

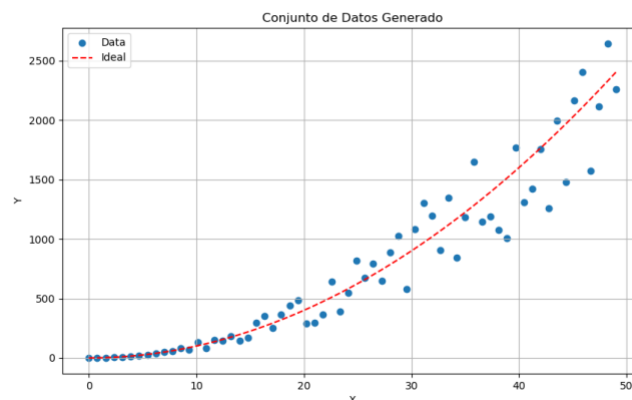
Práctica 6. Aprendizaje automático en la práctica

Datos de entrenamiento

En esta práctica se utilizarán datos artificiales obtenidos en la siguiente función proporcionada en el enunciado de la práctica `gen_data()`:

```
1 def gen_data(m, seed = 1, scale = 0.7):
2     """ generate a data set based on a  $x^2$  with added noise """
3     c = 0
4     x_train = np.linspace(0, 49, m)
5     np.random.seed(seed)
6     y_ideal = x_train ** 2 + c
7     y_train = y_ideal + scale * y_ideal * (np.random.sample((m, )) - 0.5)
8     x_ideal = x_train
9     return x_train, y_train, x_ideal, y_ideal
```

Si se muestra la gráfica:



Como el método `gen_data()` devuelve una lista unidimensional y las funciones de *scikit-learn* que se usarán necesitan arrays bidimensionales se deberá de añadir una nueva dimensión siempre que sea necesario.

Sobreajuste a los ejemplos de entrenamiento

A continuación, se muestra el código donde se realiza el sobreajuste a los ejemplos de entrenamiento junto con su respectiva explicación:

```
1 def overfitting_example():
2     x_train, y_train, x_ideal, y_ideal = gen_data(m = 64)
3     X_train, X_test, y_train, y_test = train_test_split(x_train, y_train, test_size=0.33, random_state=1)
4
5     poly = PolynomialFeatures(degree=15, include_bias=False)
6     X_train_poly = poly.fit_transform(X_train[:, np.newaxis])
7     X_test_poly = poly.transform(X_test[:, np.newaxis])
8
9     scaler = StandardScaler()
10    X_train_scaled = scaler.fit_transform(X_train_poly)
11    X_test_scaled = scaler.transform(X_test_poly)
12
13    model = LinearRegression()
14    model.fit(X_train_scaled, y_train)
15
16    y_train_pred = model.predict(X_train_scaled)
17    y_test_pred = model.predict(X_test_scaled)
18
19    mse_train = error(y_train_pred, y_train)
20    mse_test = error(y_test_pred, y_test)
21
22    print("Error on training set:", mse_train)
23    print("Error on test set:", mse_test)
24
25    plt.scatter(X_train, y_train, label='train')
26    plt.scatter(X_train, y_train_pred, label='predicted')
27    plt.plot(x_ideal, y_ideal, label='y_ideal', color='orange')
28    plt.legend()
29    plt.show()
```

Para poder generar el error del modelo tanto de los datos de entrenamiento como los datos de prueba se aplicará la siguiente fórmula:

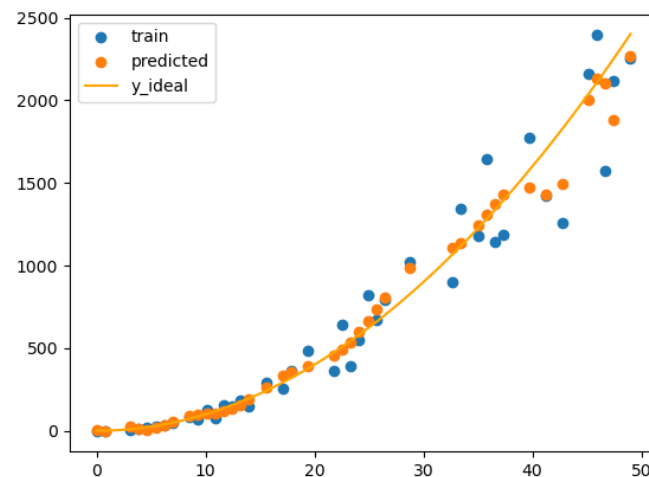
$$J(\vec{w}, b) = \frac{1}{2m} \sum_{i=1}^m (f_{\vec{w}, b}(\vec{x}^{(i)}) - y^{(i)})^2$$

