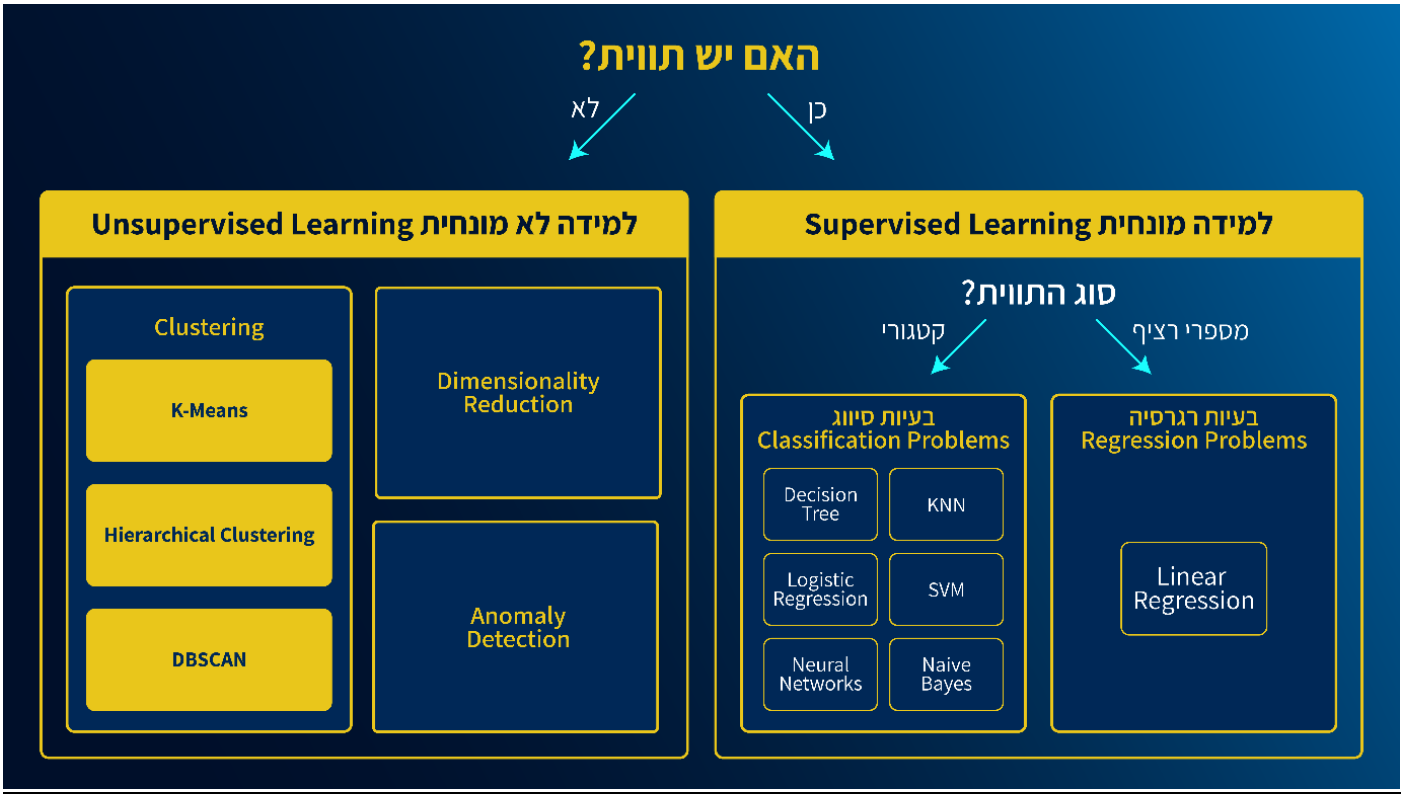
סיכום למידת מכונה (Machine Learning) (מבוסס על קורס מדע הנתונים, Campus IL)

סוגים של למידת מכונה

- **למידה מונחית (Supervised Learning):** מתן הנחייה מדויקת למחשב על התוכן הנלמד עם מתן דוגמאות.
- **למידה לא מונחית (Unsupervised Learning):** מתן נתונים למחשב בלי הנחייה מדויקת. המחשב צריך למצוא בעצמו דפוסים ונתונים.
- **למידה באמצעות חיזוקים (Reinforcement Learning):** המחשב לומד תוך כדי קבלת משוב מהסביבה. משתפר ומשתפר.

