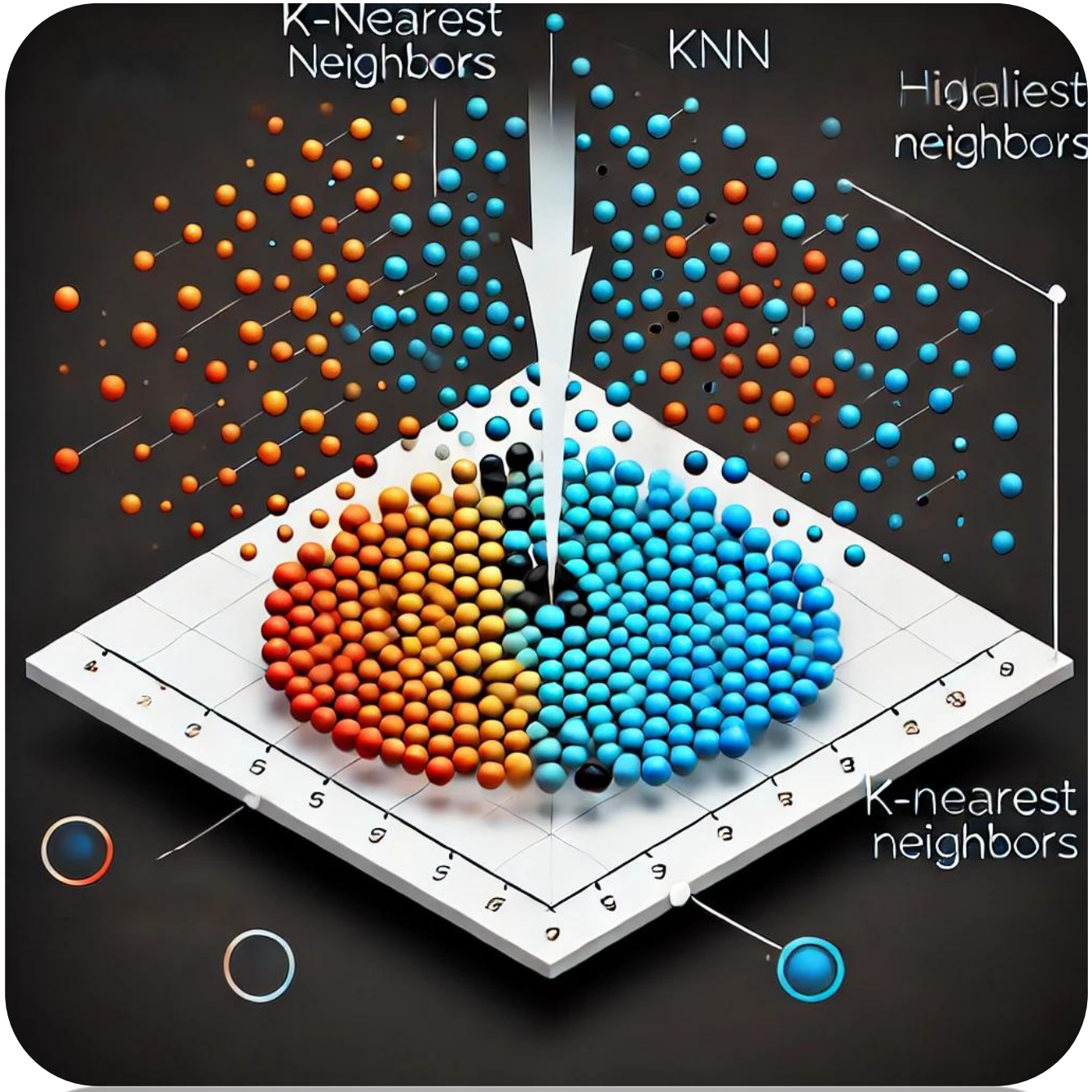
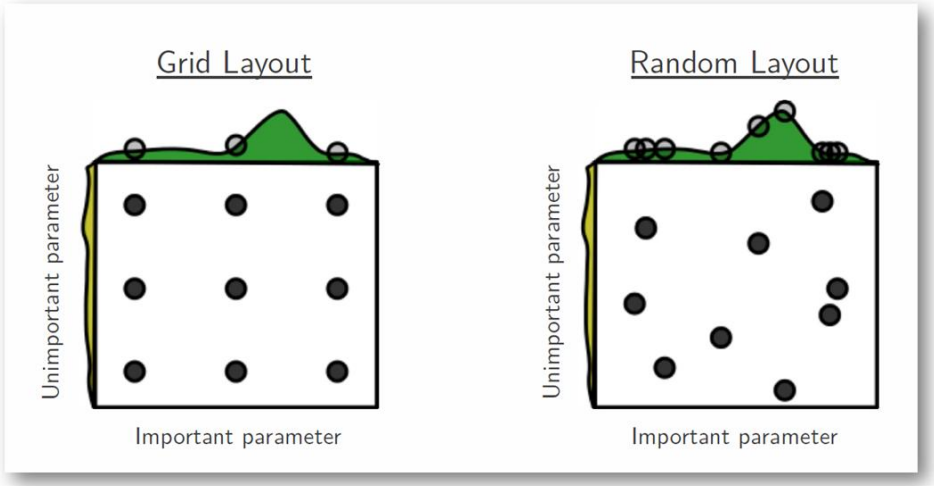

