Lecture 13
Model Selection and Hyperparameter Tuning

Andrew Ilyas (Slide Credit: Dennis Sun)
Stanford University
DATASCI 112

May 2, 2025

1. Recap
2. Model Selection and Hyperparameter Tuning
3. Grid Search

1. Recap
2. Model Selection and Hyperparameter Tuning
3. Grid Search

Here's a machine learning model.

```
from sklearn.pipeline import make_pipeline
from sklearn.preprocessing import StandardScaler
from sklearn.neighbors import KNeighborsRegressor

pipeline = make_pipeline(
    StandardScaler(),
    KNeighborsRegressor(n_neighbors=5, metric="euclidean"))
X_train = df_train[["win", "summer"]]
y_train = df_train["price"]
```

Here's a machine learning model.

```
from sklearn.pipeline import make_pipeline
from sklearn.preprocessing import StandardScaler
from sklearn.neighbors import KNeighborsRegressor

pipeline = make_pipeline(
    StandardScaler(),
    KNeighborsRegressor(n_neighbors=5, metric="euclidean"))
X_train = df_train[["win", "summer"]]
y_train = df_train["price"]
```

The right way to evaluate machine learning models is *test error*,

Here's a machine learning model.

```
from sklearn.pipeline import make_pipeline
from sklearn.preprocessing import StandardScaler
from sklearn.neighbors import KNeighborsRegressor

pipeline = make_pipeline(
    StandardScaler(),
    KNeighborsRegressor(n_neighbors=5, metric="euclidean"))
X_train = df_train[["win", "summer"]]
y_train = df_train["price"]
```

The right way to evaluate machine learning models is *test error*, which is estimated using cross-validation.

Here's a machine learning model.

```
from sklearn.pipeline import make_pipeline
from sklearn.preprocessing import StandardScaler
from sklearn.neighbors import KNeighborsRegressor

pipeline = make_pipeline(
    StandardScaler(),
    KNeighborsRegressor(n_neighbors=5, metric="euclidean"))
X_train = df_train[["win", "summer"]]
y_train = df_train["price"]
```

The right way to evaluate machine learning models is *test error*, which is estimated using cross-validation.

```
from sklearn.model_selection import cross_val_score
scores = cross_val_score(
    pipeline,
    X=X_train, y=y_train,
    scoring="neg_mean_squared_error",
    cv=4)
-scores.mean()
```

Here's a machine learning model.

```
from sklearn.pipeline import make_pipeline
from sklearn.preprocessing import StandardScaler
from sklearn.neighbors import KNeighborsRegressor

pipeline = make_pipeline(
    StandardScaler(),
    KNeighborsRegressor(n_neighbors=5, metric="euclidean"))
X_train = df_train[["win", "summer"]]
y_train = df_train["price"]
```

The right way to evaluate machine learning models is *test error*, which is estimated using cross-validation.

```
from sklearn.model_selection import cross_val_score
scores = cross_val_score(
    pipeline,
    X=X_train, y=y_train,
    scoring="neg_mean_squared_error",
    cv=4)
-scores.mean()
```

```
375.27166666666665
```

Here's a machine learning model.

```
from sklearn.pipeline import make_pipeline
from sklearn.preprocessing import StandardScaler
from sklearn.neighbors import KNeighborsRegressor
                                         scaler
pipeline = make_pipeline(
    StandardScaler(),
    KNeighborsRegressor(n_neighbors=5, metric="euclidean"))
X_train = df_train[["win", "summer"]]
y_train = df_train["price"]
```

The right way to evaluate machine learning models is *test error*, which is estimated using cross-validation.

```
from sklearn.model_selection import cross_val_score
scores = cross_val_score(
    pipeline,
    X=X_train, y=y_train,
    scoring="neg_mean_squared_error",
    cv=4)
-scores.mean()
```

