CS4619: Artificial Intelligence II

Non-Linear Models

Derek Bridge
School of Computer Science and Information Technology
University College Cork

Initialization

In [1]: %load_ext autoreload
   %autoreload 2
   %matplotlib inline

In [2]: import warnings
from sklearn.exceptions import DataConversionWarning
warnings.filterwarnings(action='ignore', category=DataConversionWarning)
warnings.filterwarnings(action='ignore', category=FutureWarning)
warnings.filterwarnings(action='ignore', category=DeprecationWarning)

In [3]: import pandas as pd
import numpy as np
import matplotlib.pyplot as plt

In [4]: from sklearn.compose import ColumnTransformer
   from sklearn.pipeline import Pipeline
   from sklearn.preprocessing import StandardScaler
   from sklearn.preprocessing import FunctionTransformer
   from sklearn.preprocessing import LabelEncoder
   from sklearn.preprocessing import PolynomialFeatures
   from sklearn.linear_model import LinearRegression
   from sklearn.linear_model import SGDRegressor
   from sklearn.linear_model import LogisticRegression
   from sklearn.model_selection import cross_val_score

Overview of CS4619 Artificial Intelligence II

- We will discuss what's coming up
- And we will discuss what aptitudes you need if you are to be successful in CS4619
Revision

- Prediction
  - Regression
  - Classification
    - Binary classification
    - Multiclass classification

Supervised learning of a model from a labeled training set

- "learning a model from the data"/
  "training the algorithm on the data"/
  "fitting a model to the data"/
- minimization of a loss function

Unknown target function \( f \)

Training set \( \{(x^{(1)}, y^{(1)}), \ldots, (x^{(m)}, y^{(m)})\} \)

Hypotheses \( \mathcal{H} \)

Learning algorithm

Final hypothesis \( h \approx f \)

Revision

- Error estimation (accuracy estimation)
  - Often we use a scoring function that is different from the loss function
  - Training error and test error
    - Holdout
    - \( k \)-fold cross-validation
    - Stratified holdout (for classification)
    - Stratified \( k \)-fold cross-validation (for classification)
Revision

- Linear regression
  \[ y = \beta_0 x_0 + \beta_1 x_1 + \beta_2 x_2 + \cdots + \beta_n x_n \]
  or
  \[ y = x\beta \]
  In effect, the learning algorithm must find the values of the vector \( \beta \)
- OLS regression is linear regression using mean-squared-error as the loss function
  - The Normal Equation
  - Gradient Descent (which can be used with many different loss functions)
- Similarly, logistic regression
  - For binary classification:
    \[ \text{Prob}(y = 1|x) = \sigma(x\beta) \]
    where \( \sigma(z) = \frac{1}{1+e^{-z}} \) and the loss function is the cross-entropy function (also called the log-loss function)
  - For multiclass classification,
    - One-versus-rest
    - One-versus-one
    - Multinomial logistic regression:
      \[ \text{Prob}(y = c|x) = \sigma(x\beta_c) = \frac{e^{x\beta_c}}{\sum_{c'\in C} e^{x\beta_{c'}}} \]
      and cross-entropy is the loss function

Revision

- **Parameters** of the model
  - Their values are found by the learning algorithm
  - E.g. \( \beta \) for linear/logistic regression
- **Hyperparameters** are parameters of the learning algorithm
  - Their values are set by us (but see later lecture on grid search)
  - E.g. none in the case of the Normal Equation
  - E.g. learning rate, number of iterations and others in the case of Gradient Descent

Revision: Linear Regression for the Cork Property Prices Dataset

In [5]: # Use pandas to read the CSV file into a DataFrame
cork_df = pd.read_csv("datasets/dataset_corkA.csv")

In [6]: # Shuffle
cork_df = cork_df.take(np.random.permutation(len(cork_df)))

In [7]: # Get the target values
cork_y = cork_df["price"].values

In [8]: # The features we want to select - we will only use the numeric features
cork_numeric_features = ["flarea", "bdrms", "bthrms", "floors"]
In [9]: # Create the preprocessor
cork_simple_preprocessor = ColumnTransformer([  
    ("num", FunctionTransformer(lambda x : x), cork_numeric_features),  
    ("drop")
])

