Automatic Detection of Hyperparameters in Datasets

Predictions for Nearest Neighbor Models

Alexis Rosenfeld & François Delafontaine

Fall 2024

Abstract

This report investigates meta-learning techniques for predicting the optimal k in k-Nearest Neighbors (KNN) models. By leveraging dataset meta-features, we propose a method to reduce the computational cost of hyperparameter selection. The work includes simulations to generate synthetic datasets, feature extraction, regression modeling, and validation of performance metrics. Additionally, we discuss the theoretical foundations of meta-learning, practical methodologies, and potential applications for reducing machine learning complexity.

Contents

1 Introduction

Hyperparameter selection is a fundamental step in machine learning, often addressed using computationally expensive methods such as grid search or cross-validation. This study focuses on k, the key hyperparameter in KNN models, and investigates its prediction through dataset meta-features. The primary objective is to reduce the computational cost of finding k from $O(n^2)$ to $O(1)$ using a predictive model.

2 What is Meta-Learning?

Meta-learning, often referred to as "learning to learn," is the process of improving machine learning systems by leveraging experiences across multiple tasks. It involves systematically observing algorithm performance on various datasets and using this knowledge to adapt quickly to new tasks.

2.1 Key Components of Meta-Learning

A critical aspect of meta-learning is the extraction of **meta-features**, descriptive characteristics summarizing dataset properties, such as:

- Dataset size (n_rows)
- Dimensionality ($n\_features$)
- Statistical moments (mean, variance, skewness)
- Correlation or class imbalance

By analyzing the relationship between meta-features and algorithm performance, meta-learning systems can predict the best hyperparameters or algorithms for new datasets. This approach offers several benefits:

1. Reduces computational costs by avoiding exhaustive searches.
2. Generalizes well to unseen tasks by learning from prior experiences.
3. Facilitates automated machine learning pipelines.

> Meta-learning, or *learning to learn*, is the science of systematically observing how different machine learning approaches perform on a wide range of learning tasks, and then learning from this experience, or meta-data, to learn new tasks much faster than otherwise possible.

— Hutter, F., Kotthoff, L., & Vanschoren, J. (2019). *Automated Machine Learning: Methods, Systems, Challenges* (p. 219). Springer Nature.

3 Data Sources and Simulation

To support this study, three types of datasets were used:

1. **Real datasets:** Sourced from publicly available repositories.
2. **Simulated datasets:** Generated with k pre-defined, followed by the calculation of meta-features.
3. **Test datasets:** Meta-features are created first, and k is obtained through KNN iterations.

This report primarily relies on the simulated datasets due to their flexibility for model training and testing.

3.1 Meta-Features

The meta-features extracted include:

- n_rows: Number of samples.
- $n\_classes$: Number of unique labels in the target variable.
- $n\_features$: Number of predictors.
- $n\_nonrand$: Count of non-random features (weighted by correlation).
- $distr\_type$: Distribution type (linear, quadratic, etc.).
- $distr\_noise$: Overlap likelihood of feature values.
- $mean\_var$: Weighted mean of feature variances.
- $mean\_skew$: Weighted mean of feature skewness.
- $mean\_kurtosis$: Weighted mean of feature kurtosis.

4 Simulation and Dataset Generation

4.1 The Use of Simulation in Meta-Learning

Simulation plays a critical role in machine learning by enabling data scientists to evaluate models in controlled environments before applying them to real-world data. This process is particularly useful for:

- Understanding how specific models behave under various conditions.
- Testing the ability of a model to identify true relationships in a hypothetical dataset.
- Simplifying model selection by generating diverse datasets without the need for extensive data collection.

While simulation provides flexibility and scalability, it also comes with the challenge of realism. Simulated data must closely mimic real-world patterns to ensure that the insights gained are applicable beyond the simulation.

4.2 Simulation in This Study

For this project, the simulation process diverged from the traditional approach of hypothesizing relationships between features and labels. Instead, we leveraged the known relationship between the dataset's characteristics (meta-features) and the optimal hyperparameter k in KNN models.

