wwtglzth8

September 11, 2023

##Name - Indrnail bain ##Enrollment No. - 2020CSB039 ##Assignment - 4 (TITANIC Dataset)

1. Download Titanic Dataset (https://www.kaggle.com/heptapod/titanic/version/1#) and do initial pre-processing including normalization, na or zero column handling, train test split, and others (Write an explanation of each in the report).

[3]:
```
from google.colab import drive
drive.mount('/content/drive')
```

```
Mounted at /content/drive
```

[8]:
```
BASE_PATH = "/content/drive/MyDrive/CSV Files - COLAB/train_and_test2.csv"
```

[9]:
```
import pandas as pd
titanic_df = pd.read_csv(BASE_PATH)
titanic_df
```

[9]:
```
      Passengerid   Age      Fare  Sex  sibsp  zero  zero.1  zero.2  zero.3  \
0               1  22.0    7.2500    0      1     0       0       0       0
1               2  38.0   71.2833    1      1     0       0       0       0
2               3  26.0    7.9250    1      0     0       0       0       0
3               4  35.0   53.1000    1      1     0       0       0       0
4               5  35.0    8.0500    0      0     0       0       0       0
…             …   …         …   …      …   …     …     …     …
1304         1305  28.0    8.0500    0      0     0       0       0       0
1305         1306  39.0  108.9000    1      0     0       0       0       0
1306         1307  38.5    7.2500    0      0     0       0       0       0
1307         1308  28.0    8.0500    0      0     0       0       0       0
1308         1309  28.0   22.3583    0      1     0       0       0       0

      zero.4  …  zero.12  zero.13  zero.14  Pclass  zero.15  zero.16  \
0          0  …        0        0        0       3        0        0
1          0  …        0        0        0       1        0        0
2          0  …        0        0        0       3        0        0
3          0  …        0        0        0       1        0        0
4          0  …        0        0        0       3        0        0
…       …  …      …      …      …     …      …      …
1304       0  …        0        0        0       3        0        0
```

```
1305       0  …        0        0        0       1        0        0
1306       0  …        0        0        0       3        0        0
1307       0  …        0        0        0       3        0        0
1308       0  …        0        0        0       3        0        0

      Embarked  zero.17  zero.18  2urvived
0          2.0        0        0         0
1          0.0        0        0         1
2          2.0        0        0         1
3          2.0        0        0         1
4          2.0        0        0         0
…          …        …        …         …
1304       2.0        0        0         0
1305       0.0        0        0         0
1306       2.0        0        0         0
1307       2.0        0        0         0
1308       0.0        0        0         0

[1309 rows x 28 columns]
```

[10]:
```
titanic_df.dropna()
```

[10]:
```
      Passengerid   Age      Fare  Sex  sibsp  zero  zero.1  zero.2  zero.3  \
0               1  22.0    7.2500    0      1     0       0       0       0
1               2  38.0   71.2833    1      1     0       0       0       0
2               3  26.0    7.9250    1      0     0       0       0       0
3               4  35.0   53.1000    1      1     0       0       0       0
4               5  35.0    8.0500    0      0     0       0       0       0
…             …     …        …    …      …     …       …       …
1304         1305  28.0    8.0500    0      0     0       0       0       0
1305         1306  39.0  108.9000    1      0     0       0       0       0
1306         1307  38.5    7.2500    0      0     0       0       0       0
1307         1308  28.0    8.0500    0      0     0       0       0       0
1308         1309  28.0   22.3583    0      1     0       0       0       0

      zero.4  …  zero.12  zero.13  zero.14  Pclass  zero.15  zero.16  \
0          0  …        0        0        0       3        0        0
1          0  …        0        0        0       1        0        0
2          0  …        0        0        0       3        0        0
3          0  …        0        0        0       1        0        0
4          0  …        0        0        0       3        0        0
…        …  …      …        …        …       …        …        …
1304       0  …        0        0        0       3        0        0
1305       0  …        0        0        0       1        0        0
1306       0  …        0        0        0       3        0        0
1307       0  …        0        0        0       3        0        0
1308       0  …        0        0        0       3        0        0
```

	Embarked	zero.17	zero.18	2urvived
0	2.0	0	0	0
1	0.0	0	0	1
2	2.0	0	0	1
3	2.0	0	0	1
4	2.0	0	0	0
…	…	…	…	…
1304	2.0	0	0	0
1305	0.0	0	0	0
1306	2.0	0	0	0
1307	2.0	0	0	0
1308	0.0	0	0	0

