

Makine Öğrenmesi

Tekrar Örnekleme ve Model Değerlendirme

İlker Birbil ve Utku Karaca

Erasmus Üniversitesi Rotterdam

İstanbul'da Makine Öğrenmesi

27 Ocak – 2 Şubat, 2020



Makine Öğrenmesi

```
graph TD; A[Makine Öğrenmesi] --> B[Doğrusal Bağlanım]; A --> C[Boyut Küçültme ve Düzenleştirme]; A --> D[Tekrar Örnekleme ve Model Değerlendirme]; A --> E[Sınıflandırma ve Ağaçlar]; A --> F[GÜdÜmsüz Öğrenme]; A --> G[Yapay Sinir Ağları ve Derin Öğrenme]; A --> H[ ];
```

Doğrusal Bağlanım

Boyut Küçültme
ve
Düzenleştirme

Tekrar Örnekleme
ve
Model Değerlendirme

Sınıflandırma
ve
Ağaçlar

GÜdÜmsüz
Öğrenme

Yapay Sinir Ağları
ve
Derin Öğrenme

Sıkça Sorulan Sorular

- Hangi tahminleyiciyi/sınıflandırıcıyı kullanmalıyız?
- Performans genellemesi nedir?
- Görülmemiş (gelecek) veri seti derken neyi kastediyoruz?
- Algoritmaların parametrelerini nasıl seçeriz?
- Birden fazla algoritmayı nasıl deneriz ve birbirleriyle kıyaslarız?
- Adil bir performans değerlendirmesi ve karşılaştırması nasıl olur?

Bu ders notlarında, Sebastian Ranschka'nın çalışmasından* pek çok fikir kullandım. Aynı zamanda, Python kodlarını yazarken de yine kendisinin kodlarından faydalandım.

**Model Evaluation, Model Selection, and Algorithm Selection in Machine Learning*, S. Raschka, arXiv:1811.1280v2, 3 Aralık 2018.

Son Ürüne Doğru Adımlar (Steps to Delivery)

- **Performans Genellemesi** (algoritma ve parametreleri belirlenmiş)

Örnek 1: K -En Yakın Komşu (sabit K)

Örnek 2: m . dereceden polinom kullanarak eğri uydurumu (sabit m)

- **Model Seçimi** (algoritma belirlenmiş, ‘en iyi’ parametreler aranıyor)

Hiperparametreler: $K=?$, $m=?$, $\lambda=?$, $\alpha=?$, ...

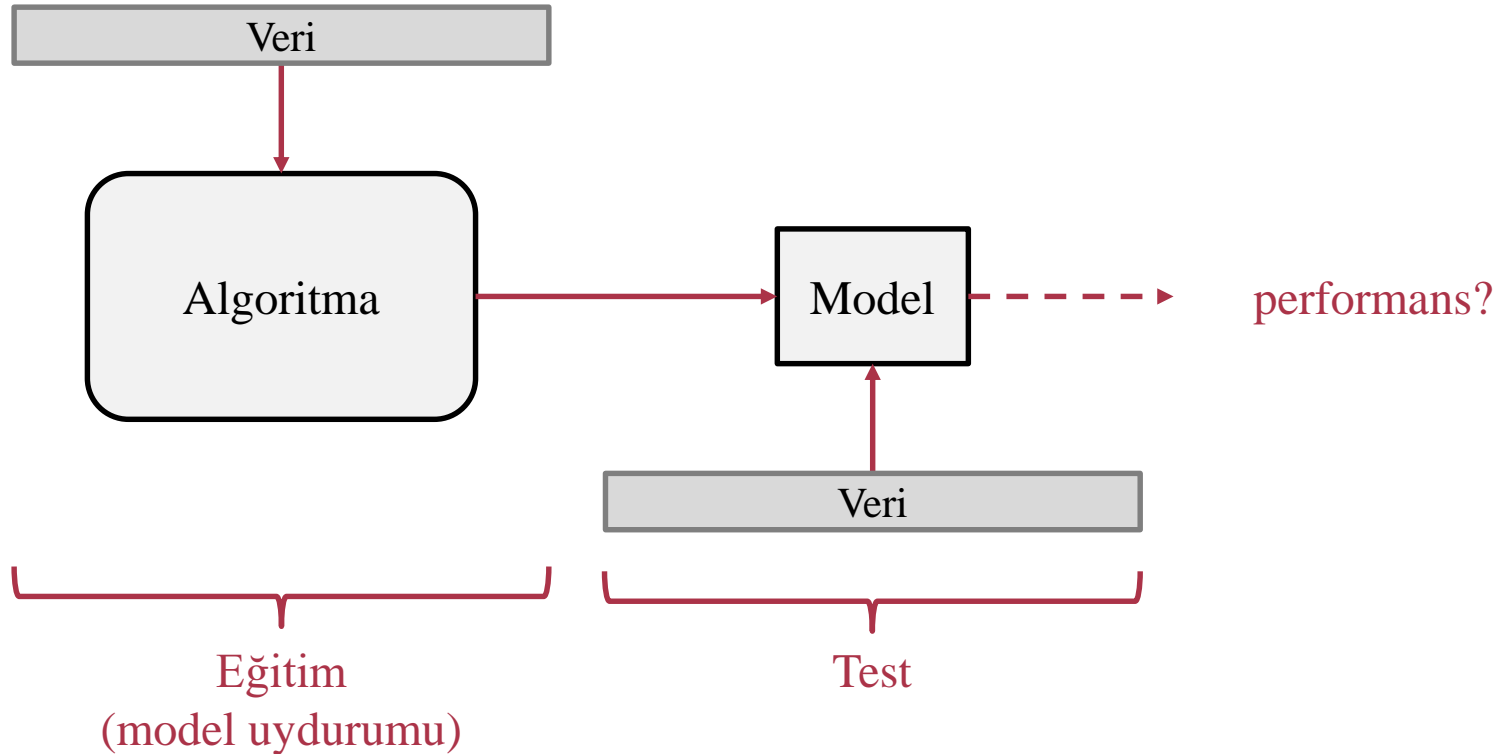
Farklı hiperparametreler (\neq model parametreleri), farklı modeller demek

- **Algoritma Seçimi**

K -EYK?, eğri uydurumu?, lasso?, yapay sinir ağları?, ...

Performans Genellemesi

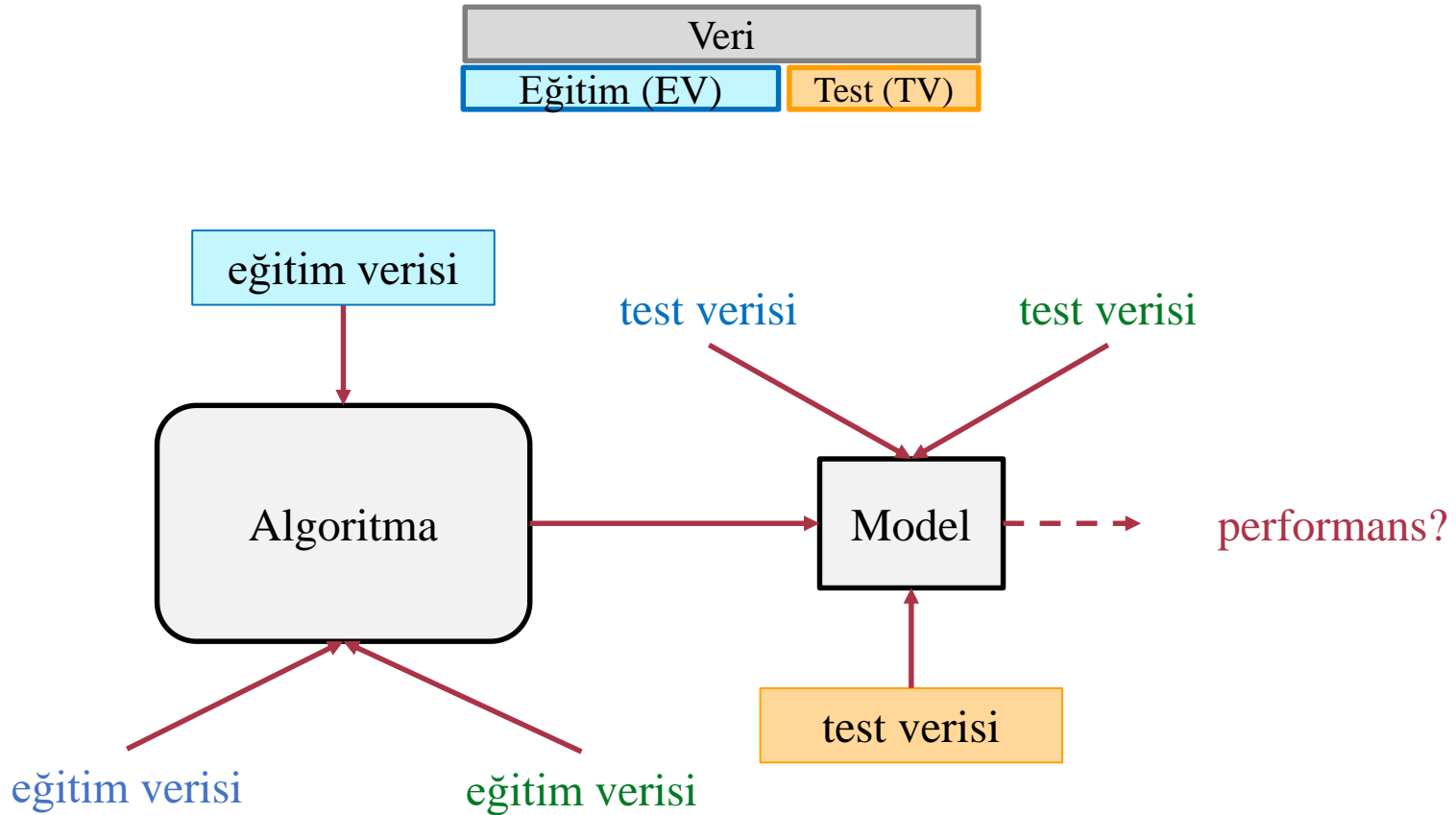
Tekrar Yerine Koyma (Resubstitution)