375.2716666666665

Here's a machine learning model.

```
from sklearn.pipeline import make_pipeline
from sklearn.preprocessing import StandardScaler
from sklearn.neighbors import KNeighborsRegressor

pipeline = make_pipeline(
    StandardScaler(),
    KNeighborsRegressor(n_neighbors=5, metric="euclidean"))
X_train = df_train[["win", "summer"]]
y_train = df_train["price"]
```

scaler method

The right way to evaluate machine learning models is *test error*, which is estimated using cross-validation.

```
from sklearn.model_selection import cross_val_score
scores = cross_val_score(
    pipeline,
    X=X_train, y=y_train,
    scoring="neg_mean_squared_error",
    cv=4)
-scores.mean()
```

375.2716666666665

Here's a machine learning model.

```
from sklearn.pipeline import make_pipeline
from sklearn.preprocessing import StandardScaler
from sklearn.neighbors import KNeighborsRegressor
```

scaler method k

```
pipeline = make_pipeline(
    StandardScaler(),
    KNeighborsRegressor(n_neighbors=5, metric="euclidean"))
X_train = df_train[["win", "summer"]]
y_train = df_train["price"]
```

The right way to evaluate machine learning models is *test error*, which is estimated using cross-validation.

```
from sklearn.model_selection import cross_val_score
scores = cross_val_score(
    pipeline,
    X=X_train, y=y_train,
    scoring="neg_mean_squared_error",
    cv=4)
-scores.mean()
```

375.2716666666665

Here's a machine learning model.

```
from sklearn.pipeline import make_pipeline
from sklearn.preprocessing import StandardScaler
from sklearn.neighbors import KNeighborsRegressor
```

scaler method k metric

```
pipeline = make_pipeline(
    StandardScaler(),
    KNeighborsRegressor(n_neighbors=5, metric="euclidean"))
X_train = df_train[["win", "summer"]]
y_train = df_train["price"]
```

The right way to evaluate machine learning models is *test error*, which is estimated using cross-validation.

```
from sklearn.model_selection import cross_val_score
scores = cross_val_score(
    pipeline,
    X=X_train, y=y_train,
    scoring="neg_mean_squared_error",
    cv=4)
-scores.mean()
```

```
375.2716666666665
```

Here's a machine learning model.

```
from sklearn.pipeline import make_pipeline
from sklearn.preprocessing import StandardScaler
from sklearn.neighbors import KNeighborsRegressor
```

scaler method metric

k

```
pipeline = make_pipeline(
    StandardScaler(),
    KNeighborsRegressor(n_neighbors=5, metric="euclidean"))
X_train = df_train[["win", "summer"]]
y_train = df_train["price"]
```

variables

The right way to evaluate machine learning models is *test error*, which is estimated using cross-validation.

```
from sklearn.model_selection import cross_val_score
scores = cross_val_score(
    pipeline,
    X=X_train, y=y_train,
    scoring="neg_mean_squared_error",
    cv=4)
-scores.mean()
```

```
375.2716666666665
```

Here's a machine learning model.

```
from sklearn.pipeline import make_pipeline
from sklearn.preprocessing import StandardScaler
from sklearn.neighbors import KNeighborsRegressor

pipeline = make_pipeline(
    StandardScaler(),
    KNeighborsRegressor(n_neighbors=5, metric="euclidean"))
X_train = df_train[["win", "summer"]]
y_train = df_train["price"]
```

scaler method k metric

variables

The right way to evaluate machine learning models is *test error*, which is estimated using cross-validation.

```
from sklearn.model_selection import cross_val_score
scores = cross_val_score(
    pipeline,
    X=X_train, y=y_train,
    scoring="neg_mean_squared_error",
    cv=4)
-scores.mean()
```

```
375.27166666666665
```

How do we choose between all the options (scaler, k, etc.)?

Two Related Problems

Two Related Problems

Model Selection refers to the choice of:

Two Related Problems

Model Selection refers to the choice of:

- which input features to include (e.g., winter rainfall, summer temperature)

Two Related Problems

Model Selection refers to the choice of:

- which input features to include (e.g., winter rainfall, summer temperature)
- what preprocessing to do (e.g., scaler)

Two Related Problems

Model Selection refers to the choice of:

- which input features to include (e.g., winter rainfall, summer temperature)
- what preprocessing to do (e.g., scaler)
- what machine learning method to use (e.g., k-nearest neighbors)

Two Related Problems

Model Selection refers to the choice of:

- which input features to include (e.g., winter rainfall, summer temperature)
- what preprocessing to do (e.g., scaler)
- what machine learning method to use (e.g., k-nearest neighbors)

Hyperparameter Tuning refers to the choice of parameters in the machine learning method.

