EX4

September 29, 2021

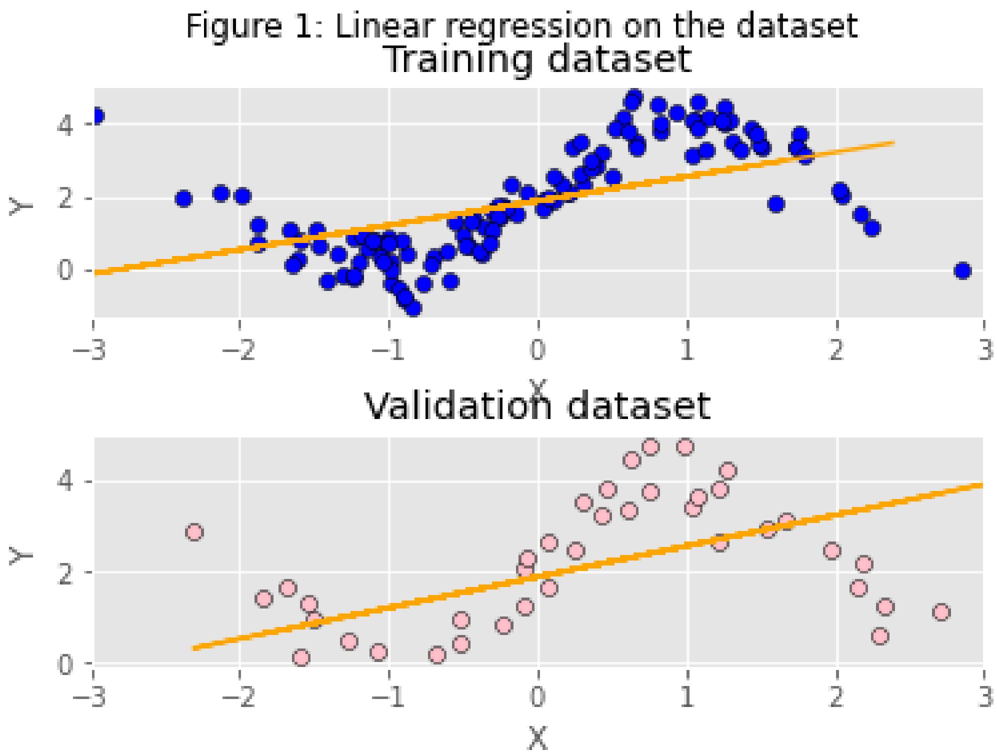
1 Exercise 1

[1]:
```
from sklearn.metrics import mean_squared_error as MSE, accuracy_score
from sklearn.neighbors import KNeighborsClassifier
from sklearn.preprocessing import PolynomialFeatures
import pandas as pd
import numpy as np
from sklearn import datasets
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn import metrics
import seaborn as sns
from matplotlib.colors import ListedColormap
import warnings
import matplotlib.pyplot as plt
import matplotlib.lines as mlines
from sklearn.feature_selection import SelectKBest
from sklearn.model_selection import cross_val_score
from sklearn.linear_model import LogisticRegression
from sklearn.pipeline import make_pipeline
import plotly.graph_objects as go
import plotly.express as px
import plotly.io as pio
from sklearn.pipeline import make_pipeline
from sklearn import preprocessing
from sklearn import svm
pio.renderers.default = "notebook+pdf"
```

[2]:
```
warnings.simplefilter(action="ignore", category=FutureWarning)

# EXERCISE 1
data = pd.read_csv('regression_nonlin.csv')
plt.style.use('ggplot')

# Let's say we want to split the data in 60:20:20 for train:valid:test dataset
train_size = 0.6
```

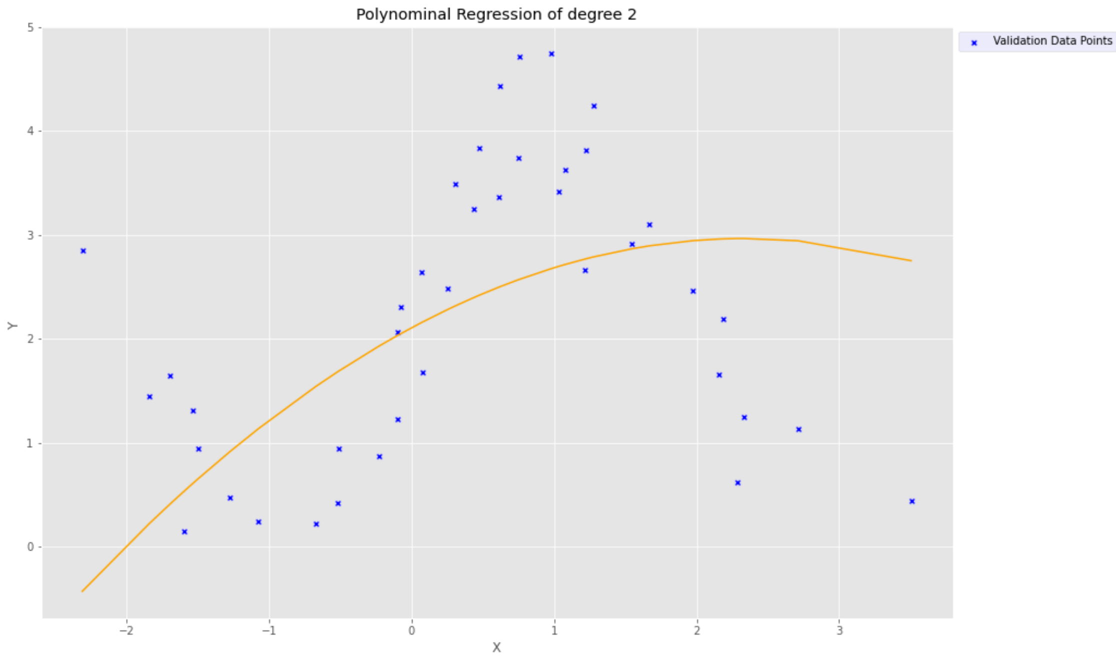
```python
x = np.array(data.X).reshape((-1, 1))
y = np.array(data.y).reshape((-1, 1))

# Validation set is different from test set. Validation set actually can be␣
↪regarded as a part of training se
# In the first step we will split the data in training and remaining dataset
X_train, X_rem, y_train, y_rem = train_test_split(x, y, train_size=0.6)

# Now since we want the valid and test size to be equal (10% each of overall␣
↪data).
# we have to define valid_size=0.5 (that is 50% of remaining data)
test_size = 0.5
X_valid, X_test, y_valid, y_test = train_test_split(
    X_rem, y_rem, test_size=0.5, random_state=20)

# Fit the model over the training dataset
model = LinearRegression()
model.fit(X_train, y_train)

fig = plt.figure()

plt.subplots_adjust(wspace=0.2, hspace=0.5)

fig.suptitle('Figure 1: Linear regression on the dataset')

ax1 = fig.add_subplot(211)
ax1.scatter(X_train, y_train,  color='blue', edgecolors='black')
ax1.set_title('Training dataset')
ax1.set_ylabel("Y")
ax1.set_xlabel("X")
ax1.set_xlim(-3, 3)
plt.plot(X_test, model.predict(X_test), color='orange')


ax2 = fig.add_subplot(212)
ax2.scatter(X_valid, y_valid,  color='pink',
            edgecolors='black', label='Oil&Gas')
ax2.set_title('Validation dataset')
ax2.set_ylabel("Y")
ax2.set_xlabel("X")
ax2.set_xlim(-3, 3)
plt.plot(X_valid, model.predict(X_valid), color='orange')

y_pred_test = model.predict(X_test)
y_pred_valid = model.predict(X_valid)
```

```
# compute the Mean Square Error on both datasets.

test_MSE = metrics.mean_squared_error(y_test, y_pred_test)
valid_MSE = metrics.mean_squared_error(y_valid, y_pred_valid)
```

