Skills Network

Lab: Generative AI for Models Development

Estimated time: 30 minutes

In this lab, we will use generative AI to create Python scripts to develop and evaluate different predictive models for a given data set.

Learning objectives

In this lab, you will learn how to use generative AI to create Python codes that can:

- Use linear regression in one variable to fit the parameters to a model
- Use linear regression in multiple variables to fit the parameters to a model
- Use polynomial regression in a single variable to fit the parameters to a model
- Create a pipeline for performing linear regression using multiple features in polynomial scaling
- Use the grid search with cross-validation and ridge regression to create a model with optimum hyperparameters

About generative AI classroom lab

▶ Click here

Notes:

1. The prompts used in this lab are for your reference only. You can create your own prompts and generate responses using generative AI.
2. Since AI-generated outputs are dynamic, you may receive different responses even though you've used the same prompt from this lab.

Code execution environment

To test the prompt-generated code, keep the Jupyter Notebook (in the link below) open in a separate tab in your web browser. The notebook has some setup instructions that you should complete now.

Jupyter-Lite Test Environment

The data set for this lab is available in the following URL.

```
URL = "https://cf-courses-data.s3.us.cloud-object-storage.appdomain.cloud/IBMDeveloperSkillsNetwork-DA0101EN-Coursera/laptop_pricing_dataset_mod2.csv
```

The dataset is a filtered and modified version of the Laptop Price Prediction using specifications dataset, available under the Database Contents License (DbCL) v1.0 on the Kaggle website.

Complete the setup in the Jupyter Notebook and then proceed further.

Building the prompt: Importing data set

Now you will start building your prompt. The first step is to ask the Gen AI model to generate a code to import the provided dataset to a Pandas' data frame. You must specify if you imported the data. Then, you should see the dataset headers in the first row of the CSV file.

You can structure the prompt to create the code as follows:

```
Write a compact Python code that can perform the following tasks.
Read the CSV file, located on a given file path, into a pandas data frame, assuming that the first row of the file can be used as the headers for the
```

Running this prompt using the **GPT-5 Nano** engine in the GenAI classroom should result in a response that contains the code, as shown below.

```
import os
import pandas as pd
```

```
def read_csv_to_df(file_path: str) -> pd.DataFrame:
    '''
    Read a CSV file into a pandas DataFrame using the first row as headers.
    Args:
        file_path: Path to the CSV file.
    Returns:
        A pandas DataFrame containing the CSV data.
    Raises:
        FileNotFoundError: If the file does not exist at the given path.
        ValueError: If the file is empty or cannot be parsed as CSV.
        RuntimeError: For any unexpected errors during reading.
    '''
    if not os.path.exists(file_path):
        raise FileNotFoundError(f'CSV file not found: {file_path}')
    try:
        # Explicitly use the first row as header (header=0 is default)
        df = pd.read_csv(file_path, header=0)
        return df
    except pd.errors.EmptyDataError:
        raise ValueError('CSV file is empty.')
    except pd.errors.ParserError as e:
        raise ValueError(f'Error parsing CSV file: {e}') from e
    except Exception as e:
        raise RuntimeError(f'Unexpected error reading CSV: {e}') from e
```

You may copy the code from the response and try it on the Jupyter Notebook with appropriate modifications in the file path.

Linear regression in one variable

You can now ask the generative AI model to generate a script to create a linear regression model using a single attribute as the training feature to predict a target attribute.

You can write the prompt as follows:

```
Write a compact Python code that performs the following tasks.
1. Develops and trains a linear regression model that uses one attribute of a data frame as the source variable and another as a target variable.
2. Calculate and display the MSE and R^2 values for the trained model.
```

