Practical - Overfitting & Underfitting

- Shivam Tawari A-58

Aim: Write a python program to evaluate a overfitting and underfitting on dataset.

Theory:

Bias and variance are two terms you need to get used to if constructing statistical ~~methods~~ models, such as those in machine learning.

There is a tension between wanting to construct a model which is complex enough to capture the system that we are modelling, but not so complex that we start to fit to noise in the training data.

Underfitting:
If we have an underfitted model, this means that we do not have enough parameters to capture the trends in the underlying system. For example, we have data that is parabolic in nature, but we try to fit this with linear function, with just one parameter.

Because the function does not have the required complexity to fit the data (two parameters), we end up with a poor predictor. In this case, the model will have high bias. This means that we will get consistent answers, but consistently wrong.

Overfitting:
If we have, this means that we have too many parameters to be justified by the actual underlying data and therefore build an overly complex model. Imagine that the true system is a parabola, but we used a higher order polynomial to fit to it. Because we have natural noise in the data used to it. The result is a model that has high variance. This means that we will not get consistent predictions of future results.

Conclusion: Hence, we have successfully performed a python program to evaluate a overfitting and underfitting.

Code:

```
[1]  # Shivam Tawari A-58
     import pandas as pd
     import numpy as np
     import matplotlib.pyplot as plt
     from sklearn.ensemble import RandomForestClassifier
     from sklearn.model_selection import GridSearchCV
     from sklearn.model_selection import cross_val_score, learning_curve, validation_curve
```

```
[2]  import matplotlib as mpl
     mpl.rcParams['xtick.labelsize'] = 22
     mpl.rcParams['ytick.labelsize'] = 22
     mpl.rcParams['figure.figsize'] = (10, 8)
     mpl.rcParams['axes.facecolor'] = (0.9,0.9,0.9)
     mpl.rcParams['lines.linewidth'] = 2
     mpl.rcParams['axes.grid'] = True
     mpl.rcParams['grid.color'] = 'w'
     mpl.rcParams['xtick.top'] = True
     mpl.rcParams['ytick.right'] = True
     mpl.rcParams['grid.linestyle'] = '--'
     mpl.rcParams['legend.fontsize'] = 22
     mpl.rcParams['legend.facecolor'] = [1,1,1]
     mpl.rcParams['legend.framealpha'] = 0.75
     mpl.rcParams['axes.labelsize'] = 22
```

```
[3]  df_train = pd.read_csv('train.csv')
     df_test = pd.read_csv('test.csv')
```

```
[3]  df_comb = df_train.append(df_test)
     x = pd.DataFrame()
```

```
[4]  def encode_sex(x):
       return 1 if x=='female' else 0

     def family_size(x):
       size = x.SibSp + x.Parch
       return 4 if size > 3 else size

     x['Sex'] = df_comb.Sex.map(encode_sex)
     x['Pclass'] = df_comb.Pclass
     x['FamilySize'] = df_comb.apply(family_size, axis=1)
```

```
[5]  fare_median = df_train.groupby(['Sex','Pclass']).Fare.median()
     fare_median.name = 'FareMedian'
     age_mean = df_train.groupby(['Sex','Pclass']).Age.mean()
     age_mean.name = 'AgeMean'

     def join(df, stat):
       return pd.merge(df, stat.to_frame(), left_on=['Sex','Pclass'], right_index=True, how='left')

     x['Fare'] = df_comb.Fare.fillna(join(df_comb, fare_median).FareMedian)
     x['Age'] = df_comb.Age.fillna(join(df_comb, age_mean).AgeMean)
```

```
[6]  x.head()
```

[6]

	Sex	Pclass	FamilySize	Fare	Age
0	0	3	1	7.2500	22.0
1	1	1	1	71.2833	38.0
2	1	3	0	7.9250	26.0
3	1	1	1	53.1000	35.0
4	0	3	0	8.0500	35.0

[7]
```
def quantiles(series, num):
  return pd.qcut(series, num, retbins=True)[1]

def discretize(series, bins):
  return pd.cut(series, bins, labels=range(len(bins)-1), include_lowest=True)

x['Fare'] = discretize(x.Fare, quantiles(df_comb.Fare, 10))
x['Age'] = discretize(x.Age, quantiles(df_comb.Age, 10))
```

[8]
```
x_train = x.iloc[:df_train.shape[0]]
x_test = x.iloc[df_train.shape[0]:]
x_train
```

	Sex	Pclass	FamilySize	Fare	Age
0	0	3	1	0	2

[9]
```
y_train = df_train.Survived
y_train
```

```
0      0
1      1
2      1
3      1
4      0
      ..
886    0
887    1
888    0
889    1
890    0
Name: Survived, Length: 891, dtype: int64
```

[10]
```
clf_1 = RandomForestClassifier(n_estimators=100, bootstrap=True, random_state=0)
clf_1.fit(x_train, y_train)
num_folds = 7
```

[11]
```
def plot_curve(ticks, train_scores, test_scores):
  train_scores_mean = -1*np.mean(train_scores,axis=1)
  train_scores_std = -1*np.std(train_scores,axis=1)
  test_scores_mean = -1*np.mean(test_scores,axis=1)
  test_scores_std = -1*np.std(test_scores,axis=1)
  plt.figure()
  plt.fill_between(ticks,
                   train_scores_mean - train_scores_std,
                   train_scores_mean + train_scores_std, alpha = 0.1,
```

```
[11]    plt.fill_between(ticks,
                         test_scores_mean - test_scores_std,
                         test_scores_mean + test_scores_std, alpha = 0.1,
                         color = "r")
        plt.plot(ticks, train_scores_mean, 'b-', label='Training Score')
        plt.plot(ticks, test_scores_mean, 'r-', label='Test Score')
        plt.legend(fancybox=True, facecolor='w')
        return plt.gca()
```

```
def plot_validation_curve(clf, x, y, param_name, param_range, scoring='roc_auc'):
  plt.xkcd()
  ax = plot_curve(param_range, *validation_curve(clf,x,y,cv=num_folds,scoring=scoring,
                                                 param_name=param_name,param_range=param_range,
                                                 n_jobs=-1))
  ax.set_title('')
  ax.set_xticklabels([])
  ax.set_yticklabels([])
  ax.set_xlim(2,12)
  ax.set_ylim(-0.97, -0.83)
  ax.set_ylabel('Error')
  ax.set_xlabel('Model Complexity')
  ax.text(9, -0.94, 'Overfitting', fontsize=22)
  ax.text(3, -0.94, 'Underfitting', fontsize=22)
  ax.axvline(7, ls='--')
  plt.tight_layout()

plot_validation_curve(clf_1, x_train, y_train,
                      param_name='max_depth', param_range=range(2,13))
```