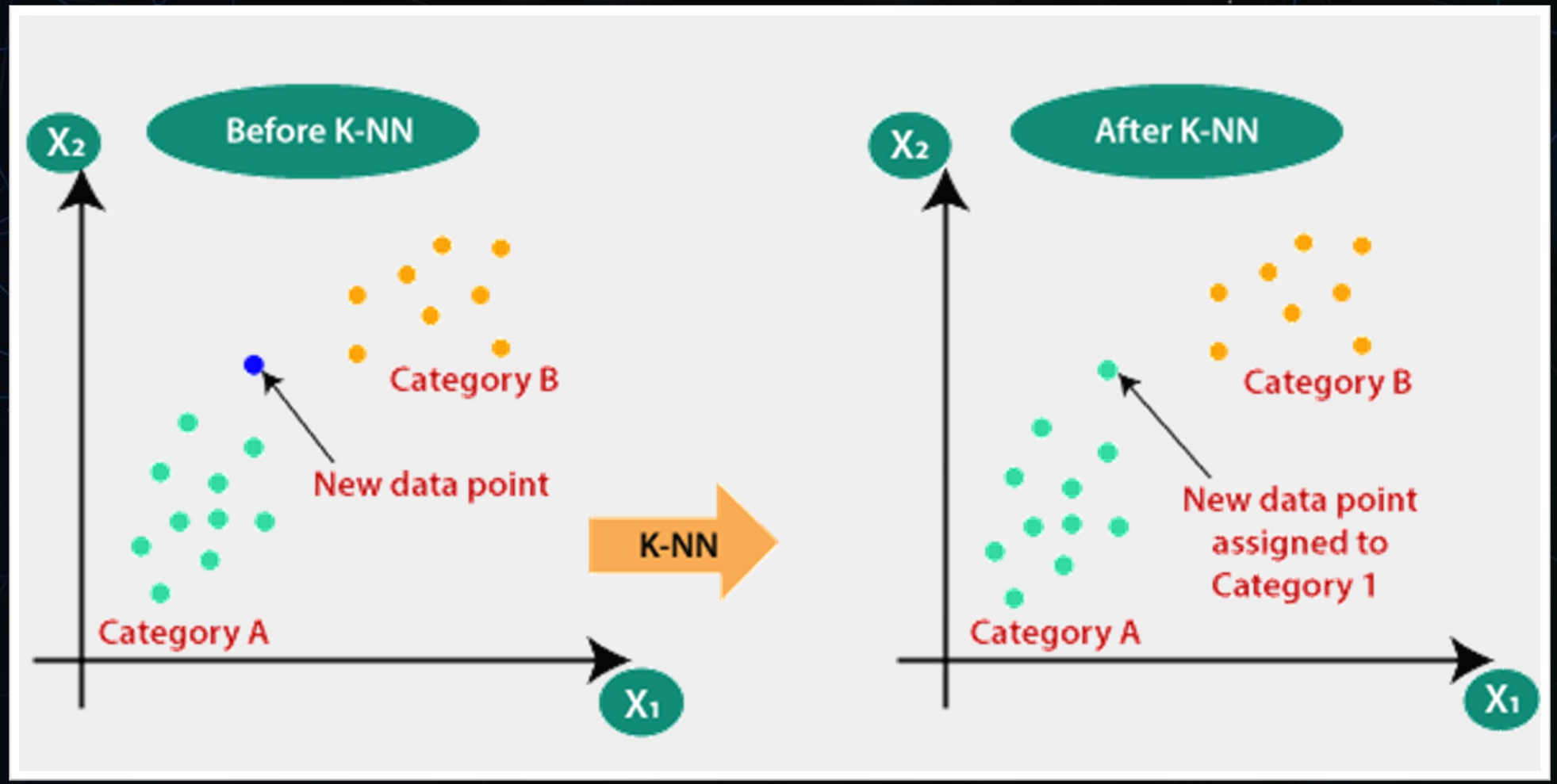
Vežbe br. 9

MAŠINSKO UČENJE

„K Najbližih Suseda“ (K Nearest Neighbors)