שימושים בלמידת מכונה

- חיזוי תוך בניית מודלים.
- משימות מורכבות/צורכות זמן רב.
- התאמה לסביבה משתנה.



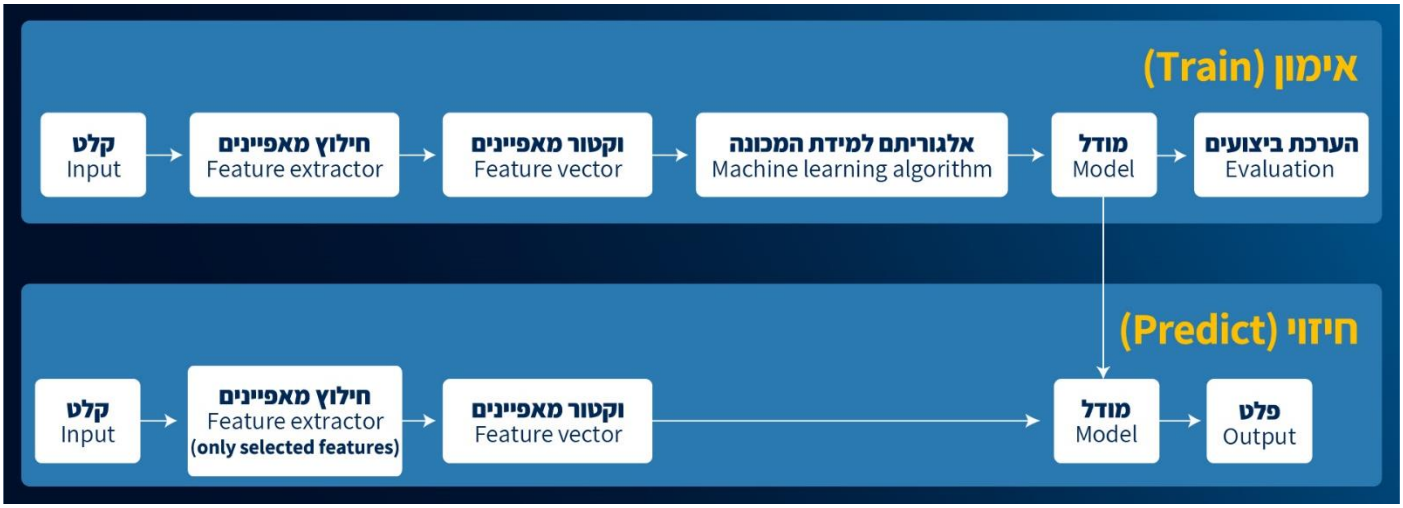
שלבי למידת מכונה

- **אימון (Train):** בניית מודל.

קלט (Input, קבלת המידע והנתונים) ← **חילוץ מאפיינים** (Feature Extractor, סידור המאפיינים) ← **וקטור מאפיינים** (Feature Vector, סידור המופעים לפי המאפיינים) ← **אלגוריתם** (ML Algorithm, בחירת והפעלת אלגוריתם) ← **מודל** (Model, קבלת מודל שמאפשר חיזוי עבור המופעים הבאים) ← **הערכת ביצועים** (Evaluation, בתחילה נחלק את הנתונים ל־train (70%-80% מהמידע) ול־test (20%-30%). את הערכת הביצועים נשווה ל־test).

- **חיזוי (Prediction):** הפעלת המודל על מופעים (Instances) חדשים.

אותם שלבים מ**קלט** עד **וקטור מאפיינים**. אחר־כך, במקום להשתמש ב**אלגוריתם**, נשתמש במודל שיצרנו ונפעיל אותו על ה־DataFrame של הנתונים החדשים.