[1307 rows x 28 columns]

[11]:
```python
titanic_df.columns
```

[11]:
```
Index(['Passengerid', 'Age', 'Fare', 'Sex', 'sibsp', 'zero', 'zero.1',
       'zero.2', 'zero.3', 'zero.4', 'zero.5', 'zero.6', 'Parch', 'zero.7',
       'zero.8', 'zero.9', 'zero.10', 'zero.11', 'zero.12', 'zero.13',
       'zero.14', 'Pclass', 'zero.15', 'zero.16', 'Embarked', 'zero.17',
       'zero.18', '2urvived'],
      dtype='object')
```

[12]:
```python
titanic_df = titanic_df[
    filter(
lambda colName: "zero" not in colName,
        titanic_df.columns
    )
]
titanic_df = titanic_df.drop("Passengerid", axis=1)
titanic_df
```

[12]:

	Age	Fare	Sex	sibsp	Parch	Pclass	Embarked	2urvived
0	22.0	7.2500	0	1	0	3	2.0	0
1	38.0	71.2833	1	1	0	1	0.0	1
2	26.0	7.9250	1	0	0	3	2.0	1
3	35.0	53.1000	1	1	0	1	2.0	1
4	35.0	8.0500	0	0	0	3	2.0	0
…	…	…	…	…	…	…	…	…
1304	28.0	8.0500	0	0	0	3	2.0	0
1305	39.0	108.9000	1	0	0	1	0.0	0
1306	38.5	7.2500	0	0	0	3	2.0	0
1307	28.0	8.0500	0	0	0	3	2.0	0
1308	28.0	22.3583	0	1	1	3	0.0	0

[1309 rows x 8 columns]

[13]:
```
from sklearn.preprocessing import OneHotEncoder
from sklearn.preprocessing import StandardScaler

def one_hot_encode(X: pd.DataFrame, col_name: str) -> pd.DataFrame:
    """
    Alters X by one-hot-encoding the values of the col_name using␣
 ↪OneHotEncoder(),
    returns the altered DataFrame
    """
    encoder = OneHotEncoder()
    encoded_df = pd.DataFrame(
        encoder.fit_transform(X[[col_name]]).toarray(),
        index=X.index,
        columns=encoder.get_feature_names_out([col_name])
    )
    X = X.join(encoded_df)
    X = X.drop(col_name, axis=1)
    return X

def standardize(df: pd.DataFrame, col_name: str) -> pd.DataFrame:
    """
    Alters df by standardizing the values of the col_name using␣
 ↪StandardScaler(),
    returns the altered DataFrame
    """
    scaler = StandardScaler()
    df[[col_name]] = pd.DataFrame(
        data=scaler.fit_transform(df[[col_name]]),
        index=df.index,
        columns=[col_name]
    )
    return df
```

[15]:
```
# Pclass has value ranging from 0 to 3 (doing OneHotEncoding)
# Sex has value ranging from 0 to 2 (doing OneHotEncoding)
# Embarked has value ranging from 0 to 3 (doing OneHotEncoding)
columns_to_encode = ["Pclass", "Embarked", "Sex"]

for column in columns_to_encode:
  titanic_df = one_hot_encode(titanic_df, column)
titanic_df
```

[15]:
```
        Age     Fare  sibsp  Parch  2urvived  Pclass_1  Pclass_2  Pclass_3  \
0      22.0   7.2500      1      0         0       0.0       0.0       1.0
1      38.0  71.2833      1      0         1       1.0       0.0       0.0
```

```
2      26.0    7.9250       0       0         1       0.0       0.0       1.0
3      35.0   53.1000       1       0         1       1.0       0.0       0.0
4      35.0    8.0500       0       0         0       0.0       0.0       1.0
…     …        …     …     …       …       …       …       …
1304   28.0    8.0500       0       0         0       0.0       0.0       1.0
1305   39.0  108.9000       0       0         0       1.0       0.0       0.0
1306   38.5    7.2500       0       0         0       0.0       0.0       1.0
1307   28.0    8.0500       0       0         0       0.0       0.0       1.0
1308   28.0   22.3583       1       1         0       0.0       0.0       1.0

      Embarked_0.0  Embarked_1.0  Embarked_2.0  Embarked_nan  Sex_0  Sex_1
0              0.0           0.0           1.0           0.0    1.0    0.0
1              1.0           0.0           0.0           0.0    0.0    1.0
2              0.0           0.0           1.0           0.0    0.0    1.0
3              0.0           0.0           1.0           0.0    0.0    1.0
4              0.0           0.0           1.0           0.0    1.0    0.0
…             …            …            …            …     …     …
1304           0.0           0.0           1.0           0.0    1.0    0.0
1305           1.0           0.0           0.0           0.0    0.0    1.0
1306           0.0           0.0           1.0           0.0    1.0    0.0
1307           0.0           0.0           1.0           0.0    1.0    0.0
1308           1.0           0.0           0.0           0.0    1.0    0.0

[1309 rows x 14 columns]
```