- Algoritam nadgledanog učenja, najčešće se koristi za klasifikaciju
- Zasniva se na pretpostavci da će podaci koji pripadaju istoj klasi biti slični i smešteni jedan blizu drugog u nekom prostoru
- Novi podatak koji treba da klasifikujemo biće smešten u onu kategoriju u kojoj se nalaze i njegove „komšije“

„K Najbližih Suseda“ (K Nearest Neighbors)



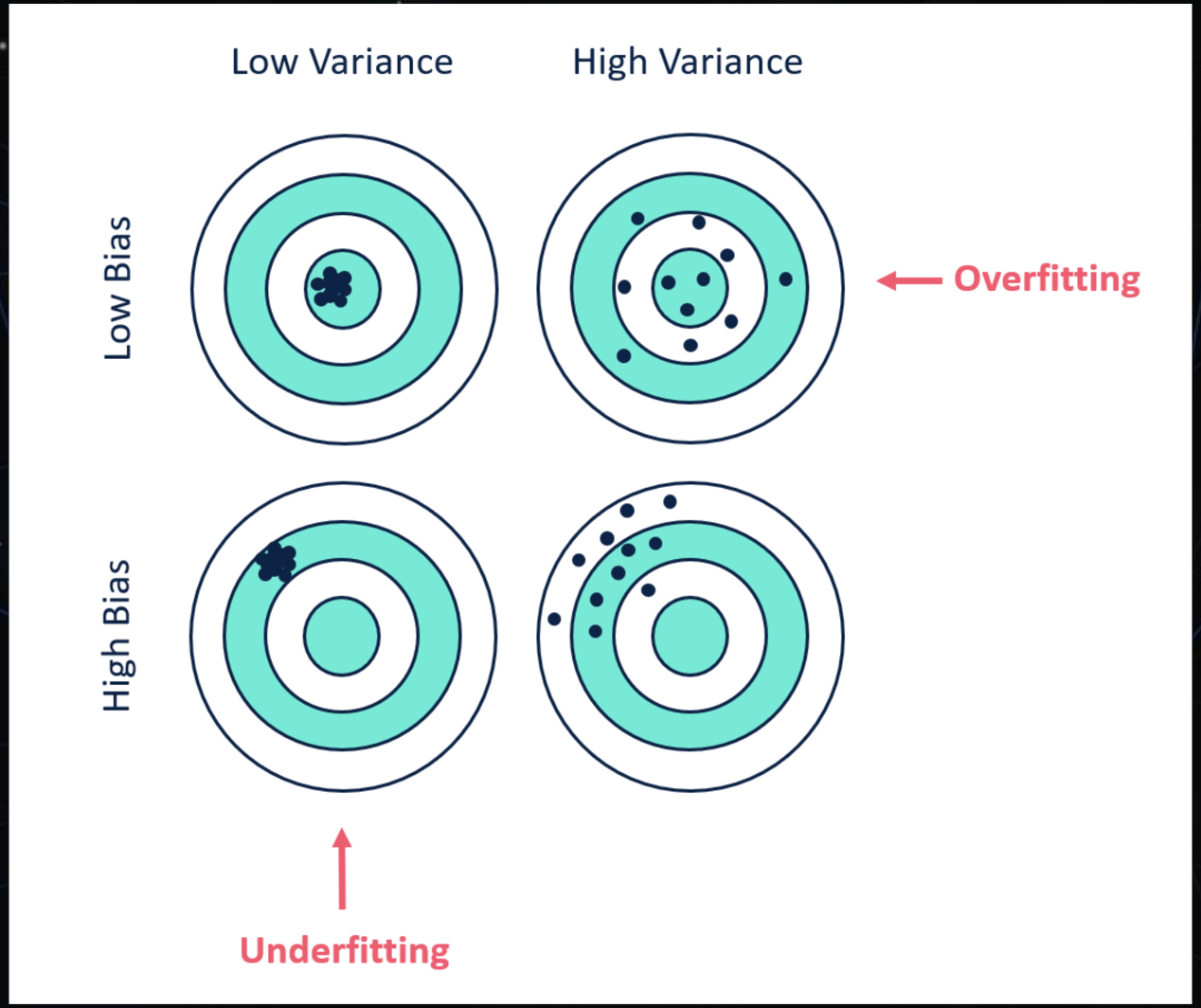
„K Najbližih Suseda“ (K Nearest Neighbors)