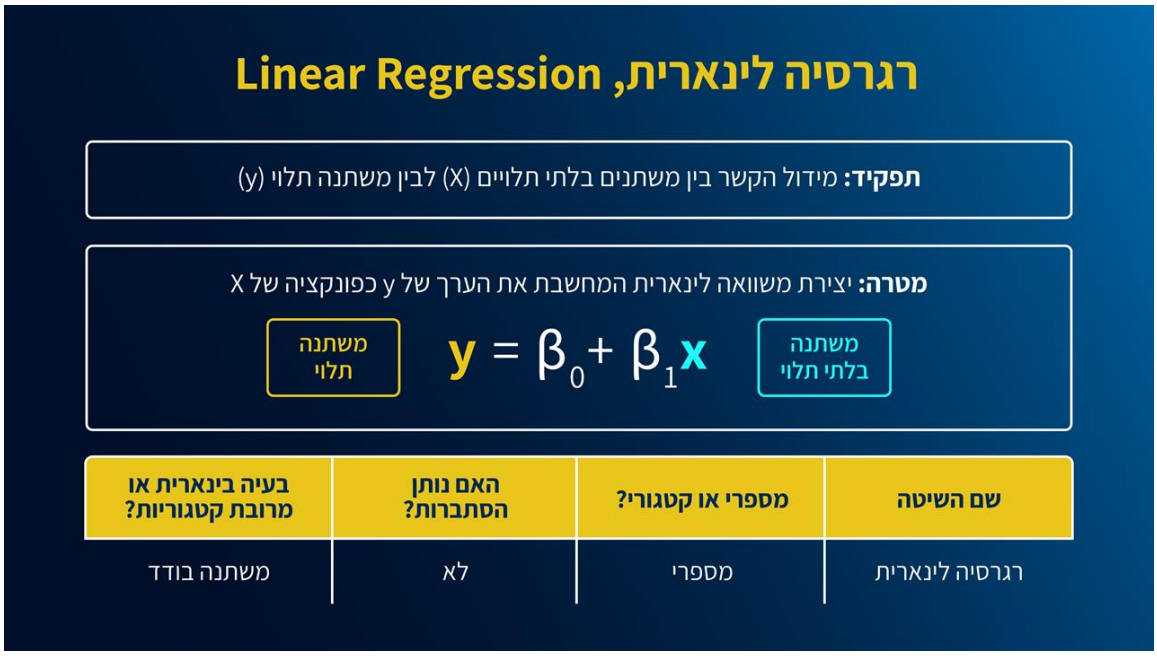


פונקציות מרכזיות של sklearn

חלוקת הנתונים	xtr, xts, ytr, yts = train_test_split(x, y, test_size=...)
אימון (בניית המודל)	x = algo() model = x.fit()
חיזוי (הפעלת המודל)	y_pred = model.predict(x_test)
איחוד אימון וחיזוי	fit_predict()
נירמול הנתונים (איחוד הנתונים לאותה הסקאלה)	scaler = MinMaxScaler() scaler = StandardScaler() scaled = scaler.fit_transform(x)

[Sklearn Functions Notebook](#) | [Intro to ML](#) | [Intro to sklearn](#) | [sklearn doc](#) | [sklearn History](#)

רגרסיה לינארית (Linear Regression)



- נכניס את כל המופעים לגרף (הערה: i=מספר מופע, לפי שורה ב־df).
- איך מוצאים משוואה לינארית נכונה? מביאים למינימום את ההפרש בין הערך האמיתי של המשתנה המוסבר לערך המחושב לפי המשוואה (חיזוי). כלומר המטרה היא למצוא קו אופטימלי בין המופעים, הנקודות.

מימוש עם פייתון

```
from sklearn.linear_model import LinearRegression
regressor = LinearRegression()
X_train = df[[f1, f2, f3, ...]]
y_train = df[target_feature]
model = regressor.fit(X_train, y_train)
y_pred = model.predict(X_test)
```

LinearRegrssion doc | More about Linear Regression

הערכת תוצאות

סכום ריבועי המרחקים בין הערכים החזויים לאמיתיים (SSE – Sum of Squared Errors)

ה־SSE עוזר לדעת מה המודל בטוב ביותר בהשוואה למודלים אחרים שנבנו. אם יש לי שני מודלים, אני אפעיל SSE על שניהם ואראה למי יש את התוצאה הנמוכה ביותר. הוא המודל הטוב.

$$\mathrm{SSE} = \sum_{i=1}^{n} (y_i - (\beta_0 + \beta_1 x_i))^2$$

y_i = **ערך אמיתי/נצפה**

$(\beta_0 + \beta_1 x_i)$ = **חישוב ערך חזוי**

את הסוגריים המרכזיות נעלה בריבוע, ואת זאת נעשה עבור כל המופעים ב־df מ־1 עד n, ונסכום כל פעם.

```
from sklearn.metrics import mean_squared_error
mean_squared_error(y_test, y_pred) * len(y_test)
```

מה אם נרצה לדעת אם מודל מסוים הוא הטוב ביותר לא רק בהשוואה לאחרים, שהוא האופטימלי ביותר?