# Create a pipeline that combines the preprocessor with the estimator
cork_lr = Pipeline([  
    ("drop", cork_simple_preprocessor),  
    ("est", LinearRegression())
])

In [10]: # Error estimation
np.mean(cross_val_score(cork_lr, cork_df, cork_y, scoring="neg_mean_absolute_error", cv=10))

Out[10]: -61.030256726209814

- Above, we used scikit-learn's LinearRegression class, which uses the Normal Equation
  - Hence, no need for scaling
  - And no hyperparameters to worry about
- Instead, we could have used the SGDRegressor class for Stochastic Gradient Descent
  - Scaling is advisable
  - And we can set the hyperparameters (or take their default values), e.g. learningRate, max_iter

In [11]: # Create another preprocessor
    cork_scaling_preprocessor = ColumnTransformer([  
        ("num", StandardScaler(), cork_numeric_features),  
        ("drop")
    ])

# Create another OLS pipeline that combines the new preprocessor with the estimator
    cork_sgd = Pipeline([  
        ("drop", cork_scaling_preprocessor),  
        ("est", SGDRegressor(penalty="None", max_iter=1000))
    ])

In [12]: # Error estimation
np.mean(cross_val_score(cork_sgd, cork_df, cork_y, scoring="neg_mean_absolute_error", cv=10))

Out[12]: -61.01611323924634

Revision: Binary Classification for the CS1109 Dataset

In [13]: # Use pandas to read the CSV file into a DataFrame
    cs1109_df = pd.read_csv("datasets/dataset_cs1109.csv")

In [14]: # Shuffle
    cs1109_df = cs1109_df.take(np.random.permutation(len(cs1109_df)))

In [15]: # Get the target values
    cs1109_y = cs1109_df["outcome"][values]

In [16]: # The features we want to select
    cs1109_features = ["lect", "lab", "cao"]
In [17]: # Create a Logistic Regression pipeline
cs1109_lr = Pipeline[
    ("pre", ColumnTransformer([("num", StandardScaler(), cs1109_features)], remainder="drop")),
    ("est", LogisticRegression())
]

In [18]: # Accuracy estimation
np.mean(cross_val_score(cs1109_lr, cs1109_df, cs1109_y, scoring="accuracy", cv=10))

Out[18]: 0.794978354978355

- scikit-learn's LogisticRegression class is already using SGDClassifier for Stochastic Gradient Descent
- Hence, there is no point us trying SGDClassifier ourselves

Revision: Multiclass Classification for the Iris Dataset

In [19]: # Use pandas to read the CSV file into a DataFrame
iris_df = pd.read_csv("datasets/dataset_iris.csv")

In [20]: # Shuffle
iris_df = iris_df.take(np.random.permutation(len(iris_df)))

In [21]: # Get the target values
iris_y = iris_df["label"].values
encoder = LabelEncoder()
iris_y_encoded = encoder.fit_transform(iris_y)

In [22]: # The features we want to select
iris_features = ["sepal_length", "sepal_width", "petal_length", "petal_width"]

In [23]: # Create a one-versus-rest pipeline
iris_ovr = Pipeline[
    ("pre", ColumnTransformer([("num", StandardScaler(), iris_features)], remainder="drop")),
    ("est", LogisticRegression())
]

In [24]: # Accuracy estimation
np.mean(cross_val_score(iris_ovr, iris_df, iris_y_encoded, scoring="accuracy", cv=10))

Out[24]: 0.8933333333333333

- scikit-learn's LogisticRegression class by default gives us one-versus-rest for multiclass classification
- But we can request multinomial logistic regression
# Create the cross-entropy pipeline

```python
iris_crossent = Pipeline([  
    ("pre", ColumnTransformer([  
        ("num", StandardScaler(), iris_features),  
        ("est", LogisticRegression(multi_class="multinomial", solver="newton-cg"))  
    ])),  
]), remainsder="drop")
```

# Accuracy estimation
```
np.mean(cross_val_score(iris_crossent, iris_df, iris_y_encoded, scoring="accuracy", cv=10))
```

```
Out[26]: 0.9533333333333334
```

## Non-Linearity
- What if the true relationship between the features and the target values is non-linear?
- A linear model may not be good enough:
  - The test error may be too high
  - Even the training error may be too high!
- What we need is a more complex model
  - Roughly, a model is more complex if it has more parameters
  - E.g. quadratic functions for a dataset with just three features ($x_1$, $x_2$ and $x_3$):
    $$
    \beta_0 + \beta_1 x_1 + \beta_2 x_2 + \beta_3 x_3 + \beta_4 x_1^2 + \beta_5 x_2^2 + \beta_6 x_3^2 + \beta_7 x_1 x_2 + \beta_8 x_1 x_3 + \beta_9 x_2 x_3
    $$
  - What would cubic functions look like (again in the case where there are just three features)?