Two Related Problems

Model Selection refers to the choice of:

- which input features to include (e.g., winter rainfall, summer temperature)
- what preprocessing to do (e.g., scaler)
- what machine learning method to use (e.g., k-nearest neighbors)

Hyperparameter Tuning refers to the choice of parameters in the machine learning method.
For k-nearest neighbors, hyperparameters include:

Two Related Problems

Model Selection refers to the choice of:

- which input features to include (e.g., winter rainfall, summer temperature)
- what preprocessing to do (e.g., scaler)
- what machine learning method to use (e.g., k-nearest neighbors)

Hyperparameter Tuning refers to the choice of parameters in the machine learning method.
For k-nearest neighbors, hyperparameters include:

- k

Two Related Problems

Model Selection refers to the choice of:

- which input features to include (e.g., winter rainfall, summer temperature)
- what preprocessing to do (e.g., scaler)
- what machine learning method to use (e.g., k-nearest neighbors)

Hyperparameter Tuning refers to the choice of parameters in the machine learning method.
For k-nearest neighbors, hyperparameters include:

- k
- metric (e.g., Euclidean distance)

Two Related Problems

Model Selection refers to the choice of:

- which input features to include (e.g., winter rainfall, summer temperature)
- what preprocessing to do (e.g., scaler)
- what machine learning method to use (e.g., k-nearest neighbors)

Hyperparameter Tuning refers to the choice of parameters in the machine learning method.
For k-nearest neighbors, hyperparameters include:

- k
- metric (e.g., Euclidean distance)

The distinction isn't important. We always use cross-validation and pick the model / hyperparameter with the smallest test error.

Example of Model Selection

Which input features should we include?

Example of Model Selection

Which input features should we include?

- winter rain, summer temp

Example of Model Selection

Which input features should we include?

- winter rain, summer temp
- winter rain, summer temp, harvest rain

Example of Model Selection

Which input features should we include?

- winter rain, summer temp
- winter rain, summer temp, harvest rain
- winter rain, summer temp, harvest rain, Sept. temp

Example of Model Selection

Which input features should we include?

- winter rain, summer temp
- winter rain, summer temp, harvest rain
- winter rain, summer temp, harvest rain, Sept. temp

```
for features in [["win", "summer"],
                 ["win", "summer", "har"],
                 ["win", "summer", "har", "sep"]]:
  scores = cross_val_score(
      pipeline,
      X=df_train[features],
      y=df_train["price"],
      scoring="neg_mean_squared_error",
      cv=4)
  print(features, -scores.mean())
```

Example of Model Selection

Which input features should we include?

- winter rain, summer temp
- winter rain, summer temp, harvest rain
- winter rain, summer temp, harvest rain, Sept. temp

```
for features in [["win", "summer"],
                 ["win", "summer", "har"],
                 ["win", "summer", "har", "sep"]]:
  scores = cross_val_score(
      pipeline,
      X=df_train[features],
      y=df_train["price"],
      scoring="neg_mean_squared_error",
      cv=4)
  print(features, -scores.mean())
```

```
['win', 'summer'] 375.27166666666665
['win', 'summer', 'har'] 363.04047619047617
['win', 'summer', 'har', 'sep'] 402.4507142857142
```

Example of Model Selection

Which input features should we include?

- winter rain, summer temp
- winter rain, summer temp, harvest rain
- winter rain, summer temp, harvest rain, Sept. temp

```
for features in [["win", "summer"],
                 ["win", "summer", "har"],
                 ["win", "summer", "har", "sep"]]:
  scores = cross_val_score(
      pipeline,
      X=df_train[features],
      y=df_train["price"],
      scoring="neg_mean_squared_error",
      cv=4)
  print(features, -scores.mean())
```

```
['win', 'summer'] 375.27166666666665
['win', 'summer', 'har'] 363.04047619047617    ✓
['win', 'summer', 'har', 'sep'] 402.4507142857142
```

Example of Hyperparameter Tuning

What is the best value of k?

```
X_train = df_train[["win", "summer", "har"]]
```

Example of Hyperparameter Tuning

What is the best value of k?

```
X_train = df_train[["win", "summer", "har"]]
```

```
ks, test_mses = range(1, 7), []
for k in ks:
  pipeline = make_pipeline(
      StandardScaler(),
      KNeighborsRegressor(n_neighbors=k, metric="euclidean"))
  scores = cross_val_score(
      pipeline, X_train, y_train,
      scoring="neg_mean_squared_error", cv=4)
  test_mses.append(-scores.mean())
```

Example of Hyperparameter Tuning

What is the best value of k?

```
X_train = df_train[["win", "summer", "har"]]
```

```
ks, test_mses = range(1, 7), []
for k in ks:
  pipeline = make_pipeline(
      StandardScaler(),
      KNeighborsRegressor(n_neighbors=k, metric="euclidean"))
  scores = cross_val_score(
      pipeline, X_train, y_train,
      scoring="neg_mean_squared_error", cv=4)
  test_mses.append(-scores.mean())
```

```
pd.Series(test_mses, index=ks).plot.line()
```

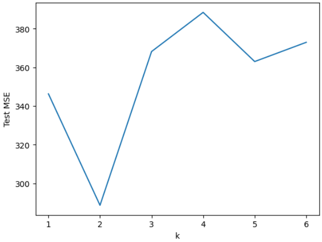
Example of Hyperparameter Tuning

What is the best value of k?

```
X_train = df_train[["win", "summer", "har"]]
```

```
ks, test_mses = range(1, 7), []
for k in ks:
  pipeline = make_pipeline(
      StandardScaler(),
      KNeighborsRegressor(n_neighbors=k, metric="euclidean"))
  scores = cross_val_score(
      pipeline, X_train, y_train,
      scoring="neg_mean_squared_error", cv=4)
  test_mses.append(-scores.mean())
```

```
pd.Series(test_mses, index=ks).plot.line()
```



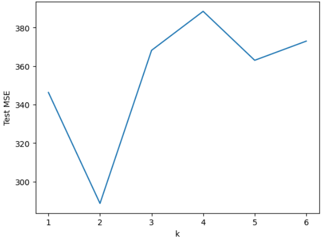
Example of Hyperparameter Tuning

What is the best value of k?

```
X_train = df_train[["win", "summer", "har"]]
```

```
ks, test_mses = range(1, 7), []
for k in ks:
  pipeline = make_pipeline(
      StandardScaler(),
      KNeighborsRegressor(n_neighbors=k, metric="euclidean"))
  scores = cross_val_score(
      pipeline, X_train, y_train,
      scoring="neg_mean_squared_error", cv=4)
  test_mses.append(-scores.mean())
```

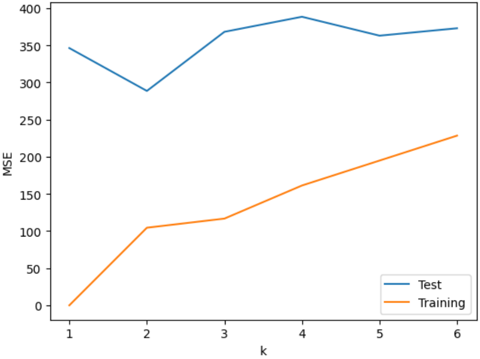
```
pd.Series(test_mses, index=ks).plot.line()
```



The best value of k is 2.

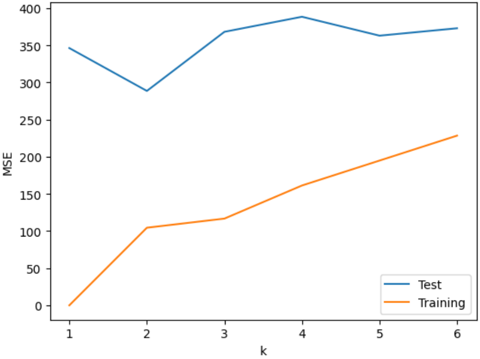
Training vs. Test Error

Here are the training and test MSEs on the same graph.