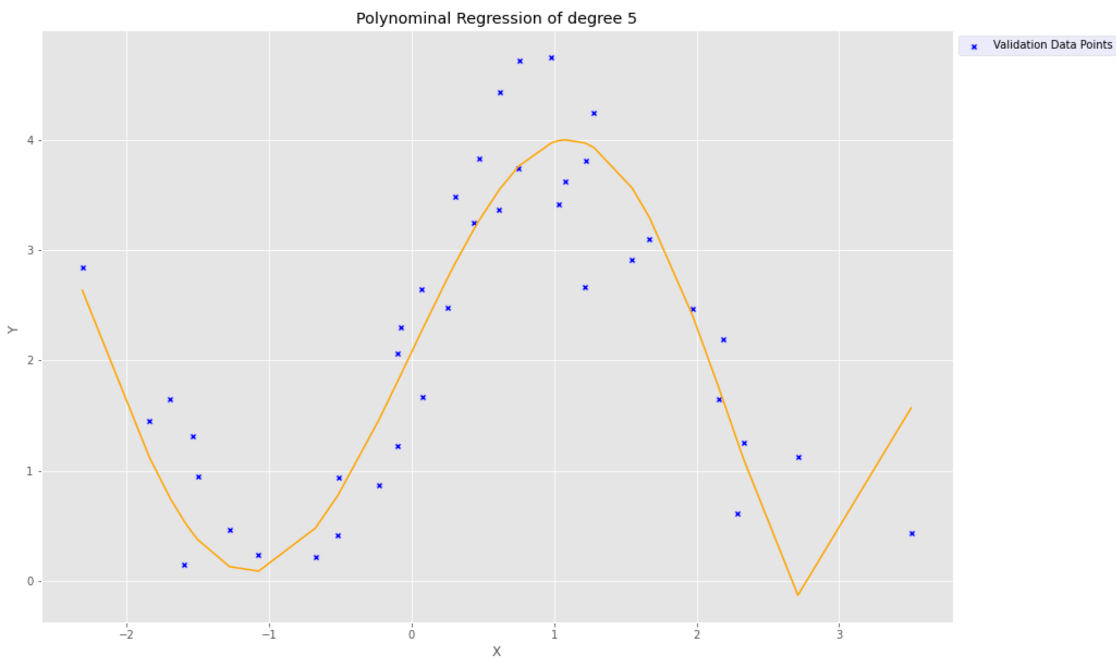


[3]:
```
# Importing the dataset

plt.style.use('ggplot')
df = pd.DataFrame()
degrees = ([2, 5, 10, 20, 25])

summary = pd.DataFrame()
summary['index name'] = ["Validation", "Train"]
summary = pd.DataFrame(summary.set_index('index name'))

X_train = np.array(X_train).reshape(-1, 1)
X_valid = np.array(X_valid).reshape(-1, 1)

# Visualizing the Polymonial Regression results
def viz_polymonial(deg):
    poly_X_train = PolynomialFeatures(deg).fit_transform(X_train)
    poly_X_valid = PolynomialFeatures(deg).fit_transform(X_valid)
```