- Kako funkcioniše algoritam:
  - Definisati broj K
  - Izračunati Euklidsko rastojanje od susednih tačaka
  - Izabrati K najbližih suseda
  - Videti kojoj klasi pripada tih K suseda i smestiti novi podatak u većinsku klasu

„K Najbližih Suseda“ (K Nearest Neighbors)

- Kako odabrati K?
  - Malo K: mala greška, ali velika varijansa
  - Veliko K: velika greška, mala varijansa
- Težimo modelu koji ima malu grešku i malu varijansu, ali je to izuzetno teško postići („Bias – Variance tradeoff“)

„K Najbližih Suseda“ (K Nearest Neighbors)

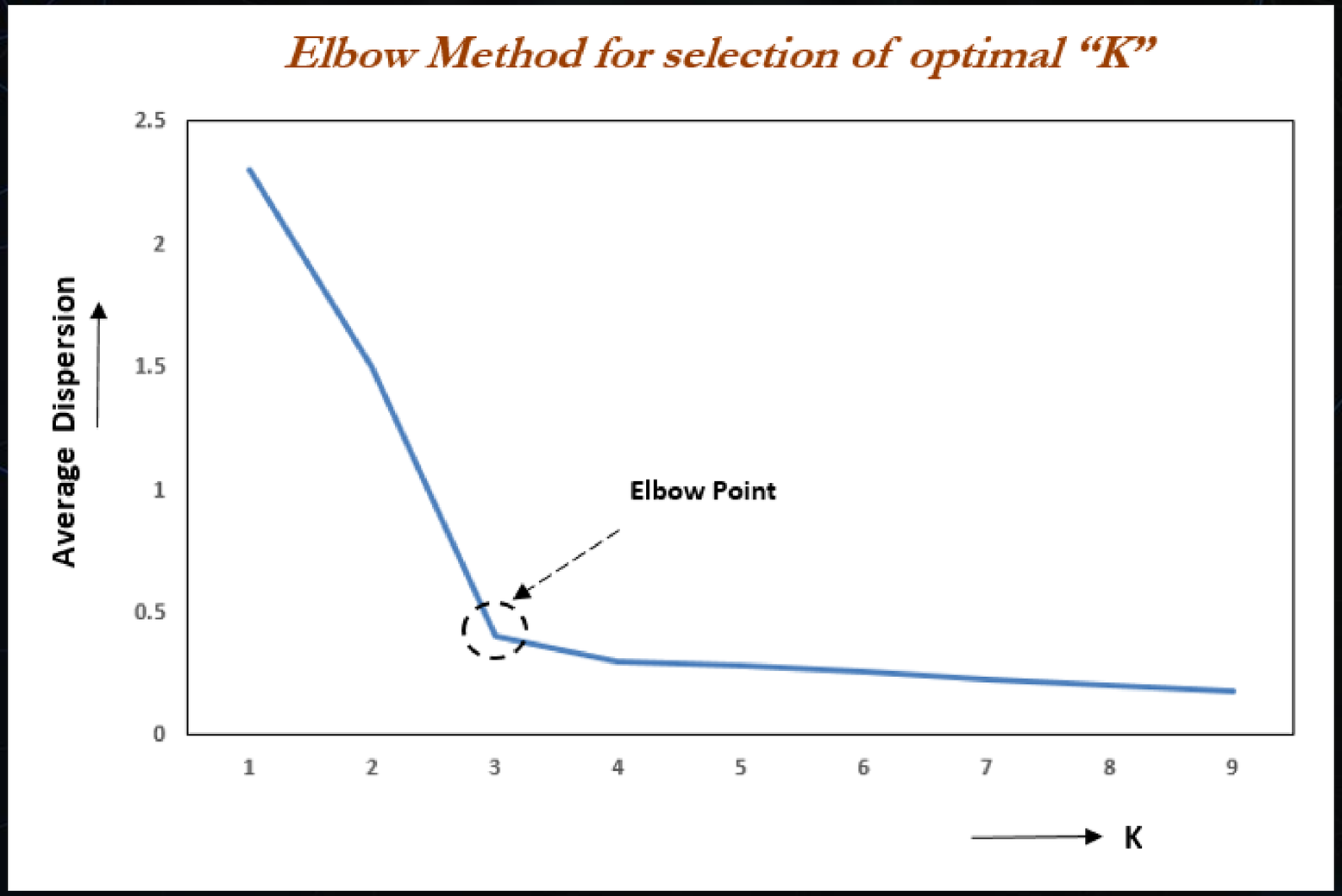