Training vs. Test Error

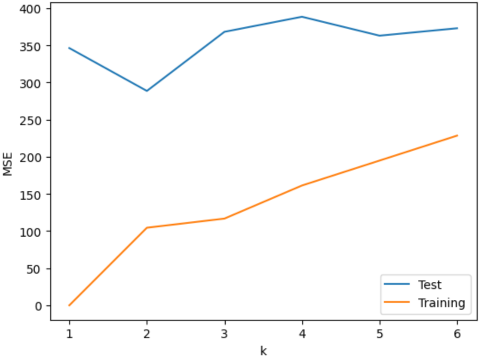
Here are the training and test MSEs on the same graph.



Notice that training MSE only goes down as we decrease k.

Training vs. Test Error

Here are the training and test MSEs on the same graph.

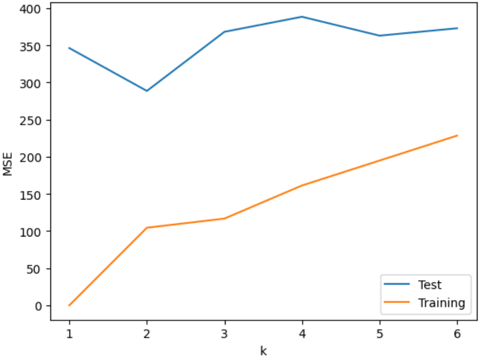


Notice that training MSE only goes down as we decrease k.

If we optimize for training MSE, then we will pick $k = 1$, but this has worse test MSE.

Training vs. Test Error

Here are the training and test MSEs on the same graph.



Notice that training MSE only goes down as we decrease k.

If we optimize for training MSE, then we will pick $k = 1$, but this has worse test MSE.

In other words, the $k = 1$ model has **overfit** to the training data.

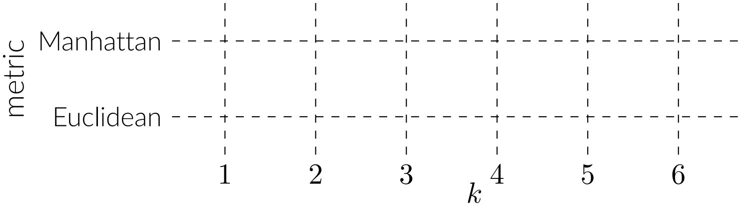
Grid Search

Suppose we want to choose k and the distance metric (Euclidean or Manhattan).

Grid Search

Suppose we want to choose k and the distance metric (Euclidean or Manhattan).

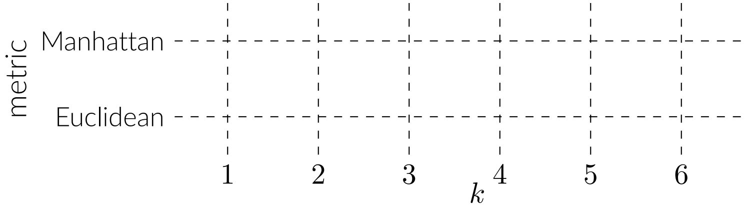
We need to try all 12 combinations on the following grid:



Grid Search

Suppose we want to choose k and the distance metric (Euclidean or Manhattan).

We need to try all 12 combinations on the following grid:



Scikit-Learn's `GridSearchCV` automates the creation of a grid with all combinations.

Grid Search in Scikit-Learn

Let's try out `GridSearchCV` in a Colab.

Challenges with Grid Search

Why can't all machine learning be automated by grid search?

Challenges with Grid Search

Why can't all machine learning be automated by grid search?

There were 5 input features in the original data (summer temp, harvest rainfall, winter rainfall, Sept. temperature, age).

Challenges with Grid Search

Why can't all machine learning be automated by grid search?

There were 5 input features in the original data (summer temp, harvest rainfall, winter rainfall, Sept. temperature, age).
How many combinations of features would we need to try?

Challenges with Grid Search

Why can't all machine learning be automated by grid search?

There were 5 input features in the original data (summer temp, harvest rainfall, winter rainfall, Sept. temperature, age).
How many combinations of features would we need to try?
$2^5 = 32$

Challenges with Grid Search

Why can't all machine learning be automated by grid search?