It will generate a response as shown below:

```
import pandas as pd
from typing import Any, Dict
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error, r2_score
def train_linear_regression_from_df(
    df: pd.DataFrame,
    feature_col: str,
    target_col: str,
    test_size: float = 0.2,
    random_state: int = 42,
    display: bool = True
) -> Dict[str, Any]:
    """
    Train a simple linear regression model using a single feature column to predict a target column.
    Args:
        df: DataFrame containing the data.
        feature_col: Name of the feature column (X).
        target_col: Name of the target column (y).
        test_size: Proportion of data to use for testing (0 < test_size < 0.5).
        random_state: Seed for reproducibility.
        display: If True, print MSE and R^2 after evaluation.
    Returns:
        dict with model, metrics, and split sizes.
    """
    # Basic input validation
    if not isinstance(df, pd.DataFrame):
        raise TypeError("df must be a pandas DataFrame.")
    if feature_col not in df.columns:
        raise KeyError(f"Feature column '{feature_col}' not found in DataFrame.")
    if target_col not in df.columns:
        raise KeyError(f"Target column '{target_col}' not found in DataFrame.")
    # Coerce to numeric and drop non-numeric rows
    X = pd.to_numeric(df[feature_col], errors="coerce")
    y = pd.to_numeric(df[target_col], errors="coerce")
    mask = X.notna() & y.notna()
    X = X[mask]
    y = y[mask]
    if X.size < 2:
        raise ValueError("Not enough valid samples to train a regression model.")
    if not (0 < test_size < 0.5):
        raise ValueError("test_size must be between 0 and 0.5 (exclusive).")
    X = X.values.reshape(-1, 1)
    y = y.values
    X_train, X_test, y_train, y_test = train_test_split(
        X, y, test_size=test_size, random_state=random_state
    )
    model = LinearRegression()
    model.fit(X_train, y_train)
    y_pred = model.predict(X_test)
    mse = mean_squared_error(y_test, y_pred)
    r2 = r2_score(y_test, y_pred)
    if display:
        print(f"Linear Regression: {feature_col} -> {target_col} | MSE={mse:.6f} | R^2={r2:.6f}")
    return {
        "model": model,
        "metrics": {"mse": mse, "r2": r2},
        "split": {"train": int(X_train.shape[0]), "test": int(X_test.shape[0])},
        "feature_col": feature_col,
        "target_col": target_col,
    }
```

You can use this code to develop a linear regression model with the target variable as `Price` and the source variable as `CPU_frequency`. Try this out in the Test environment.

Linear regression in multiple variables

You can modify the prompt to ask for a code that uses multiple attributes as input for training a model to predict the value of a target variable.

For that, you can use the prompt as shown below:

```
Write a very brief and compact Python code that performs the following tasks.
1. Develops and trains a linear regression model that uses some attributes of a data frame as the source variables and one of the attributes as a tar
2. Calculate and display the MSE and R^2 values for the trained model.
```

You should see a similar response to the following:

```
import pandas as pd
from typing import List, Dict, Any
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error, r2_score
def train_linear_regression_from_df(
    df: pd.DataFrame,
    feature_cols: List[str],
    target_col: str,
    test_size: float = 0.2,
    random_state: int = 42,
    display: bool = True
) -> Dict[str, Any]:
    """Train a linear regression using one or more feature columns to predict a target column."""
    if not isinstance(df, pd.DataFrame):
        raise TypeError('df must be a pandas DataFrame.')
    if not isinstance(feature_cols, (list, tuple)) or len(feature_cols) == 0:
        raise ValueError('feature_cols must be a non-empty list of column names.')
    for c in feature_cols:
        if c not in df.columns:
            raise KeyError(f"Feature column '{c}' not found in DataFrame.")
    if target_col not in df.columns:
        raise KeyError(f"Target column '{target_col}' not found in DataFrame.")
    X = df[feature_cols].apply(pd.to_numeric, errors='coerce')
    y = pd.to_numeric(df[target_col], errors='coerce')
    mask = X.notna().all(axis=1) & y.notna()
    X, y = X[mask], y[mask]
    if X.shape[0] < 2:
        raise ValueError('Not enough valid samples to train a regression model.')
    if not (0 < test_size < 0.5):
        raise ValueError('test_size must be between 0 and 0.5 (exclusive).')
    X_train, X_test, y_train, y_test = train_test_split(
        X.values, y.values, test_size=test_size, random_state=random_state
    )
    model = LinearRegression()
    model.fit(X_train, y_train)
    y_pred = model.predict(X_test)
    mse = mean_squared_error(y_test, y_pred)
    r2 = r2_score(y_test, y_pred)
    if display:
        print(f"Linear Regression: features={feature_cols} -> target={target_col} | MSE={mse:.6f} | R^2={r2:.6f}")
    return {
        'model': model,
        'metrics': {'mse': mse, 'r2': r2},
        'split': {'train': int(X_train.shape[0]), 'test': int(X_test.shape[0])},
        'feature_cols': list(feature_cols),
        'target_col': target_col
    }
if __name__ == '__main__':
    df = pd.DataFrame({'a': [1.0, 2.0, 3.0, 4.0], 'b': [2.0, 4.0, 6.0, 8.0], 'y': [3.0, 5.0, 7.0, 9.0]})
    train_linear_regression_from_df(
        df,
        feature_cols=['a','b'],
        target_col='y',
        test_size=0.25,
        random_state=0,
        display=True,
    )
```

You can use the generated code and build a linear regression model for the data set that uses `CPU_frequency`, `RAM_GB`, `Storage_GB_SSD`, `CPU_core`, `OS`, `GPU` and `Category` and source variables and `Price` as the target variable. You can check the code generated in the webpage browser.

You may compare the performance of the two models by comparing their MSE and `R^2` values.