Traducido a código será de la siguiente forma:

```
1 def error(predictions, labels):
2     m = len(labels)
3     mse = np.sum((predictions - labels) ** 2) / (2 * m)
4     return mse
```

Una vez realizado todo el proceso de sobreajuste se obtienen los siguientes valores de error, tanto para el entrenamiento como de prueba:

- Error on training set: 11855.0447622640992
- Error on test set: 48579.4289101656



Elección del grado de polinomio usando un conjunto de validación

Generando los datos de la misma forma que en el apartado anterior esta vez usaremos el 60% de los datos para entrenamiento, el 20% para validación y el restante 20% para prueba.

Entrenando modelos con transformaciones polinómicas de grado 1 a 10 deberíamos encontrar que el de menor error genera predicciones.

Con todo esto la función en la que se obtiene el grado óptimo del polinomio es la siguiente:

```

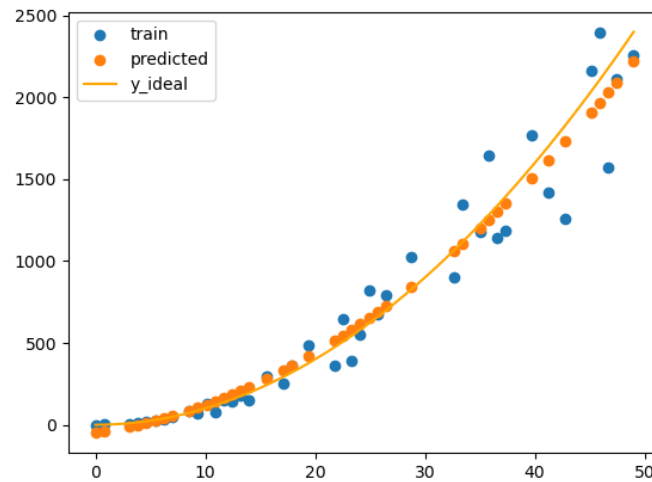
1  def select_polynomial_degree():
2      x_train, y_train, _, _ = gen_data(64)
3      X_train, X_val, y_train, y_val = train_test_split(x_train, y_train, test_size=0.2, random_state=1)
4
5      degrees = range(1, 11)
6      best_degree = None
7      min_val_error = float('inf')
8
9      for degree in degrees:
10         poly = PolynomialFeatures(degree=degree, include_bias=False)
11         X_train_poly = poly.fit_transform(X_train[:, np.newaxis])
12         X_val_poly = poly.transform(X_val[:, np.newaxis])
13
14         model = LinearRegression()
15         model.fit(X_train_poly, y_train)
16
17         y_val_pred = model.predict(X_val_poly)
18         val_error = error(y_val_pred, y_val)
19
20         if val_error < min_val_error:
21             min_val_error = val_error
22             best_degree = degree
23
24     print("Best polynomial degree:", best_degree)
25     return best_degree

```

El resultado de esto, uniéndolo con el método del apartado anterior es el siguiente:

- Best polynomial degree: 2

- Error on training set: 14877.96905472397
- Error on test set: 28209.12655628797



Elección del parámetro λ

Se utilizará una transformación polinómica de grado 15 y evaluaremos sobre el conjunto de validación cruzada los valores de λ .

Una vez explicados los valores que tendrán el grado del polinomio y el parámetro de regularización se desarrolla el método:

```

1  def select_lambda(degree = 15):
2      x_train, y_train, _, _ = gen_data(64)
3      X_train, X_val_test, y_train, y_val_test = train_test_split(x_train, y_train, test_size=0.4, random_state=1)
4      X_val, X_test, y_val, y_test = train_test_split(X_val_test, y_val_test, test_size=0.5, random_state=1)
5
6      lambdas = [1e-6, 1e-5, 1e-4, 1e-3, 1e-2, 1e-1, 1, 10, 100, 300, 600, 900]
7      best_lambda = None
8      min_test_error = float('inf')
9
10     for lambda_ in lambdas:
11         poly = PolynomialFeatures(degree=degree, include_bias=False)
12         X_train_poly = poly.fit_transform(X_train[:, np.newaxis])
13         X_val_poly = poly.transform(X_val[:, np.newaxis])
14
15         model = Ridge(alpha=lambda_)
16         model.fit(X_train_poly, y_train)
17
18         y_val_pred = model.predict(X_val_poly)
19         test_error = error(y_val_pred, y_val)
20
21         if test_error < min_test_error:
22             min_test_error = test_error
23             best_lambda = lambda_
24
25     print("Best lambda:", best_lambda)
26     return best_lambda

```

Proporcionando la mejor lambda: 1e-06

Elección de hiper-parámetros

A continuación se implementará una búsqueda exhaustiva de la mejor combinación de grado del polinomio y valor de λ para un conjunto de 750 valores, usando el 60% de los datos para entrenamiento, el 20% para validación y el restante 20% para prueba.