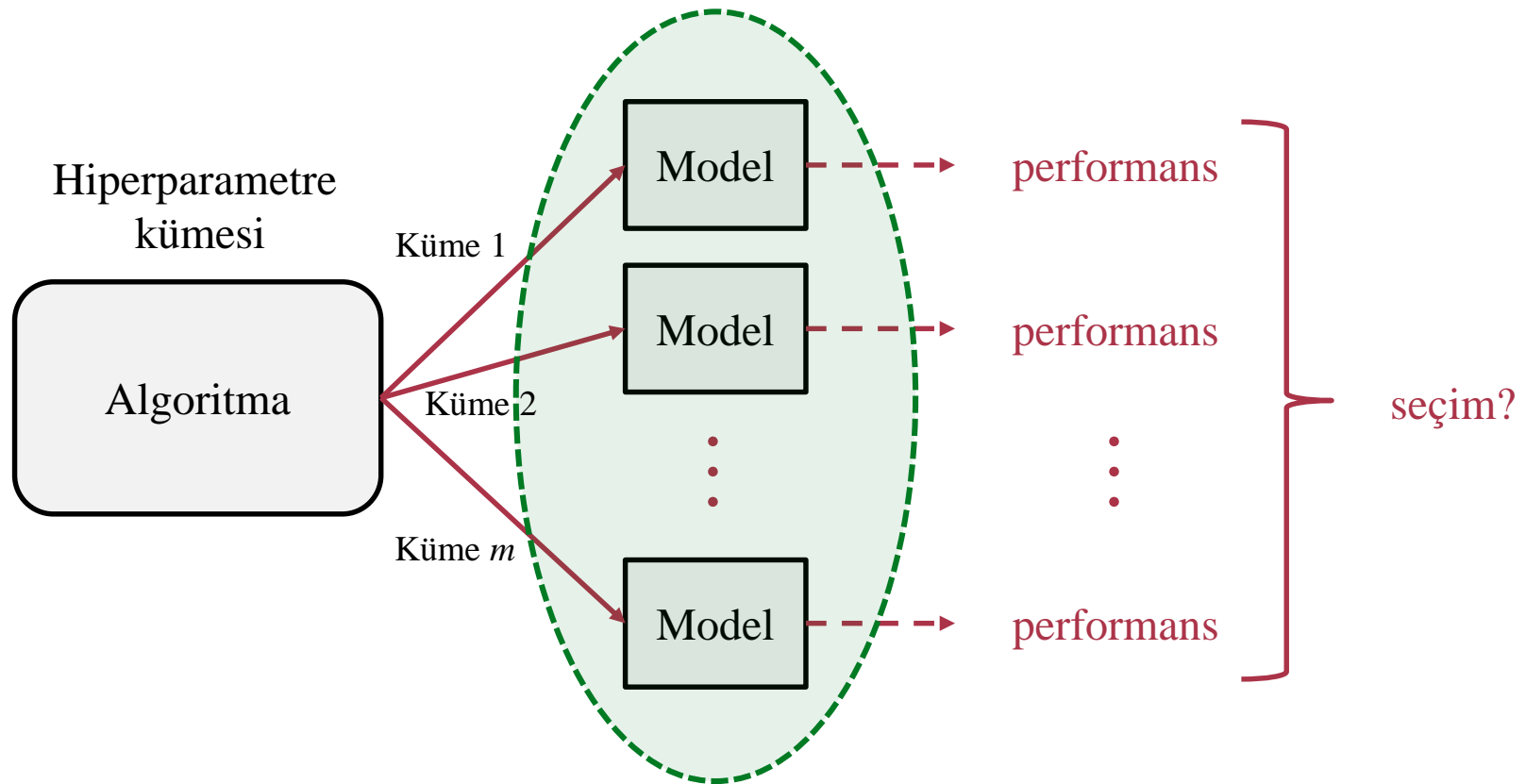
iyimser yanlılık
(optimistic bias)

Performans Genellemesi

Ayırma (Holdout)



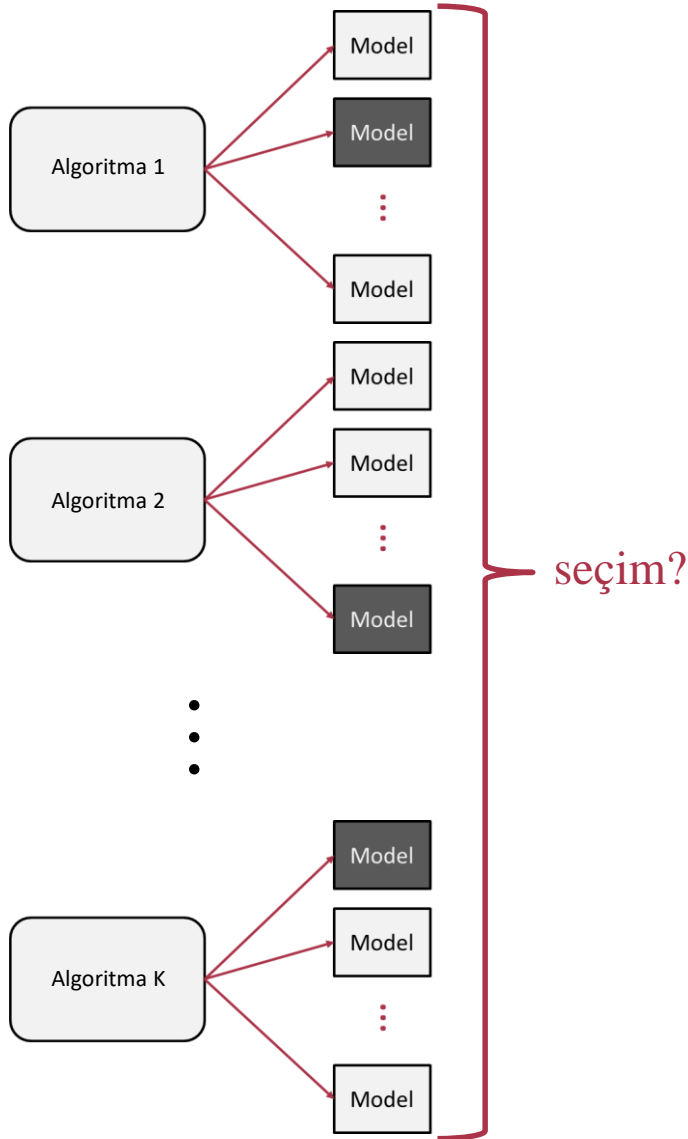
Model Seçimi



Hipotez Uzayı
(Hipotez \cong Model)

$$\hat{Y} = \hat{f}(X)$$

Algoritma Seçimi



- Eğitim (EV) - Doğrulama (DV) - Test (TV)

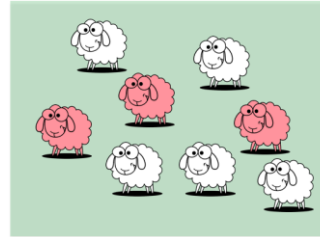
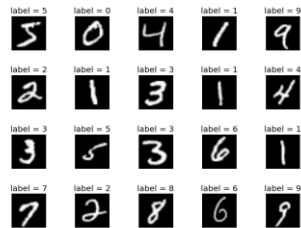


- İstatistiksel testler
- Tekrar örnekleme
 - Çapraz Geçerlilik Sınaması (Cross-validation)
 - Zorlama Tekniği (Boostrapping)
- Yanlılık – Varyans ikilemi

Ayarlar ve Kurallar

$$\{(x_i, y_i) : 1, \dots, n\}$$

- Bağımsız özdeşçe dağılmış veri ile sınıflandırma problemi



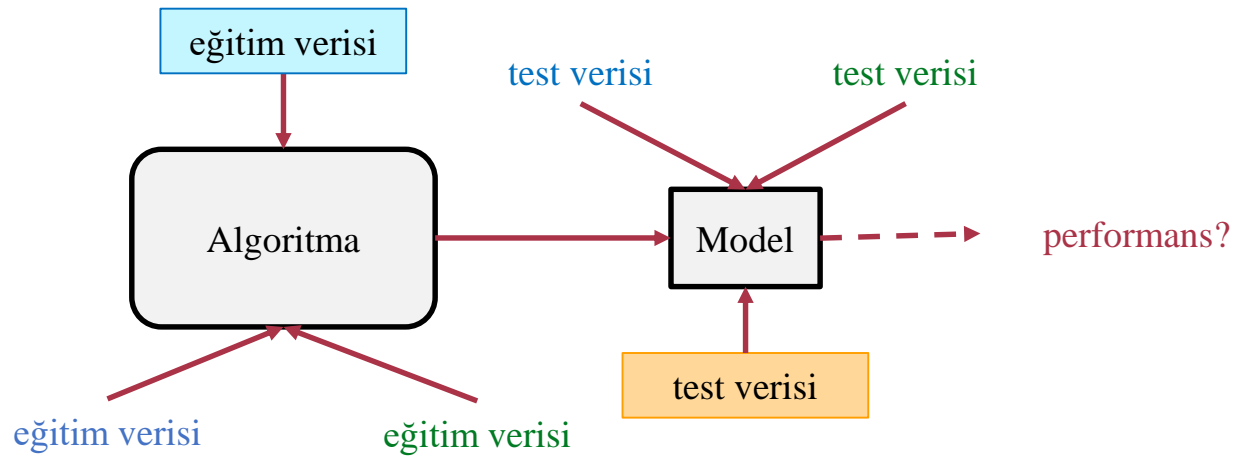
- **Doğruluk:** Doğru sınıflandırılmış verilerin oranı

$$\delta = 1 - \frac{1}{n} \sum_{i=1}^n I(y_i \neq \hat{y}_i)$$

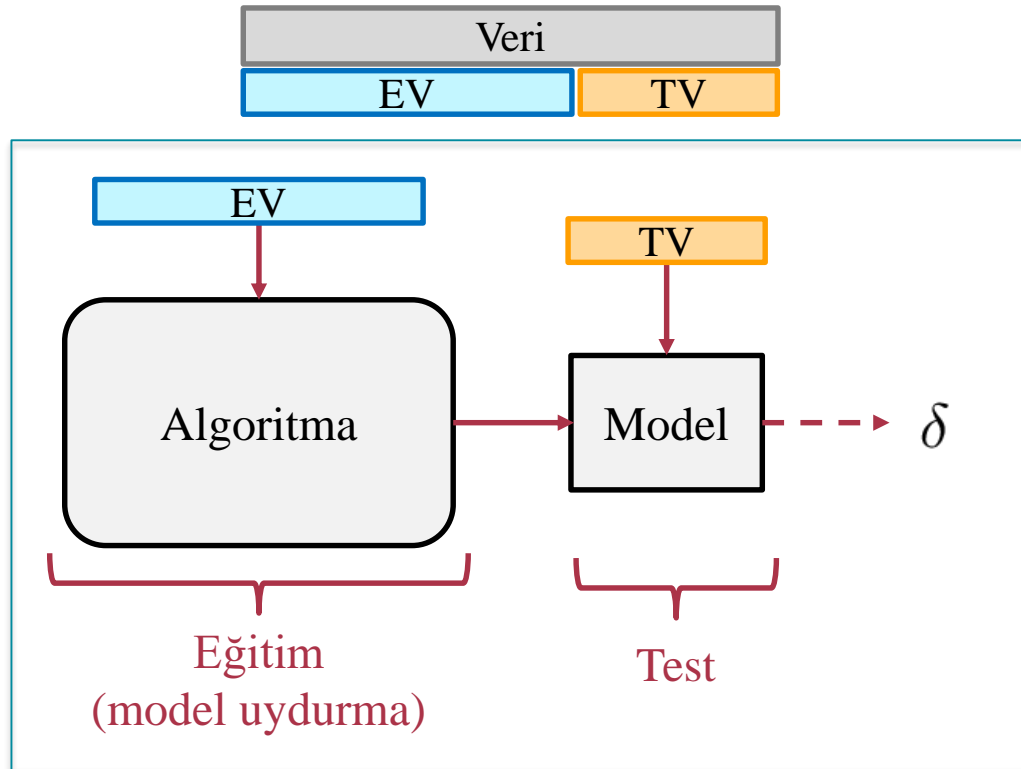
- Yanlılığı önlemek için **test setini sadece bir kere kullan**

Performans Genellemesi

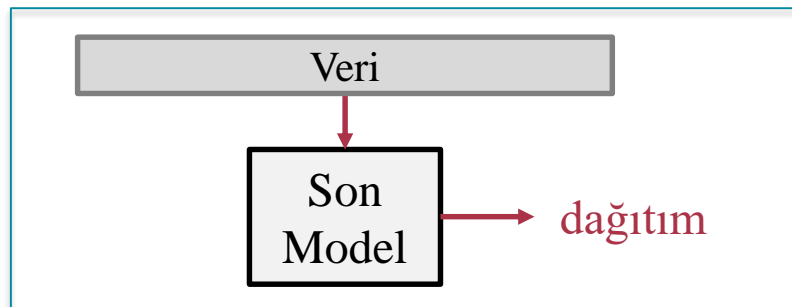
(algoritma ve parametreleri belirlenmiş)



Ayırma (Holdout)



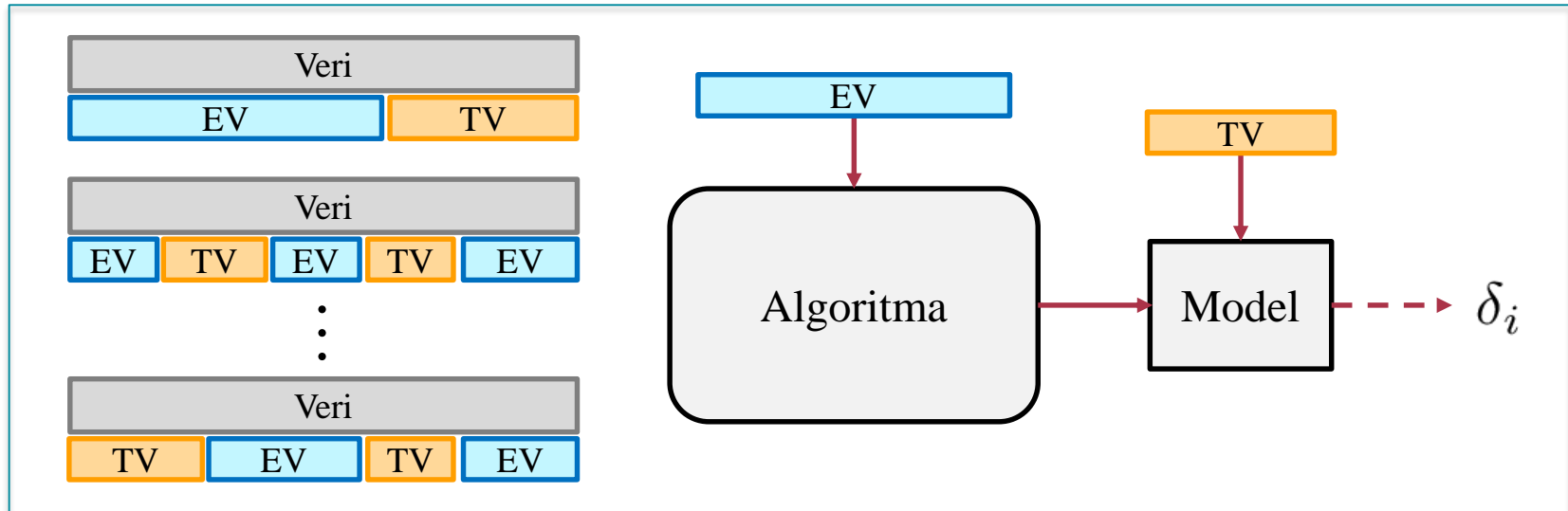
Son Eğitim



Verinin rastgele bölünmesinden dolayı ortaya çıkan belirsizlik?

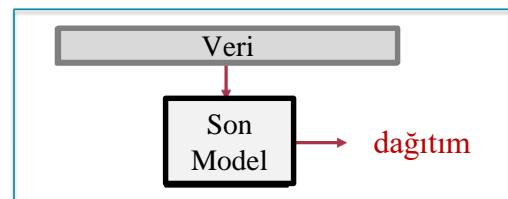
Tekrarlı Ayırma (Repeated Holdout)

$i = 1, \dots, k$



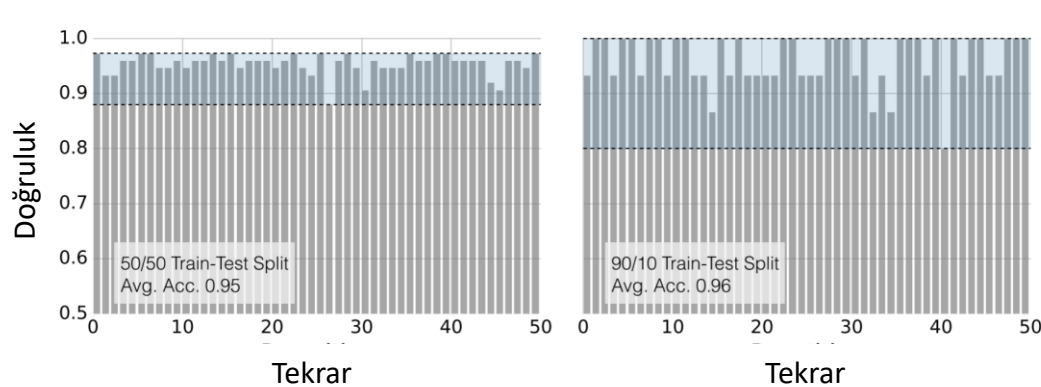
$$\delta = \frac{1}{k} \sum_{i=1}^k \delta_i$$

Son eğitim



(Tekrarlı) Ayırma Üzerine Notlar

- Tek bölme: nokta tahmini (yüksek varyans)
- Tekrarlı bölmeler: ortalama tahmin (düşük varyans)
- ‘Büyük Test Verisi – Kötümser Yanlılık vs. Küçük Test Verisi – Yüksek Varyans



- Katmanlama: Eğitim ve test verilerinde sınıf oranlarını koruma
- Diğer tekrar örnekleme yöntemleri: çapraz geçerlilik sınaması, zorlama tekniği (bootstrap)