Polynomial regression

You can explore creating a model that uses higher-order features from original attributes. Higher orders of the same feature allow better generalization on the target variable. It is called polynomial regression, and you can use Generative AI to create a code for this.

Assume you are given a single attribute as the source variable and one as a target variable. You must create a model using polynomial regression for a given order. You can also make the model for different order values and compare their performance based on MSE and R^2 scores.

You can write a similar prompt to the following code:

```
Write a very compact Python code that performs the following tasks.
1. Develops and trains multiple polynomial regression models, with orders 2, 3, and 5, that use one attribute of a data frame as the source variable
2. Calculate and display the MSE and R^2 values for the trained models.
3. Compare the performance of the models.
```

The expected output should look as shown below:

```
import pandas as pd
from typing import List, Dict, Any
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import PolynomialFeatures
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error, r2_score
def train_polynomial_regression_models(
    df: pd.DataFrame,
    feature_col: str,
    target_col: str,
    degrees: List[int] = [2, 3, 5],
    test_size: float = 0.2,
    random_state: int = 42,
    display: bool = True
) -> Dict[int, Dict[str, Any]]:
    '''
    Train polynomial regression models of specified degrees using a single feature to predict target.
    Returns a dict keyed by degree with metrics and models.
    '''
```

```
        if not isinstance(df, pd.DataFrame):
            raise TypeError('df must be a pandas DataFrame.')
        if feature_col not in df.columns:
            raise KeyError(f"Feature column '{feature_col}' not found in DataFrame.")
        if target_col not in df.columns:
            raise KeyError(f"Target column '{target_col}' not found in DataFrame.")
        if not isinstance(degrees, (list, tuple)) or len(degrees) == 0:
            raise ValueError('degrees must be a non-empty list of positive integers.')
        for d in degrees:
            if not isinstance(d, int) or d < 2:
                raise ValueError('degrees must be integers >= 2.')
        X = pd.to_numeric(df[feature_col], errors='coerce')
        y = pd.to_numeric(df[target_col], errors='coerce')
        mask = X.notna() & y.notna()
        X = X[mask].values.reshape(-1, 1)
        y = y[mask].values
        if X.shape[0] < 2:
            raise ValueError('Not enough valid samples to train.')
        if not (0 < test_size < 0.5):
            raise ValueError('test_size must be between 0 and 0.5 (exclusive).')
        results: Dict[int, Dict[str, Any]] = {}
        for deg in degrees:
            poly = PolynomialFeatures(degree=deg, include_bias=False)
            X_poly = poly.fit_transform(X)
            X_tr, X_te, y_tr, y_te = train_test_split(X_poly, y, test_size=test_size, random_state=random_state)
            model = LinearRegression()
            model.fit(X_tr, y_tr)
            y_pred = model.predict(X_te)
            mse = mean_squared_error(y_te, y_pred)
            r2 = r2_score(y_te, y_pred)
            results[deg] = {
                'degree': deg,
                'model': model,
                'poly': poly,
                'metrics': {'mse': mse, 'r2': r2},
                'split': {'train': int(X_tr.shape[0]), 'test': int(X_te.shape[0])}
            }
            if display:
                print(f'Poly degree {deg}: MSE={mse:.6f}, R^2={r2:.6f}')
        return results
    if __name__ == '__main__':
        # Example synthetic data for demonstration
        import numpy as np
        rng = np.random.default_rng(0)
        x = np.linspace(-2, 8, 50)
        y = 1.0 + 2.0*x + 0.5*x**2 + rng.normal(scale=4.0, size=x.shape)
        df = pd.DataFrame({'x': x, 'y': y})
        results = train_polynomial_regression_models(
            df,
            feature_col='x',
            target_col='y',
            degrees=[2, 3, 5],
            test_size=0.2,
            random_state=42,
            display=True
        )
        best = max(results.keys(), key=lambda d: results[d]['metrics']['r2'])
        print(f'Best degree by R^2: {best} -> R^2={results[best]['metrics']['r2']:.6f}')
```

You can use the relevant part of the code in your script.

You can see that the model can generate sophisticated code using functions to create and train models with different orders and evaluate their performance for each of them.

Try to run the generated code on the testing interface with the source variable as `CPU frequency` and the target variable as `Price`.

Creating a Pipeline

Pipelines are processes containing a sequence of steps that lead to creating a trained model.

You will now use the Generative AI model to create a pipeline for performing feature scaling, creating polynomial features for multiple attributes, and performing linear regression using these variables.