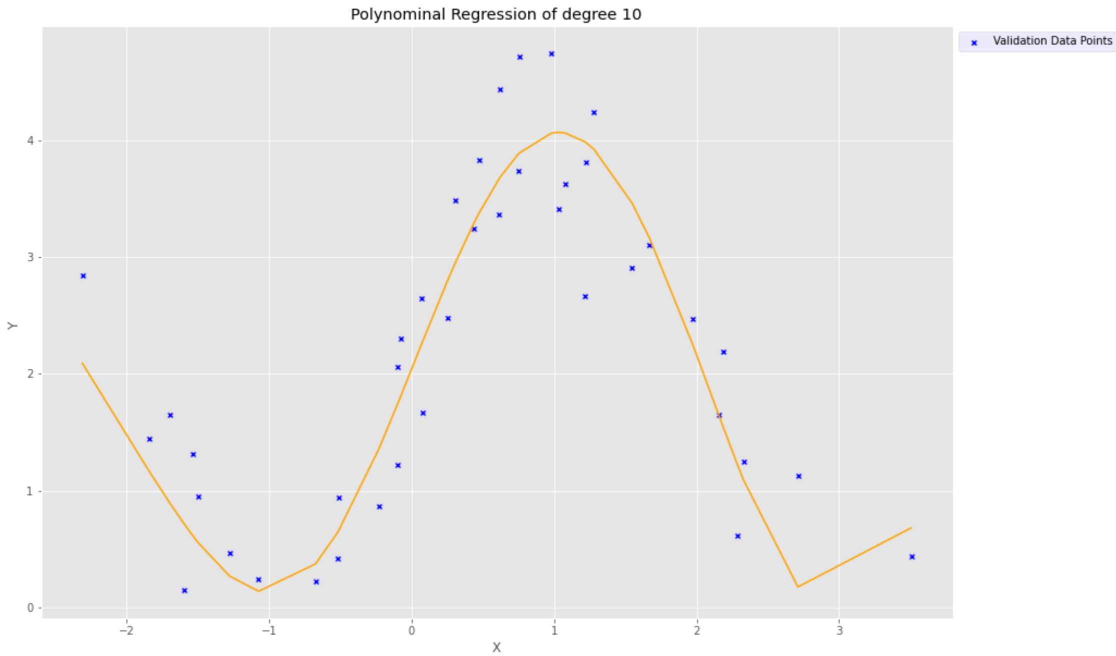
```
    poly = LinearRegression().fit(poly_X_train, y_train)
    y_train_poly = poly.predict(poly_X_train)
    y_valid_poly = poly.predict(poly_X_valid)
    plt.figure(figsize=(15, 10))
    X = X_valid[:, 0]
    Y = y_valid[:, 0]

    sns.scatterplot(x=X, y=Y, color='blue', edgecolors='blue',
                    marker="X", label="Validation Data Points")
    sns.lineplot(X_valid[:, 0], y_valid_poly[:, 0],  color='orange')

    # ERROR
    summary[str(deg) + " MSE"] = [metrics.mean_squared_error(y_valid,␣
 ↪y_valid_poly),
                                  metrics.mean_squared_error(y_train,␣
 ↪y_train_poly)]
    plt.title('Polynominal Regression of degree ' + str(deg))
    plt.xlabel('X')
    plt.ylabel('Y ')
    plt.legend(bbox_to_anchor=(1, 0.8, 0.3, 0.2),
               loc='upper left', facecolor='lavender')
    plt.show()
    return


degrees = ([2, 5, 10, 20, 25])

for i in range(0, len(degrees)):
    viz_polymonial(degrees[i])

summary['Linear'] = [valid_MSE, test_MSE]
```