Pratikte (ayırma (holdout))

```
1 import numpy as np
2 from sklearn.datasets import load_digits
3 from sklearn.model_selection import train_test_split
4 from sklearn.neighbors import KNeighborsClassifier
5
6 digits = load_digits()
7 print(digits.data.shape)
8
9 X, y = digits.data, digits.target
10
11 X_train, X_test, y_train, y_test = train_test_split(X, y,
12                                                    test_size=0.3,
13                                                    random_state=1,
14                                                    stratify=y)
15
16
17 KNN_classifier = KNeighborsClassifier(n_neighbors=3,
18                                     weights='uniform',
19                                     algorithm='kd_tree',
20                                     leaf_size=30,
21                                     p=2,
22                                     metric='minkowski',
23                                     metric_params=None,
24                                     n_jobs=1)
25
26 # Obtain a model with the training set
27 KNN_classifier.fit(X_train, y_train)
28 # Predict with the trained model
29 KNN_y_pred = KNN_classifier.predict(X_test)
30 # Evaluate the prediction accuracy
31 KNN_y_pred_acc = np.mean(y_test == KNN_y_pred)
32 # OR
33 # KNN_y_pred_acc = KNN_classifier.score(X_test, y_test)
34 print("Holdout prediction accuracy: ", KNN_y_pred_acc)
35
36 # Final model trained on the entire data set (deployment)
37 KNN_classifier.fit(X, y)
```

paketler ve fonksiyonlar

MNIST verisinin yüklenmesi

katmanlı veri ayrımı
(eğitim: %70 - test: %30)

tek algoritma (K-EYK)
belirli hiperparametreler

model eğitilmesi

model test edilmesi

performans değerlendirmesi

son model elde edilmesi

Pratikte (tekrarlı ayırma)

```
1 import numpy as np
2 from sklearn.datasets import load_digits
3 from sklearn.model_selection import train_test_split
4 from sklearn.neighbors import KNeighborsClassifier
5
6 digits = load_digits()
7 X, y = digits.data, digits.target
8
9 KNN_classifier = KNeighborsClassifier(n_neighbors=3,
10                                     weights='uniform',
11                                     algorithm='kd_tree',
12                                     leaf_size=30,
13                                     p=2,
14                                     metric='minkowski',
15                                     metric_params=None,
16                                     n_jobs=1)
17
18 rng = np.random.RandomState(seed=12345)
19 seeds = np.arange(10*5); rng.shuffle(seeds); seeds = seeds[:50] # Select the first 50
20
21 accuracies = []
22 # The repeated holdout starts here
23 for i in seeds:
24     X_train, X_test, y_train, y_test = train_test_split(X, y,
25                                                         test_size=split_ratio,
26                                                         random_state=i,
27                                                         stratify=y)
28     KNN_classifier.fit(X_train, y_train)
29     KNN_y_pred_i_acc = KNN_classifier.score(X_test, y_test)
30     accuracies.append(KNN_y_pred_i_acc)
31
32 accuracies = np.asarray(accuracies)
33 print("Repeated holdout average prediction accuracy: ", accuracies.mean())
34 # Final model trained on the entire data set (deployment)
35 KNN_classifier.fit(X, y)
36 print('Resubstitution (optimistic) prediction accuracy: ', KNN_classifier.score(X, y))
```

paket ve fonksiyonlar

MNIST verisinin yüklenmesi

tek algoritma (K-EYK)
belirlenmiş hiperparametreler

Tekrarlar için rastgele seed'ler

katmanlı veri ayrımı
(eğitim: %70 - test: %30)

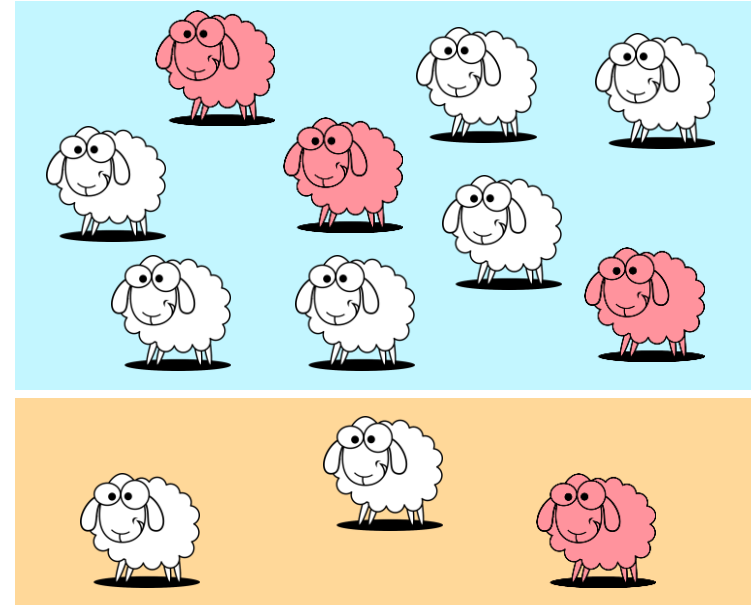
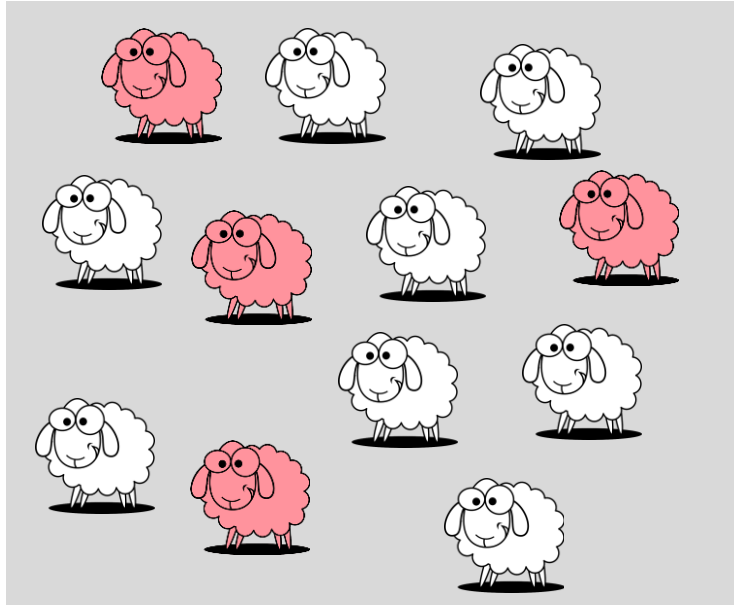
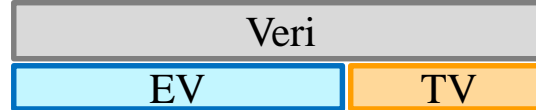
her bir model eğitilip,
test edilmesi

modellerin ortalama performansı

son model elde edilmesi

Tekrar Örnekleme

Katmanlama (Stratification)

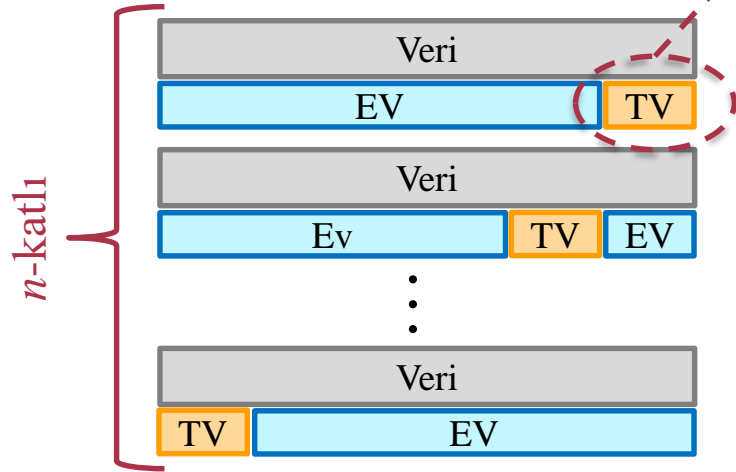


Tekrar Örnekleme

Çapraz Geçerlilik Sınaması (ÇGS)

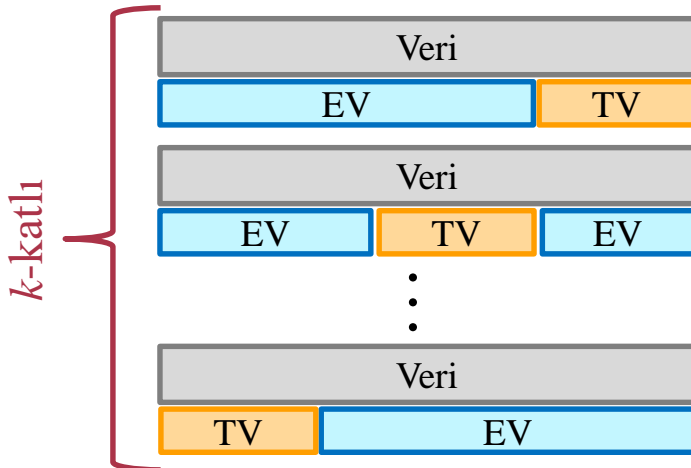
$$\{(x_i, y_i) : 1, \dots, n\}$$

Tek satır
veri



Biri-Hariç (Leave-One-Out) ÇGS

$$\delta = 1 - \frac{1}{n} \sum_{i=1}^n I(y_i \neq \hat{y}_i)$$

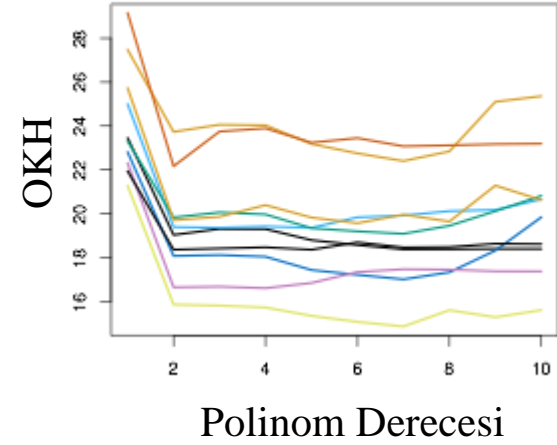


k -Katlı (k -Fold) ÇGS

$$\delta = \frac{1}{k} \sum_{i=1}^k \delta_i$$

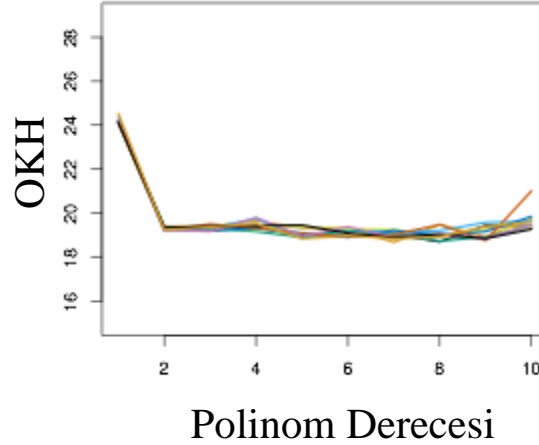
Genellikle ÇGS uygulamadan önce veri karıştırılır

