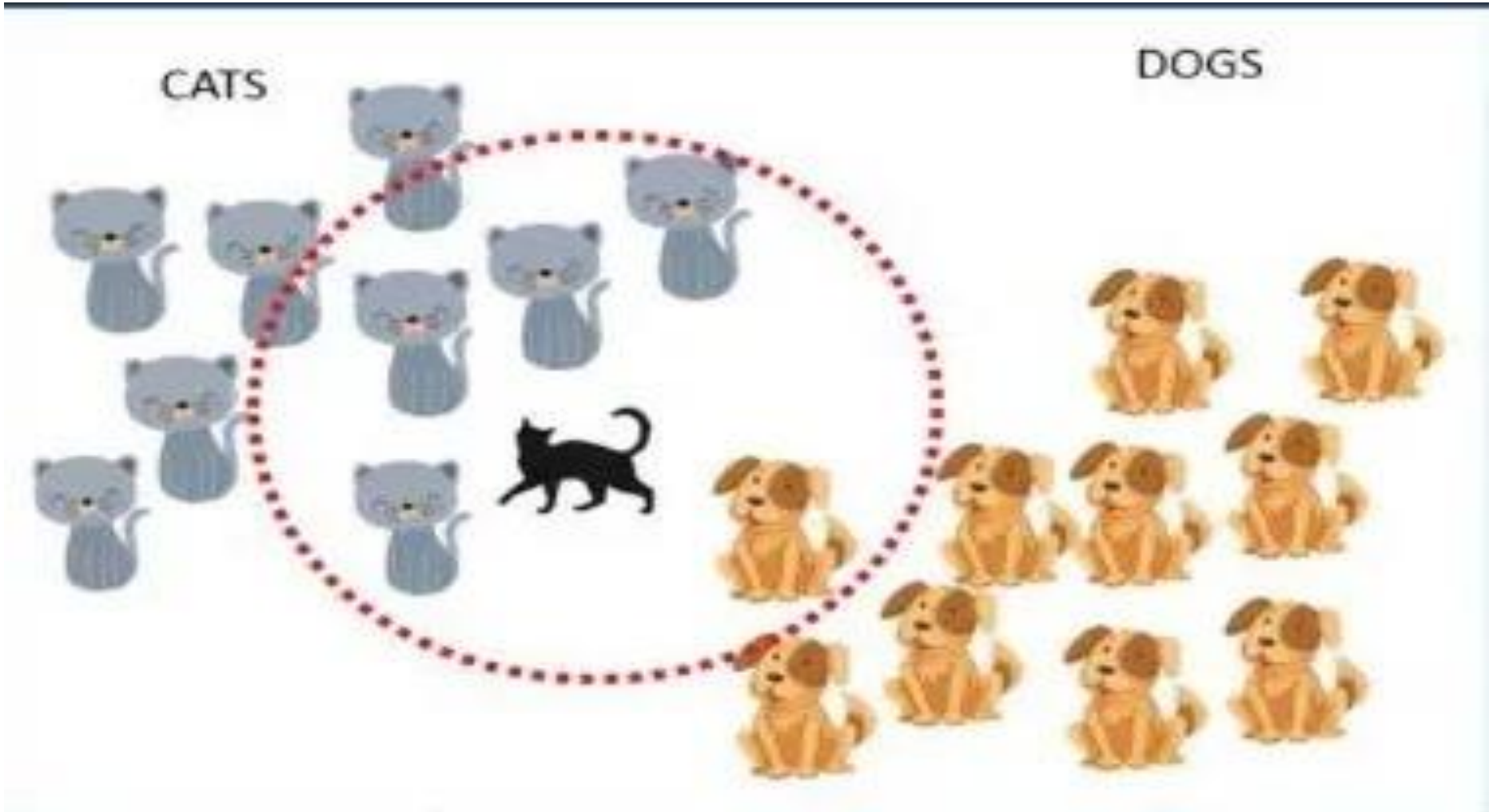
Makina Öğrenmesi İçin K-Nearest Neighbors (KNN) En Yakın Komşu Algoritması


