

מתודולוגיות עבודה ב-DS

דגשים בתהליך העבודה - הרצת אלגוריתמים

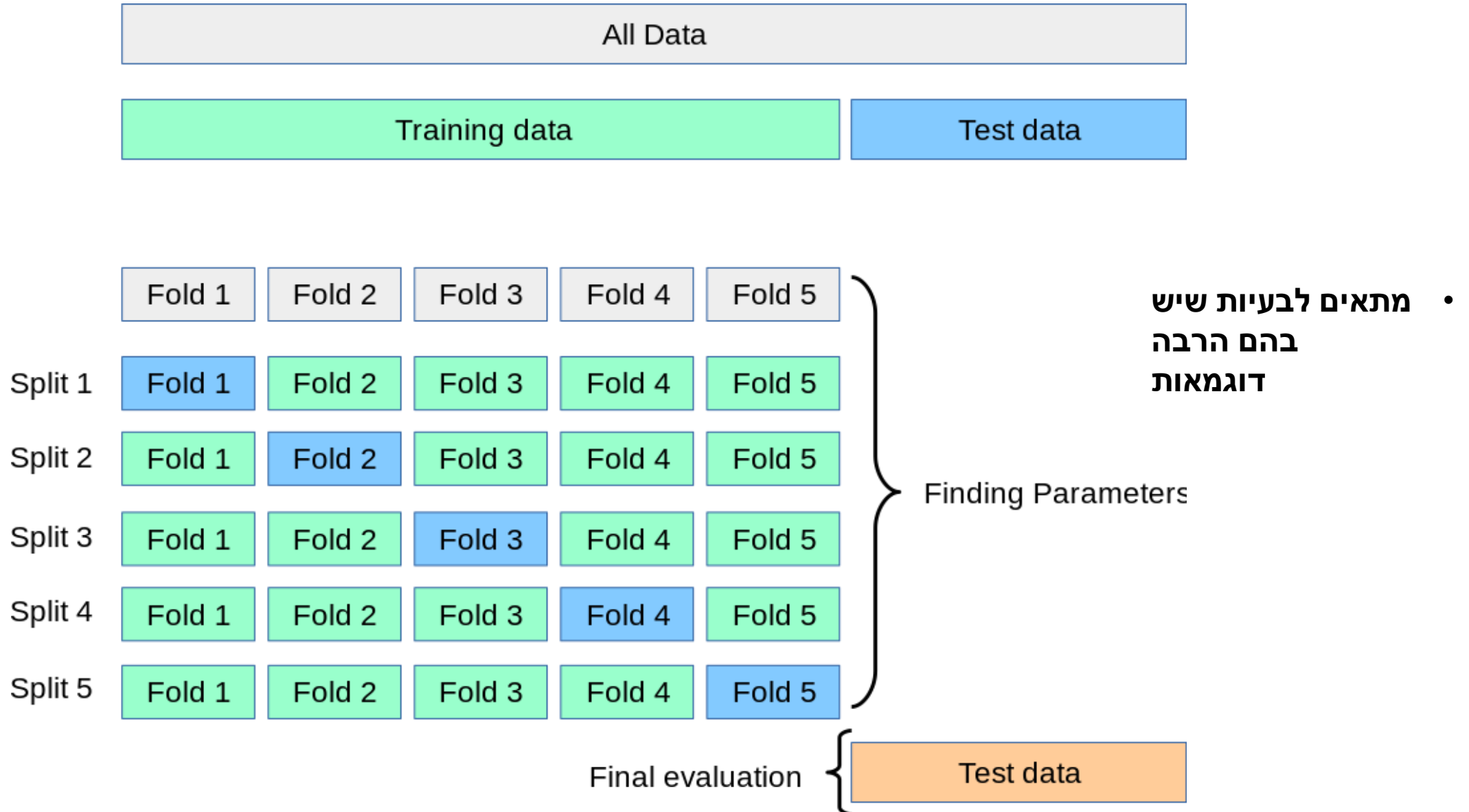
תוכן עניינים

- Test Harness
- K-Fold Cross Validation
- Hyperparameters
- דגשים לסיווג
- Confusion Matrix
- Receiver operating characteristic
- Area under the ROC Curve
- Ensembles

Test Harness

- בכדי להבין איזה אלגוריתם פותר את הבעיה בצורה הטובה ביותר יש לבצע את הבדיקות כך שיתאפשר להעריך את ביצועי המכונה גם במקרי אמת
- לצורך כך יש לממש בדיקות באופן כזה שיאפשרו להעריך ביצועים של מכונות שונות
- ניתן לממש Test Harness ע"י חלוקה נכונה של ה-Data עליו מבוצע הבדיקות
- מימוש של Test Harness יכול להיעשות במס' צורות
 - חלוקה ל-Train & Test
 - חלוקה ל-Train, Cross & Test
 - K-Fold Cross Validation
- ההרצה הסופית מבוצעת על ה-Test בלבד למניעת leakage

K-Fold Cross Validation



K-Fold Cross Validation

```
x, y = load_iris(return_X_y=True)
clf = svm.SVC(kernel='linear', C=1, random_state=0)
scores = cross_validate(clf, x, y, cv=5, scoring='accuracy')
print("Test Score")
print(scores['test_score'])

print("Accuracy: %0.2f (+/- %0.2f)" % (scores['test_score'].mean(),
scores['test_score'].std() * 2))
```

