Out of Fold Predictions:

1. OOFP, dağıtım örnekleri üzerinde k-cross-validation (kCV) uygulanırken modeller tarafından yapılan tahminlerdir. OOFP, doğrudan out-of-sample prediction (OOSP) ile örtüşür. Her iki yaklaşımda da, tahminler modeli eğitmek için kullanılan örnekler üzerinden değil, yeni bir veri üzerinde tahmin yapmak için kullanılan modelin performansını tahmin etmek için kullanılır.
2. Sonuç olarak OOFP, OOSP'nin bir türevidir ve kCV kullanarak bir modelin içeriğini değerlendirir.
3. OOSP'de, tahminler bir model tarafından, modelin eğitimi sırasında kullanılmayan bir veri üzerinden yapılır.
4. OOFP'nin iki ana kullanım yaklaşımı bulunur:
   - 4.a: Daha önce görülmemiş bir veri üzerinde modelin performansını accuracy veya error skoruna bakarak değerlendirmek. Her fold'da modelin tahminleri üzerinden modele bir puan verilir ve son olarak puanların ortalaması alınır.
   - 4.b: Bir grup modeli uygulamak. Bütün tahminler toplanır ve beklenen değer ile karşılaştırılarak bir skor hesaplanır.

ÖRN: 2 yöntem arasındaki farkı pekiştirmek için bir örnek uygulayalım:

⇒ make_blobs() fonksiyonu, 1000 örnekli, 100 özellikli ve iki sınıflı bir denetimli sınıflandırma problemi oluşturur.

Code:

```
1. X, y = make_blobs(n_samples = 1000, centers=2, n_features = 100,
                     cluster_std = 20)
   (X: input, y: output)
2. print(X.shape, y.shape)
```

Output:

```
(1000, 100)  (1000,)
```

(1000: input data, 100: input features, (1000,): corresponding classification labels)

⇒ kNeighborsClassifier modeli değerlendirmek için KCV kullanırız. Bu örnekte k=10 alınır, her değerlendirmenin skoru bir listede tutulur ve sonuç olarak bütün skorların ortalaması ve standart sapması hesaplanır.

Code < >

```
1. X, y = make_blobs(n_samples=1000, centers=2, n_features=100,
                     cluster_std=20)
   (X: input, y: output)
2. scores = list()
3. kfold = KFold(n_splits=10, shuffle=True)
4. for train_ix, test_ix in kfold.split(X)
5.     train_X, test_X = X[train_ix], X[test_ix]
6.     train_y, test_y = y[train_ix], y[test_ix]
7.     model = kNeighborsClassifier()
8.     model.fit(train_X, train_y)
9.     yhat = model.predict(test_X)
10.    acc = accuracy_score(test_y, yhat)
11.    scores.append(acc)
12. mean_s, std_s = mean(scores), std(scores)
```

(5–6: → test giriş ve çıktı dizileri; 8: → eğitim giriş ve çıktı dizileri; model eğitim seti üzerinde uygulanır. 9: → test verisi üzerindeki modelin tahmini. 10: → modelin doğruluk değeri; gerçek sonuç; modelin tahmini. 12: → ortalama; standart sapma)

⇒ Diğer yaklaşımda, tüm tahminler tek bir grup olarak değerlendirilir. Her aşamada modeli değerlendirmek yerine tahminler yapılır ve listede saklanır. Çalışma sonunda tahminler her set için beklenen değerler ile karşılaştırılır ve tek bir doğruluk skoru elde edilir.

Code < >

```
10. data_y.extend(test_y)
11. data_yhat.extend(yhat)
12. acc = accuracy_score(data_y, data_yhat)
```

OOFP for Ensembles:

* OOFP'nin bir diğer yaygın kullanımı bir grup model keşfinde kullanılmasıdır. Aynı datasetinde birden fazla modelin tahminlerinin birleşiminden oluşturulan bir makine öğrenme modeline bir ensemble (grup) denir.

* OOFP, modeli eğitmek için kullanılmadığında, modelin eğitim veri kümesindeki her örnek üzerinde gösterdiği performans ile ilgili veri toplar. Bu veriler, tahminleri doğrulamak veya geliştirmek için bir model eğitilirken kullanılabilir.