K-Nearest
Neighbors
KNN
Higaliest
neighbors
K-nearest
neighbors

Los hiperparámetros son propiedades específicas del modelo que se "fijan" incluso antes de que el modelo se entrene o se pruebe con los datos. Por ejemplo: en el caso de un bosque aleatorio, los hiperparámetros incluyen la cantidad de árboles de decisión en el bosque; en el caso de una red neuronal, existe la tasa de aprendizaje, la cantidad de capas ocultas, la cantidad de unidades en cada capa y varios otros parámetros.

- *Búsqueda en cuadrícula*
- *Búsqueda aleatoria*

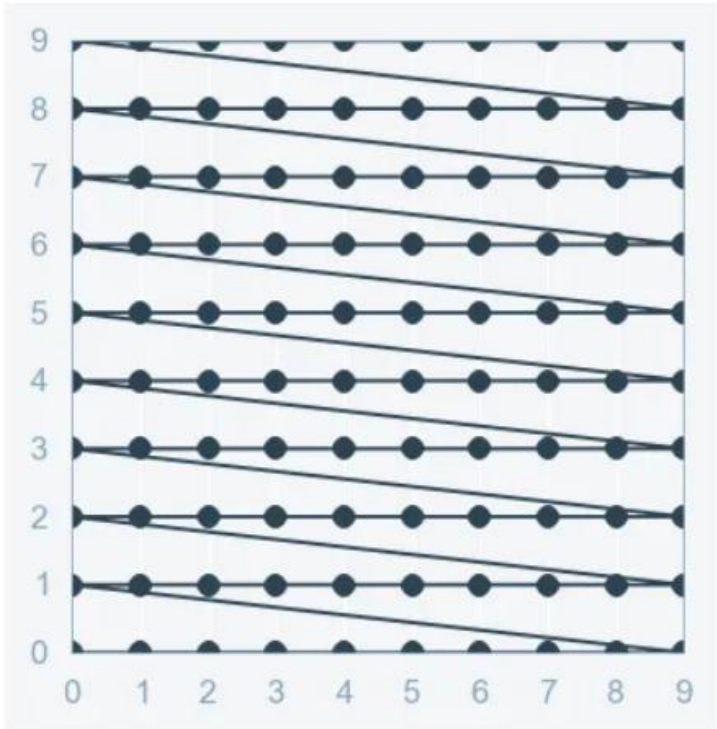



```
parameters = {'max_depth':[1,2,3,4,5],
              'min_samples_leaf':[1,2,3,4,5],
              'min_samples_split':[2,3,4,5],
              'criterion' : ['gini','entropy']}
```

```
In [ ]: search_obj = GridSearchCV(model, parameters,cv=5, scoring='f1_macro')
        fit_obj = search_obj.fit(xtrain, ytrain)
        print(fit_obj.cv_results_['mean_test_score'])
        best_model = fit_obj.best_estimator_
```

Búsqueda en cuadrícula

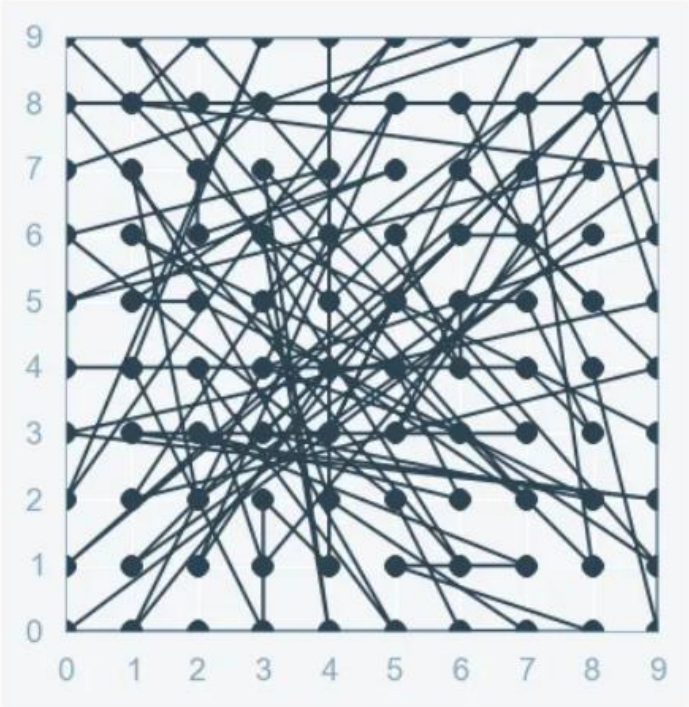
En la búsqueda en cuadrícula, probamos todas las combinaciones de una lista preestablecida de valores de los hiperparámetros y evaluamos el modelo para cada combinación. El patrón que se sigue aquí es similar al de la cuadrícula, donde todos los valores se colocan en forma de matriz. Se tiene en cuenta cada conjunto de parámetros y se anota la precisión. Una vez que se evalúan todas las combinaciones, se considera que el modelo con el conjunto de parámetros que ofrece la mayor precisión es el mejor.