K-Fold Cross Validation

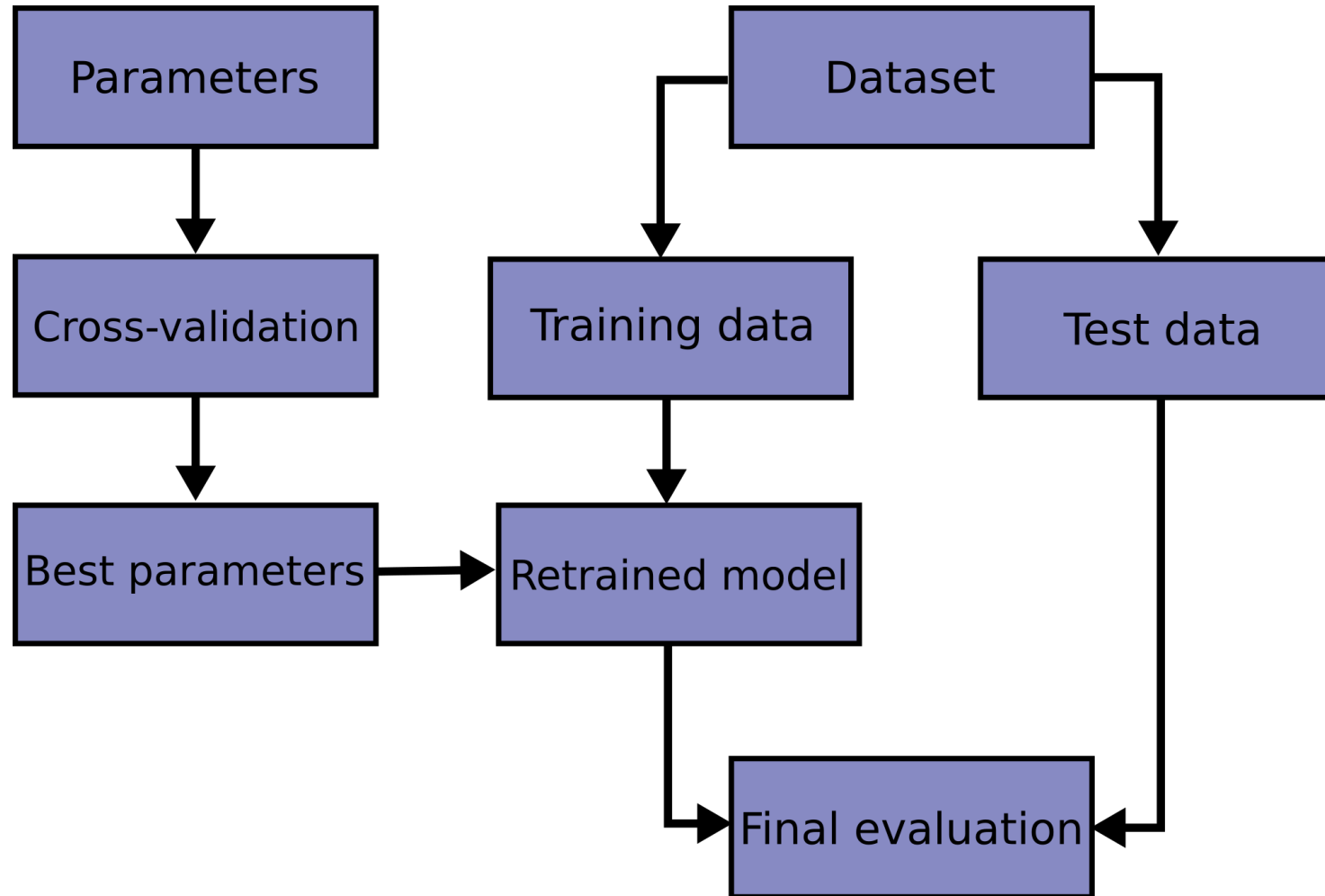
`sklearn.model_selection.cross_val_score`

```
sklearn.model_selection. cross_val_score (estimator, X, y=None, groups=None, scoring=None, cv='warn',  
n_jobs=None, verbose=0, fit_params=None, pre_dispatch='2*n_jobs', error_score='raise-deprecating') ¶ \[source\]
```

Hyperparameters

- מטרת ה-Hyperparameters היא לימוד הפרמטרים הבונים את המודל ומועברים אליו, שאינם נלמדים במסגרת המודל, לדוגמא: C , Kernel & Gamma במודל של SVC
- הלימוד של הפרמטרים מבוצע ע"י הרצה של כל הקומבינציות האפשריות של הפרמטרים, והבאת הציון הגבוה ביותר
- חובה לבצע Hyperparameters למציאת התוצאה הטובה ביותר עבור ה-Cross Validation

Hyperparameters



Hyperparameters

```
#GridSearchCV
```

```
parameters = {'kernel':('linear', 'rbf'), 'C':[1, 10]}
```

```
svc = svm.SVC(gamma="scale")
```

```
clf = GridSearchCV(svc, parameters, cv=5)
```

```
clf.fit(x, y)
```

```
sorted(clf.cv_results_.keys())
```

```
clf.best_estimator_
```

```
clf.best_params_
```

```
clf.best_score_
```

Hyperparameters

`sklearn.model_selection.GridSearchCV`

```
class sklearn.model_selection. GridSearchCV (estimator, param_grid, scoring=None, fit_params=None,  
n_jobs=None, iid='warn', refit=True, cv='warn', verbose=0, pre_dispatch='2*n_jobs', error_score='raise-deprecating',  
return_train_score='warn')
```

[\[source\]](#)

דגשים לסיווג - Confusion Matrix

		True condition				
Total population		Condition positive	Condition negative	Prevalence = $\frac{\Sigma \text{Condition positive}}{\Sigma \text{Total population}}$	Accuracy (ACC) = $\frac{\Sigma \text{True positive} + \Sigma \text{True negative}}{\Sigma \text{Total population}}$	
Predicted condition	Predicted condition positive	True positive , Power	False positive , Type I error	Positive predictive value (PPV), Precision = $\frac{\Sigma \text{True positive}}{\Sigma \text{Predicted condition positive}}$	False discovery rate (FDR) = $\frac{\Sigma \text{False positive}}{\Sigma \text{Predicted condition positive}}$	
	Predicted condition negative	False negative , Type II error	True negative	False omission rate (FOR) = $\frac{\Sigma \text{False negative}}{\Sigma \text{Predicted condition negative}}$	Negative predictive value (NPV) = $\frac{\Sigma \text{True negative}}{\Sigma \text{Predicted condition negative}}$	
		True positive rate (TPR), Recall, Sensitivity, probability of detection = $\frac{\Sigma \text{True positive}}{\Sigma \text{Condition positive}}$	False positive rate (FPR), Fall-out, probability of false alarm = $\frac{\Sigma \text{False positive}}{\Sigma \text{Condition negative}}$	Positive likelihood ratio (LR+) = $\frac{\text{TPR}}{\text{FPR}}$	Diagnostic odds ratio (DOR) = $\frac{\text{LR+}}{\text{LR-}}$	F ₁ score = $2 \cdot \frac{\text{Precision} \cdot \text{Recall}}{\text{Precision} + \text{Recall}}$
		False negative rate (FNR), Miss rate = $\frac{\Sigma \text{False negative}}{\Sigma \text{Condition positive}}$	Specificity (SPC), Selectivity, True negative rate (TNR) = $\frac{\Sigma \text{True negative}}{\Sigma \text{Condition negative}}$	Negative likelihood ratio (LR-) = $\frac{\text{FNR}}{\text{TNR}}$		