⇒ İlk olarak KCV her temel model için uygulanır ve OOFP tahminleri toplanır. Önemli nokta eğitim verisinin aynı bölümüne (kfold) her modelin uygulanmasıdır. Her model için bir toplam grup tahminine sahip olunur.

Temel model: Modeller eğitim veri kümesi üzerinde KCV kullanılarak değerlendirilir ve tahminler (OOFP) saklanır.

⇒ Ardından meta-model olarak adlandırılan, ikincil dereceli model diğer modellerin yaptığı tahminler üzerinden eğitilir. Meta model tahminler yaparken diğer modellerin girdi verilerini kullanabilir de kullanmayabilirde. Bu modelin amacı, OOFP'leri kullanarak diğer modellerin tahminlerini en iyi şekilde nasıl birleştireceğini ve düzelteceğini öğrenmektir.

Meta model: Bu model, bir veya daha fazla model tarafından üretilen OOFP'leri girdi olarak alarak, tahminlerin en iyi şekilde nasıl birleştirileceğini ve doğrulanacağını göstermektir.

⇒ Bu yöntemi özetleyecek olursak:

1. KCV'yi kullanarak, OOF tahminlerini topla. (Her model için)
2. Bütün modellerden elde edilen OOF tahminleri üzerinde meta-modeli eğit.
3. Bütün eğitim verisetı üzerinde her temel modeli eğit.

Bu aktı stacked generalization veyor stacking for short olarak adlandırılır. meta model olarak a linear weighted sum (doğrusal ağırlıklı toplam) kullanılması yaygın olduğundan, bu işleme bazen blending (karıştırma) denir.

Super Learner For Regression:

```
1.  def get_models():                                   modeller tanımlanır
2.      < ... >
3.  def out_of_fold_predictions(X, y, models):          modeller için OOFP değerleri
4.      meta_X, meta_y = list(), list()                 elde edilir.
5.      kfold = KFold(n_splits=10, shuffle=True)
6.      for train_ix, test_ix in kfold.split(X):
7.          fold_yhats = list()
8.          train_X, test_X = X[train_ix], X[test_ix]
9.          train_y, test_y = y[train_ix], y[test_ix]
10.         meta_y.extend(test_y)
11.         for model in models:
12.             model.fit(train_X, train_y) //model oluşturulur
13.         yhat = model.predict(test_X) //model test verisi ile test edilir
14.         fold_yhats.append(yhat.reshape(len(yhat), 1))
15.     meta_X.append(hstack(fold_yhats)) //Liste meta_model'in girdi olacak şekilde belirlenir.
16.     return vstack(meta_X), asarray(meta_y)
17. def fit_base_models(X, y, models):                  modeller tüm sete uygulanır.
18.     for model in models:
19.         model.fit(X, y)
20. def fit_meta_model(X, y):                           meta_model tüm sete uygulanır.
21.     model = LinearRegression()
22.     model.fit(X, y)
23.     return model
24. def evaluate_models(X, y, models):                  modeller ayrı ayrı değerlendirilir.
25.     for model in models:
26.         yhat = model.predict(X)
27.         rmse = sqrt(mean_squared_error(y, yhat))
28. def super_learner_predictions(X, models, meta_model):   Super learner in
29.     meta_X = list()                                    tahmini üretilir.
30.     for model in models:
31.         yhat = model.predict(X)
32.         meta_X.append(yhat.reshape(len(yhat),1))
33.     meta_X = hstack(meta_X)
34.     return meta_model.predict(meta_X)
```

yhat: modelin tahmini
fold_yhats: aynı fold için bütün modellerin tahminlerini tutan liste
meta_X: meta modelin girdisi
meta_y: meta modelin doğrulama yapacağı veriler.

```
35. X, y = make_regression(n_s =1000, n_f = 100, noise= 0.5)
36. X, X_val, y - y_val = train_test_split(X, y, test_size = 0.50)
37. models = get_models()
40. meta_X, meta_y = get_out_of_fold_predictions(X, y, models)
41. fit_base_models(X, y, models)
42. meta_model = fit_meta_model(meta_X, meta_Y)
43. evaluate_models(X_val, y_val, models)
44. yhat = super_learner_predictions(X_val, models, meta_model)
```