Representación visual de la búsqueda en cuadrícula

Búsqueda aleatoria

La búsqueda aleatoria es una técnica en la que se utilizan combinaciones aleatorias de hiperparámetros para encontrar la mejor solución para el modelo construido.Intenta combinaciones aleatorias de un rango de valores. Para optimizar la búsqueda aleatoria, la función se evalúa en una cierta cantidad de configuraciones aleatorias en el espacio de parámetros.



Representación visual de una búsqueda aleatoria

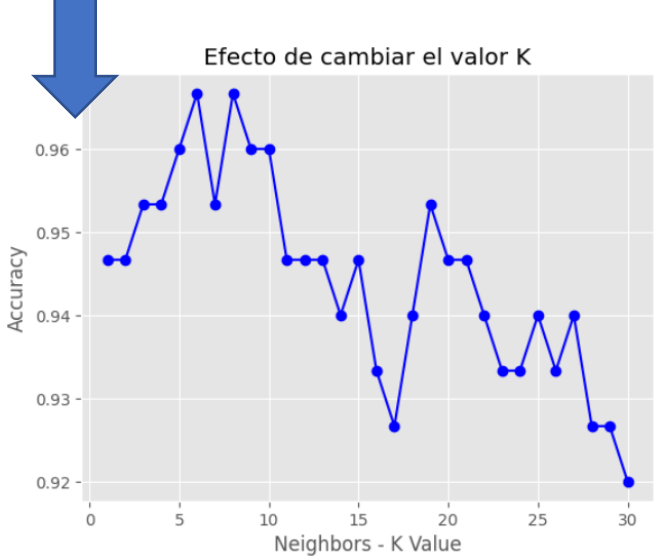
En el artículo Random Search for Hyper-Parameter Optimization de Bergstra y Bengio, los autores muestran empírica y teóricamente que la búsqueda aleatoria es n eficiente para la optimización de parámetros que la búsqueda en cuadrícula

2) Método de redecilla o Grid search

```
k_values = list(range(1, 31)) # From 1 to 30

from sklearn.neighbors import KNeighborsClassifier
KNN = KNeighborsClassifier()

hyperparameter_space = {'n_neighbors':k_values}

from sklearn.model_selection import GridSearchCV # Validación cruzada con Busqueda en redecilla de la K ideal
gs = GridSearchCV(KNN, param_grid=hyperparameter_space,
                  scoring='accuracy', cv=5) #Este código hace una búsqueda de Redecilla o Grid Search

gs.fit(X_scaled, y)

print("Mejor valor de K: ", gs.best_params_)
print("Accuracy promedio del mejor valor para K: ", gs.best_score_)
```

```
Mejor valor de K:  {'n_neighbors': 6}
Accuracy promedio del mejor valor para K:  0.9666666666666668
```

2) Método de redecilla o Grid search




```
k_values = list(range(1, 31)) # From 1 to 30

from sklearn.neighbors import KNeighborsClassifier
KNN = KNeighborsClassifier()

hyperparameter_space = {'n_neighbors':k_values}

from sklearn.model_selection import GridSearchCV # Validación cruzada con Busqueda en redecilla de la K ideal
gs = GridSearchCV(KNN, param_grid=hyperparameter_space,
                  scoring='accuracy', cv=5) #Este código hace una búsqueda de Redecilla o Grid Search

gs.fit(X_scaled, y)

print("Mejor valor de K: ", gs.best_params_)
print("Accuracy promedio del mejor valor para K: ", gs.best_score_)
```

```
Mejor valor de K:  {'n_neighbors': 6}
Accuracy promedio del mejor valor para K:  0.9666666666666668
```

"Pues, puedo cambiar la resolución de esta gráfica ajustando el número de vecinos *K*. En síntesis, el algoritmo K-Nearest Neighbors (*KNN*) es útil tanto para tareas de clasificación como de regresión. Funciona de manera estándar utilizando la distancia euclidiana, evaluando los vecinos más cercanos para hacer predicciones, ya sean numéricas o categóricas. Este algoritmo puede ajustarse mediante validación cruzada, variando el número de vecinos (*K*) o modificando la métrica de distancia según las necesidades específicas.

El valor de *K* es importante; aunque *K* es útil, al utilizarlo, se debe tener en cuenta que KNN es un algoritmo supervisado. Sin embargo, cuando no tenemos etiquetas, podemos utilizar técnicas de clustering no supervisado que evalúan distancias promedio entre vecinos o similares. Así que, la clave está en identificar si un conjunto de datos necesita un enfoque supervisado o no supervisado."

Cross-Validation


```
In [11]: scores.mean()

Out[11]: 0.6259887797548376
```