דגשים לסיווג - Confusion Matrix

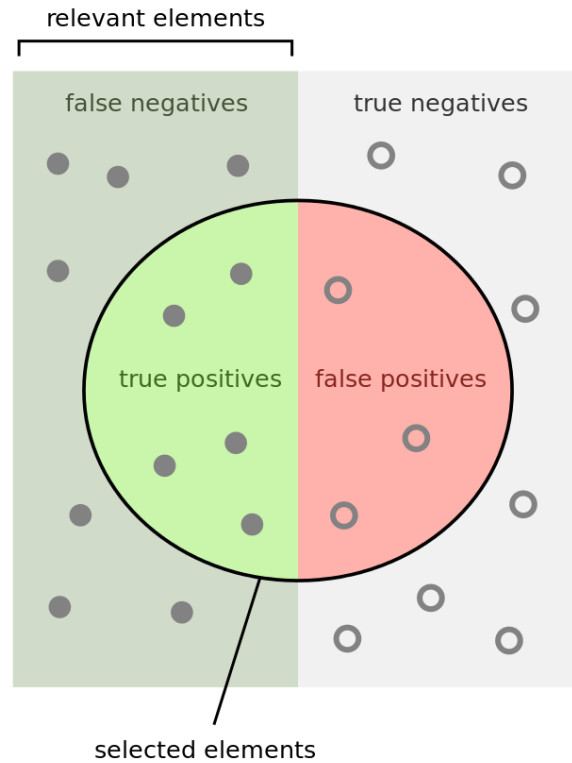
	לקוחות עם קשיי תשלום	לקוחות ללא קשיי תשלום
חיזוי קשיי תשלום חיובי	59	12
חיזוי קשיי תשלום שלילי	34	150

• TP – 59, NT – 150, PF – 12, NF – 34

• $\text{Recall} = (59 + 34) / 59$

$\text{persicion} = (12 + 59) / 150$

דגשים לסיווג - Recall & percision



How many selected items are relevant?

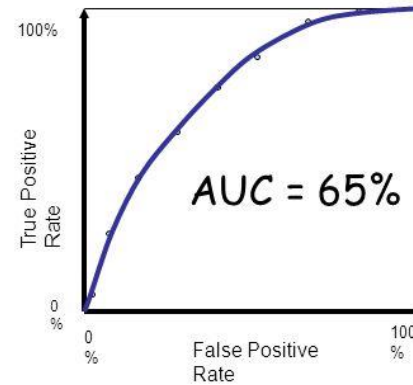
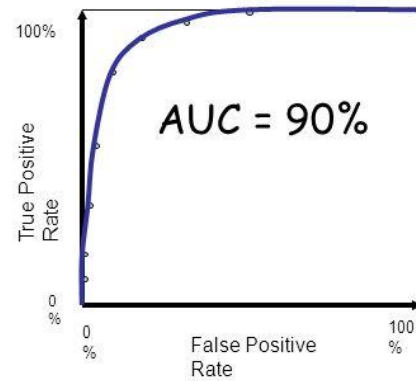
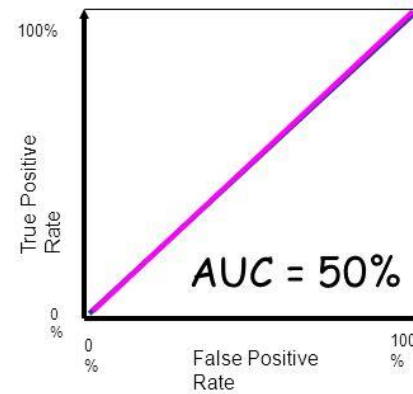
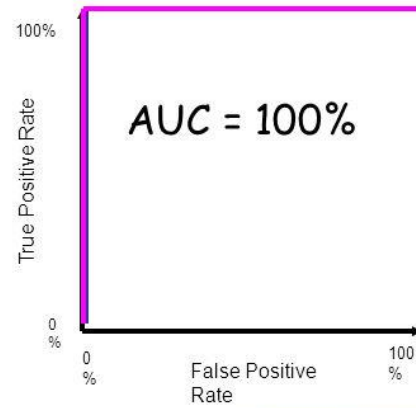
$$\text{Precision} = \frac{\text{true positives}}{\text{true positives} + \text{false positives}}$$

How many relevant items are selected?

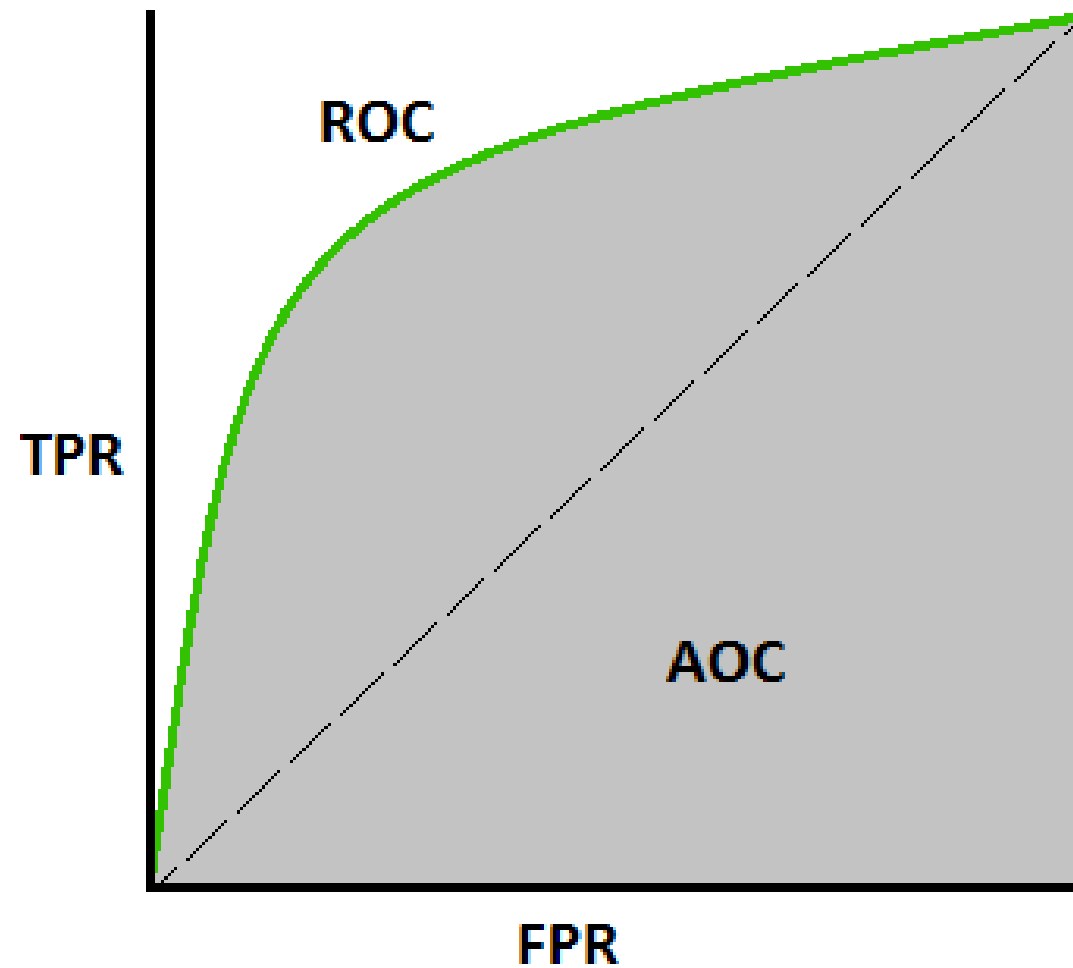
$$\text{Recall} = \frac{\text{true positives}}{\text{true positives} + \text{false negatives}}$$