$k = 2$



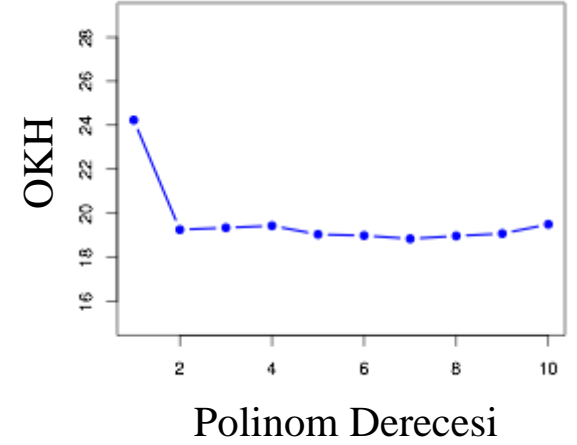
Geçerleme Kümesi Yaklaşımı
(Validation Set Approach)

$k = 10$



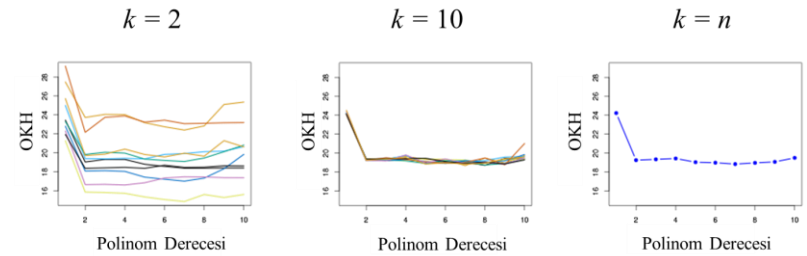
10-Katlı ÇGS

$k = n$



Biri-Hariç ÇGS

Yanlılık-Varyans İkilemi



Test Hatası

k



Varyans



Yanlılık



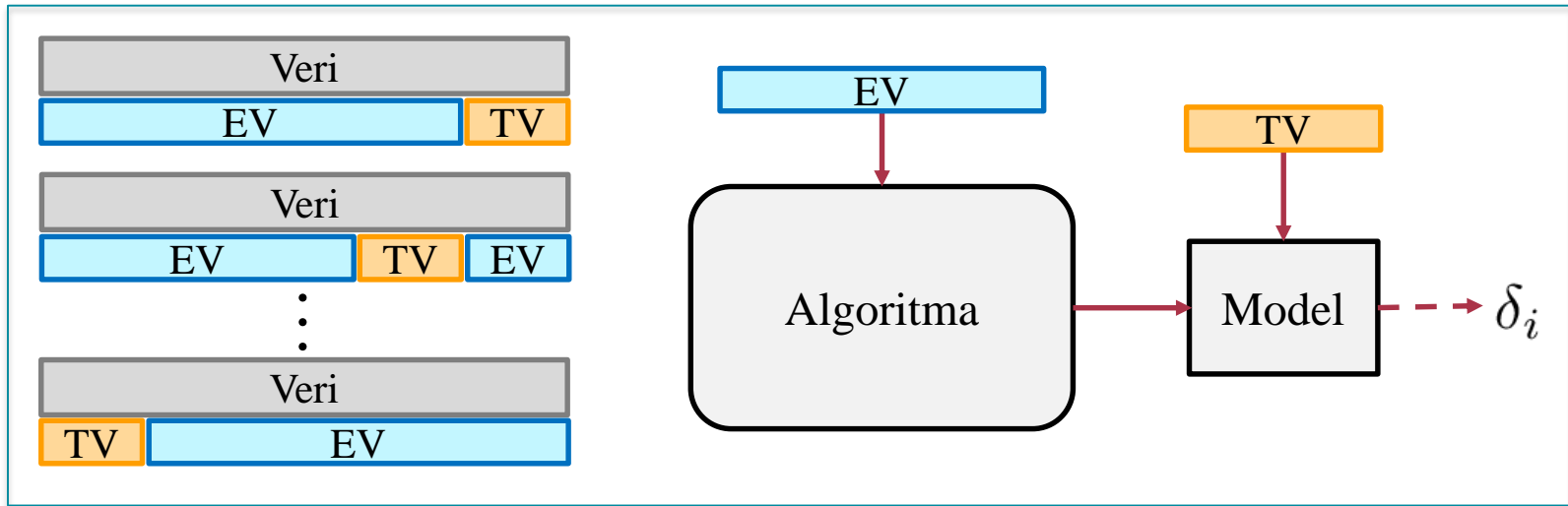
$$\left(\frac{k-1}{k}\right)n : \text{eğitim verisi boyutu}$$

$$\delta = \frac{1}{k} \sum_{i=1}^k \delta_i$$

$$\text{Var}(X + Y) = \text{Var}(X) + \text{Var}(Y) + 2\text{Cov}(X, Y)$$

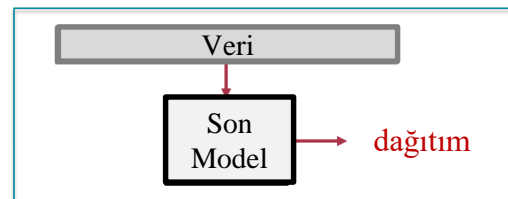
k -katlı Çapraz Geçerlilik Sınaması

$i = 1, \dots, k$

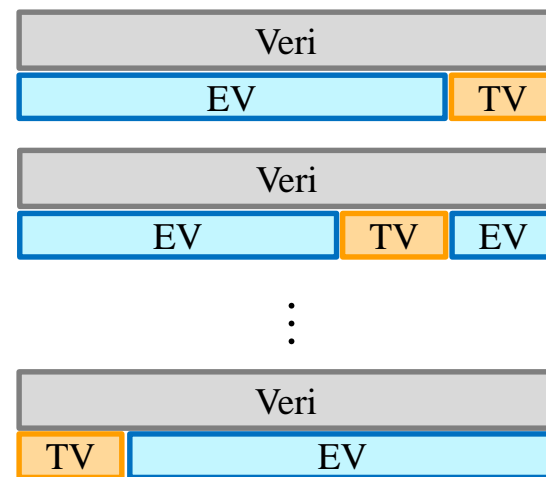
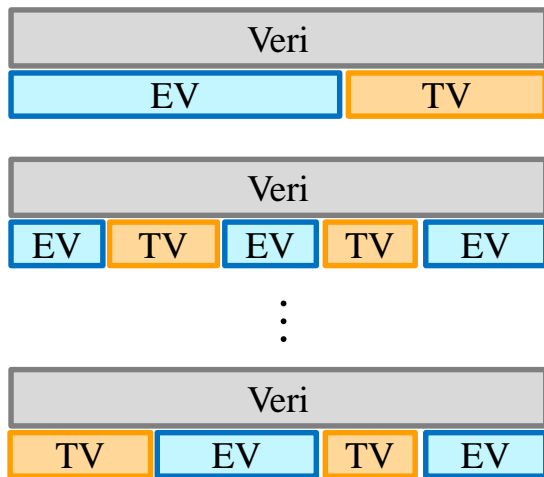


$$\delta = \frac{1}{k} \sum_{i=1}^k \delta_i$$


Son eğitim



Tekrarlı Ayırma vs. k -katlı ÇGS



Zorlama Tekniği (Bootstrap)

1	2	3		n
				
③	9	5	③	9
8	9	5	12	2
			⋮	
1	9	5	6	37

- Yerine koyarak
- **eğitim** verisi sayısı genellikle n
- dışarıda kalanlar (out-of-bag) **test** verisi
- İstatistik toplama (örn., bağlantım parametrelerinin varyansları)
- veri az olduğunda tercih sebebi

Tekrar Örnekleme

İstatistikler

iyimser yanlılık

$$\delta^r = \frac{1}{b} \sum_{j=1}^b \delta_j^r$$

tekrar yerine koyma doğruluğu
(resubstitution accuracy)

$$\delta^h = \frac{1}{b} \sum_{j=1}^b \delta_j^h$$

ayırma
doğruluğu

kötümser yanlılık

$$\delta^\bullet \pm t \sqrt{\frac{1}{b-1} \sum_{j=1}^b (\delta_j^\bullet - \delta^\bullet)^2}$$

güven aralığı
(normallik varsayımı altında)

Örnek : $b = 100, t_{95} = 1.984$

Tekrar Örnekleme

.632 Tahmini (Estimate)

$$\mathbb{P}(\text{bir örnek seçilmesi}) = 1 - \left(1 - \frac{1}{n}\right)^n \underset{n \gg 0}{\approx} 0.632$$

$$\begin{aligned}\delta &= \frac{1}{b} \sum_{j=1}^b (0.632 \delta_j^h + 0.368 \delta_j^r) \\ &= 0.632 \delta^h + 0.368 \delta^r\end{aligned}$$

biraz daha
iyimser yanlılık

.632+ yöntemi (Efron and Tibshirani, 1997)

