COMP4220: Machine Learning, Spring 2021, Assignment 3

Due: Wednsday, March 10, 11pm

Please submit one pdf file for all questions.

You can type your answer for the first two questions in the below cell of each question using "Markdown" option!

**When turning in assignments after the due date, please clearly specify the number of late hours used.

1. Suppose you are using Polynomial Regression. You plot the learning curves and you notice that there is a large gap between the training error and the validation error. What is happening? What are three ways to solve this?

In this instance, the model will be overfitting the training set of data, this is shown since the validation error is greater thanthe training error. There are a few ways that this can be fixed. For starters, you can add a penalty to the cost function in an attempt to regularize your model.A greater training set may also solve the issue. You can also reduce the degree of your polynomial to reduce overfitting.

2. How does the Stochastic Gradient Descent

algorithm work?

Regular gradient descent iteratively starts at a point on a function and travels down the slope until it finds a slope of 0 (minimum value). Stochastic gradient descent introduces a random element to the gradient descent algorithm. In the phase of selecting data points, this algorithm picks a data point from the entire set each iteration.

3. Programming assignment (Linear Regression)

In [1]:
```
import matplotlib.pyplot as plt
import numpy as np
from sklearn import datasets, linear_model
```

Create a two-dimensional plot of linear regression using one feature of the diabetes dataset

In [21]:
```
# Load the diabetes dataset
diabetes_X, diabetes_y = datasets.load_diabetes(return_X_y=True)
```

P3.1: Find the number of data instances(# of rows) and features(# of columns) in this data set

(Show the result using print function)

In [26]:
```
print ("Number of X is %s", (len(diabetes_X)))
print ("Number of Y is %s", (len(diabetes_y)))
```

```
Number of X is %s 442
Number of Y is %s 442
```

Using just one feature and splitting dataset into training/testing sets

In [27]:
```
# Use only one feature
diabetes_X = diabetes_X[:, np.newaxis, 2]
diabetes_X.shape
```

Out[27]:
```
(442, 1)
```

In [28]:
```
# Split the data into training/testing sets
diabetes_X_train = diabetes_X[:-20]
diabetes_X_test = diabetes_X[-20:]
```

In [29]:
```
# Split the targets into training/testing sets
diabetes_y_train = diabetes_y[:-20]
diabetes_y_test = diabetes_y[-20:]
```

P3.2: Create linear regression object

In [32]:
```
regr = linear_model.LinearRegression()
```

P3.3: Train the model using the training sets

In [33]:
```
regr.fit(diabetes_X_train, diabetes_y_train)
```

Out[33]:
```
LinearRegression()
```

P3.4: Make predictions using the testing set

In [34]:
```
diabetes_y_pred = regr.predict(diabetes_X_test)
print(diabetes_y_pred)
```

```
[225.9732401  115.74763374 163.27610621 114.73638965 120.80385422
 158.21988574 236.08568105 121.81509832  99.56772822 123.83758651
 204.73711411  96.53399594 154.17490936 130.91629517  83.3878227
 171.36605897 137.99500384 137.99500384 189.56845268  84.3990668 ]
```

P3.5: Find the coefficients and intercept (Showing the result using print function)

In [35]:
```
coefficients = regr.coef_
intercept = regr.intercept_
print('The coefficient is {} and the interception is {}'.format(coeffic
ients[0],intercept))
```

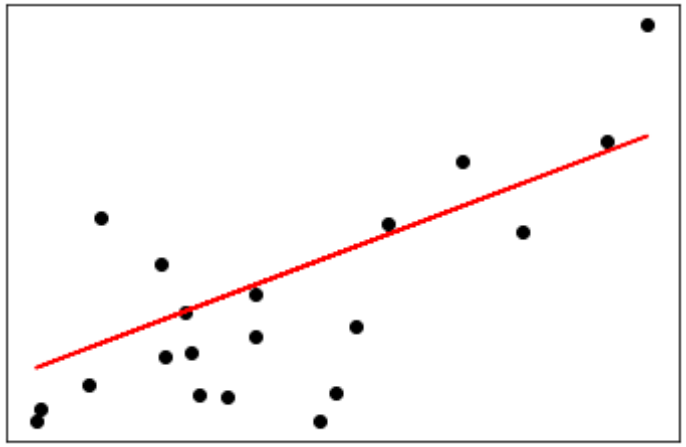
```
The coefficient is 938.2378612512633 and the interception is 152.918861
82616167
```

P3.6: Plot the test data points using plt.scatter and draw "Line of Best Fit" using plt.plot()

Make sure that you label both axes with appropriate names

In [81]:
```
plt.scatter(diabetes_X_test, diabetes_y_test, color = 'black')
plt.plot(diabetes_X_test, diabetes_y_pred, color = 'red')
```

```
plt.xticks(())
plt.yticks(())
plt.show()
```



P3.7: We like to measure the generalization error for the trained model. You can find a wide range of evaluation

metrics at https://scikit-learn.org/stable/modules/model_evaluation.html

Pick only one suitable evaluation metric from this list and report the generalization error using the test data set

In [98]:
```
from sklearn.svm import SVC
clf = regr.fit(diabetes_X_train, diabetes_y_train)
```

```
clf.score(diabetes_X_test, diabetes_y_test)
```

Out[98]:
```
0.47257544798227136
```

4. Programming assignment (Polynomial Regression)

Polynomial Regression is appropriate when data are nonlinearly distributed (meaning you can't fit a straight line between y and x).