This approach enabled us to:

- Directly calculate the label (k) from the raw dataset.
- Focus on generating variations in raw datasets to explore the robustness of our meta-learning model.
- Avoid creating hypothetical relationships, thus ensuring that the results were grounded in quantifiable data properties.

4.3 Simulation Process and Dataset Types

We generated synthetic datasets $D = y + X$, where:

- y: Target variable representing the labels.
- X: Matrix of predictors with controlled statistical properties.

From these datasets, we derived meta-features (X_s) to form the meta-dataset $D_t = y_k + X_s$, where y_k represents the optimal k values calculated using traditional KNN iterations.

4.4 Key Characteristics of Simulated Datasets

To ensure robust predictions, the simulation process incorporated variations in:

- Distribution types (linear, quadratic, etc.).
- Noise levels ($distr\_noise$): Controlling the likelihood of feature overlap.
- Dimensionality ($n\_features$) and dataset size (n_rows).
- Feature correlations and their impact on k.

4.5 Advantages of Simulation in Meta-Learning

The primary advantages of using simulation in this study are:

- The ability to test model behavior across a wide range of dataset characteristics.
- The creation of diverse training data for our regression model.
- The elimination of hypothetical assumptions, as the optimal k is directly derived from the data.

By focusing on realistic variations in raw datasets, our simulation process provided a robust foundation for predicting the hyperparameter k through meta-learning.

5 Methodology and Pipeline

5.1 Preprocessing

The preprocessing steps included:

1. **One-Hot Encoding:** Transforming categorical variables.
2. **Scaling:** Standardizing feature values for numerical stability.
3. **Feature Selection:** Retaining features with significant correlation.

5.2 Regression Model

A linear regression model was chosen as the baseline:

$$k = \beta_0 + \beta_1 X_1 + \beta_2 X_2 + \cdots + \beta_n X_n$$

The model predicts k based on the extracted meta-features.

6 Results and Evaluation

6.1 Performance Metrics

The model's performance was evaluated using:

- Mean Squared Error (MSE)
- Coefficient of Determination (R^2)

Cross-validation was used to ensure robustness.

6.2 Delta Accuracy Metric

The **delta accuracy metric** quantifies the performance gap between the predicted k and the optimal k:

$$\Delta\text{Accuracy} = \text{Accuracy}(\text{optimal } k) - \text{Accuracy}(\hat{k})$$

7 Discussion

7.1 Challenges

The model struggled with highly noisy datasets and those with imbalanced feature distributions. Future iterations should explore advanced regression models or neural networks.

7.2 Ethical Considerations

Real-world applications must address fairness and privacy concerns, particularly when using sensitive datasets.

8 Conclusion

This study successfully demonstrates the potential of meta-learning to predict KNN hyperparameters efficiently. By leveraging dataset meta-features, we significantly reduce computational costs. Future research should refine feature selection and extend testing to real-world datasets.

A Python Code for Simulation

```python
import numpy as np

def simulate_dataset(n_samples, n_features, noise_level):
    X = np.random.normal(0, 1, size=(n_samples, n_features))
    noise = np.random.normal(0, noise_level, size=X.shape)
    y = (X.sum(axis=1) > 0).astype(int)
    X += noise
    return X, y
```

Listing 1: Dataset Simulation Code

B References

References

[1] F. Hutter, L. Kotthoff, J. Vanschoren, *Automated Machine Learning: Methods, Systems, Challenges*, Springer, 2019.