CATS
DOGS

•Makina öğreniminde sıkça kullanılan KNN algoritmasının temel mantığı şu şekildedir bir gözlem biriminin kendine en yakın K adet gözlem birimi seçilir ve bu K adet gözlem biriminin bağımlı değişkeni neyse onun üzerinden ilgili gözlem için tahmin yapılır

- Hem regresyon hemde sınıflanırma problemleri için kullanılan bu yöntemi her ikisi içinde inceleyelim

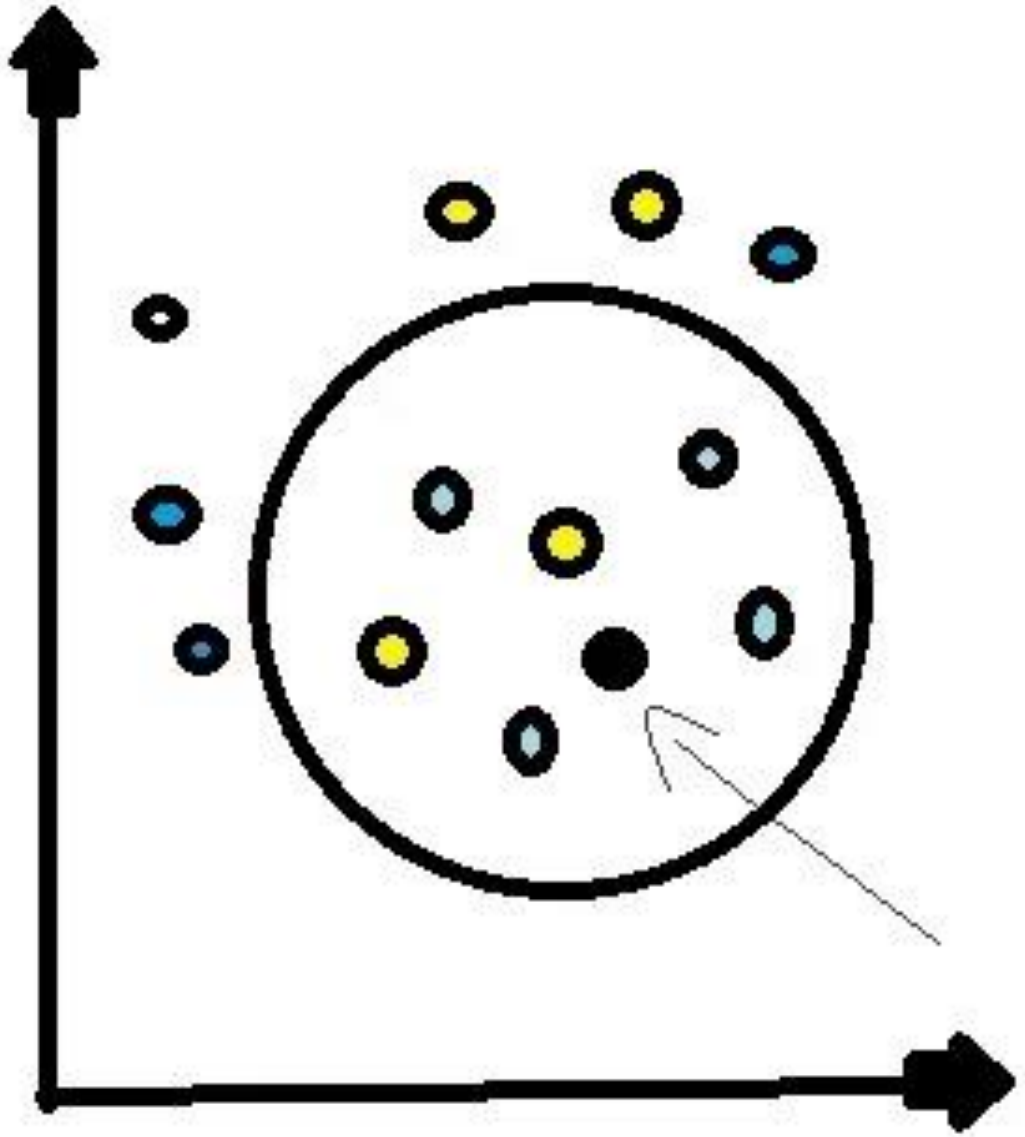
- Regresyon problemleri için KNN
- Eğer bir değeri Knn ile tahmin etmek istediğinize aşağıda resimde görüldüğü gibi belirlediğiniz k değeri üzerinden ona en yakın ‘k’ adet gözlemi seçer ve ortalamasını alırsınız