„K Najbližih Suseda“ (K Nearest Neighbors)

- Kako odabrati K?
  - Kod binarne klasifikacije K mora biti neparni broj
  - Nekada se K postavlja da bude kvadratni koren broja uzoraka u trening skupu
  - „Metoda lakta“ - najčešća tehnika za odabir K

- Obično K na kraju dobije vrednost između 3 i 10

„K Najbližih Suseda“ (K Nearest Neighbors)



„K Najbližih Suseda“ (K Nearest Neighbors)

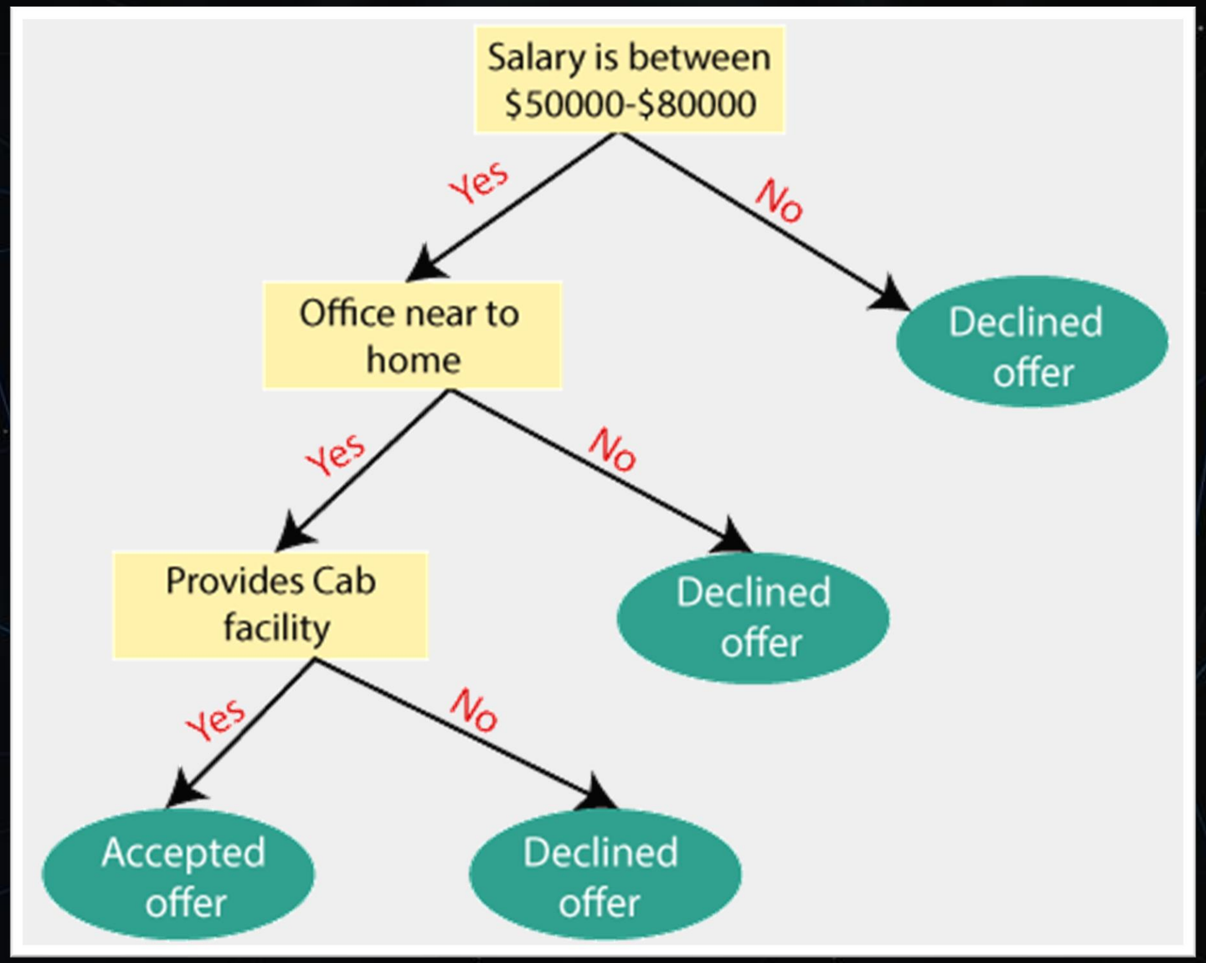
- Prednosti:
  - Lak za razumevanje
  - Lako se prilagođava podacima
  - Lak za kreiranje (malo hiperparametara)