You can build a similar prompt to the following code:

```
Write a very brief and compact Python code that performs the following tasks.
1. Create a pipeline that performs parameter scaling, Polynomial Feature generation, and Linear regression. Use the set of multiple features as befor
2. Calculate and display the MSE and R^2 values for the trained model.
```

The expected response is as shown below:

```
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.pipeline import Pipeline
from sklearn.preprocessing import StandardScaler, PolynomialFeatures
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error, r2_score
# Example dataset: two features (X) with a simple linear relation to y
X = np.array([[1, 2], [2, 3], [3, 4], [4, 5], [5, 6]])
y = np.array([3, 5, 7, 9, 11])
try:
    X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
    model = Pipeline([
        ('scaler', StandardScaler()),
        ('poly', PolynomialFeatures(degree=2, include_bias=False)),
        ('lr', LinearRegression())
    ])
    model.fit(X_train, y_train)
    y_pred = model.predict(X_test)
    mse = mean_squared_error(y_test, y_pred)
    r2 = r2_score(y_test, y_pred)
    print(f"MSE: {mse:.4f}, R^2: {r2:.4f}")
except Exception as e:
    print(f"Error: {e}")
```

You can use the relevant part in your code.

Make appropriate changes to the attributes you will use in the code. Consider the same set of attributes as used for multiple feature linear regression, which is `CPU_frequency`, `RAM_GB`, `Storage_GB_SSD`, `CPU_core`, `OS`, `GPU` and `Category` as source variables, and `Price` as the target variable.

Grid search and Ridge regression

An improved way to train your model is to use ridge regression instead of linear regression. You can use the polynomial features of multiple attributes. One of the key factors of ridge regression is using the parameter `alpha` as a hyperparameter for training. Using grid search, one can determine the optimum value of the hyperparameter for the given set of features. Grid search also uses cross-validation training to train and prepare the optimum model.

You can use generative AI to create the Python code to perform a grid search for the optimum ridge regression model, which uses polynomial features generated from multiple parameters.

You may use the following prompt to create this:

```
Write a very brief nad compact Python code that performs the following tasks.
1. Use polynomial features for some of the attributes of a data frame.
2. Perform Grid search on a ridge regression model for a set of values of hyperparameter alpha and polynomial features as input.
3. Use cross-validation in the Grid search.
4. Evaluate the resulting model's MSE and R^2 values.
```

The resulting code will look as shown below:

```
import numpy as np
import pandas as pd
from sklearn.model_selection import train_test_split, GridSearchCV
from sklearn.pipeline import Pipeline
from sklearn.compose import ColumnTransformer
from sklearn.preprocessing import PolynomialFeatures
from sklearn.linear_model import Ridge
from sklearn.metrics import mean_squared_error, r2_score
# Generate synthetic data (DataFrame with features A, B, C)
rng = np.random.default_rng(42)
n = 80
A = rng.normal(size=n)
B = rng.normal(size=n)
C = rng.normal(size=n)
```

```
y = 2*A - 1.5*B + 0.5*A*B + 0.3*A**2 + 0.05*C + rng.normal(scale=0.5, size=n)
X = pd.DataFrame({'A': A, 'B': B, 'C': C})
# Train-test split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
# Use polynomial features on a subset of attributes (A and B only)
poly_cols = ['A', 'B']
preprocess = ColumnTransformer(
    transformers=[
        ('poly', PolynomialFeatures(degree=2, include_bias=False), poly_cols)
    ],
    remainder='passthrough'
)
# Pipeline with Ridge regression
pipe = Pipeline([
    ('prep', preprocess),
    ('ridge', Ridge())
])
# Grid search over polynomial degree and Ridge alpha with 5-fold CV
param_grid = {
    'prep__poly__degree': [2, 3],
    'ridge__alpha': [0.1, 1.0, 10.0]
}
grid = GridSearchCV(pipe, param_grid, cv=5, scoring='neg_mean_squared_error')
try:
    grid.fit(X_train, y_train)
    best = grid.best_estimator_
    y_pred = best.predict(X_test)
    mse = mean_squared_error(y_test, y_pred)
    r2 = r2_score(y_test, y_pred)
    print(f"MSE: {mse:.4f}, R^2: {r2:.4f}")
except Exception as e:
    print(f"Error: {e}")
```

You can test this code for the data set on the testing environment.

You make use of the following parametric values for this purpose.

Source Variables: `CPU_frequency`, `RAM_GB`, `Storage_GB_SSD`, `CPU_core`, `OS`, `GPU` and `Category`
Target Variable: `Price`
Set of values for alpha: `0.0001,0.001,0.01, 0.1, 1, 10`
Cross Validation: 4-fold
Polynomial Feature order: 2

Conclusion

Congratulations! You have completed the lab on Data preparation.

With this, you have learned how to use generative AI to create Python codes that can:

- Implement linear regression in single variable
- Implement linear regression in multiple variables
- Implement polynomial regression for different orders of a single variable
- Create a pipeline that implements polynomial scaling for multiple variables and performs linear regression on them
- Apply a grid search to create an optimum ridge regression model for multiple features

Author(s)

Abhishek Gagneja