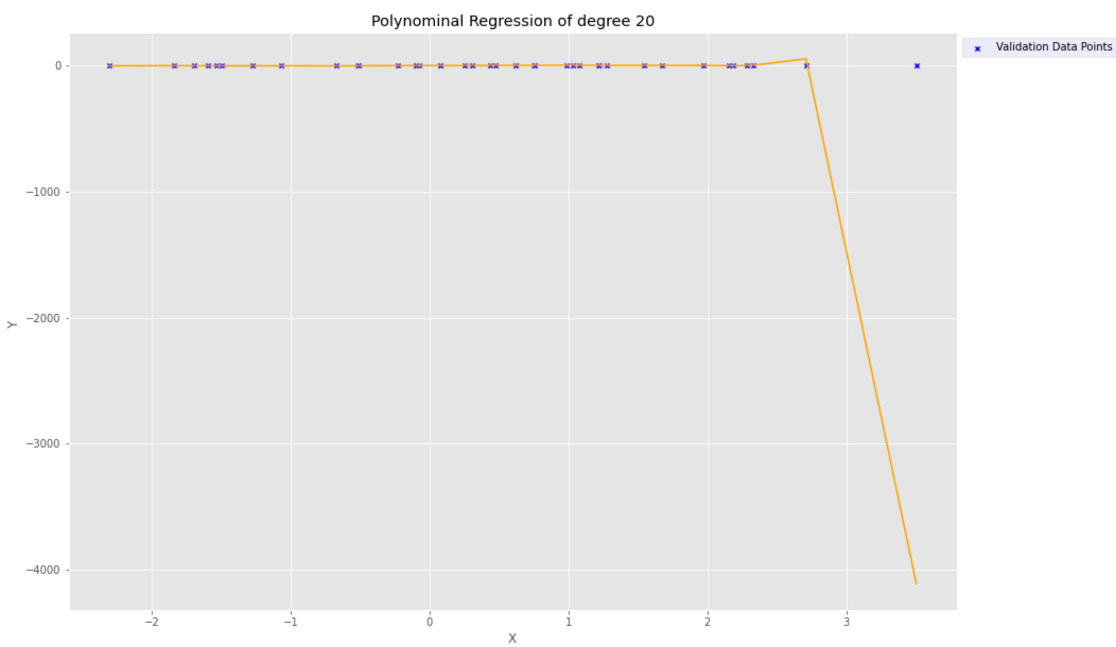


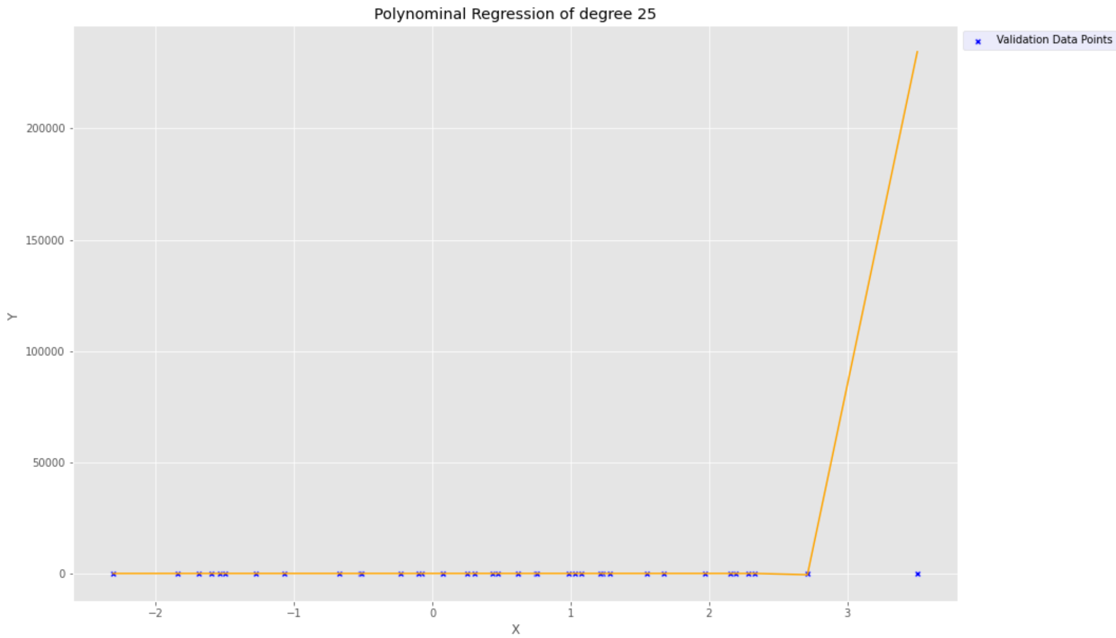
Polynominal Regression of degree 2
Validation Data Points
Y
X



Polynominal Regression of degree 5
Validation Data Points
Y
X


Polynominal Regression of degree 10
Validation Data Points
Y
X


Polynominal Regression of degree 20
Validation Data Points
Y
X



[4]:
```
summary
```

[4]:
```
                2 MSE     5 MSE    10 MSE          20 MSE         25 MSE  \
index name
Validation   1.619176  0.336511  0.270407  422722.686608  1.374592e+09
Train        1.585742  0.253159  0.235234       0.195400  1.838485e-01

               Linear
index name
Validation   2.052759
Train        2.776337
```

We can determine whether a predictive model is underfitting or overfitting the training data by looking at the prediction error on the training data and the evaluation data.

Model is:

- *Underfitting* - the training data when the model performs poorly on the training data. This is because the model is unable to capture the relationship between the input examples (often called X) and the target values (often called Y).
- *Overfitting* your training data when you see that the model performs well on the training data but does not perform well on the evaluation data. This is because the model is memorizing the data it has seen and is unable to generalize to unseen examples.

A model that is **underfit** will have high training and high testing error while an **overfit** model will have extremely low training error but a high testing error.

- Overfitted - 20, 25 Polynominal Regression model?

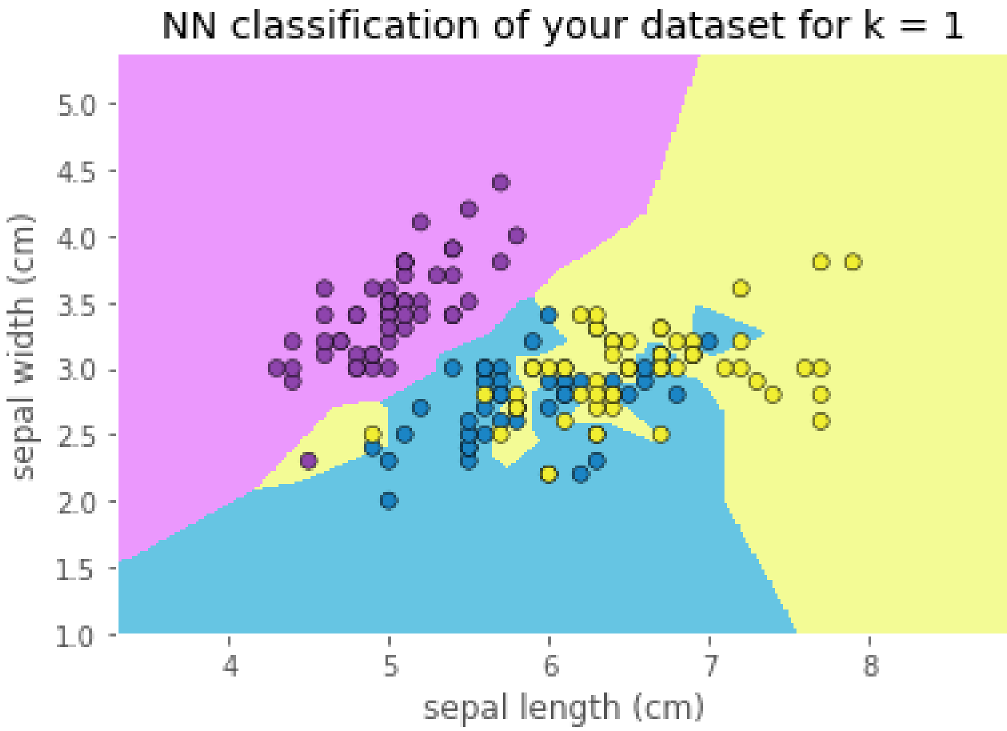
Neighbors-based classification is simple classification model when classification is computed from a simple majority vote of the nearest neighbors of each point: a query point is assigned the data class which has the most representatives within the nearest neighbors of the point.