## Polynomial Regression
- There are learning algorithms for non-linear models (including neural networks, see later)
- But there’s a really neat trick for using a linear model to get some of the same effect!
  - We add extra features to our dataset
    - Some of the new features will be powers of the original features, e.g. a new feature $x_4 = x_1^2$
    - Others will be products of the original features, e.g. a new feature $x_7 = x_1 x_2$ (these are often called interaction features)
  - Then learn a linear model on the new dataset
This is called Polynomial Regression
- E.g.
  - We want to learn models of this form
    $$
    \beta_0 + \beta_1 x_1 + \beta_2 x_2 + \beta_3 x_3 + \beta_4 x_1^2 + \beta_5 x_2^2 + \beta_6 x_3^2 + \beta_7 x_1 x_2 + \beta_8 x_1 x_3 + \beta_9 x_2 x_3
    $$
  - But instead we learn linear models of this form
    $$
    \beta_0 + \beta_1 x_1 + \beta_2 x_2 + \beta_3 x_3 + \beta_4 x_4 + \beta_5 x_5 + \beta_6 x_6 + \beta_7 x_7 + \beta_8 x_8 + \beta_9 x_9
    $$
  - but where
    $$
    x_4 = x_1^2 \\
    x_5 = x_2^2 \\
    x_6 = x_3^2 \\
    x_7 = x_1 x_2 \\
    x_8 = x_1 x_3 \\
    x_9 = x_2 x_3
    $$
Polynomial Regression in scikit-learn

- There's a class called PolynomialFeatures
- We simply include it in our pipeline
- Bear in mind that it may or may not make things better — which we will discuss in upcoming lectures

```python
In [27]: # Cork Property Prices polynomial regression
cork_poly = Pipeline([  
    ('pre', cork_scaling_preprocessor),  
    ('poly', PolynomialFeatures(degree=2, include_bias=False)),  
    ('est', LinearRegression())])
```

```python
In [28]: # Error estimation
np.mean(cross_val_score(cork_poly, cork_df, cork_y, scoring="neg_mean_absolute_error", cv=10))
Out[28]: -60.22062025549383
```

```python
In [29]: # CS1109 polynomial regression
cs1109_poly = Pipeline([  
    ('pre', ColumnTransformer([("num", StandardScaler(), cs1109_features)], remainder="drop")),  
    ('poly', PolynomialFeatures(degree=2, include_bias=False)),  
    ('est', LogisticRegression())])
```

```python
In [30]: # Accuracy estimation
np.mean(cross_val_score(cs1109_poly, cs1109_df, cs1109_y, scoring="accuracy", cv=10))
Out[30]: 0.7864069264069264
```

```python
In [31]: # Iris polynomial regression
iris_poly = Pipeline([  
    ('pre', ColumnTransformer([("num", StandardScaler(), iris_features)], remainder="drop")),  
    ('poly', PolynomialFeatures(degree=2, include_bias=False)),  
    ('est', LogisticRegression(multi_class="multinomial", solver="newton-cg"))])
```

```python
In [32]: # Accuracy estimation
np.mean(cross_val_score(iris_poly, iris_df, iris_y_encoded, scoring="accuracy", cv=10))
Out[32]: 0.9666666666666668
```

- Try some other values for the degree argument
Conclusions

- Polynomial Regression gives a combinatorial explosion in the number of features
  - `PolynomialFeatures(degree=d, include_bias=False)` transforms a dataset that had $n$ features into one that has $\frac{(n+d)!}{d!n!}$ features
- Surprisingly, allowing more complex models does not necessarily lead to lower error/higher accuracy
  - See underfitting and overfitting in upcoming lectures

In [ ]: