SEN608

Data Analytics and Statistics

Week 5: Regression and Prediction

Assoc. Prof. Dr. Onur Polat

Source: Bruce, P., Bruce, A., Gedeck, P. (2020). Practical Statistics for Data Scientists, 2nd Ed., O'Reilly — Chapter 4

OVERVIEW

This Week's Agenda

- Simple Linear Regression
- Multiple Linear Regression
- Model Assessment and Selection
- Weighted Regression
- Factor Variables (Dummy Variables)
- Interpreting the Regression Equation
- Regression Diagnostics
- Polynomial and Spline Regression, GAM

KEY CONCEPTS

Regression Analysis — Key Terms

- Response Variable (Y): The dependent variable to be predicted (target, outcome)
- Predictor Variable (X): The feature used for prediction (independent variable)
- Fitted Values: The model's predicted values y-hat
- Residuals: Difference between observed and predicted: ei = yi - y-hat-i
- Ordinary Least Squares (OLS): Minimizes the sum of squared residuals
- RMSE (Root Mean Squared Error): Standard measure of prediction error
- R-squared: Proportion of variance explained by the model (0 to 1)

SECTION I

Simple Linear Regression

The Regression Equation

- Models the linear relationship between one predictor (X) and one response (Y)
- b0 (intercept): Expected value of Y when X = 0
- b1 (slope): Change in Y for a one-unit increase in X

$$Y = b0 + b1\,X + e$$

- OLS: Minimizes sum of squared residuals = sum(yi - y-hat-i)^2
- Example Dataset: LungDisease.csv — Exposure (years) vs. PEFR

Simple Linear Regression in Python

```
from sklearn.linear_model import LinearRegression
import pandas as pd

lung = pd.read_csv('LungDisease.csv')
model = LinearRegression()
model.fit(lung[['Exposure']], lung['PEFR'])

print(f'Intercept (b0): {model.intercept_:.3f}')
print(f'Slope (b1): {model.coef_[0]:.3f}')

# Fitted values and residuals
fitted = model.predict(lung[['Exposure']])
residuals = lung['PEFR'] - fitted
```

Output: b0 ~ 424.58, b1 ~ -4.19 → PEFR decreases as exposure increases

Fitted Values and Residuals

- Fitted value (y-hat-i): The model's prediction for each observation
- Residual (ei = yi - y-hat-i): Difference between observed and predicted
- Residuals represent information the model failed to capture
- In a good model, residuals should be randomly distributed with no systematic pattern
- Prediction vs. Explanation: Regression can be used for both forecasting and understanding relationships

SECTION II

Multiple Linear Regression

Multiple Regression Equation

$$Y = b0 + b1\ X1 + b2\ X2 + \ldots + bp\ Xp + e$$

- Prediction using multiple independent variables
- Each coefficient (bj) shows the marginal effect of Xj on Y, holding all other predictors constant
- Example: King County House Sales data (house_sales.csv)
- Response variable: AdjSalePrice (adjusted sale price)
- Predictors: SqFtTotLiving, SqFtLot, Bathrooms, Bedrooms, BldgGrade

King County House Sales — Multiple Regression

```
house = pd.read_csv('house_sales.csv', sep='\t')

predictors = ['SqFtTotLiving', 'SqFtLot', 'Bathrooms',
              'Bedrooms', 'BldgGrade']
outcome = 'AdjSalePrice'

house_lm = LinearRegression()
house_lm.fit(house[predictors], house[outcome])

print(f'Intercept: {house_lm.intercept_:.3f}')
for name, coef in zip(predictors, house_lm.coef_):
    print(f'  {name}: {coef:.3f}')
```

Each 1 sq.ft. increase in living area → ~$228 increase in sale price

Model Assessment: RMSE and R-squared

- RMSE (Root Mean Squared Error): Average magnitude of prediction errors
- R-squared: Proportion of variance explained by the model (0 to 1)
- Adjusted R-squared: R-squared penalized for number of predictors

```
from sklearn.metrics import r2_score, mean_squared_error
import numpy as np

fitted = house_lm.predict(house[predictors])
RMSE = np.sqrt(mean_squared_error(house[outcome], fitted))
r2 = r2_score(house[outcome], fitted)
print(f'RMSE: {RMSE:.0f}')    # ~261,000
print(f'R2: {r2:.4f}')        # ~0.54
```