Pratikte (zorlama tekniği)

```
1 import numpy as np
2 import seaborn as sns
3 from sklearn.datasets import load_digits
4 from sklearn.model_selection import train_test_split
5 from sklearn.neighbors import KNeighborsClassifier
6
7 digits = load_digits()
8 X, y = digits.data, digits.target
9
10 KNN_classifier = KNeighborsClassifier(n_neighbors=3,
11                                     weights='uniform',
12                                     algorithm='kd_tree',
13                                     leaf_size=30,
14                                     p=2,
15                                     metric='minkowski',
16                                     metric_params=None,
17                                     n_jobs=1)
18
19
20 rng = np.random.RandomState(seed=12345)
21
22 idx = np.arange(y.shape[0])
23 accuracies = []
24 for i in range(200):
25     train_idx = rng.choice(idx, size=idx.shape[0], replace=True)
26     # Out-of-bag sample as the test set
27     test_idx = np.setdiff1d(idx, train_idx, assume_unique=False)
28     boot_train_X, boot_train_y = X[train_idx], y[train_idx]
29     boot_test_X, boot_test_y = X[test_idx], y[test_idx]
30     KNN_classifier.fit(boot_train_X, boot_train_y)
31     acc = KNN_classifier.score(boot_test_X, boot_test_y)
32     accuracies.append(acc)
33
34 mean = np.mean(accuracies)
35 print("Bootstrap average prediction accuracy: ", mean)
36
37 # Final model trained on the entire data set (deployment)
38 KNN_classifier.fit(X, y)
```

paketler ve fonksiyonlar

MNIST verisinin yüklenmesi

tek algoritma (K-EYK)
belirli hiperparametreler

zorlama tekniği için rastgele seed'ler

yerine koyarak örnekleme(eğitim verisi)
kalanları test için işaretleme

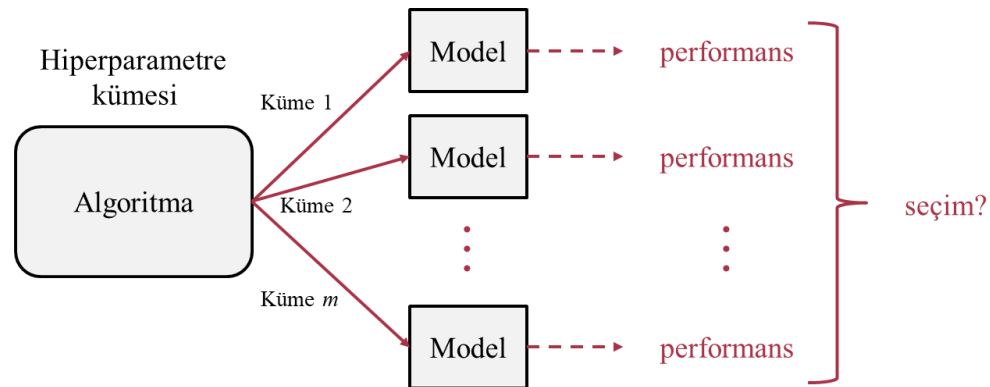
her bir model eğitilip, test edilmesi

modellerin ortalama performansı

son model elde edilmesi

Model Seçimi

(algoritma belirlenmiş, ‘en iyi’ parametreler aranıyor)



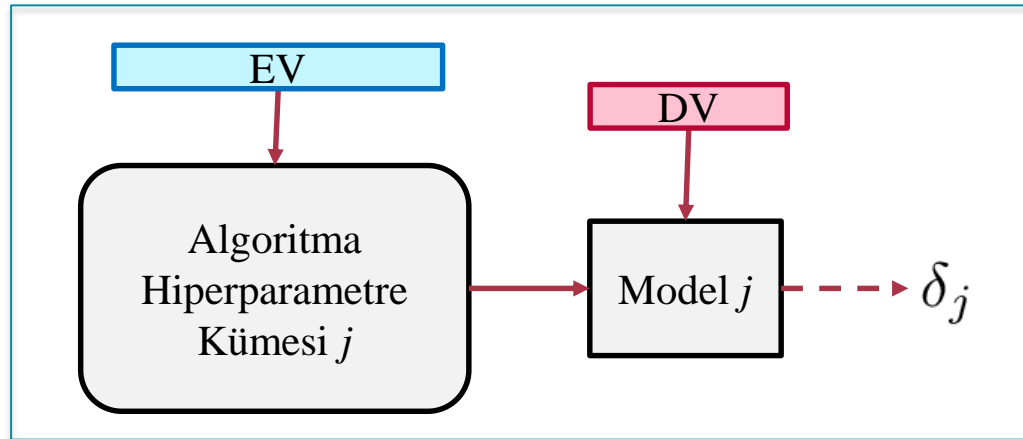
Hipotez Uzayı
(Hipotez \cong Model)