מדד R^2 (R-Squared)

$R^2 = 1 - \frac{SSE}{TSS}$ **(SSE = סכום ריבועי המרחקים, TSS = השונות של כל המופעים)**

$$\mathrm{TSS} = \sum_{i=1}^{n} (y_i - \bar{y})^2$$

$\bar{y}$ = **ממוצע y**

$$0 \le \frac{SSE}{TSS} \le 1$$

ככל שמדד R^2 קרוב יותר ל־1, המודל יותר טוב.

```
from sklearn.metrics import r2_score
r2_score(y_test, y_pred)
```

הערה: אפשרות נחמדה לשיפור המודל, היא Feature Engineering, להוסיף עוד מאפיין/ים ברשימה של X, שהוא כפולות של כמה מאפיינים, סכום וכו'...

הערה נוספת: התאמת יתר (Overfitting) הוא מצב בו בונים מודל יותר מדי ספציפי, והוא גורם לבעיה בחיזוי.

רגרסיה לוגיסטית (Logistic Regression)

רגרסיה לוגיסטית, Logistic Regression

תפקיד: חיזוי ערך קטגוריאלי, פתרון לבעיות סיווג (קלאסיפיקציה)

תכונות חשובות: שיטת לימוד חזקה עבור בעיות בינאריות, היכולת לתרגם את התשובה לאחוזים

שם השיטה	מספרי או קטגורי?	האם נותן הסתברות?	בעיה בינארית או מרובת קטגוריות?
רגרסיה לינארית	מספרי	לא	משתנה בודד
רגרסיה לוגיסטית	קטגורי	כן	בעיה בינארית

רגרסיה לוגיסטית היא פונקציה סיגמואית, פונקציה רציפה ומונטונית שעולה בין 0 ל־1. היא שואפת ל־0 עבור ערכי x נמוכים, ול־1 עבור גבוהים. היא מחזירה הסתברות לכך שהתווית היא 1.

מימוש עם פייתון

```
from sklearn.linear_model import LogisticRegression
regressor = LogisticRegression()
X_train = df[[f1, f2, f3, ...]]
y_train = df[target_feature]
model = regressor.fit(X_train, y_train)
y_pred = model.predict(X_test)
```

LogisticRegression doc | More About Logistic Regression

הערכת ביצועים

הצגת סיכום הסיווגים החזויים לעומת האמיתיים (Confusion Matrix)

טבלה ששורותיה מציגות את הערכים האמיתיים, ועמודותיה מציגים את הערכים החזויים. באמצעות הטבלה ניתן לראות מתי יש טעויות, אילו טעויות ואת הנתונים על רמת הדיוק.

במקרה של רגרסיה לוגיסטית, הטבלה תהיה 2x2, והיא תכלול 4 תאים:

		Predicted	
		P	N
Actual	P	TP	FN
	N	FP	TN

TP, TN, FP, FN (T = True / F = False / P = Positive / N = Negative)

True ו־False ייצגו האם המחשב צדק, ו־Positive ו־Negative מייצגים מה הערכים האמיתיים (1 או 0).

