Machine Learning

K-Nearest Neighbors

Mostafa S. Ibrahim
Teaching, Training and Coaching for more than a decade!

Artificial Intelligence & Computer Vision Researcher
PhD from Simon Fraser University - Canada
Bachelor / MSc from Cairo University - Egypt
Ex-(Software Engineer / ICPC World Finalist)

K-Nearest Neighbors (KNN)

- Nearest Neighbors (**KNN**) is one of the simplest **supervised** machine learning algorithms used for both **classification and regression** tasks.
- **Goal**: find the **closest K training examples** and predicts the answer based on them
  - In **classification**: use the label with the most votes
  - In **regression**: average or median of the k-nearest neighbors
- K is a number given by the user

Procedure

- **Calculate** the Euclidean **distance** (or any other distance metric) between the input data point and all other points in the **training data set.**
- **Sort** the distances and pick the nearest k neighbors.
- For the classification, count the frequency of the labels and use the most voted label
  - For example with K = 5 we have classes {2, 2, 2, 2, 7}
    - So class 2 is repeated 4 times: the most
- Tips
  - Scale the features (similar to K-Means)

Pros and Cons

- Pros
  - Simple
  - No training
- Cons
  - Significantly slower as the number of examples and/or features increase
    - More efficient data structures like KD-Trees can be used
  - Need value of K
    - A smaller k can capture noise in the data
    - A larger k can smooth over far data points which might result in underfitting
      - **Cross-validation** can be used to find the optimal k

Create a dataset

```
X, y = make_classification(n_samples=10000, n_features=10, random_state=42)

# Split the data into training and testing sets
X_train, X_test, y_train, y_test = \
    train_test_split(X, y, test_size=0.2, random_state=42)

# Scale the features
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)
```

Trying several Ks

```python
for k in [3, 4, 5, 6, 7, 10, 15, 30]:
    # Train a KNN classifier
    knn = KNeighborsClassifier(n_neighbors=k)
    knn.fit(X_train, y_train)

    # Predict on the test set
    y_pred = knn.predict(X_test)

    # Evaluate the performance
    accuracy = accuracy_score(y_test, y_pred)
    print(f"K = {k} - Accuracy: {accuracy * 100:.2f}%")
```

```
K = 3 - Accuracy: 90.20%
K = 4 - Accuracy: 89.55%
K = 5 - Accuracy: 91.40%
K = 6 - Accuracy: 90.90%
K = 7 - Accuracy: 92.00%
K = 10 - Accuracy: 91.85%
K = 15 - Accuracy: 91.80%
K = 30 - Accuracy: 91.95%
```

```python
class KNN:
    def __init__(self, k=3, task='classification'):
        self.k, self.task = k, task

    def fit(self, X, y):
        self.X_train, self.y_train = X, y

    def predict(self, X):
        return np.array([self._predict(x) for x in X])

    def _predict(self, x):
        distances = [np.sum((x - x_train) ** 2) for x_train in self.X_train]
        k_indices = np.argsort(distances)[:self.k]

        k_nearest_outputs = [self.y_train[i] for i in k_indices]
        if self.task == 'classification':
            most_common = Counter(k_nearest_outputs).most_common(1)
            return most_common[0][0]
        elif self.task == 'regression':
            return np.mean(k_nearest_outputs)
```

Anomaly Detection

- We can also try using it for anomaly detection.
- Find its k-neighbours. The kth one shouldn’t be bigger than a threshold

```
def is_anomaly(self, x):
    threshold = 1.0
    distances = [np.sum((x - x_train) ** 2) for x_train in self.X_train]
    kth_distance = np.sort(distances)[self.k-1]
    return kth_distance > threshold * threshold
```

Notes

- Our KNN is slower than SKlearn KNN
- KNN is much a common coding task in interviews

"Acquire knowledge and impart it to the people."

"Seek knowledge from the Cradle to the Grave."