- Mane:
  - Teško odrediti K
  - „Lenji algoritam“
  - „Prokletstvo dimenzionalnosti“

„Stablo odlučivanja“ (Decision Tree)

- Algoritam nadgledanog učenja, koristi se za klasifikaciju
- Na neki način imitira proces koji se dešava u ljudskom mozgu prilikom donošenja neke odluke
- Funkcioniše po principu pitanja i odgovora
- U zavisnosti od osobina podatka koji treba klasifikovati donosi se odluka o izlazu klasifikacije

„Stablo odlučivanja“ (Decision Tree)



„Stablo odlučivanja“ (Decision Tree)

- Prednosti:
  - Bez problema radi i sa nelinearnim podacima
  - Ne zahteva normalizaciju, standardizaciju...
  - Može da odradi „Feature selection“

- Mane:
  - Uglavnom kompleksniji od većine drugih algoritama
  - Uglavnom sporiji od većine drugih algoritama
  - Podložan velikim promenama za male promene podataka

Zadatak za vežbu

- Kreirati modele, zasnovane na KNN i stablu odlučivanja, koji će klasifikovati cveće u odgovarajuće klase, u zavisnosti od osobina cveta. Koristiti ugrađeni skup podataka „Iris“. Prikazati uspešnost kreiranih klasifikatora. Kod klasifikatora zasnovanog na stablu odlučivanja, prikazati proces klasifikacije podataka.

Rešenje

```
from sklearn.datasets import load_iris
from sklearn.metrics import accuracy_score
from sklearn.model_selection import train_test_split
from sklearn.neighbors import KNeighborsClassifier
from sklearn.tree import DecisionTreeClassifier
import matplotlib.pyplot as plt
from sklearn import tree

iris = load_iris()
X = iris.data[:, 2:]
y = iris.target
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, stratify=y)

knn = KNeighborsClassifier()
knn.fit(X_train, y_train)
y_predict = knn.predict(X_test)
print("Tačnost KNN (u %)", "%0.3f" % (accuracy_score(y_test, y_predict) * 100))

tree_clf = DecisionTreeClassifier()
tree_clf.fit(X_train, y_train)
y_predict = tree_clf.predict(X_test)
print("Tačnost stabla odluke (u %)", "%0.3f" % (accuracy_score(y_test, y_predict) * 100))

fig = plt.figure(figsize=(6, 6))
_ = tree.plot_tree(tree_clf,
                   feature_names=iris.feature_names,
                   class_names=iris.target_names,
                   filled=True)
plt.show()
```

Podešavanje hiperparametara modela

- Svaki model mašinskog učenja sadrži parametre i hiperparametre
  - Parametri modela: podešavaju se interno, tokom obučavanja modela
  - Hiperparametri modela: podešavaju se ručno

- Cilj: pronaći kombinaciju hiperparametara koja će učiniti model najboljim mogućim

Podešavanje hiperparametara modela

sklearn.neighbors.KNeighborsClassifier

class sklearn.neighbors.**KNeighborsClassifier**(*n_neighbors=5, *, weights='uniform', algorithm='auto', leaf_size=30, p=2, metric='minkowski', metric_params=None, n_jobs=None*) [source]

Classifier implementing the k-nearest neighbors vote.