דגשים לסיווג

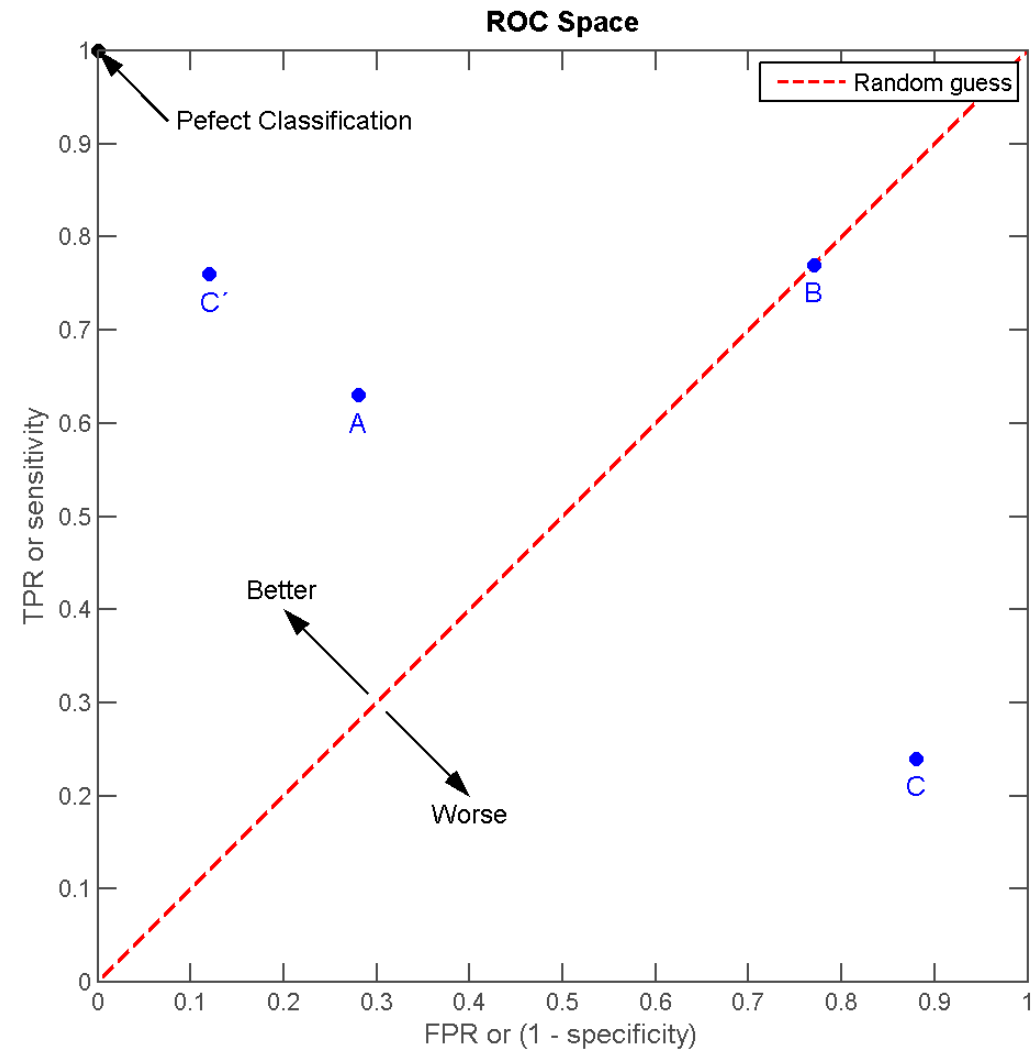
AUC for ROC curves



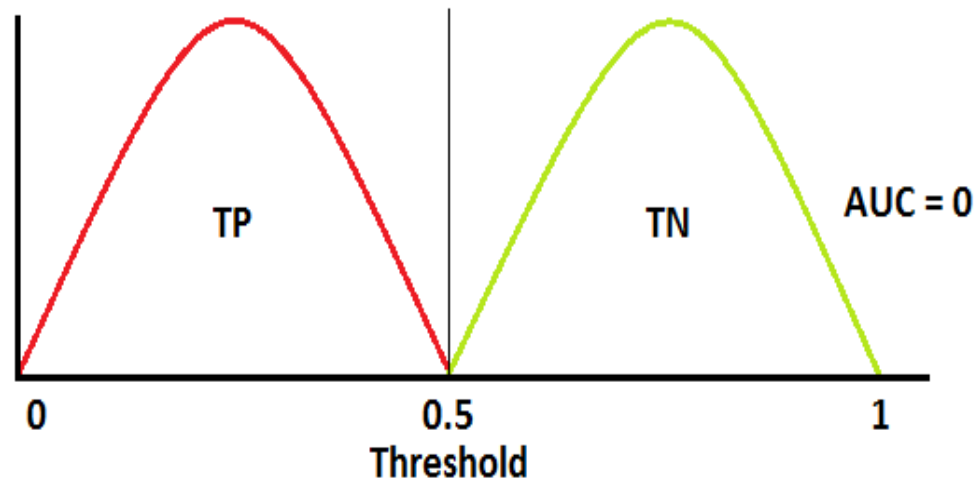