X1	X2	Y
150	55	100
266	78	178
349	45	200
345	55	210

•Makina öğreniminde sıkça kullanılan KNN algoritmasının temel mantığı şu şekildedir bir gözlem biriminin kendine en yakın K adet gözlem birimi seçilir ve bu K adet gözlem biriminin bağımlı değişkeni neyse onun üzerinden ilgili gözlem için tahmin yapılır

- Hem regresyon hemde sınıflanırma problemleri için kullanılan bu yöntemi her ikisi içinde inceleyelim

- Regresyon problemleri için KNN
- Eğer bir değeri Knn ile tahmin etmek istediğinize aşağıda resimde görüldüğü gibi belirlediğiniz k değeri üzerinden ona en yakın ‘k’ adet gözlemi seçer ve ortalamasını alırsınız



•Python Üzerinde inceleme

- Diyabet veri seti üzerinden bir knn çalışması yapacağız veri setinin içinde hastaların diyabet olup olmadıklarını gösteren 1 ve 0 sütunun olduğu ‘Outcome’ sütunu ve diyabete etki eden diğer unsurlar bulunmaktadır Örneğin: Kan basıncı,Deri kalınlığı,Glukoz değerleri,inulin değerleri,hamile olma durumu gibi değerler
- Gerekli kütüphaneleri import aşaması

•import pandas as pd

- from sklearn.metrics import classification_report, roc_auc_score
- from sklearn.model_selection import GridSearchCV, cross_validate
- from sklearn.neighbors import KNeighborsClassifier # KNN uygulamaları
- from sklearn.preprocessing import StandardScaler # Veriyi standartlaştırmak

Veriye Göz Atmak

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Age	Outcome
0	6	148	72	35	0	33.6	0.627	50	1
1	1	85	66	29	0	26.6	0.351	31	0
2	8	183	64	0	0	23.3	0.672	32	1
3	1	89	66	23	94	28.1	0.167	21	0
4	0	137	40	35	168	43.1	2.288	33	1

•Veriyi İşleme Ve Özellik Mühendisiği

- • Knn uzaklık temelli bir yöntemdir ve uzaklık temelli yöntemlerde daha doğru sonuçlar çıkması açısından verinin standarlaştırılması önemlidir

•y = df["Outcome"] # modelin bağımlı ve bağımsız değişkenlerini
- X = df.drop(["Outcome"], axis=1) # belirleme
- X_sc

•y = df["Outcome"] # modelin bağımlı ve bağımsız değişkenlerini
- X = df.drop(["Outcome"], axis=1) # belirleme
- X_sc

- Model kurma aşaması
- knn_model = KNeighborsClassifier().fit(X, y) # fit edilen model
- Deneme aşaması

- Modele rasgele 5 kişi seçtirdim ve onların diyabet olup olmadıklarını sorgulattım
- random_user = X.sample(5) # rasgele 5 kişi
- knn_model.predict(random_user)
- array([1, 1, 0, 0, 0], dtype=int64)
- İlk ikisin diyabet diğer üçünün diyabet olmadığı çıktısını verdi

•Modelin başarısını değerlendirme aşaması

- Karmaşıklık matrisi üzerinden diğer başarı ölçütlerimize ulaşacağız accuary,F1 score gibi metrikleri bulacağız
- y_pred = knn_model.predict(X) # karmaşıklık matrisi için y_predict
- y_prob = knn_model.predict_proba(X)[:, 1]
- print(classification_report(y, y_pred))

	precision	recall	f1-score	support
0	0.85	0.90	0.87	500
1	0.79	0.70	0.74	268
accuracy			0.83	768
macro avg	0.82	0.80	0.81	768
weighted avg	0.83	0.83	0.83	768