[17]:

```python
# Age, Fare, sibsp, Parch needs to be standardized as their values
# are not in the range of 0 to 1
columns_to_standardize = ['Age', "Fare", 'sibsp', "Parch"]

for column in columns_to_standardize:
    titanic_df = standardize(titanic_df, column)
titanic_df
```

[17]:

```
           Age      Fare     sibsp     Parch  2urvived  Pclass_1  Pclass_2  \
0    -0.581628 -0.503291  0.481288 -0.445000         0       0.0       0.0
1     0.658652  0.734744  0.481288 -0.445000         1       1.0       0.0
2    -0.271558 -0.490240 -0.479087 -0.445000         1       0.0       0.0
3     0.426099  0.383183  0.481288 -0.445000         1       1.0       0.0
4     0.426099 -0.487824 -0.479087 -0.445000         0       0.0       0.0
…          …       …       …       …       …       …       …
1304 -0.116523 -0.487824 -0.479087 -0.445000         0       0.0       0.0
1305  0.736169  1.462034 -0.479087 -0.445000         0       1.0       0.0
1306  0.697411 -0.503291 -0.479087 -0.445000         0       0.0       0.0
1307 -0.116523 -0.487824 -0.479087 -0.445000         0       0.0       0.0
1308 -0.116523 -0.211184  0.481288  0.710763         0       0.0       0.0

      Pclass_3  Embarked_0.0  Embarked_1.0  Embarked_2.0  Embarked_nan  Sex_0  \
```

```
0          1.0          0.0          0.0          1.0          0.0    1.0
1          0.0          1.0          0.0          0.0          0.0    0.0
2          1.0          0.0          0.0          1.0          0.0    0.0
3          0.0          0.0          0.0          1.0          0.0    0.0
4          1.0          0.0          0.0          1.0          0.0    1.0
…         …          …          …          …          …    …
1304       1.0          0.0          0.0          1.0          0.0    1.0
1305       0.0          1.0          0.0          0.0          0.0    0.0
1306       1.0          0.0          0.0          1.0          0.0    1.0
1307       1.0          0.0          0.0          1.0          0.0    1.0
1308       1.0          1.0          0.0          0.0          0.0    1.0

      Sex_1
0       0.0
1       1.0
2       1.0
3       1.0
4       0.0
…     …
1304    0.0
1305    1.0
1306    0.0
1307    0.0
1308    0.0

[1309 rows x 14 columns]
```

2. Train the SVM using the below kernels with parameters, present the support vectors in the table of the comparison of the model along with accuracy.

- Linear
- Polynomial: where degree d is set to 2, 3 and 5
- RBF
- Sigmoid

[18]:
```python
from sklearn.svm import SVC
import pandas as pd

def trainSVC(
    X_train: pd.DataFrame,
    X_test: pd.DataFrame,
    y_train: pd.DataFrame,
    y_test: pd.DataFrame,
    kernel: str,
    degree: int = 3,
    return_model: bool = False
):
    """
```

```python
    degree is ignored if kernel is not 'poly'
    """
    model = SVC(kernel=kernel, degree=degree)
    model.fit(X_train, y_train.iloc[:,0])
    accuracy = model.score(X_test, y_test)

    if return_model:
        return model

    return [kernel, degree if kernel=='poly' else 'None', accuracy, model.
↪support_vectors_]
```

[19]:
```python
from sklearn.model_selection import train_test_split
X = titanic_df.drop('2urvived', axis=1)
y = titanic_df[['2urvived']]
X_train, X_test, y_train, y_test = train_test_split(X,y, test_size=0.2)
trainSVC(X_train, X_test, y_train, y_test, 'linear')
```