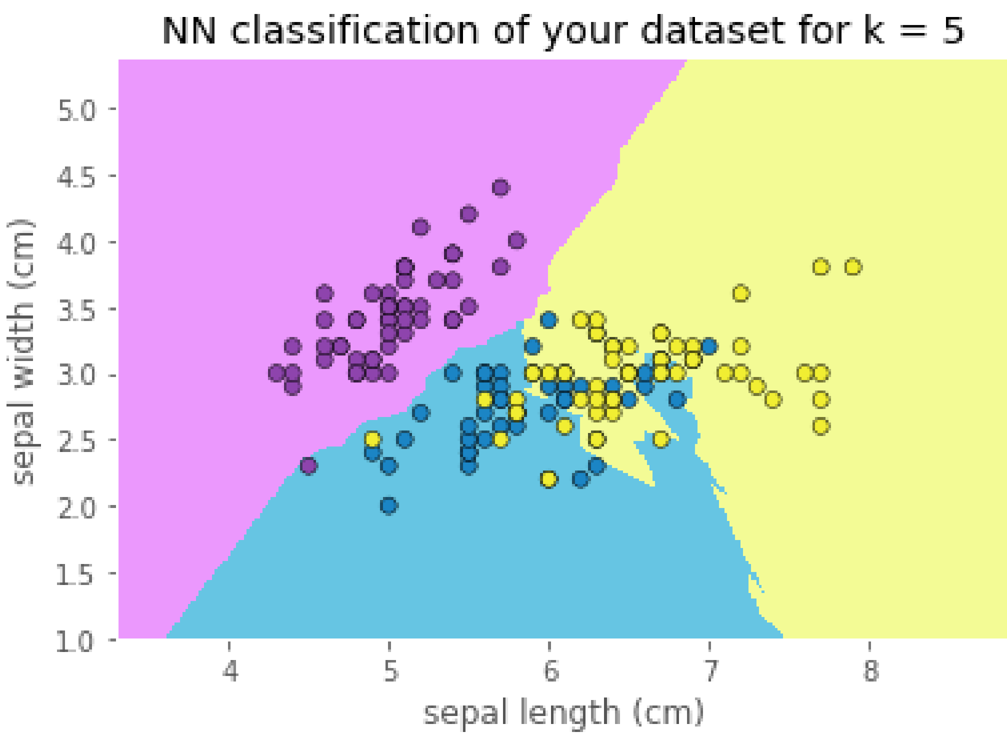
We are feeding our model with data that are allready with correct labels so this algorithm relies on labeled input data to learn a function that produces an appropriate output when given new unlabeled data.