Read more in the User Guide.

Parameters:

n_neighbors : *int, default=5*
Number of neighbors to use by default for `kneighbors` queries.

weights : *{'uniform', 'distance'}, callable or None, default='uniform'*
Weight function used in prediction. Possible values:

- 'uniform' : uniform weights. All points in each neighborhood are weighted equally.
- 'distance' : weight points by the inverse of their distance. in this case, closer neighbors of a query point will have a greater influence than neighbors which are further away.
- [callable] : a user-defined function which accepts an array of distances, and returns an array of the same shape containing the weights.

algorithm : *{'auto', 'ball_tree', 'kd_tree', 'brute'}, default='auto'*
Algorithm used to compute the nearest neighbors:

- 'ball_tree' will use `BallTree`
- 'kd_tree' will use `KDTree`
- 'brute' will use a brute-force search.
- 'auto' will attempt to decide the most appropriate algorithm based on the values passed to `fit` method.

Note: fitting on sparse input will override the setting of this parameter, using brute force.

leaf_size : *int, default=30*
Leaf size passed to BallTree or KDTree. This can affect the speed of the construction and query, as well as the memory required to store the tree. The optimal value depends on the nature of the problem.

p : *int, default=2*
Power parameter for the Minkowski metric. When p = 1, this is equivalent to using manhattan_distance (l1), and euclidean_distance (l2) for p = 2. For arbitrary p, minkowski_distance (l_p) is used.

metric : *str or callable, default='minkowski'*
Metric to use for distance computation. Default is "minkowski", which results in the standard Euclidean distance when p = 2. See the documentation of scipy.spatial.distance and the metrics listed in `distance_metrics` for valid metric values.

If metric is "precomputed", X is assumed to be a distance matrix and must be square during fit. X may be a sparse graph, in which case only "nonzero" elements may be considered neighbors.

If metric is a callable function, it takes two arrays representing 1D vectors as inputs and must return one value indicating the distance between those vectors. This works for Scipy's metrics, but is less efficient than passing the metric name as a string.

metric_params : *dict, default=None*
Additional keyword arguments for the metric function.

n_jobs : *int, default=None*
The number of parallel jobs to run for neighbors search. `None` means 1 unless in a `joblib.parallel_backend` context. `-1` means using all processors. See Glossary for more details. Doesn't affect `fit` method.

Podešavanje hiperparametara modela

- **Grid Search**
  - Najpoznatija tehnika za podešavanje hiperparametara modela
  - Definišu se hiperparametri koje želimo da podesimo i njihove moguće vrednosti, a onda se pravi kombinacija „svaki sa svakim“ dok se ne dođe do najbolje kombinacije
  - Prednosti: jednostavnost, efikasnost
  - Mana: složenost, limitiranost definisanim vrednostima

Zadatak za vežbu

- Podesiti hiperparametre stabla odlučivanja kreiranog u prethodnom zadatku. Uporediti rezultate dobijene nakon podešavanja hiperparametara sa početnim rezultatima. Samostalno uraditi zadatak i za K najbližih suseda.

Rešenje

```
from sklearn.model_selection import GridSearchCV

tuned_parameters = [
   {'criterion': ["gini", "entropy"],
    'ccp_alpha': [0, 0.01, 0.02, 0.05, 0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.8, 1.0]},
]

cv_clf = GridSearchCV(
   tree_clf, tuned_parameters, cv=10
)
cv_clf.fit(X_train, y_train)
print("Najbolji parametri: ", cv_clf.best_params_)

tree_clf.criterion = cv_clf.best_params_['criterion']
tree_clf.ccp_alpha = cv_clf.best_params_['ccp_alpha']

finalModel = tree_clf.fit(X_train, y_train)
y_predict = finalModel.predict(X_test)
print("Tačnost stabla odluke nakon podešavanja hiperparametara (u %)", "%0.3f" % (accuracy_score(y_test, y_predict) * 100))
```