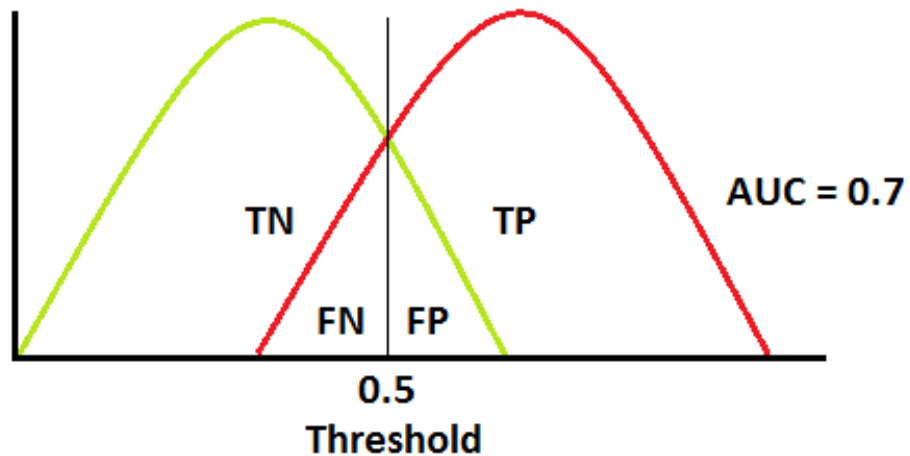
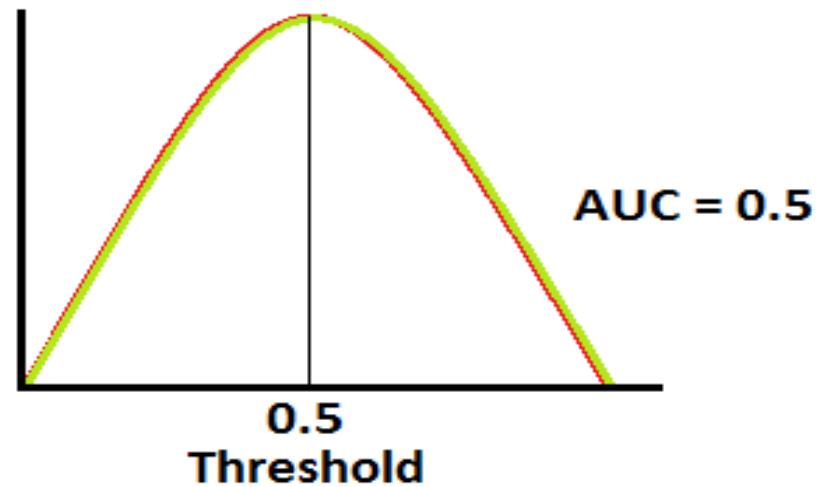
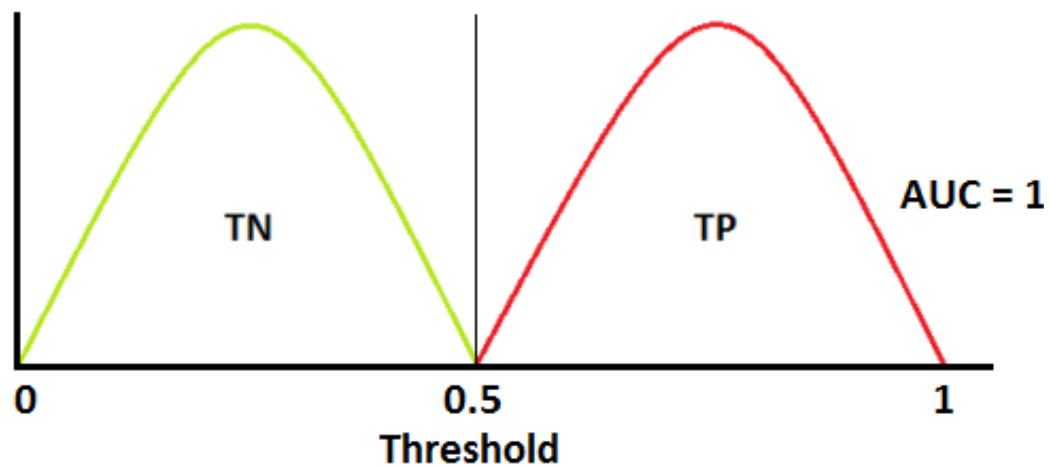
דגשים לסיווג