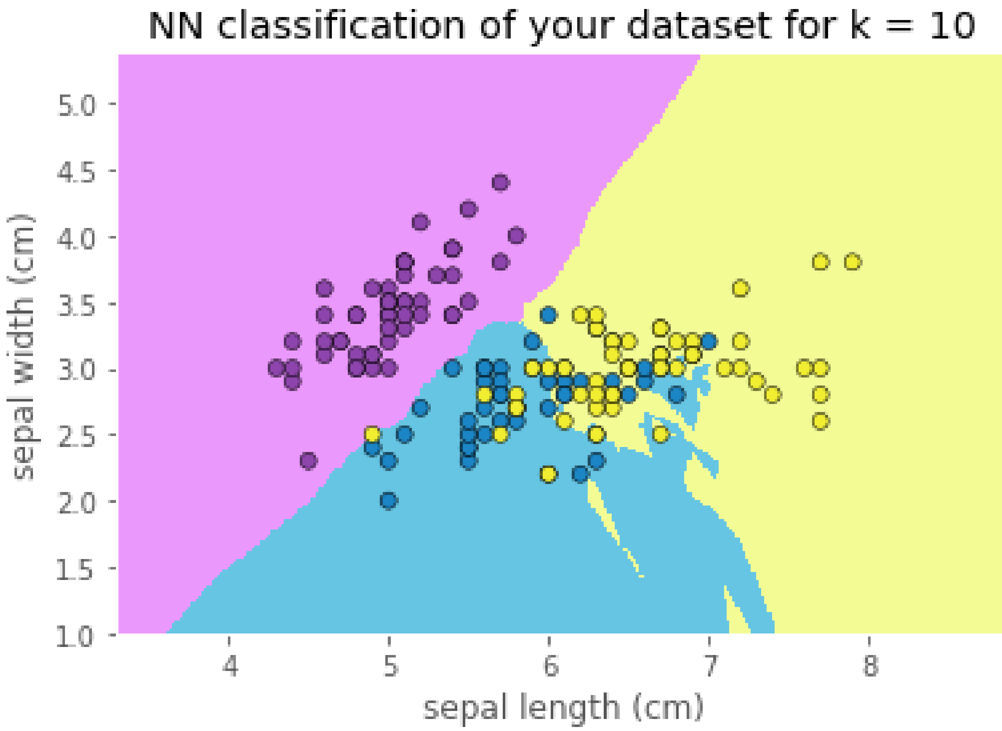
1.1 Exercise 2

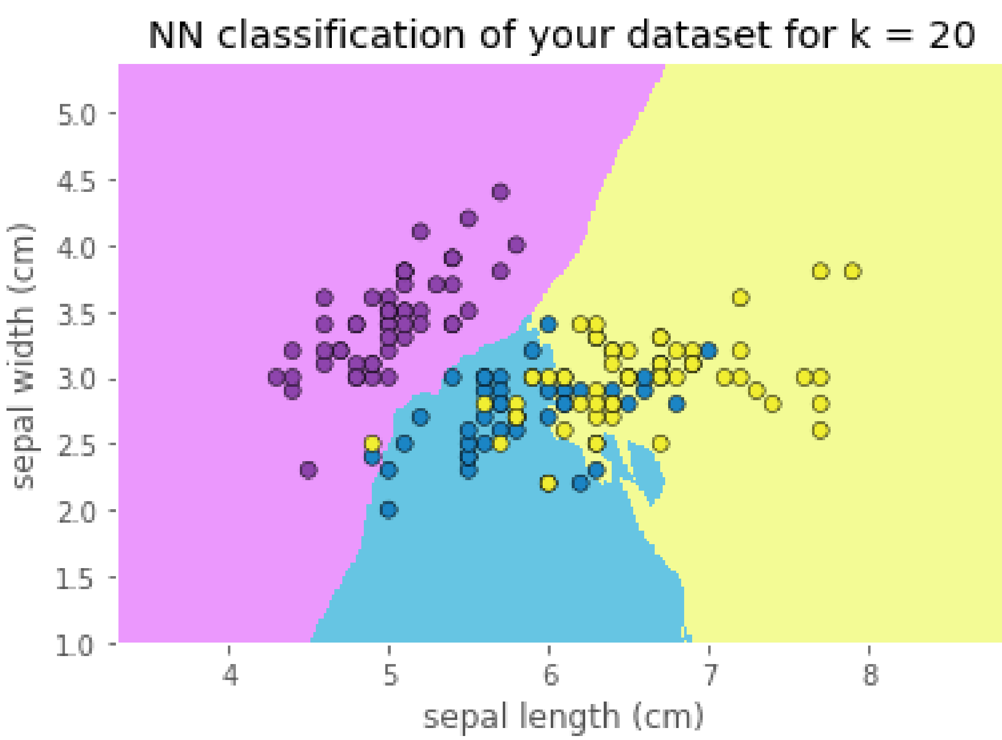
[10]:
```
# %%



# Visualization:
cmap_light = ListedColormap(['#EB98FD', '#66C5E3', '#F3FB95'])
cmap_dark = ListedColormap(['#8E44AD', '#1B85C5', '#F1EE32'])
k = [1, 5, 10, 20, 30]


iris = datasets.load_iris()
X = iris.data[:, :2]  # we only take the first two features.
Y = iris.target
h = 0.02
summary = pd.DataFrame()
summary['index name'] = ["Validation", "Train", "K"]
summary = pd.DataFrame(summary.set_index('index name'))

# Split data
X_train, X_rem, y_train, y_rem = train_test_split(X, Y, train_size=0.6)

test_size = 0.5
X_valid, X_test, y_valid, y_test = train_test_split(
    X_rem, y_rem, test_size=0.5, random_state=20)


def viz_classification(k):
    model = KNeighborsClassifier(n_neighbors=k)
    model.fit(X_train, y_train)

    x_min, x_max = X[:, 0].min() - 1, X[:, 0].max() + 1
    y_min, y_max = X[:, 1].min() - 1, X[:, 1].max() + 1
    xx, yy = np.meshgrid(np.arange(x_min, x_max, h),
                         np.arange(y_min, y_max, h))
    Z = model.predict(np.c_[xx.ravel(), yy.ravel()])
    valid_score = accuracy_score(y_valid, model.predict(X_valid))
    train_score = accuracy_score(y_train, model.predict(X_train))
    summary[str(k) + " Accurancy Score"] = [valid_score, train_score, k]
```

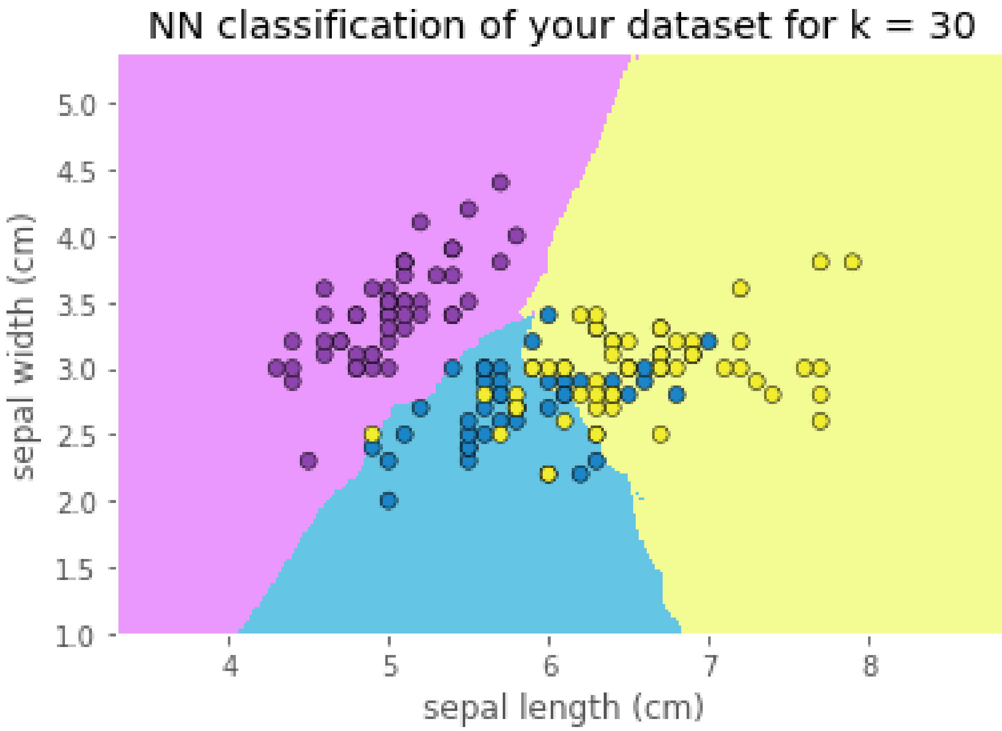
```python
    # Put the result into a color plot
    Z = Z.reshape(xx.shape)
    plt.figure()
    plt.pcolormesh(xx, yy, Z, cmap=cmap_light, shading='auto')

    # Plot also the training points
    plt.scatter(X[:, 0], X[:, 1], c=Y, cmap=cmap_dark, edgecolors='black')

    plt.xlim(xx.min(), xx.max())
    plt.ylim(yy.min(), yy.max())
    plt.title('NN classification of your dataset for k = ' + str(k))
    plt.xlabel('sepal length (cm)')
    plt.ylabel('sepal width (cm)')
    return


for i in range(0, len(k)):
    viz_classification(k[i])

plt.figure()
plt.subplots_adjust(left=0.1)
plt.plot(summary.iloc[2], summary.iloc[1], label="Validation")
plt.plot(summary.iloc[2], summary.iloc[0], label="Train")
plt.title("Change in as a function of different K")

plt.legend(loc='best', facecolor='lavender')
plt.show()
```