Polynomial Regression is linear with respect to coefficients since we don't have any power of the coefficients (all the coefficients are raised to the power of 1). However, Polynomial Regression is a nonlinear function of the input x, since we have the inputs raised to several powers.

In [37]:
```
# Importing the libraries
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
```

In [38]:
```
# Importing the dataset
dataset = pd.read_csv('Position_Salaries.csv') #the dataset is availabl
e in the folder of HW3 on Blackboard
X = dataset.iloc[:, 1:-1].values
y = dataset.iloc[:, -1].values
```

In [39]:
```
# Training the Linear Regression model on the whole dataset
from sklearn.linear_model import LinearRegression
lin_reg = LinearRegression()
lin_reg_1 = lin_reg.fit(X, y)
y_pred1 = lin_reg_1.predict(X)
y_pred1
```

Out[39]:
```
array([-114454.54545455,  -33575.75757576,   47303.03030303,
        128181.81818182,  209060.60606061,  289939.39393939,
```

```
370818.18181818,  451696.96969697,  532575.75757576,
613454.54545455])
```

P4.1: Training the Polynomial Regression model on the whole dataset (degree=3)

In [42]:
```
from sklearn.preprocessing import PolynomialFeatures
# Use "PolynomialFeatures" to generate polynomial features
regression1 = PolynomialFeatures(degree = 3)
# Solve the linear regression model using the polynomial features
regression1_X = regression1.fit_transform(X)
regression1.fit(regression1_X, y)
lin_reg_2 = LinearRegression()
lin_reg_2.fit(regression1_X, y)
```

Out[42]:
```
LinearRegression()
```

P4.2: Training the Polynomial Regression model on the whole dataset (degree=4) as above and call the model lin_reg_4

In [43]:
```
# Use "PolynomialFeatures" to generate polynomial features
regression2 = PolynomialFeatures(degree = 4)
# Solve the linear regression model using the polynomial features
regression2_X = regression2.fit_transform(X)
regression2.fit(regression2_X, y)
lin_reg_4 = LinearRegression()
lin_reg_4.fit(regression2_X, y)
```

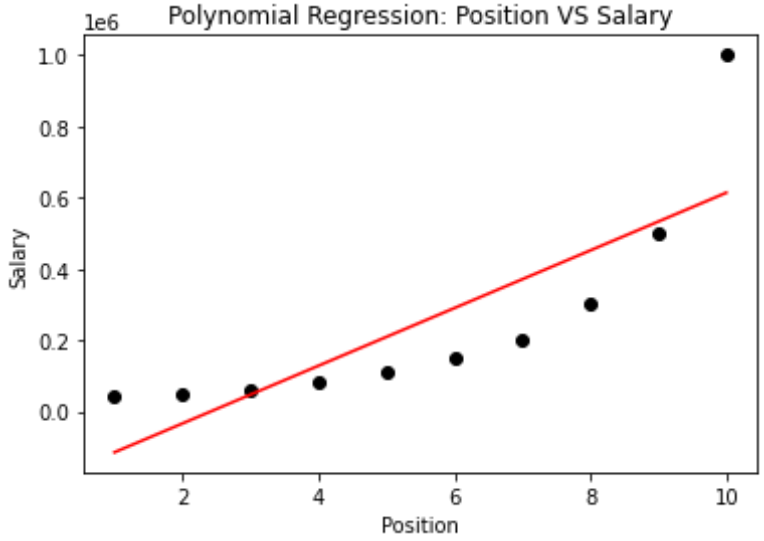
Out[43]:
```
LinearRegression()
```

P4.3: Visualising the Linear Regression

results

Name both axes and don't forget to use "title" as we have several plots

In [53]:
```
plt.scatter(X, y, color = 'black')
plt.plot(X, lin_reg_1.predict(X), color = 'red')
plt.xlabel('Position')
plt.ylabel('Salary')
plt.title('Polynomial Regression: Position VS Salary')
plt.show()
```