There were 5 input features in the original data (summer temp, harvest rainfall, winter rainfall, Sept. temperature, age). How many combinations of features would we need to try?

$2^5 = 32$

Now, combine this with the choice of k, distance metric, and scaler.

Challenges with Grid Search

Why can't all machine learning be automated by grid search?

There were 5 input features in the original data (summer temp, harvest rainfall, winter rainfall, Sept. temperature, age).
How many combinations of features would we need to try?
$2^5 = 32$

Now, combine this with the choice of k, distance metric, and scaler.

- 6 choices of k

Challenges with Grid Search

Why can't all machine learning be automated by grid search?

There were 5 input features in the original data (summer temp, harvest rainfall, winter rainfall, Sept. temperature, age).
How many combinations of features would we need to try?
$2^5 = 32$

Now, combine this with the choice of k, distance metric, and scaler.

- 6 choices of k
- 2 choices of distance metric (Euclidean, Manhattan)

Challenges with Grid Search

Why can't all machine learning be automated by grid search?

There were 5 input features in the original data (summer temp, harvest rainfall, winter rainfall, Sept. temperature, age).
How many combinations of features would we need to try?
$2^5 = 32$

Now, combine this with the choice of k, distance metric, and scaler.

- 6 choices of k
- 2 choices of distance metric (Euclidean, Manhattan)
- 2 choices of scaler (`StandardScaler`, `MinMaxScaler`)

Challenges with Grid Search

Why can't all machine learning be automated by grid search?

There were 5 input features in the original data (summer temp, harvest rainfall, winter rainfall, Sept. temperature, age).
How many combinations of features would we need to try?
$2^5 = 32$

Now, combine this with the choice of k, distance metric, and scaler.

- 6 choices of k
- 2 choices of distance metric (Euclidean, Manhattan)
- 2 choices of scaler (`StandardScaler`, `MinMaxScaler`)

That's already $32 \times 6 \times 2 \times 2 = 768$ models.

Challenges with Grid Search

Why can't all machine learning be automated by grid search?

There were 5 input features in the original data (summer temp, harvest rainfall, winter rainfall, Sept. temperature, age). How many combinations of features would we need to try?

$2^5 = 32$

Now, combine this with the choice of k, distance metric, and scaler.

- 6 choices of k
- 2 choices of distance metric (Euclidean, Manhattan)
- 2 choices of scaler (`StandardScaler`, `MinMaxScaler`)

That's already $32 \times 6 \times 2 \times 2 = 768$ models.

And that's not even considering models besides k-nearest neighbors!

Heuristics for Parameter Tuning

For large data sets, it is impossible to try every combination of models and parameters.

Heuristics for Parameter Tuning

For large data sets, it is impossible to try every combination of models and parameters.

So instead we use *heuristics*, which do not guarantee the best model but tend to work well in practice.

Heuristics for Parameter Tuning

For large data sets, it is impossible to try every combination of models and parameters.

So instead we use *heuristics*, which do not guarantee the best model but tend to work well in practice.

- **randomized search**: try random combinations of parameters, implemented in Scikit-Learn as `RandomizedSearchCV`.

Heuristics for Parameter Tuning

For large data sets, it is impossible to try every combination of models and parameters.

So instead we use *heuristics*, which do not guarantee the best model but tend to work well in practice.

- **randomized search**: try random combinations of parameters, implemented in Scikit-Learn as `RandomizedSearchCV`.
- **coordinate optimization**:
  - start with guesses for all parameters,
  - try all values for *one* parameter (holding the rest constant) and find the best value of that parameter,
  - cycle through the parameters.

Heuristics for Parameter Tuning

For large data sets, it is impossible to try every combination of models and parameters.

So instead we use *heuristics*, which do not guarantee the best model but tend to work well in practice.

- **randomized search**: try random combinations of parameters, implemented in Scikit-Learn as `RandomizedSearchCV`.
- **coordinate optimization**:
  - start with guesses for all parameters,
  - try all values for *one* parameter (holding the rest constant) and find the best value of that parameter,
  - cycle through the parameters.

You will have the chance to practice this on Lab 4, which is a kaggle competition to build the best machine learning model. There will be prizes for the winners!