דגשים לסיווג

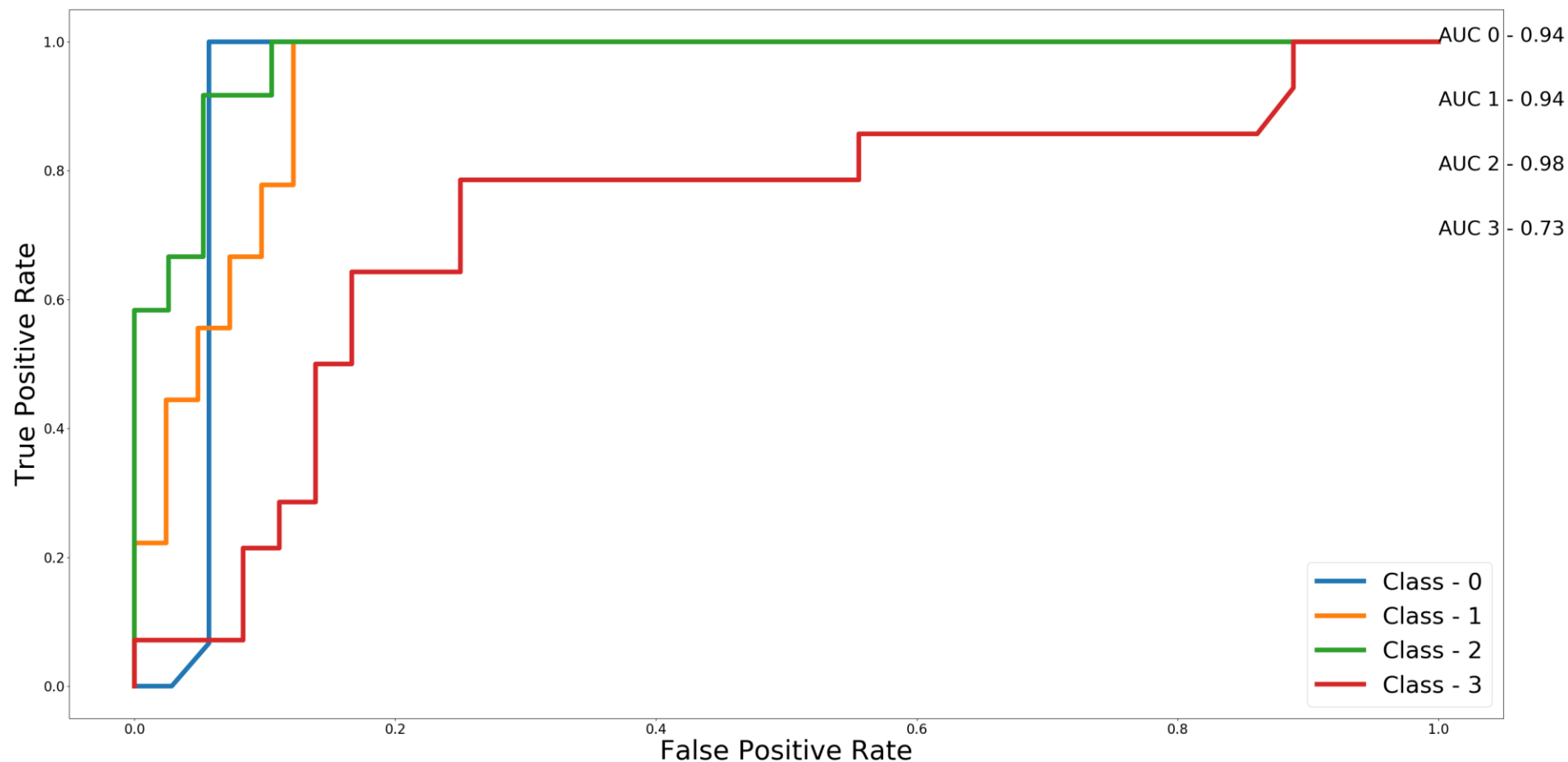


דגשים לסיווג



דגשים לסיווג - מימוש roc auc ב-python

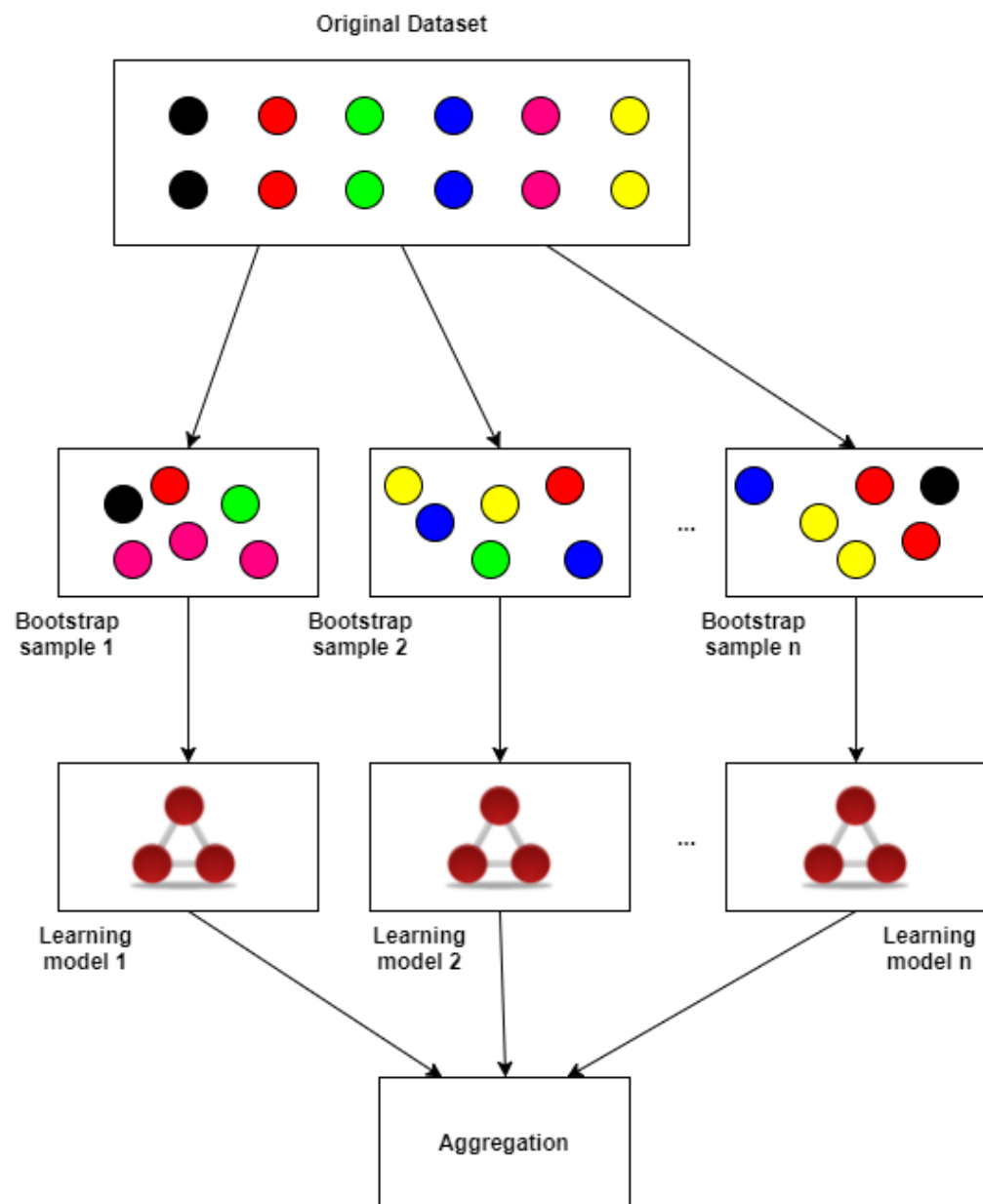
ROC Curve



- הגרף מחושב עבור תנאי סף שונים
- threshold והוא אינו ליניארי

Ensembles - חוכמת ההמונים

- Ensembles – שילוב של מספר מודלים מוצלחים, לקבלת תוצאה אופטימאלית
- Bagging (Bootstrap Aggregation) - אימון מודל זהה על דוגמאות שונות מתוך ה - training set
- Boosting - אימון מודל זהה על דוגמאות שונות מתוך ה - training set בשרשור, ומתן דגש על לימוד דוגמאות שסווגו לא נכון
- Blending (Stacking) – הכנסת תוצרי אימון של מודלים שונים כקלט ולימודם ע"י אלגוריתם חדש לקבלת תוצאה משוקללת