Detailed Analysis with statsmodels

```
import statsmodels.api as sm

model = sm.OLS(house[outcome],
               house[predictors].assign(const=1))
results = model.fit()
print(results.summary())

# Output includes:
# - t-statistic and p-value for each coefficient
# - F-statistic (overall model significance)
# - AIC / BIC information criteria
# - R-squared and Adj. R-squared
```

statsmodels provides much more detailed statistical output than scikit-learn

SECTION III

Model Selection and Weighted Regression

Stepwise Regression

- Forward Selection: Start with empty model, add the best predictor at each step
- Backward Elimination: Start with full model, remove the weakest predictor
- AIC (Akaike Information Criterion): Penalizes model complexity
- Lower AIC → better model (balances accuracy and parsimony)

```
from dmba import stepwise_selection, AIC_score

def train_model(variables):
    model = LinearRegression()
    model.fit(X[variables], y)
    return model

def score_model(model, variables):
    return AIC_score(y, model.predict(X[variables]), model)

best_model, best_vars = stepwise_selection(
    X.columns, train_model, score_model)
```

Weighted Regression

- Some observations are given more weight in the model fitting
- Example: More recent sales receive higher weight
- Use cases: Correcting heteroskedasticity, temporal weighting

```
# Weight by year: newer data gets higher weight
house['Year'] = house.DocumentDate.apply(
    lambda d: int(d.split('-')[0]))
house['Weight'] = house.Year - 2005

house_wt = LinearRegression()
house_wt.fit(house[predictors], house[outcome],
             sample_weight=house.Weight)
```

SECTION IV

Factor Variables

Dummy Variable Encoding

- Categorical variables cannot enter regression directly → numeric encoding needed
- A factor with P levels → P-1 dummy (binary) variables
- Reference level: The category with no dummy variable (drop_first=True)
- Each dummy coefficient represents the difference from the reference

```
# PropertyType: Multiplex, Single Family, Townhouse
print(pd.get_dummies(house['PropertyType'],
                     drop_first=True, dtype=int).head())

# Output: 0/1 columns for Single Family and Townhouse
# Reference category: Multiplex
```

Factors with Many Levels

- Variables like ZipCode have many levels → too many dummy variables
- Solution: Group levels into bins
- Textbook approach: Group zip codes into 5 bins based on residual medians
- This captures location information with a manageable number of variables

```
# Group zip codes by residual patterns
zip_groups['ZipGroup'] = pd.qcut(
    zip_groups['cum_count'], 5,
    labels=False, retbins=False)

# Add ZipGroup to the model
house = house.join(
    zip_groups[['ZipCode','ZipGroup']].set_index('ZipCode'),
    on='ZipCode')
```

SECTION V

Interpreting the Regression Equation

Correlated Predictors and Multicollinearity

- Correlated Predictors: Correlation among predictors can change coefficient signs
- Multicollinearity: When predictors are highly correlated, coefficients become unstable
- Symptom: Adding or removing a variable dramatically changes other coefficients
- Example: When SqFtTotLiving is in the model, Bedrooms coefficient can turn negative

Confounding Variables and Interactions

- Confounding Variable: A lurking variable that affects both X and Y, masking the true effect
- Example: Location (ZipGroup) affects both price and house features
- Interaction: The effect of X1 depends on the level of X2
- Example: SqFtTotLiving x ZipGroup — the value of extra space varies by location

```
import statsmodels.formula.api as smf

# Interaction term: SqFtTotLiving * ZipGroup
model = smf.ols(
    formula='AdjSalePrice ~ SqFtTotLiving*ZipGroup + '
            'SqFtLot + Bathrooms + Bedrooms + BldgGrade + '
            'PropertyType',
    data=house)
results = model.fit()
```

SECTION VI

Regression Diagnostics

Outliers and Influential Observations

- Outlier: An observation far from the model's expectation
- Standardized Residuals: $|e_i^*| > 2.5$ flags a potential outlier
- Leverage (Hat Value): How extreme an observation is in predictor space — Threshold: $2(p+1)/n$
- Cook's Distance: Combined measure of outlier status and leverage — Threshold: $4/(n-p-1)$
- Influence Plot: Hat values vs. studentized residuals, point size = Cook's distance