C Appendix - R Code

```
    import os, time, re, joblib
    import numpy as np
    import pandas as pd
    import matplotlib.pyplot as plt
    from scipy import stats
    from scipy.stats import skew, kurtosis
    from sklearn.neighbors import KNeighborsClassifier as sk_kNN
    from sklearn.metrics import accuracy_score as sk_acc
    from sklearn.metrics import mean_squared_error as sk_mse
    from sklearn.metrics import r2_score as sk_r2
    from sklearn.model_selection import train_test_split as sk_tts
    from sklearn.model_selection import cross_val_score as sk_cross
    from sklearn.preprocessing import OneHotEncoder, StandardScaler
    from sklearn.linear_model import LinearRegression as sk_lm
    from sklearn.neural_network import MLPRegressor as sk_nn

    class KG_base():
        def __init__(self):
            self.log_path = ""
            self.head = ['best_k', 'n_rows', 'n_classes', 'n_features',
                         'n_nonrand', 'distr_type', 'distr_noise',
                         'mean_var', 'mean_skew', 'mean_kurtosis']

        def _ifds(self, x, y=None):
            if y is not None:
                return x, y
            elif isinstance(x, np.ndarray):
                return x[:,1:], x[:,0]
            elif isinstance(x, pd.core.frame.DataFrame):
                yn = x.columns[0]
                return x.drop([yn]), x[yn]
            else:
                y = []
                for i, ix in x:
                    y = x[i][0]; x[i] = x[i][1:]
                return np.array(x), np.array(y)

        def log(self, txt, f="", end="\n"):
            f = self.log_path if not f else f
            if f:
                mode = "w" if not os.path.isfile(f) else "a"
                with open(f, mode=mode, encoding="utf-8") as wf:
                    wf.write(txt+end)
            else:
                print(txt, end=end)

        def save(self, f, dat, columns=None):
            columns = self.head if columns is None else columns
            if not os.path.isfile(f):
                pd.DataFrame(dat, columns=columns).to_excel(f, index=False)
            else:
                df = pd.read_excel(f)
```

```
                nr = df[df.columns[0]].count()
                for i, row in enumerate(dat):
                    df.loc[nr+i] = row
                df.to_excel(f, index=False)

        def load(self, f):
            if not os.path.isfile(f):
                return np.array([])
            return pd.read_excel(f).to_numpy()

        def params(self, d_params={}):
            d_cpy = {}
            for k, v in self.__dict__.items():
                if k.startswith("_"):
                    continue
                d_cpy[k] = v
            for k, v in d_params.items():
                if k in d_cpy:
                    d_cpy[k] = self.__dict__[k] = v
            return d_cpy

        def show(self, x, y=None, f="", columns=[]):
            x, y = self._ifds(x, y)
            columns = self.head if not columns else columns
            if isinstance(y, np.ndarray):
                df = pd.DataFrame(np.hstack((y.reshape(-1, 1), x)), columns=
                    columns)
            else:
                df = pd.concat([y, x], axis=1, join="inner")
            self.log(df.head(), f)

        def select_features(self, x, y=None, tol=0.2, delete=False):
            x, y = self._ifds(x, y)
            l_index = []
            for i in range(x.shape[1]):
                ix = x[:,i] if isinstance(x, np.ndarray) else \
                     pd.DataFrame(x.iloc[:,i].values.reshape(-1, 1))
                corr = np.corrcoef(y, ix)[0][1]
                corr = abs(corr)
                if corr > tol:
                    l_index.append(i)
            if delete:
                l_rem = list(filter(None, [i if i not in l_index else None
                                           for i in range(x.shape[1])]))
                if isinstance(x, pd.core.frame.DataFrame):
                    x.drop(x.columns[l_rem], axis=1, inplace=True)
                elif isinstance(x, np.ndarray):
                    x = np.delete(x, l_rem, 1)
                return x
            return l_index

    class KG_test(KG_base):
        def __init__(self):
            super().__init__()
            self.lim_row = [100, 10000]           # Nombre de lignes
            self.lim_ycl = [3, 5]                 # Nombre de classes de y
            self.lim_nbx = [3, 9]                 # Nombre de variables x
            self.distr_type = -1                  # Type de distribution
            self.distr_noise = -1                 # Niveau de bruit

        def generate(self, view=False, cheat=True, save_path=""):
            pass

```

```
115         def get_bestk(self, x, y=None):
116             # Calcule le meilleur k (nombre de voisins)
117             pass
118 
119         def get_features(self, x, y=None, d_vi=None):
120             pass
121 
122     if __name__ == "__main__":
123         kg_test = KG_test()
124         dat, k = kg_test.sim(100, "", True, "")
125         print("Done.")
```

Listing 2: Exemple de classe Python