$$\hat{Y} = \hat{f}(X)$$

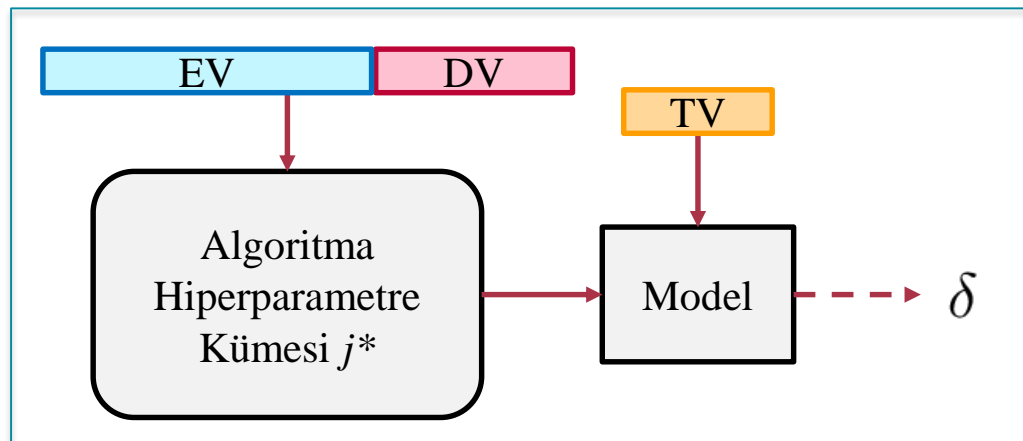
Üçlü Ayırma



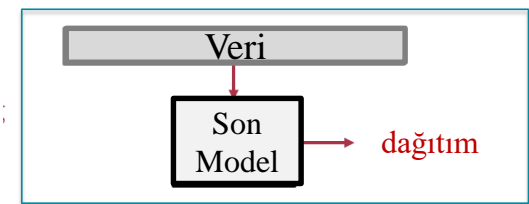
$j = 1, \dots, m$



$$j^* = \arg \max \{ \delta_j : j = 1, \dots, m \}$$

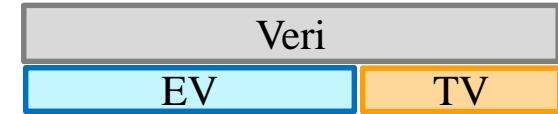


Son eğitim

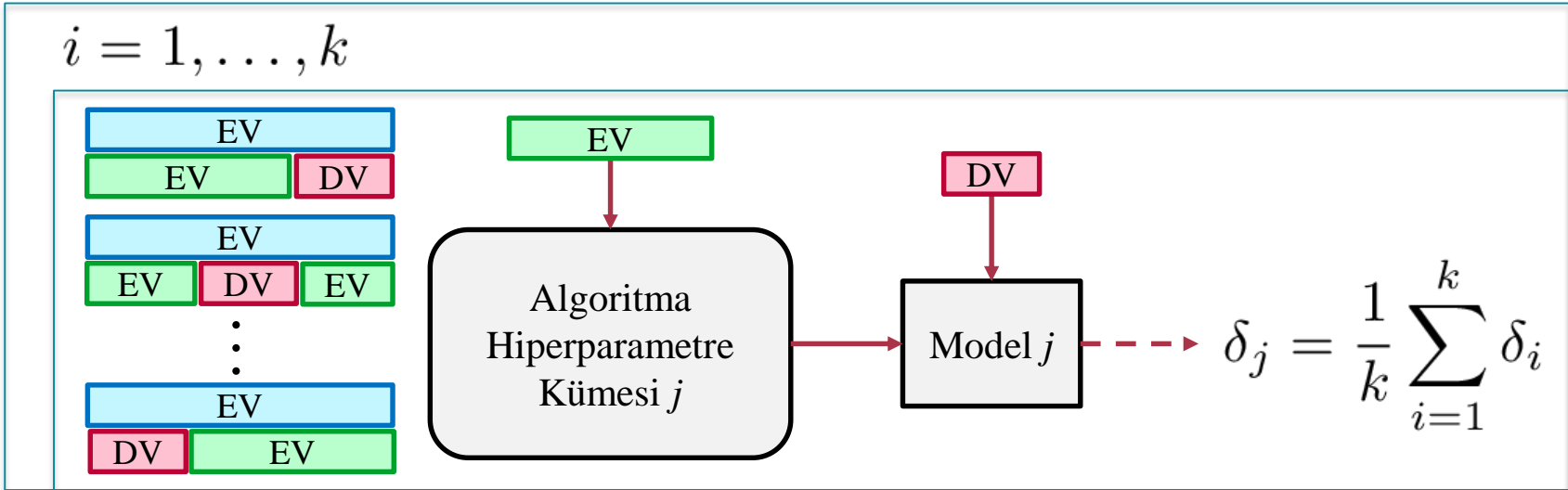


k -katlı ÇGS

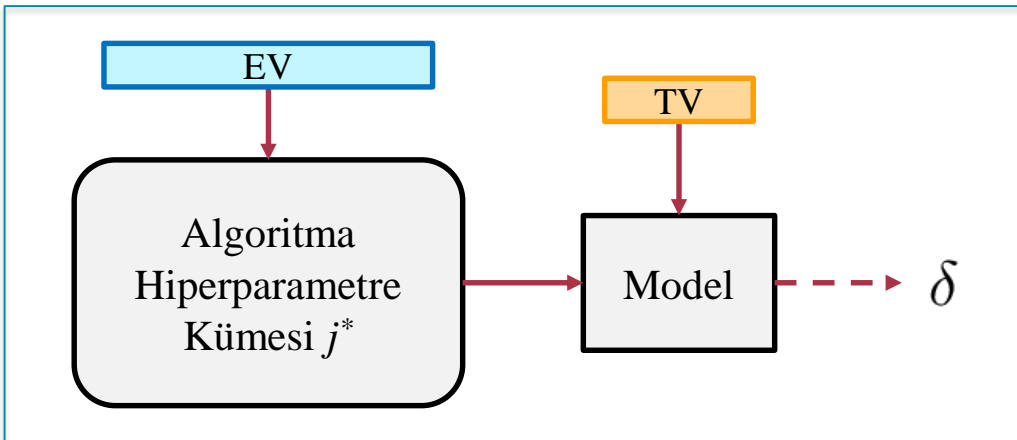
$j = 1, \dots, m$



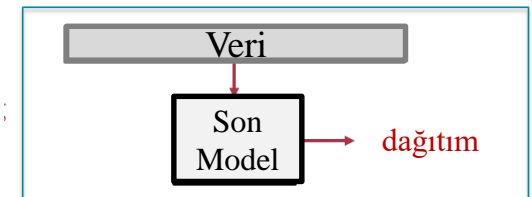
$i = 1, \dots, k$



$$j^* = \arg \max \{ \delta_j : j = 1, \dots, m \}$$



Son eğitim



Pratikte (k -katlı ÇGS)

```
1 import numpy as np
2 from sklearn.datasets import load_digits
3 from sklearn.model_selection import train_test_split
4 from sklearn.model_selection import StratifiedKFold
5 from sklearn.neighbors import KNeighborsClassifier
6
7 digits = load_digits()
8 X, y = digits.data, digits.target
9 # Hyperparameter K = 1, 2, ..., 7
10 params = range(1, 8)
11 cv_acc, cv_std, cv_stderr = [], [], []
12 X_train, X_test, y_train, y_test = train_test_split(X, y,
13                                                    test_size=0.3,
14                                                    random_state=1,
15                                                    stratify=y)
16 # 10-fold stratified cross validation
17 cv = StratifiedKFold(n_splits=10, shuffle=True, random_state=1)
18 for c in params:
19     KNN_classifier = KNeighborsClassifier(n_neighbors=c,
20                                         weights='uniform',
21                                         algorithm='kd_tree',
22                                         leaf_size=30,
23                                         p=2,
24                                         metric='minkowski',
25                                         metric_params=None,
26                                         n_jobs=1)
27     all_acc = []
28     for train_index, valid_index in cv.split(X_train, y_train):
29         pred = KNN_classifier.fit(X_train[train_index], y_train[train_index])\
30             .predict(X_train[valid_index])
31         acc = np.mean(y_train[valid_index] == pred)
32         all_acc.append(acc)
33
34     all_acc = np.array(all_acc)
35     y_pred_cv10_mean = all_acc.mean(); y_pred_cv10_std = all_acc.std()
36     y_pred_cv10_stderr = y_pred_cv10_std / np.sqrt(10)
37     cv_acc.append(y_pred_cv10_mean); cv_std.append(y_pred_cv10_std)
38     cv_stderr.append(y_pred_cv10_stderr)
39 best_K = np.argmax(cv_acc)
40 KNN_classifier = KNeighborsClassifier(n_neighbors=params[best_K],
41                                     weights='uniform',
42                                     algorithm='kd_tree',
43                                     leaf_size=30,
44                                     p=2,
45                                     metric='minkowski',
46                                     metric_params=None,
47                                     n_jobs=1)
48 KNN_classifier.fit(X_train, y_train)
49 KNN_y_pred_acc = KNN_classifier.score(X_test, y_test)
50 print('K-fold prediction accuracy: ', KNN_y_pred_acc)
51 # Final model trained on the entire data set (deployment)
52 KNN_classifier.fit(X, y)
```

paketler ve fonksiyonlar

MNIST verisinin yüklenmesi
hiperparametre aralığı

katmanlı veri ayrımı (%70 - %30)

10-katlı ÇGS hazırlığı

tek algoritma (K -EYK)
farklı hiperparametreler

10-katlı ÇGS uygulaması

istatistikleri kaydetme

‘en iyi’ parametrelerle eğitme

performans değerlendirmesi

son modelin elde edilmesi

Pratikte (tekrarlı k -katlı ÇGS)

```
1 import numpy as np
2 import matplotlib.pyplot as plt
3 from sklearn.datasets import load_digits
4 from sklearn.model_selection import train_test_split
5 from sklearn.model_selection import StratifiedKFold
6 from sklearn.neighbors import KNeighborsClassifier
7 digits = load_digits()
8 X, y = digits.data, digits.target
9 # Hyperparameter K = 1, 2, ..., 7
10 params = range(1, 8); cv_acc, cv_std, cv_stderr = [], [], []
11 params_by_seed = []
12 X_train, X_test, y_train, y_test = train_test_split(X, y,
13                                                    test_size=0.3,
14                                                    random_state=1,
15                                                    stratify=y)
16 rng = np.random.RandomState(seed=12345); seeds = np.arange(10**5)
17 rng.shuffle(seeds); seeds = seeds[:5] # Select the first 5 seeds
18 for seed in seeds:
19     # 10-fold stratified cross validation
20     cv = StratifiedKFold(n_splits=10, shuffle=True, random_state=seed)
21     acc_by_param = []
22     for c in params:
23         KNN_classifier = KNeighborsClassifier(n_neighbors=c,
24                                             weights='uniform',
25                                             algorithm='kd_tree',
26                                             leaf_size=30,
27                                             p=2,
28                                             metric='minkowski',
29                                             metric_params=None,
30                                             n_jobs=1)
31     all_acc = []
32     for train_index, valid_index in cv.split(X_train, y_train):
33         pred = KNN_classifier.fit(X_train[train_index], y_train[train_index])\
34             .predict(X_train[valid_index])
35         acc = np.mean(y_train[valid_index] == pred)
36         all_acc.append(acc)
37     all_acc = np.array(all_acc)
38     acc_by_param.append(all_acc.mean())
39     params_by_seed.append(acc_by_param)
40 best_K = np.argmax(np.mean(params_by_seed, 0))
41 KNN_classifier = KNeighborsClassifier(n_neighbors=params[best_K],
42                                     weights='uniform',
43                                     algorithm='kd_tree',
44                                     leaf_size=30,
45                                     p=2,
46                                     metric='minkowski',
47                                     metric_params=None,
48                                     n_jobs=1)
49 KNN_classifier.fit(X_train, y_train)
50 KNN_y_pred_acc = KNN_classifier.score(X_test, y_test)
51 print("Repeated K-fold prediction accuracy: ", KNN_y_pred_acc)
52 # Final model trained on the entire data set (deployment)
53 KNN_classifier.fit(X, y)
```

paketler ve fonksiyonlar

MNIST veri setinin yüklenmesi
hiperparametrele aralığı

katmanlı veri ayrımı (%70 - %30)

tekrarlar için rastgele seed'ler
farklı seed'ler ile 10-katlı ÇGS
hazırlığı

tek algoritma (K -EYK)
farklı hiperparametreler

10-katlı ÇGS uygulaması
ve istatistiklerin kaydedilmesi

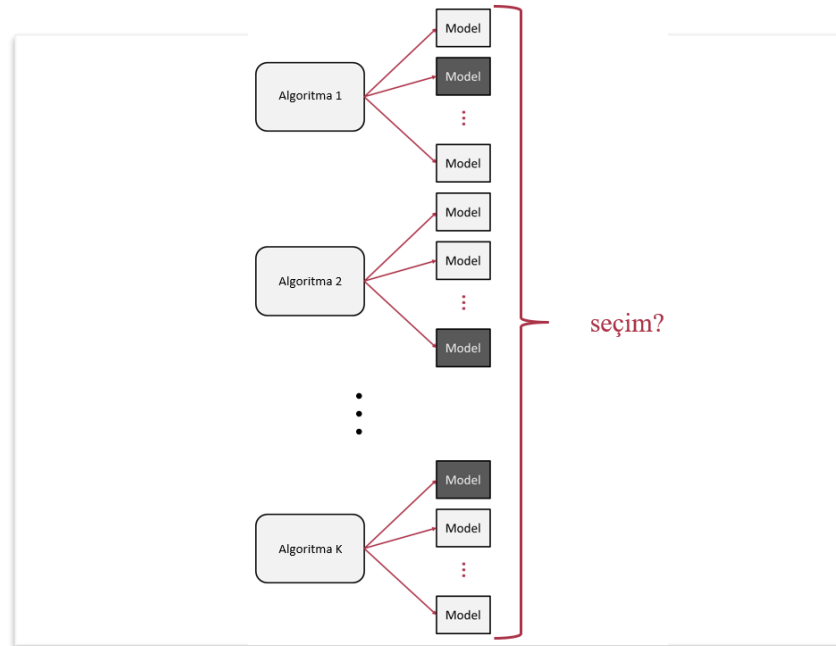
'en iyi' parametre ile eğitme

performans değerlendirmesi
son modelin elde edilmesi

Model Seçimi Notları

- Etraflı performans genellemesi test setine bağlıdır
- k -katlı ÇGS büyük veriyle veya yavaş algoritmalarla çok zaman alır
- Veri büyük olduğunda üçlü ayırma yöntemi daha hızlıdır
- Ayırma yöntemi arasına 2-katlı ÇGS olarak anılır (tam olarak doğru olmasa da)
- k -katlı ÇGS için genelgeçer bir k değeri yoktur (genellikle 5 veya 10)
- Kabaca: Biri-hariç ÇGS (küçük veri), k -katlı ÇGS ya da üçlü ayırma (büyük veri)

Algoritma Seçimi

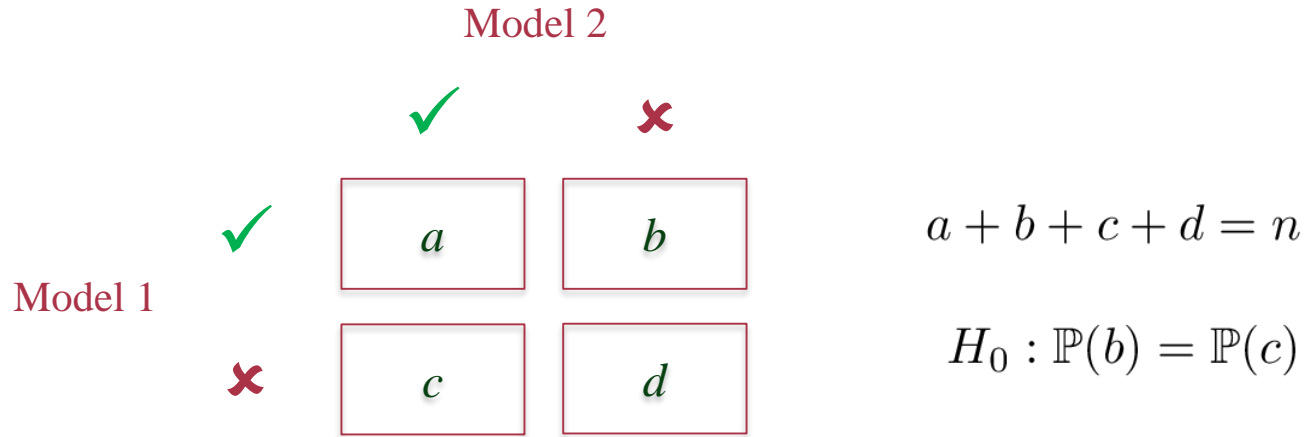


Model Kıyaslaması

- İki model kıyaslarken
 - z -puanlarına dayalı fark testleri
 - McNemar Testi
- İkiden fazla modeli kıyaslarken
 - Cochran Q Testi
 - F -testi
 - Bağımlı t -testi, birleştirilmiş F -testi
 - İç içe çapraz geçerlilik sınaması
- Ve dahası...

