genelinde %91.5 oranında doğru tahmin yapılmıştır. Bu oran, diğer modellerin (özellikle Logistic Regression, KNN, Decision Tree) çoğundan yüksektir.

Precision: 0.6499 "Evet" tahmini yapılan bireylerin yaklaşık %65'i gerçekten de kampanyaya olumlu yanıt vermiş. Bu değer VotingClassifier ve RandomForest'a yakın ama biraz daha güçlü.

Recall: 0.5540 Gerçekten kampanyaya katılan müşterilerin %55'i doğru tahmin edilebilmiştir. Bu metrikte en iyi sonucu veren modellerden biri. Recall değeri RandomForest ve VotingClassifier'dan da daha iyi.

F1-Score: 0.5981 Precision ve Recall'un dengeli bir ortalaması olan F1 puanı oldukça iyi. Bu, modelin genel dengeyi sağlamada başarılı olduğunu gösteriyor.

ROC-AUC: 0.9451 Bu değer, modelin pozitif sınıfı negatiften ayırt etme başarısını ölçer. XGBClassifier, ROC-AUC açısından en iyi sonucu elde etmiştir (VotingClassifier 0.9437 idi). Bu, modelin sınıflandırmada çok güçlü olduğunu gösterir.

Genel Yorum

XGBClassifier, tüm metriklerde (özellikle Recall, F1 ve ROC-AUC) öne çıkıyor. Özellikle dengesiz veri setlerinde önemli olan Recall ve ROC-AUC değerleri sayesinde, kampanyaya olumlu yanıt veren müşterileri tespit etmede oldukça etkili.

Bu model, Random Forest ve VotingClassifier ile birlikte en iyi performans gösteren modellerden biridir.

Özellikle pazarlama kampanyası gibi hassas alanlarda yüksek Recall değeri sayesinde potansiyel müşteriler gözden kaçırılmamış olur.

```
from sklearn.ensemble import GradientBoostingClassifier
from sklearn.metrics import accuracy_score, precision_score,
recall_score, f1_score, roc_auc_score

# GBM modelini tanımla ve eğit
gbm_clf = GradientBoostingClassifier(
    n_estimators=300,
    learning_rate=0.1,
    max_depth=5,
    random_state=42
)

gbm_clf.fit(X_train, y_train)

# Tahminler
y_pred_gbm = gbm_clf.predict(X_test)
y_prob_gbm = gbm_clf.predict_proba(X_test)[:, 1]

# Performans metrikleri hesapla
acc_gbm = accuracy_score(y_test, y_pred_gbm)
prec_gbm = precision_score(y_test, y_pred_gbm)
rec_gbm = recall_score(y_test, y_pred_gbm)
f1_gbm = f1_score(y_test, y_pred_gbm)
roc_auc_gbm = roc_auc_score(y_test, y_prob_gbm)

print(f"GBM Accuracy: {acc_gbm}")
print(f"GBM Precision: {prec_gbm}")
print(f"GBM Recall: {rec_gbm}")
print(f"GBM F1-Score: {f1_gbm}")
print(f"GBM ROC-AUC: {roc_auc_gbm}")


GBM Accuracy: 0.9156348628307842
GBM Precision: 0.6507537688442211
GBM Recall: 0.5540106951871657
GBM F1-Score: 0.5984979780473715
GBM ROC-AUC: 0.9478280481027137
```

Genel olarak model iyi performans gösteriyor. ROC-AUC değeri yüksek, bu da modelin sınıflar arasında iyi ayrım yaptığı anlamına gelir.

Ancak recall'ın düşük olması, kampanya açısından olumsuz olabilir çünkü potansiyel müşterilerin yarısını yakalayamıyor.

1. GBM Algoritmasının Mantığı ve Zayıf Öğrenici Kavramı

Gradient Boosting Machine (GBM), topluluk öğrenmesi (ensemble learning) yöntemlerinden biridir ve zayıf öğrenicilerin (weak learners) ardışık olarak eğitilmesine dayanır. Burada zayıf öğrenici, genellikle tek başına çok güçlü olmayan ama biraz daha iyi tahmin yapabilen basit modellerdir; örneğin küçük derinlikli karar ağaçları.

GBM, ilk olarak basit bir model (zayıf öğrenici) eğitir ve bu modelin hatalarını (residuals) hesaplar.

Sonra, ikinci model bu hataları öğrenmeye çalışır; yani ilk modelin yapamadığını düzeltecek şekilde eğitilir.

Bu süreç ardışık olarak devam eder, her yeni model öncekinin hatalarını azaltmak için eklenir.

Sonuçta, zayıf öğrenicilerin birleşimiyle güçlü ve daha doğru bir model ortaya çıkar.

Bu yöntem, özellikle karmaşık ve doğrusal olmayan ilişkilere sahip verilerde başarılıdır.

2. XGBoost'un GBM'den Farkları

XGBoost, GBM algoritmasının geliştirilmiş ve optimize edilmiş halidir. Temel farkları şunlardır:

Regularizasyon (L1 ve L2): XGBoost, model karmaşıklığını kontrol etmek için regularizasyon terimleri ekler. Böylece aşırı uyum (overfitting) azaltılır.

Erken Durdurma (Early Stopping): Model, doğrulama setindeki performans düşmeye başladığında eğitim otomatik olarak durdurulabilir. Bu da aşırı uyumu önler ve eğitim süresini kısaltır.

Paralel İşlem: XGBoost, ağaç oluşturma işlemini paralel yaparak çok daha hızlı çalışır.

Eksik Veri Desteği: XGBoost, eksik değerlerle doğal olarak başa çıkabilir, onları özel olarak işler.

Daha İyi Optimizasyon: İkinci türev tabanlı optimizasyon yaparak daha hızlı ve etkili öğrenme sağlar.

Sütun Örneklemesi: Rastgele olarak ağaç oluştururken kullanılan özelliklerin bir kısmını seçerek çeşitliliği artırır ve aşırı uyumu azaltır.