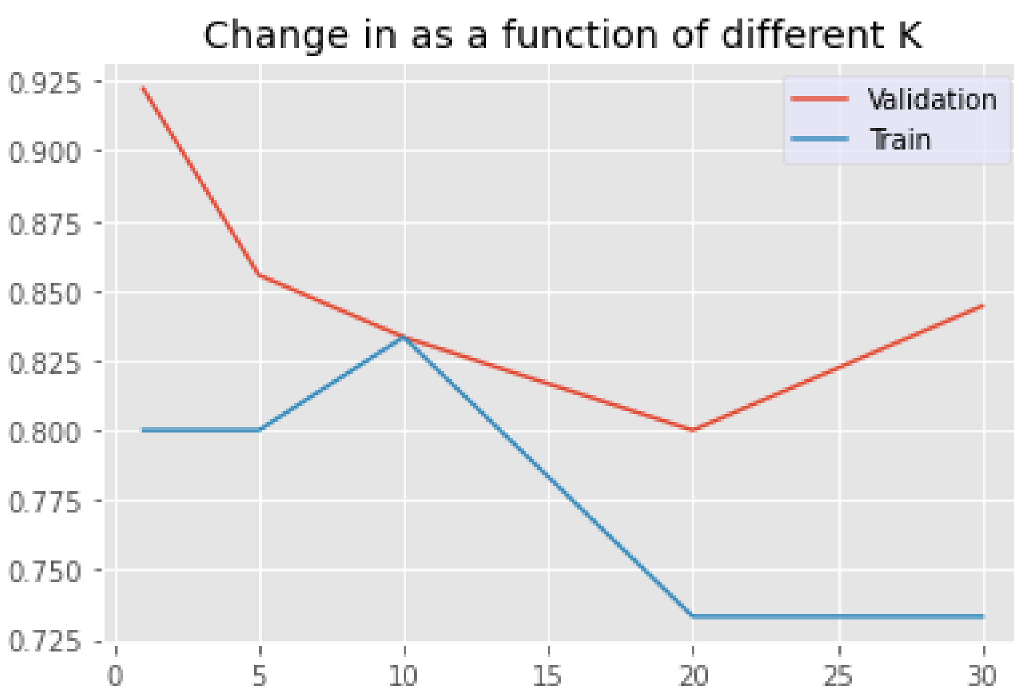

NN classification of your dataset for k = 1
sepal width (cm)
sepal length (cm)
5.0
4.5
4.0
3.5
3.0
2.5
2.0
1.5
1.0
4
5
6
7
8


NN classification of your dataset for k = 5
sepal width (cm)
sepal length (cm)
5.0
4.5
4.0
3.5
3.0
2.5
2.0
1.5
1.0
4
5
6
7
8

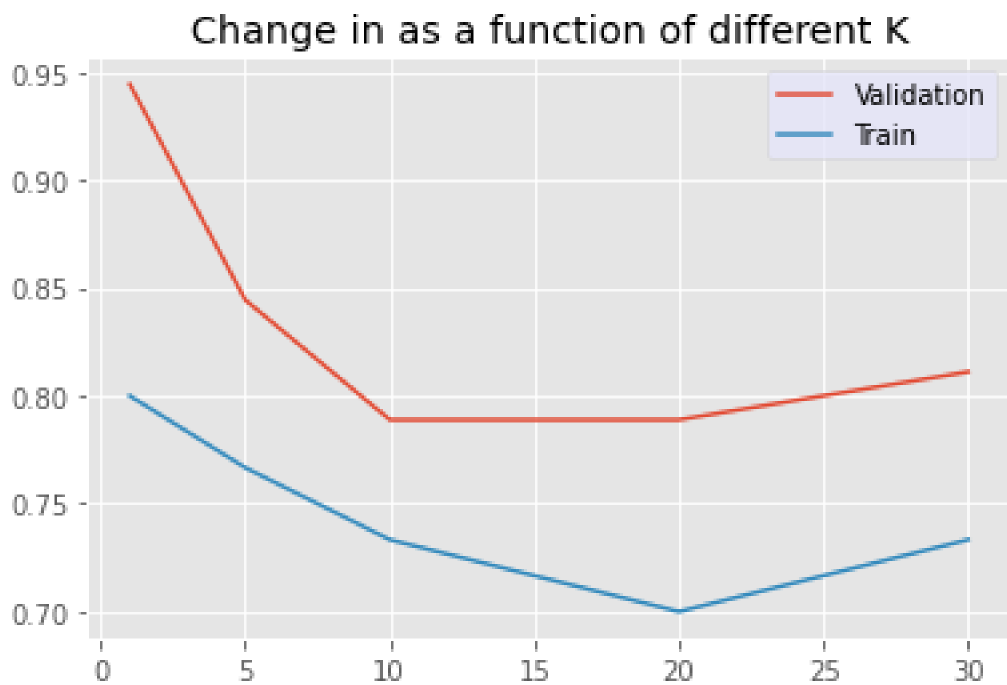

NN classification of your dataset for k = 10
sepal width (cm)
sepal length (cm)
5.0
4.5
4.0
3.5
3.0
2.5
2.0
1.5
1.0
4
5
6
7
8