Bootstrap Aggregating

- הדוגמאות מחולקות לקבוצות קטנות, על כל קבוצה מבוצעת למידה
- לאחר אימון מתקבלים מודלים עליהם מבוצעת פעולת Aggregation
- במקרה של רגרסיה החיזוי יהיה ממוצע של כלל המודלים (soft)
- במקרה של קלסיפיקציה החיזוי יתבצע ע"י פעולת Voting (hard) או ממוצע על ההסתברויות (soft)
- RF הוא דוגמא למודל שעושה Bagging - כאשר התת מודל שלו זה עצי ההחלטה

מימוש Bootstrap Aggregating -

```
#Ensembles
```

```
x, y = load_iris(return_X_y=True)
x_new , y_new = put_anomaly(x,y,i_max=50)
x = x_new
y = y_new
```

```
X_train, X_test, y_train, y_test = train_test_split(
    x, y, test_size=0.33, random_state=42)
```

```
clf = RandomForestClassifier(n_estimators=100, max_depth=2,random_state=0)
```

```
bagging = BaggingClassifier(clf,max_samples=0.5, max_features=0.5)
```

```
clf.fit(X_train,y_train).score(X_test,y_test)
bagging.fit(X_train,y_train).score(X_test,y_test)
```

Bootstrap Aggregating

```
In [1291]: bagging.fit(X_train,y_train).score(X_test,y_test)
```

```
Out[1291]: 0.74
```

```
In [1292]: clf.fit(X_train,y_train).score(X_test,y_test)
```

```
Out[1292]: 0.62
```

Bootstrap Aggregating

`sklearn.ensemble`.**BaggingClassifier**

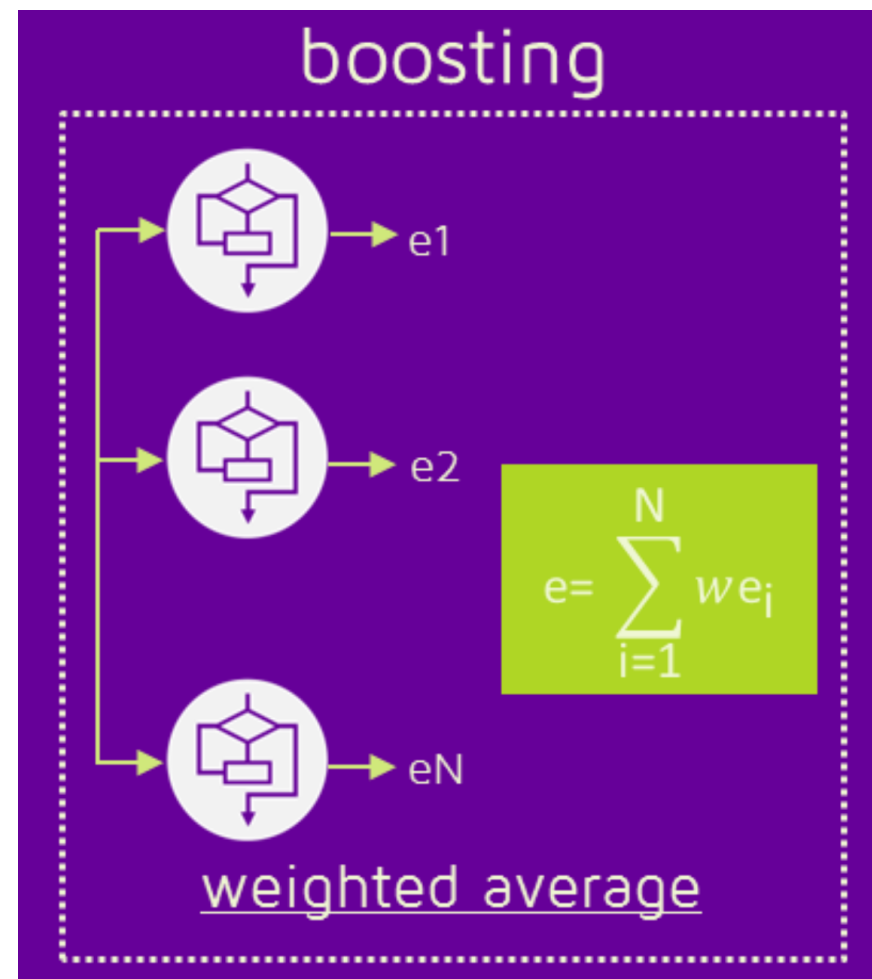
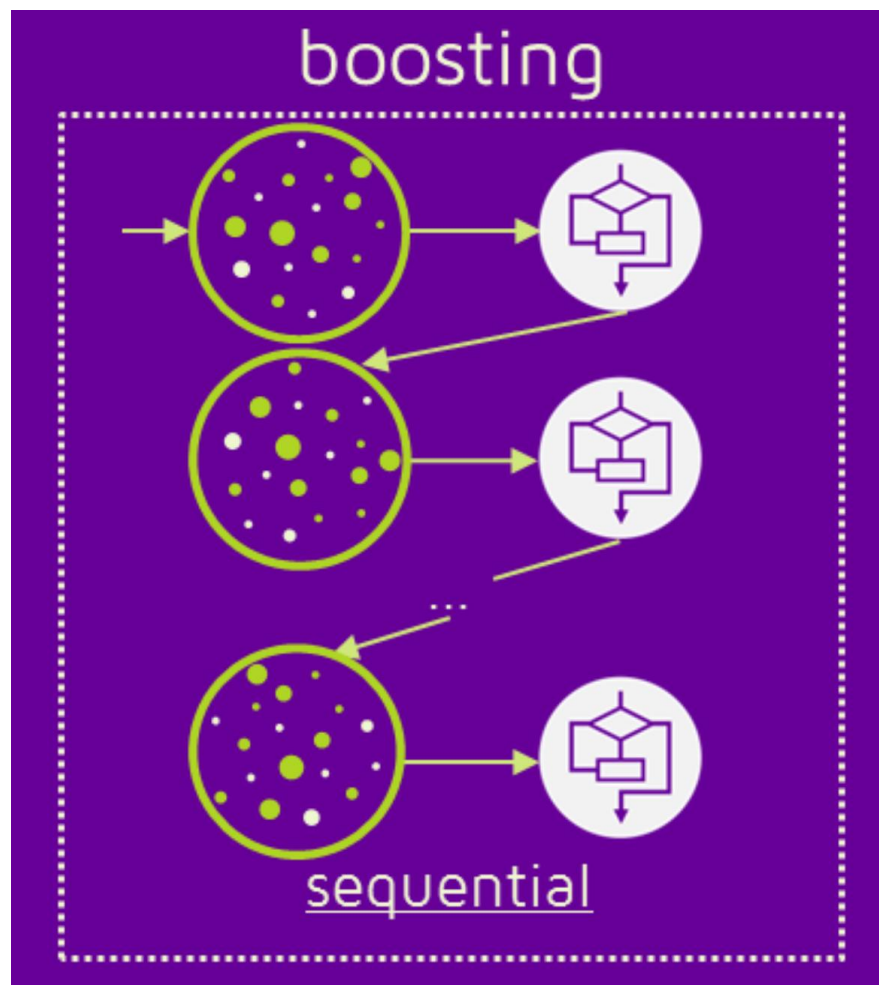
```
class sklearn.ensemble. BaggingClassifier (base_estimator=None, n_estimators=10, max_samples=1.0,  
max_features=1.0, bootstrap=True, bootstrap_features=False, oob_score=False, warm_start=False, n_jobs=None,  
random_state=None, verbose=0)
```