Probando con polinomios hasta grado 15 y valores de λ se desarrollará el método *hyperparameter_tuning()*:

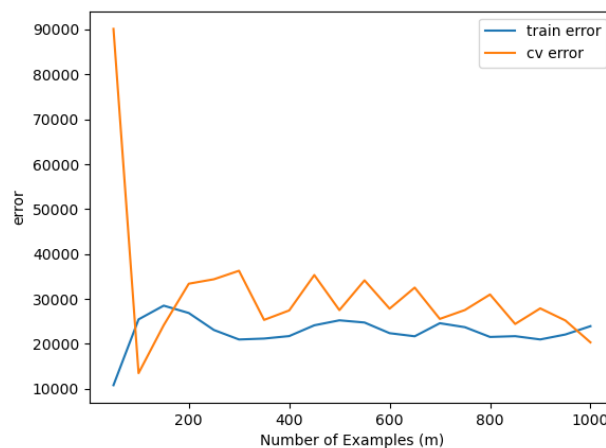
```
1  def hyperparameter_tuning():
2      x_train, y_train, _, _ = gen_data(750)
3      X_train, X_val_test, y_train, y_val_test = train_test_split(x_train, y_train, test_size=0.4, random_state=1)
4      X_val, X_test, y_val, y_test = train_test_split(X_val_test, y_val_test, test_size=0.5, random_state=1)
5
6      degrees = range(1, 16)
7      lambdas = [1e-6, 1e-5, 1e-4, 1e-3, 1e-2, 1e-1, 1, 10, 100, 300, 600, 900]
8      best_degree = None
9      best_lambda = None
10     min_test_error = float('inf')
11
12     for degree in degrees:
13         for lmbda in lambdas:
14             poly = PolynomialFeatures(degree=degree, include_bias=False)
15             X_train_poly = poly.fit_transform(X_train[:, np.newaxis])
16             X_val_poly = poly.transform(X_val[:, np.newaxis])
17
18             model = Ridge(alpha=lmbda)
19             model.fit(X_train_poly, y_train)
20
21             y_val_pred = model.predict(X_val_poly)
22             test_error = error(y_val_pred, y_val)
23
24             if test_error < min_test_error:
25                 min_test_error = test_error
26                 best_degree = degree
27                 best_lambda = lmbda
28
29     print("Best polynomial degree:", best_degree)
30     print("Best lambda:", best_lambda)
31     print('Error', min_test_error)
32     return best_degree, best_lambda
```

Proporcionando el siguiente resultado:

- Best polynomial degree: 4
- Best lambda: 1e-06
- Error 28288.38408286469

Curvas de aprendizaje

Entrenaremos un modelo con polinomios de grado 16 sin regularizar con conjuntos crecientes de datos de entrenamiento, generando entre 50 y 1000 ejemplos y usando en cada caso el 60% de los datos para entrenamiento, el 20% para validación y calculando en cada caso el error de entrenamiento y el de validación para generar una gráfica:



El código será el siguiente:

```
1 def learning_curves():
2     train_sizes = np.arange(50, 1001, 50)
3     train_errors = []
4     val_errors = []
5
6     for size in train_sizes:
7         x_train, y_train, _, _ = gen_data(size)
8         X_train, X_val, y_train, y_val = train_test_split(x_train, y_train, test_size=0.2, random_state=1)
9
10        poly = PolynomialFeatures(degree=16, include_bias=False)
11        X_train_poly = poly.fit_transform(X_train[:, np.newaxis])
12        X_val_poly = poly.transform(X_val[:, np.newaxis])
13
14        model = LinearRegression()
15        model.fit(X_train_poly, y_train)
16
17        y_train_pred = model.predict(X_train_poly)
18        y_val_pred = model.predict(X_val_poly)
19
20        train_error = error(y_train_pred, y_train,)
21        val_error = error(y_val_pred, y_val)
22
23        train_errors.append(train_error)
24        val_errors.append(val_error)
25
26    plt.plot(train_sizes, train_errors, label='train error')
27    plt.plot(train_sizes, val_errors, label='cv error')
28    plt.xlabel('Number of Examples (m)')
29    plt.ylabel('error')
30    plt.legend()
31    plt.show()
```