NN classification of your dataset for k = 20
sepal width (cm)
sepal length (cm)
5.0
4.5
4.0
3.5
3.0
2.5
2.0
1.5
1.0
4
5
6
7
8


NN classification of your dataset for k = 30
sepal width (cm)
sepal length (cm)
5.0
4.5
4.0
3.5
3.0
2.5
2.0
1.5
1.0
4
5
6
7
8


Change in as a function of different K
Validation
Train
0.925
0.900
0.875
0.850
0.825
0.800
0.775
0.750
0.725
0
5
10
15
20
25
30

[6]:
```
plt.figure()
plt.subplots_adjust(left=0.1)
plt.plot(summary.iloc[2], summary.iloc[1], label="Validation")
plt.plot(summary.iloc[2], summary.iloc[0], label="Train")
plt.title("Change in as a function of different K")

plt.legend(loc='best', facecolor='lavender')
plt.show()
```



[7]:
```
summary
```

[7]:
```
                1 Accurancy Score  5 Accurancy Score  10 Accurancy Score  \
index name
Validation               0.800000           0.766667            0.733333
Train                    0.944444           0.844444            0.788889
K                        1.000000           5.000000           10.000000

                20 Accurancy Score  30 Accurancy Score
index name
Validation                0.700000            0.733333
Train                     0.788889            0.811111
K                        20.000000           30.000000
```

How well does your best performing model on validation data, make predictions on the test dataset? Model with best performance for validation model is K = 30, whereas the model for train dataset for K=30 has the highest MSE.

[8]:
```
X_test = np.array(X_test).reshape(-1, 1)
poly_X_test = PolynomialFeatures(5).fit_transform(X_test)
```

2 Exercise 3

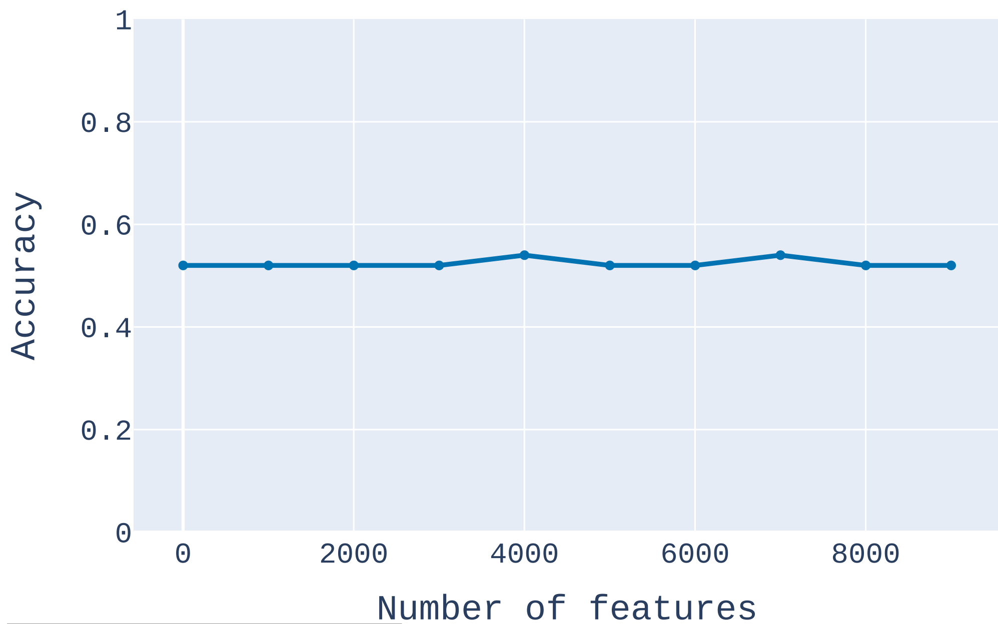
Data leakage is when information from outside the training dataset is used to create the model. When the data used to train a machine-learning algorithm happens to have the information the model is trying to predict;

I think is this because we are using all features of our dataset. We are tecahing our model with features is it about to predict.

[9]:
```
np.random.seed(0)
X_ = np.random.randn(500, 10000)
y = np.random.randint(2, size=500)
scores = []
k_list = list(range(1, 10001, 1000))

# Create a pipeline that scales the data then trains a support vector classifier
classifier_pipeline = make_pipeline(preprocessing.StandardScaler(), svm.
 ↪SVC(C=1))
for i, k in enumerate(k_list):
    print('%d from %d of list itmes are checked'%(i+1, len(k_list)))
    X_selected = SelectKBest(k=k).fit_transform(X_, y)

    #scores.append(cross_val_score(classifier_pipeline, X_selected, y, cv=10).
 ↪mean())
    scores = cross_val_score(classifier_pipeline, X_selected, y, cv=10)

fig = go.Figure()

fig.add_trace(go.Scatter(
        x=k_list,
        y=scores,
        line=dict(
                    color='#0173b2',
                    width=3
                ),
        name='Accuracy 1'
        ))

fig.update_layout(
    font=dict(
        family="Courier New, monospace",
        size=18),
```

```
                 xaxis= {'title': 'Number of features'},
                yaxis = {'title': 'Accuracy'},
                yaxis_range=[0,1])
fig.show()
```

```
1 from 10 of list itmes are checked
2 from 10 of list itmes are checked
3 from 10 of list itmes are checked
4 from 10 of list itmes are checked
5 from 10 of list itmes are checked
6 from 10 of list itmes are checked
7 from 10 of list itmes are checked
8 from 10 of list itmes are checked
9 from 10 of list itmes are checked
10 from 10 of list itmes are checked
```



Loading [MathJax]/extensions/MathMenu.js