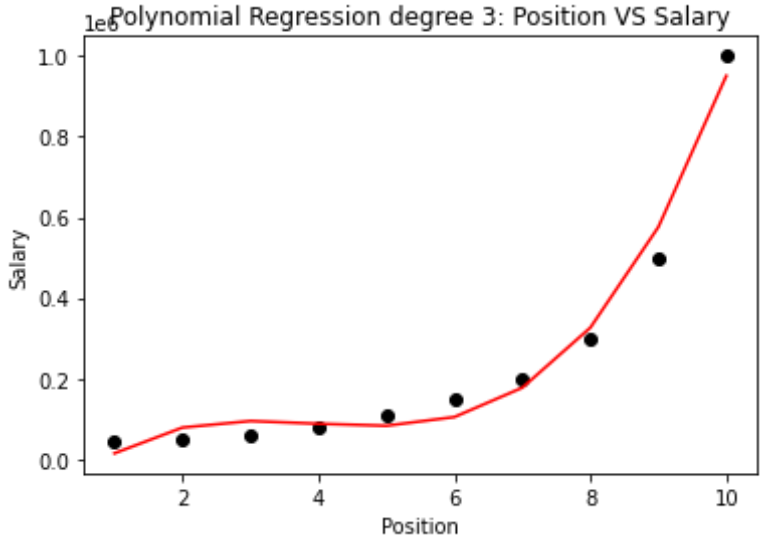
How do we find the best degree?

The main form of finding a good fit is to plot the model and see what it looks like visually. You simply test several degrees and you see which one gives you the best fit. The other option is to find the lowest root-mean-square error (RMSE) for your model, but in that case be careful not to overfit the data.

P4.4: Visualising the Polynomial Regression results (degree = 3)

Name both axes and don't forget to use "title" as we have several plots

In [54]:
```
plt.scatter(X, y, color = 'black')
plt.plot(X, lin_reg_2.predict(regression1.fit_transform(X)), color = 'r
ed')
plt.xlabel('Position')
plt.ylabel('Salary')
plt.title('Polynomial Regression degree 3: Position VS Salary')
plt.show()
```

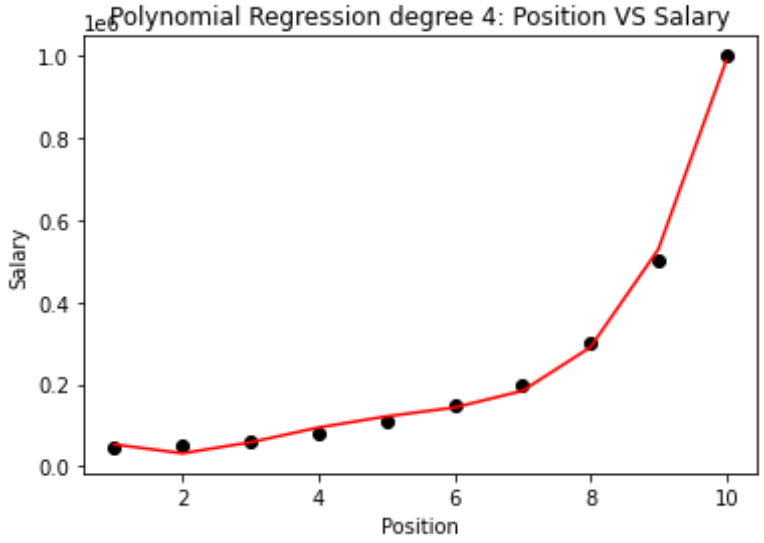


P4.5: Visualising the Polynomial Regression

results (degree = 4)

Name both axes and don't forget to use "title" as we have several plots

In [56]:
```
plt.scatter(X, y, color = 'black')
plt.plot(X, lin_reg_4.predict(regression2.fit_transform(X)), color = 'r
ed')
plt.xlabel('Position')
plt.ylabel('Salary')
plt.title('Polynomial Regression degree 4: Position VS Salary')
plt.show()
```



P4.6: Predicting a new result with Linear Regression when the input is x=19

In [60]:
```
y_with_19_years = lin_reg_1.predict([[19]])
```

```
print(y_with_19_years)
```

```
[1341363.63636364]
```

P4.7: Predicting a new result with Polynomial Regression(degree=3) when the input is x=19

In [62]:
```
y_with_19_years_3d = lin_reg_2.predict(regression1.fit_transform([[19
]]))
print(y_with_19_years_3d)
```

```
[14044517.48251703]
```

P4.8: Predicting a new result with Polynomial Regression(degree=4) when the input is x=19

In [64]:
```
y_with_19_years_4d = lin_reg_4.predict(regression2.fit_transform([[19
]]))
print(y_with_19_years_4d)
```

```
[40328199.30071028]
```

P4.9: Please explain the impact of selecting an appropriate degree parameter on generalization error based on the above values.

If the network works well in its training but does not predict well there is a good chance that it is overfitting whereas underfitting may occur if the bias is large but the variance is small. Upping

the degree polynomial can solve underfitting while reducing the degree polynomial can solve overfitting. In the degree 4 polynomial it appears to be overfitted.