- Accuary değerimize baktığımızda bu değer bize 1 sınıfını ne kadar doğru tahmin ettiğimizi gösterir yani diyabet olan hastaları ne kadar doğru tanımladık bu değer 0.83 yani %83 başarı ile modelimiz çalışmış fakat dengesiz veri problemlerinde bazen accuary değeri bizi yanıltabilir bunun için f1- score değerine bakabilirz bu değerde %74 başarı ile çalışmış başarılı sayılabilecek bir sayı son olarakta Roc auc değerine bakmakta fayda var bu değer bize farklı eşik değerlerine model ne kadar dayanıklı bunu gösterir

ROC AUC

```
roc_auc_score(y, y_prob)
```

```
0.9017686567164179
```

•Oldukça iyi bir değer farklı sınıflandırma eşik değerleri için %90 başarı gösteren bir veri seti

- Bu aşamaya kadar verimizi kendisi üzerine test ettik yani bütün veri ile bir model kurduk yine aynı veri ile test ettik acaba bu veri hiç görmediği değerlerde ne kadar başarı gösterebilir bunu test etmek için holdout ve Cross validation yntemlerine bakacağız

- Cross Validation yöntemi

- 30 katlı Cross Validation uyguladık
- cv_results = cross_validate(knn_model, X, y, cv=5, scoring=["accuracy", "f1", "roc_auc"])

```
cv_results['test_accuracy'].mean()
```

```
0.7424615384615385
```

```
cv_results['test_f1'].mean()
```

```
0.6031047988246132
```

```
cv_results['test_roc_auc'].mean()
```

```
0.7798270697167755
```

•Skorların hepsi belli ölçülerde düştü yani model hiç görmediği bir veri üzerinde biraz yalpaladı ama bunu da düzeltmenin bir yolu var biz çapraz doğrulamaya başlarken 5 katlı ile başladık belki daha fazla veya daha az katlı çapraz doğrulamalarda daha yüksek skorlar verecek en iyi sonucu veren çapraz doğrulamayı bulmak için hiper parametre optimizasyonu yapacağız

- Hiper Parametre Optimizasyonu

```
knn_model = KNeighborsClassifier()
knn_model.get_params()
```

```
{'algorithm': 'auto',
 'leaf_size': 30,
 'metric': 'minkowski',
 'metric_params': None,
 'n_jobs': None,
 'n_neighbors': 5,
 'p': 2,
 'weights': 'uniform'}
```

- Bu değerler ilk başta atanmış değerler biz bu değerleri en optimum değerlere çekeceğiz örneğin komşuluk sayısını 5 olarak vermiş
- knn_params = {"n_neighbors": range(2, 50)}
- # 2 ile 50 arası komşuluk kurmak için yapılan işlemler

```
knn_gs_best = GridSearchCV(knn_model,
                           knn_params,
                           cv=5,
                           n_jobs=-1,
                           verbose=1).fit(X, y)
```

```
Fitting 5 folds for each of 48 candidates, totalling 240 fits
```

GridSearchCV metodu ile 2 ile 50 arasındaki tüm komşuluklar denenecek yani 3 komşuluk için knn modeli kurulacak ve sonrasında skorlar incelenecek aynısı 50 ye kadar yapılacak ve en iyi sonucu veren komşuluk sayısı bulunacak

```
knn_gs_best.best_params_
```

```
{'n_neighbors': 17}
```

En iyi sonuç best params metodu ile 17 komşuluk değerinde bulundu

•knn_final = knn_model.set_params(**knn_gs_best.best_params_).fit(X, y)

• # 17 değeri best params metodunun içinde gömülü

•cv_results = cross_validate(knn_final,

• X,

• y,

• cv=5,

• scoring=["accuracy", "f1", "roc_auc"])

• Skorlara tekrardan göz atalım

```
cv_results['test_accuracy'].mean()
```

```
0.7669892199303965
```

```
cv_results['test_f1'].mean()
```

```
0.6170909049720137
```

```
cv_results['test_roc_auc'].mean()
```

```
0.8127938504542278
```

Sonuç
Skorların hepsi neredeyse 3 ila 4 puan arasında artış gösterdi böylelikle en iyi komşu sayımızı ve dolayısı ile en iyi modelimizi üretmiş olduk