[\[source\]](#)

Boosting

- מבוצע אימון כל פעם על חלק מהדוגמאות מתוך ה-Data Set
- לאחר האימון ניתן משקל לכל דוגמא כך שדוגמאות עליהם ניתן חיזוי שגוי יילמדו בפעמים הבאות
- באופן זה הדוגמאות השגויות מאומנות יותר וכך התוצאות משתפרות
- בסיום התהליך ניתן משקל גם למסווגים השונים
- קיימים סוגים שונים של אלגוריתמים Boosting ביניהם: AdaBoost, LPBoost, XGBoost, GradientBoost, BrownBoost
- מומלץ להשתמש ב-XGBoost

Boosting

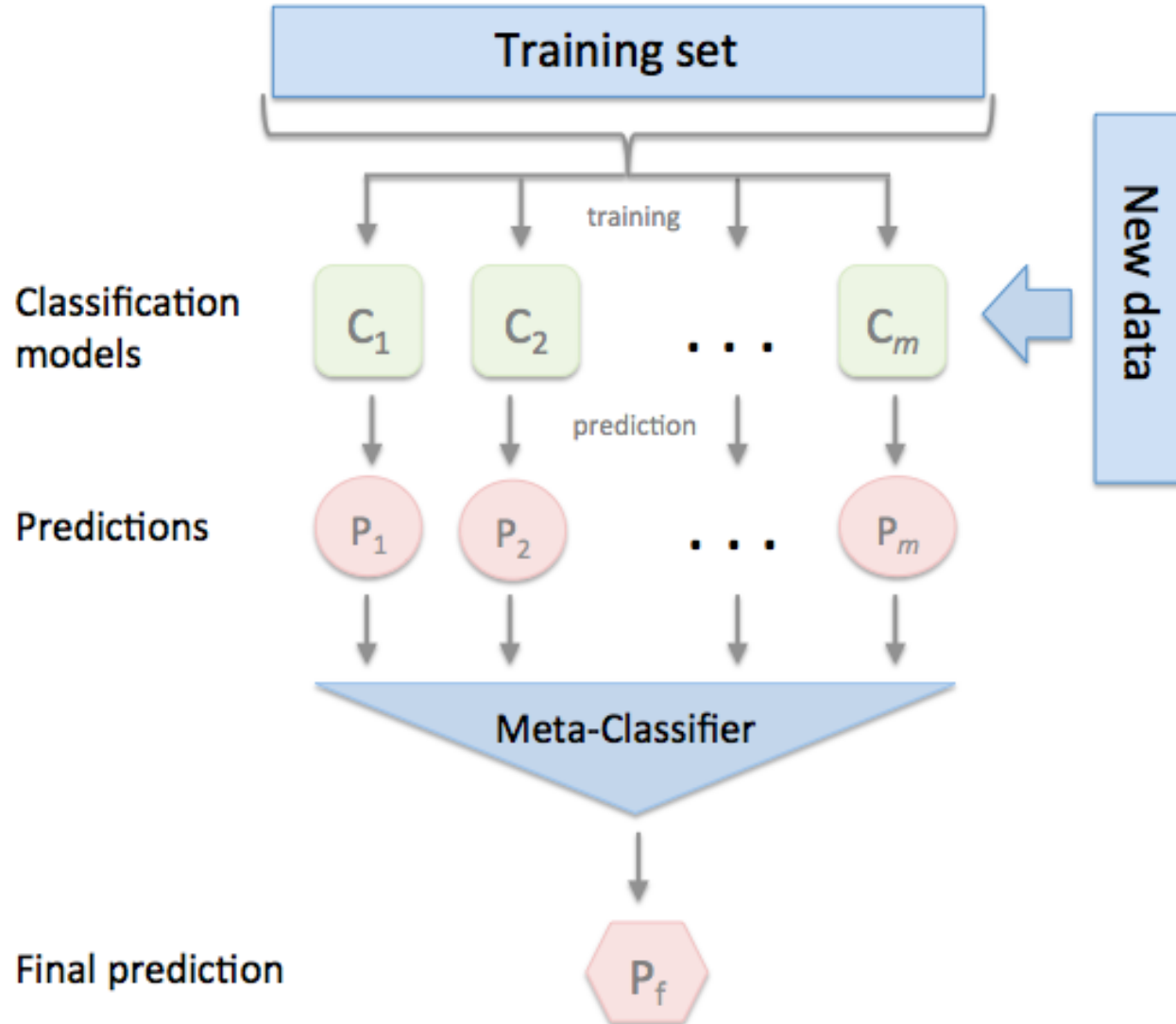


Boosting - מימוש בפייתון

#Boosting - AdaBoost

```
clf_boost = AdaBoostClassifier(clf,n_estimators=100)  
clf_boost.fit(X_train,y_train).score(X_test,y_test)
```

Blending - Stacking



- בשיטה זו משתמשים במספר מסווגים שונים (meta-classifier) על כל ה-train data
- לבסוף ישנו אלגוריתם הלומד את התוצאות של מסווגים אלו, ונותן חיזוי המורכב מכלל התוצאות של meta-classifier
- האימון יכול להיות על הסיווג עצמו או על ההסתברויות שמתקבלות מהסיווג

Blending

#Blending

```
model1 = RandomForestClassifier()
```

```
model2 = KNeighborsClassifier()
```

```
model3= LogisticRegression()
```

```
clf_voting = VotingClassifier(estimators=[  
    ('RandomForestClassifier', model1), ('KNeighborsClassifier', model2),  
    ('LogisticRegression', model3)], voting='hard')# 'soft'
```

```
clf_voting.fit(X_train,y_train).score(X_test,y_test)
```

Ensemble

decrease model variance –Bagging •

decrease model bias –Boosting •

increase predictive force of the classifier –Blending •