Outlier Analysis — Python

```
from statsmodels.stats.outliers_influence import OLSInfluence

# Subset model for ZipCode 98105
house_98105 = house.loc[house['ZipCode'] == 98105]
house_outlier = sm.OLS(house_98105[outcome],
                       house_98105[predictors].assign(const=1))
result_98105 = house_outlier.fit()

# Influence analysis
influence = OLSInfluence(result_98105)
sresiduals = influence.resid_studentized_internal

# Filter by Cook's distance
mask = [d < .08 for d in influence.cooks_distance[0]]
house_infl = house_98105.loc[mask]
```

Heteroskedasticity and Partial Residual Plots

- Heteroskedasticity: Residual variance changes with the fitted value
- Diagnosis: Look for systematic patterns in the fitted vs. |residual| plot
- Normality: Residual histogram should be roughly symmetric and bell-shaped
- Partial Residual Plots: Examine each predictor's effect individually
- If a nonlinear pattern appears → consider polynomial or spline regression

SECTION VII

Polynomial, Spline Regression and GAM

Polynomial Regression

- Add higher powers of X to capture nonlinear relationships
- Quadratic (degree 2): Y = b0 + b1 X + b2 X^2
- Textbook example: Adding SqFtTotLiving^2 captures curvature
- Caution: High-degree polynomials risk overfitting

```
model_poly = smf.ols(
    formula='AdjSalePrice ~ SqFtTotLiving + '
            'np.power(SqFtTotLiving, 2) + '
            'SqFtLot + Bathrooms + Bedrooms + BldgGrade',
    data=house_98105)
result_poly = model_poly.fit()
print(result_poly.summary())
```

Spline Regression

- Spline: Divide data range into segments, fit separate polynomials to each
- Curves join smoothly at knot points (continuous and differentiable)
- B-splines: The most common spline basis, df controls flexibility
- Advantage over polynomials: No wild oscillations at the extremes

```
formula = ('AdjSalePrice ~ '
           'bs(SqFtTotLiving, df=6, degree=3) + '
           'SqFtLot + Bathrooms + Bedrooms + BldgGrade')
model_spline = smf.ols(formula=formula,
                       data=house_98105)
result_spline = model_spline.fit()
```

GAM — Generalized Additive Models

- GAM: A separate smoothing function (smoothing spline) for each predictor
- Y = s(X1) + f(X2) + ... → Each variable can be linear or curved
- pygam library: s() for spline terms, l() for linear terms
- Partial dependence plots → visualize each variable's effect
- Advantage: Captures nonlinear relationships automatically while preserving interpretability

```
from pygam import LinearGAM, s, l

gam = LinearGAM(s(0, n_splines=12) + l(1) + l(2) + l(3) + l(4))
gam.gridsearch(X, y)
print(gam.summary())
# Use partial dependence plots to visualize
# each variable's effect separately
```

SUMMARY

What We Learned This Week

- Building prediction models with simple and multiple linear regression
- Model assessment: RMSE, R-squared, adjusted R-squared, AIC
- Automatic variable selection with stepwise regression
- Encoding categorical variables as dummies in regression
- Coefficient interpretation: correlated predictors, confounders, interactions
- Regression diagnostics: outliers, influential observations, heteroskedasticity
- Nonlinear relationships: polynomial, spline regression and GAM

References and Further Reading

- Bruce, P., Bruce, A., Gedeck, P. (2020). Practical Statistics for Data Scientists, 2nd Ed., O'Reilly — Chapter 4
- GitHub: github.com/gedeck/practical-statistics-for-data-scientists
- scikit-learn Documentation: scikit-learn.org/stable/modules/linear_model.html
- statsmodels Documentation: statsmodels.org/stable/regression.html
- Dataset: King County House Sales (house_sales.csv)

NEXT WEEK

Next Week: Classification

- Logistic Regression
- Classification metrics (accuracy, precision, recall, F1)
- ROC curve and AUC
- Naive Bayes and K-Nearest Neighbors (KNN)

Questions?