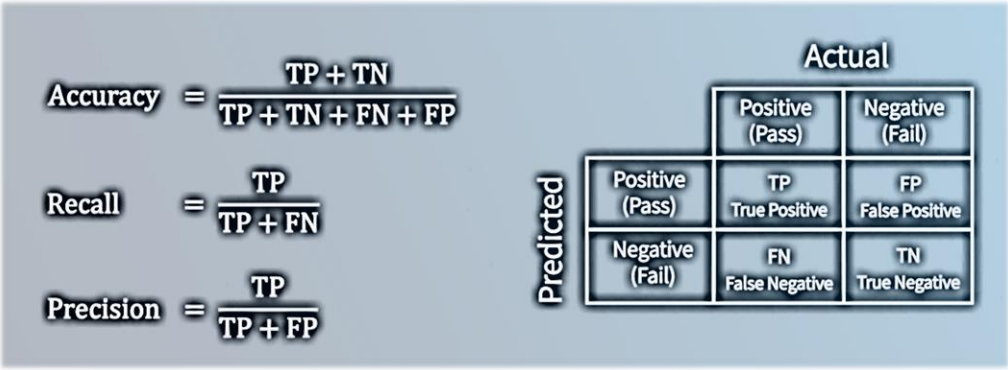
```
from sklearn.metrics import confusion_matrix
confusion_matrix(y_test, y_pred)
```

מדדים לבעיות קלסיפיקציה

מדד הדיוק (Precision) הוא כמה המודל צדק מתוך קטגוריה אחת – $\frac{TP}{TP+FP}$.

מדד הכיסוי (Recall) הוא כמה המודל צדק מתוך כל הקטגוריות חלקי כמות המקרים של הקטגוריה – $\frac{TP}{TP+FN}$.

מדד Accuracy הוא מדד דיוק כללי – $\frac{TP+TN}{TP+TN+FN+FP}$.



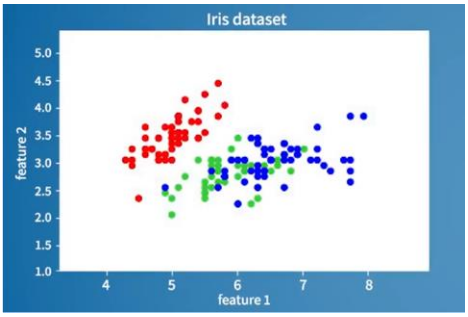
ישנו מדד נוסף, בשם F-measure, והוא $2 \cdot \frac{Precision \cdot Recall}{Precision + Recall}$, והוא משקלל את מדד הדיוק והכיסוי.

```
from sklearn import metrics
print(f"Accuracy: {metrics.accuracy_score(y_test, y_pred)}")  # Of training data: (y_train, y_pred_train)
print(f"Precision: {metrics.precision_score(y_test, y_pred)}")
print(f"Recall: {metrics.recall_score(y_test, y_pred)}")
print(f"F-Measure: {metrics.f1_score(y_test, y_pred)}")
```

[Evaluating Performance in sklearn]

שכנים קרובים (KNN - K Nearest Neighbors)

בשיטה זו, ניתן לסווג יותר מ־2 מאפיינים (בשונה מרגרסיה לוגיסטית) כמעט אין אימון על המודל. יש לזה יתרון, אך גם חיסרון (זיכרון רב).



השיטה עובדת בצורה די פשוטה. ממקמים על גרף scatter plot את כל המופעים, ומחלקים אותם לפי הקטגוריות שלהם. ניתן לראות שיש הפרדה בין כל קטגוריה לקטגוריה. כשנרצה לבדוק לאיזו קטגוריה מופע שייך, נמקם אותו על הגרף ונראה אילו שכנים יש סביבו (הקטגוריה של רוב השכנים "מנצחת").

מימוש עם פייתון

```
from sklearn.neighbors import KNeighborsClassifier
k = ?
classifier = KNeighborsClassifier(n_neighbors=k)
X_train = df[[f1, f2, f3, ...]]
y_train = df[target_feature]
classifier.fit(X_train, y_train)
y_pred = classifier.predict(X_test)
```

איך לבחור את כמות השכנים (K)?

אם נבחר K גדול, חלוקת הגרף לפי קטגוריות תהיה כוללנית; אם נבחר K קטן החלוקה תהיה ספציפית.

למה לא לבחור K=1 תמיד? Overfitting. למה לא לבחור K קיצוני? Under-fitting (ההפך מ־Overfitting). צריך למצוא את ה־K האופטימלי לפי השיקולים הבאים:

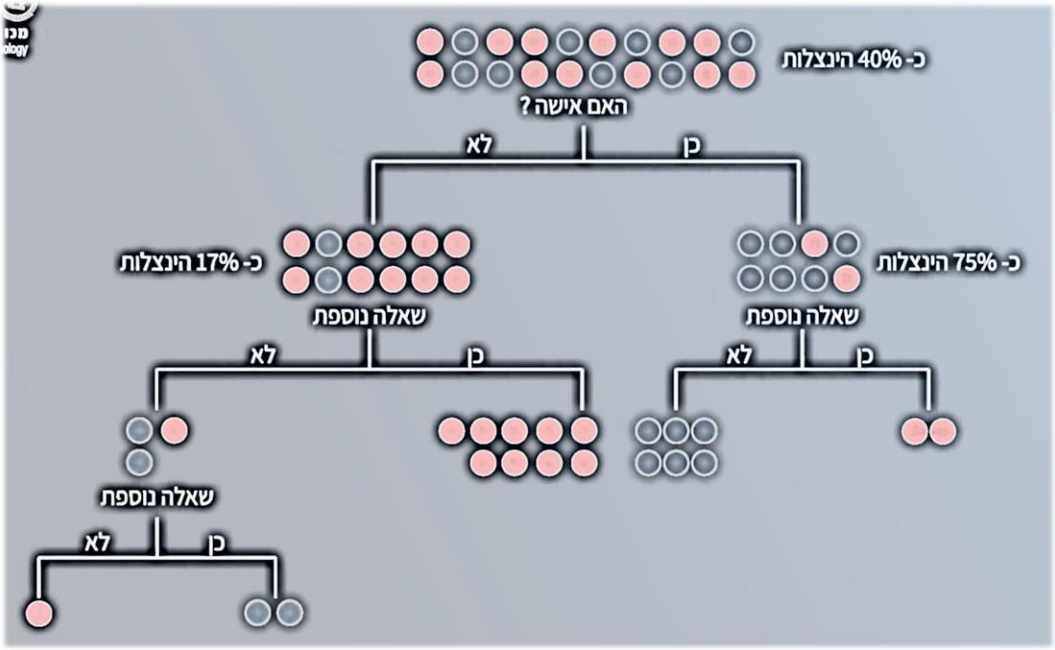
- K נמוך, אבל לא יותר מדי (כלל: K קטן משורש מספר המופעים ב־train).
- K אי־זוגי כדי למנוע מצב של תיקו.
- Cross-validation – בשיטה זו מחלקים את הנתונים למספר קבוצות, ורצים בלולאה. בלולאה מפרידים כל פעם קבוצה אחת שתשמש כ־test, ושאר קבוצות שישמשו כ־train. עליהם מריצים כל מיני Kים. בודקים איזה K נתן את הדיוק המירבי. זה ה־K האופטימלי.

```
from sklearn.model_selection import GridSearchCV
from sklearn.metrics import make_scorer

k_list = []
train_accuracies = []
test_accuracies = []
for k in range(1,25):
    clf = KNeighborsClassifier(n_neighbors=k)
    clf.fit(X_train, y_train)
    y_pred_train = clf.predict(X_train)
    y_pred_test = clf.predict(X_test)
    k_list,append(k)
    train_accuracies.append(metrics.accuracy_score(y_true=y_train, y_pred=y_pred_train))
    test_accuracies.append(metrics.accuracy_score(y_true=y_test, y_pred=y_pred))

accuracies_df = pd.DataFrame({"k": k_list, "train_accuracy": train_accuracies, "test_accuracy": test_accuracies})

parameters = {"n_neighbors": range(1,25,2) }
knn = KNeighborsClassifier()
classifier = GridSearchCV(knn, parameters, scoring=make_scorer(metrics.accuracy_score, greater_is_better=True))
classifier.fit(X_train, y_train)

print(f"Best parameter set is: {classifier.best_params_} and its score was {classifier.best_score_}")
print(classifier.cv_results_.items())  # if you want to see all iterations internal numbers uncomment the next line.
```

KNN Notebook | KNN doc | GridSearchCV doc | More About KNN

עצי החלטה (Decision Trees)

עצי החלטה הם אלגוריתם קלסיפיקציה שמאפשר סיווג בין מספר מאפיינים. האלגוריתם לוקח את המידע ומחלק את המידע לשאלות של כן ולא לפי המאפיינים. כך הוא מפצל כל פעם עד שמקבלים תוצאה ראויה.



יתרונות: נוח להסבר ולפרשנות, יעיל בצריכת זיכרון, אפשרות לשילוב מאפיינים קטגוריאליים, סיווג מהיר של מופע חדש וקבלה של ערך הסתברותי לכל סיווג ורמה בעץ.

חסרונות: פחות מדויק, בעל נטייה ל־Overfitting ורגיש לשינויים במידע.

מימוש עם פייתון

```
from sklearn.tree import DecisionTreeClassifier

decTree = DecisionTreeClassifier(max_depth=?, min_samples_split=?)
decTree = decTree.fit(X_train, y_train)
y_pred = decTree.predict(X_train)
```

מימוש ויזואליזציה של עצי החלטה עם פייתון

```
from IPython.display import Image, display
import pydotplus
from scipy import misc

def renderTree(my_tree, features):
        filename = "temp.dot"
        with open(filename, 'w') as f:
        f = tree.export_graphviz(my_tree,
                                 out_file=f,
                                 feature_names=features,
                                 class_names=["Perished", "Survived"],
                                 filled=True,
                                 rounded=True,
                                 special_characters=True)

        dot_data = ""
        with open(filename, 'r') as f:
                dot_data = f.read()

         graph = pydotplus.graph_from_dot_data(dot_data)
         image_name = "temp.png"
         graph.write_png(image_name)
```

```
        display(Image(filename=image_name))

def main():
        renderTree(decisionTree, ["Sex"])  # Example.
```

מימוש עץ אופטימלי עם פייתון

כמו ב־KNN, עם שינוי הפרטמר הראשון של הפונקצייה GridSearchCV ל־decTree.

איך למנוע Overfitting בעצי החלטה?

- **הגבלת עומק העץ:** להגדיר שלעץ לא יכול להיות יותר מכמה דרגות (כלומר שאלות, פיצולים) ובכך ליצור עץ יותר מכליל.
- לקבוע כמות ערכים מינימלית בקבוצה.
- **יער רנדומלי (Random Forest):** אלגוריתם המשלב הרבה עצים פשוטים ביחד ליצירת אופטימליות.

מימוש Random Forest עם פייתון

```
from sklearn.ensemble import RandomForestClassifier
forest = RandomForestClassifier(n_estimators=?)
```

מספר העצים = n_estimators.

ולהמשיך כרגיל (fit ו־predict)...

[Decision Trees doc](#) | [Decision Tree Explanation](#) | [Random Forest doc](#)

שיטה בייסיאנית (Naïve Bayes)

שיטה זו היא שיטת קלסיפיקציה מעולם ההסתברות המסתמכת על הסתברות מותנית (בהינתן מאורע A קרה, מה ההסתברות שמאורע B יקרה). היא בודקת איזה מבין הקטגוריות היא הכי סבירה מבין המאפיינים שלה.

כדי לחשב את ההסתברות המותנית יש את הנוסחה הבאה:

$$p(c|x) = \frac{p(x|c)p(c)}{p(x)}$$

כאשר c מייצגת קטגוריה מסוימת, ו־x מאפייני מופע מסוים (אנחנו מחפשים את p של c בהינתן x – p(c|x)).

יתרונות: מהירה, פשוטה, קלה להסבר, מתאימה לכל Data, צריכת הזיכרון די סבבה וכמעט אינה נוטה ל־overfitting.

חסרונות: מחשב את הקטגוריה הסבירה ביותר ולא את ההסתברות שלה, ובעייתי עם מאפיינים קורלטיביים.

עקרונות

- חיזוי: מציאת הקטגוריה הסבירה ביותר בהינתן סט מאפיינים.
- אימון: מציאת ההסתברות של ערכי המאפיינים עבור כל קטגוריה, ומציאת השכיחות של כל קטגוריה.

כלומר, האימון הוא חישוב הערכים של צד ימין של המשוואה; והחיזוי הוא תוצאה של צד שמאל על־ידי הפעלת ערכי הסתברות שמתקבלים לפי המופע שאותו אנחנו רוצים לסווג.

מימוש עם פייתון

הערה כללית: ב־KNN, עצי החלטה ו־Naïve Bayes, נהוג (בגלל הנטייה שלהם ל־overfitting) לעשות predict גם על ה־X_train וגם על ה־X_test ואז להעריך תוצאות לכל אחד מהם.

```
gnb = GaussianNB()
```

```
gnb.fit(X_train, y_train)
y_pred = gnb.predict(X_train)
```

[Naïve Bayes (Continuous Features) doc](#) | [Naïve Bayes (Discrete Features) doc](#) | [More About Naïve Bayes](#)

אלגוריתמים נוספים

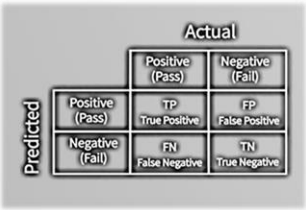
למידע נוסף לחצו כאן.

לקריאה על SVM לחצו כאן.

סיכום למידה מונחית (Supervised Learning)

	רגרסיה לינארית	רגרסיה לוגיסטית	KNN	עצי החלטה	Naïve Bayes	רשתות נוירונים	SVM
פשוט להסבר	כן	לא	כן	כן	כן	לא	לא
קל לביצוע	כן	כן	כן	כן	כן	לא	לא
צריכת זיכרון נמוכה	כן	כן	לא	כן (למעט random forest)	כן	כן	כן
ריבוי קטגוריות	לא	לא	כן	כן	כן	כן (תלוי)	לא
סכנת Overfitting	לא	כן	כן	כן	לא	כן	לא
עמיד לשינויים	כן	כן	כן	לא	כן	כן	כן
מהירות אימון	מהיר	מהיר	מינימלית	סבירה	מהיר	ארוכה	ארוכה

נוסחאות Machine Learning

שם		תפקיד	משמעות		הנוסחה
פונקציה לינארית		חישוב ערך חזוי	ה־X_i מתרבה ככל ריבוי המאפיינים.		$y = \beta_0 + \beta_1 x_i$
SSE (Sum of Squared Errors)		הערכת תוצאות יחסיות. סכום ריבועי המרחקים בין ערכים חזויים לאמיתיים.	ערך אמיתי/נצפה	y_i	$SSE = \sum_{i=1}^{n}(y_i - (\beta_0 + \beta_1 x_i))^2$
			חישוב ערך חזוי	$(\beta_0 + \beta_1 x_i)$	
TSS (Total Sum Squared)		שונות המופעים	ממוצע y	$\bar{y}$	$TSS = \sum_{i=1}^{n}(y_i - \bar{y})^2$
R-Squared (R^2)		הערכות תוצאות אבסולוטיות.			$R^2 = 1 - \frac{SSE}{TSS}$
פונקציה סיגמואית (של רגרסיה לוגיסטית)		מה ההסתברות שמופע מסוים יהיה 1. חישוב ערך קטגוריאלי (סיווג, קלסיפיקציה).	ידע כללי		$p(x) = \frac{e^{\beta_0 \beta_1 x}}{1 + e^{\beta_0 \beta_1 x}}$
מטריצת בלבול	**Accuracy**	מדד דיוק כללי – כמה המודל צדק מתוך סך הדברים.	TP = True-Positive TN = True-Negative FP = False-Positive FN = False-Negative 		$\frac{TP + TN}{TP + TN + FN + FP}$
	Precision – דיוק	כמה המודל צדק מתוך קטגוריה אחת.			$\frac{TP}{TP + FP}$
	Recall – כיסוי	כמה המודל צדק מתוך כל הקטגוריות חלקי כמות המקרים של הקטגוריה.			$\frac{TP}{TP + FN}$
	F-measure	שקלול ה־Precision וה־Recall.			$2 \cdot \frac{Precision \cdot Recall}{Precision + Recall}$
הסתברות מותנית (Naïve Bayes)		בהינתן מאורע A קרה, מה ההסתברות שמאורע B יקרה.	ההסתברות של ...	p()	$p(c\|x) = \frac{p(x\|c)p(c)}{p(x)}$
			קטגוריה מסוימת	c	
			מאפייני מופע מסוים	x	
			ההסתברות של c בהינתן x	p(c|x)	