İki modeli kıyaslama

McNemar Testi

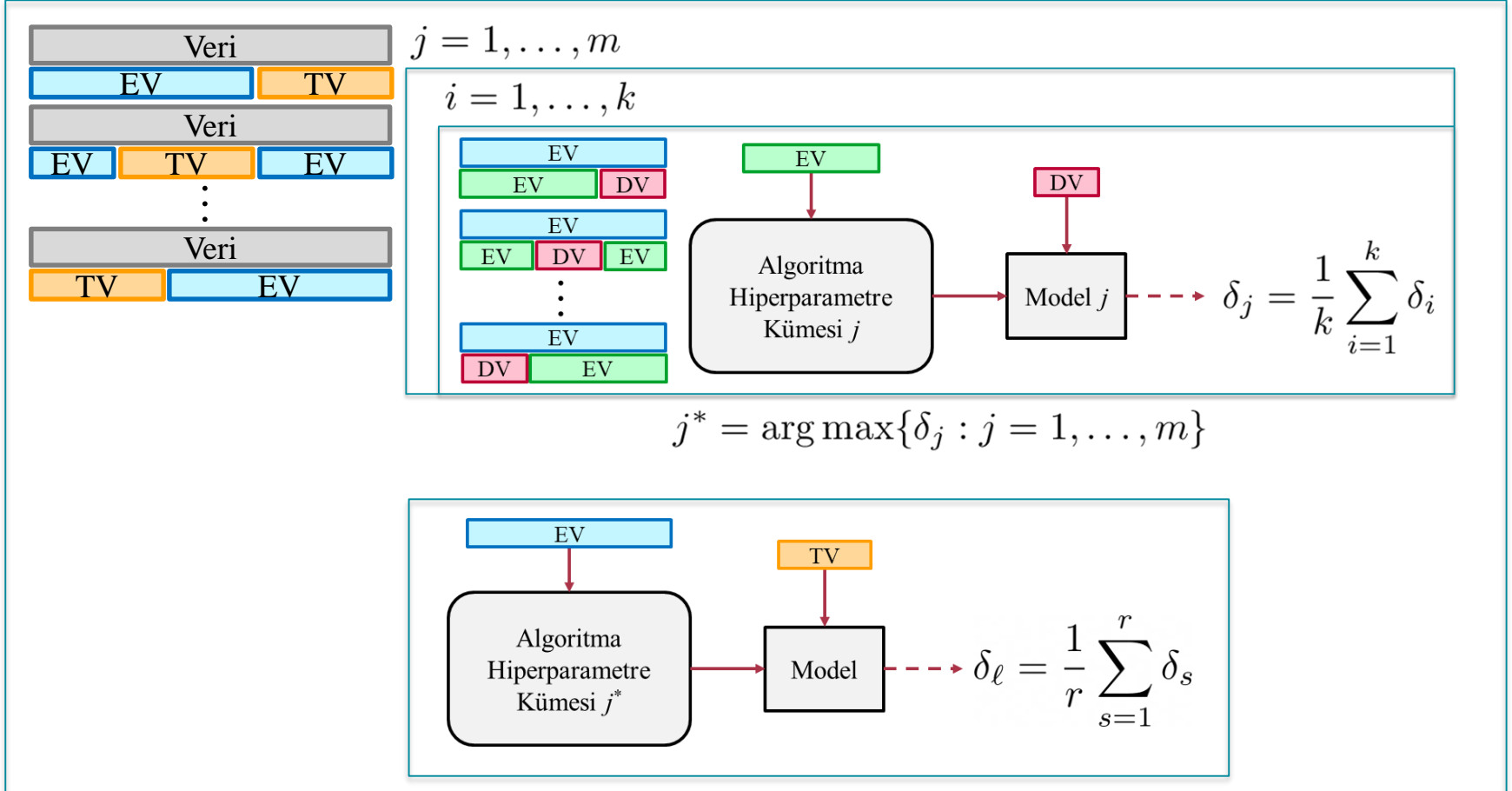


$$\text{Test istatistiği : } \chi^2 = \frac{(|b - c| - 1)^2}{b + c}$$

1. Anlamlılık düzeyi seçimi (e.g., 0.05)
2. Test: p -değeri değerlendirmesi
3. Sıfır hipotezini kabul etme ya da reddetme

İç içe k -katlı ÇGS (Nested k -fold CV)

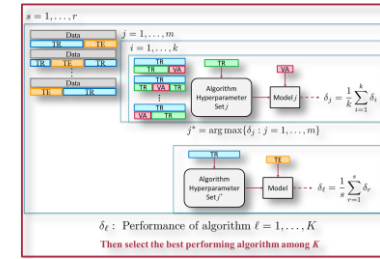
$s = 1, \dots, r$



δ_ℓ : Algoritma $\ell = 1, \dots, K$ performansı

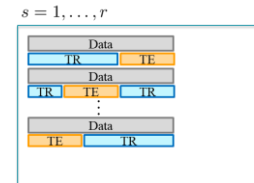
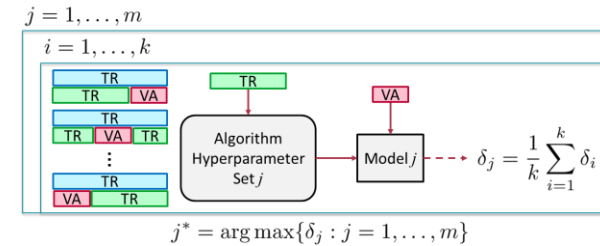
Daha sonra K algoritma arasından en iyi performans gösteren seçilir

Pratikte (iç içe k -kathı ÇGS)



aday algoritmalar

hiperparametre kümelerinin belirlenmesi



en iyi algoritmanın performansının belirlenmesi

son modelin elde edilmesi

```
1 # Initializing Classifiers
2 clf1 = LogisticRegression(...)
3 clf2 = KNeighborsClassifier(...)
4 clf3 = DecisionTreeClassifier(...)
5 clf4 = SVC(...)
6
7 # Setting up the parameter grids
8 param_grid1 = [{'clf1_penalty': ['l2'],
9                  'clf1_C': np.power(10., np.arange(-4, 4))}]
10 param_grid2 = [{'clf2_n_neighbors': list(range(1, 10)),
11                 'clf2_p': [1, 2]}]
12 param_grid3 = [{'max_depth': list(range(1, 10)) + [None],
13                 'criterion': ['gini', 'entropy']}]
14 param_grid4 = [{'clf4_kernel': ['rbf'],
15                 'clf4_C': np.power(10., np.arange(-4, 4)),
16                 'clf4_gamma': np.power(10., np.arange(-5, 0))},
17                 {'clf4_kernel': ['linear'],
18                  'clf4_C': np.power(10., np.arange(-4, 4))}]
19
20 # Setting up multiple GridSearchCV objects, one for each algorithm
21 gridcvs = {}
22 inner_cv = StratifiedKFold(n_splits=2, shuffle=True, random_state=1)
23 for pgrid, est, name in zip((param_grid1, param_grid2,
24                             param_grid3, param_grid4),
25                             (pipe1, pipe2, clf3, pipe4),
26                             ('Softmax', 'KNN', 'DTree', 'SVM')):
27     gcv = GridSearchCV(estimator=est,
28                        param_grid=pgrid,
29                        scoring='accuracy',
30                        n_jobs=1,
31                        cv=inner_cv,
32                        verbose=0,
33                        refit=True)
34     gridcvs[name] = gcv
35
36 outer_cv = StratifiedKFold(n_splits=5, shuffle=True, random_state=1)
37 for name, gs_est in sorted(gridcvs.items()):
38     nested_score = cross_val_score(gs_est,
39                                     X=X_train,
40                                     y=y_train,
41                                     cv=outer_cv,
42                                     n_jobs=1)
43     print('%s | outer ACC %.2f%% +/- %.2f' %
44           (name, nested_score.mean() * 100, nested_score.std() * 100))
45
46 # The "best" algorithm was SVM
47 best_algo = gridcvs['SVM']
48 best_algo.fit(X_train, y_train)
49 train_acc = accuracy_score(y_true=y_train, y_pred=best_algo.predict(X_train))
50 test_acc = accuracy_score(y_true=y_test, y_pred=best_algo.predict(X_test))
51
52 # Final model trained on the entire data set (deployment)
53 best_algo.fit(X, y)
```

ve...



Reza Zadeh ✓

@Reza_Zadeh

Follow



When you use a 10 layer Deep Neural Network where Logistic Regression would suffice



Özet

- Performans genelleme
- Model seçimi (hiperparametre ayarlaması (tuning))
- Ayırma yöntemleri
- Tekrar Örneklemeye: zorlama yöntemi (bootstrapping), ÇGS
- Algoritma seçimi
- Fark testleri
- Diğer yöntemler