Lab 4 – Overfitting and underfitting.Model evaluation metrics

Author: Gonçalo Aguiar 904475

1. **Analyze the overfitting.py program. Identify variables corresponding to the number of training samples, the level of "noise" causing the samples to deviate from the actual distribution, and the degrees of the polynomial. What are the results of different changes to these parameters?**

```
n_samples = 30
degrees = [1, 4, 15]
X = np.sort(np.random.rand(n_samples))
noise = 0.1
y = true_fun(X) + np.random.randn(n_samples) * noise
```

According to the last image:

- **Number of training samples**--> n_samples(int). If we increase the number of samples, the models will have more data to learn from, which can lead to better generalization performance.

- **Level of "noise"** --> noise(double). When the level of noise in the data is increased, it adds more variability and randomness to the samples, making it more challenging for the models to learn from the samples.

- **Degrees of the polynomial**--> degrees(array). By using higher degrees of polynomial will result in more complex models that can fit the data more closely, but they also increase the risk of overfitting. On the other hand, using lower degrees of polynomial will result in simpler models that may not capture the underlying patterns in the data and can lead to underfitting.

2. **Write a program for classification of the Iris dataset (binary classification for two arbitrary classes, arbitrary classifier). Evaluate the classifier on the test set using the following metrics: the confusion matrix, accuracy, F1-score, Jaccarda coefficient.**

In order to complete the task the following code was written:

```
from sklearn import datasets
import matplotlib.pyplot as plt
import numpy as np
from sklearn.linear_model import LogisticRegression
from sklearn.svm import SVC

iris = datasets.load_iris()
X = iris.data
y = iris.target

x_min, x_max = X[:, 0].min() - 0.5, X[:, 0].max() + 0.5
y_min, y_max = X[:, 1].min() - 0.5, X[:, 1].max() + 0.5

fig, ax = plt.subplots()
scatter = ax.scatter(X[:, 0], X[:, 1], c=y, cmap=plt.cm.Set1, edgecolor="k")
plt.xlabel("sepal length")
plt.ylabel("sepal width")

legend1 = ax.legend(*scatter.legend_elements(),loc="lower right", title="Class")
ax.add_artist(legend1)

plt.xlim(x_min, x_max)
plt.ylim(y_min, y_max)

# indices 0-49, 50-99, 100-149 in random order
random0 = np.random.choice(np.arange(0,50),50,replace=False)
random1 = np.random.choice(np.arange(50,100),50,replace=False)
random2 = np.random.choice(np.arange(100,150),50,replace=False)

# take 80% (40 samples) of each class to the training set

X0 = X[random0[:40],:]
X1 = X[random1[:40],:]
X2 = X[random2[:40],:]
```

```
# take the corresponding labels
y0 = y[random0[:40]]
y1 = y[random1[:40]]
y2 = y[random2[:40]]

# take 20% (10 samples) of each class to the test set
X0_test = X[random0[40:],:]
X1_test = X[random1[40:],:]
X2_test = X[random2[40:],:]

# take the corresponding labels
y0_test = y[random0[40:]]
y1_test = y[random1[40:]]
y2_test = y[random2[40:]]

# compose the training set and the test set - just features 0 and 1 and classes 0 and 1
X01_train = np.concatenate([X0[:,0:2], X1[:,0:2]])
y01_train = np.concatenate([y0, y1])
X01_test = np.concatenate([X0_test[:,0:2], X1_test[:,0:2]])
y01_test = np.concatenate([y0_test, y1_test])

# linear binary classification with a logistic regression model
clf = SVC(kernel='linear').fit(X01_train, y01_train)

y_true = y01_test
y_pred = clf.predict(X01_test)

from sklearn.metrics import accuracy_score
print(accuracy_score(y_true, y_pred)  )

from sklearn.metrics import f1_score
print(f1_score(y_true, y_pred)  )

from sklearn.metrics import confusion_matrix
print(confusion_matrix(y_true, y_pred)  )

from sklearn.metrics import jaccard_score
print(jaccard_score(y_true, y_pred)  )
```

The result was:

```
Accuracy Score-> 1.0
F1 Score-> 1.0
Confusion Matrix->
[[10  0]
 [ 0 10]]
Jaccard Score-> 1.0
```

As we can see the classification model achieved perfect scores and the confusion matrix shows us that there were no false positives or false negatives and that all the samples were correctly classified.

3. **Implement linear regression for prediction of sepal width based on sepal length. Then, evaluate the regressor on the test set using the following metrics: mean squared error (MSE), mean absolute error (MAE), explained variance score, determination coefficient R^2 .**

   In order to complete the task the following code was written:

```
from sklearn import datasets
import matplotlib.pyplot as plt
import numpy as np

iris = datasets.load_iris()
X = iris.data
Y = iris.target

X = X[Y==0][:,0:2]
Y = Y[Y==0]


random0 = np.random.choice(np.arange(0,50),50, replace=False)

train_inds = random0[:40]
test_inds = random0[40:]

x_min,x_max = X[:,0].min() - 0.5 , X[:,0].max() +0.5
y_min,y_max = X[:,1].min() - 0.5 , X[:,1].max() +0.5


fig, ax = plt.subplots()

scatter = ax.scatter(X[train_inds, 0], X[train_inds, 1])
scatter = ax.scatter(X[test_inds, 0], X[test_inds, 1])
plt.xlabel("sepal length")

plt.ylabel("sepal width")

plt.legend([ "Training examples", "Test examples"])
plt.xlim(x_min, x_max)

plt.ylim(y_min, y_max)

##------------------------------------------------------------------------

from sklearn import linear_model
regr = linear_model.LinearRegression()
regr.fit(X[train_inds,0:1],X[train_inds,1:])
```

```
# parameters of the regression line: slope and intercept
print(regr.coef_, regr.intercept_)

# draw the regression line .
line = np.arange(x_min,x_max+1)*regr.coef_[0] + regr.intercept_[0]
plt.plot(np.arange(x_min,x_max+1),line, 'r')

predictions = regr.predict(X[train_inds,0:1])

from sklearn.metrics import mean_squared_error
print("\nMean Squared Error->",mean_squared_error(predictions, X[test_inds,1:]))

from sklearn.metrics import mean_absolute_error
print("\nMean Absolute Error->",mean_absolute_error(predictions, X[test_inds,1:]))


from sklearn.metrics import explained_variance_score
print("\nExplained Variance Score->",explained_variance_score(predictions, X[test_inds,1:]))


from sklearn.metrics import r2_score
print("\nR2 Score->",r2_score(predictions, X[test_inds,1:]) )
```

The result was:

```
Mean Squared Error-> 0.06652028216323284

Mean Absolute Error-> 0.20845605924807725

Explained Variance Score-> -0.8444773054425463

R2 Score-> -0.8950706726468494
```

As we can see the linear regression model has a low mean squared error and a low mean absolute error. This indicates that the model is making reasonably accurate preditions. However, the negative values in the other two scores indicate that the model is not performing well regarding the variance in the test data probably because the linear model may not be the best one for the data that was given.