[19]:
```
['linear',
 'None',
 0.7862595419847328,
 array([[-0.50411079, -0.39211923, -0.47908676, …,  0.        ,
          1.        ,  0.        ],
        [ 1.58886213,  3.44572594,  0.48128777, …,  0.        ,
          0.        ,  1.        ],
        [-0.42659328,  4.14216046, -0.47908676, …,  0.        ,
          1.        ,  0.        ],
        …,
        [-0.11652322,  0.94524511,  0.48128777, …,  0.        ,
          0.        ,  1.        ],
        [ 0.27106436, -0.1407742 , -0.47908676, …,  0.        ,
          0.        ,  1.        ],
        [ 1.58886213, -0.1407742 , -0.47908676, …,  0.        ,
          0.        ,  1.        ]])]
```

[21]:
```python
configs = [
    ['linear'],
    ['poly', 2],
    ['poly', 3],
    ['poly', 5],
    ['rbf'],
    ['sigmoid']
]
data = [
trainSVC(X_train, X_test, y_train, y_test, *config) for config in configs
]
```

```
pd.DataFrame(data, columns=['kernel', 'degree (only for poly)',␣
↪'accuracy','support vectors'])
```

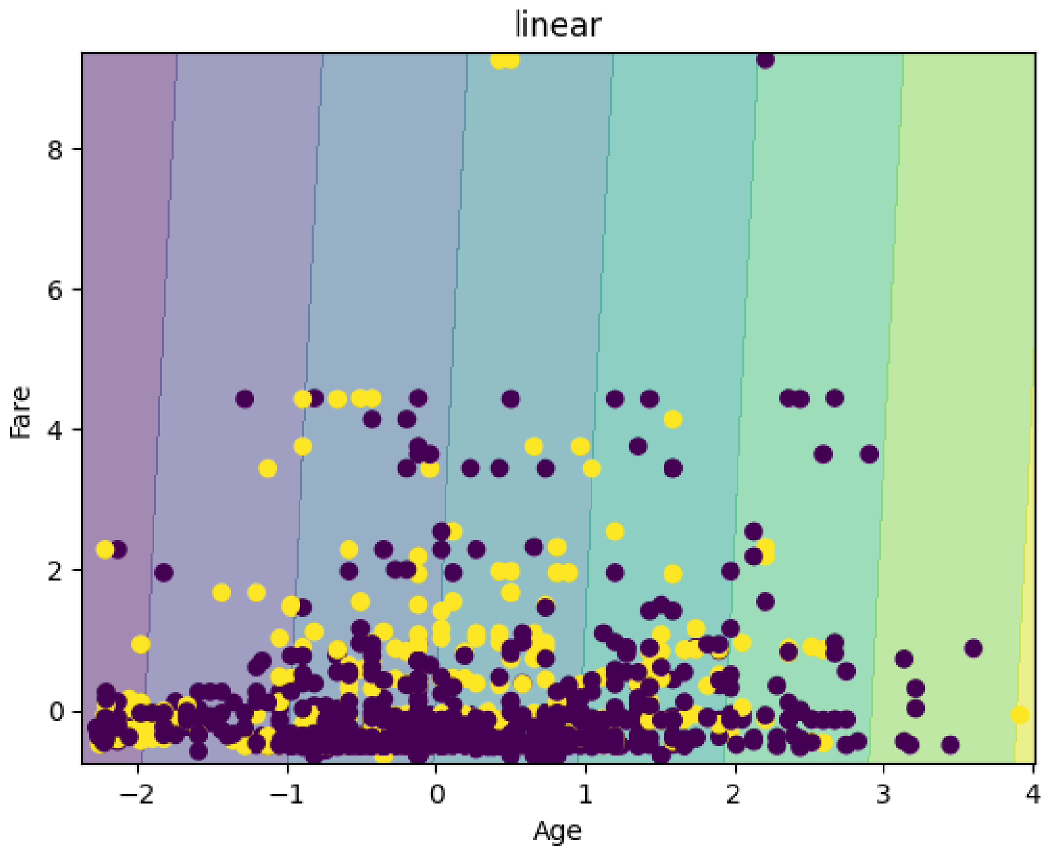
[21]:

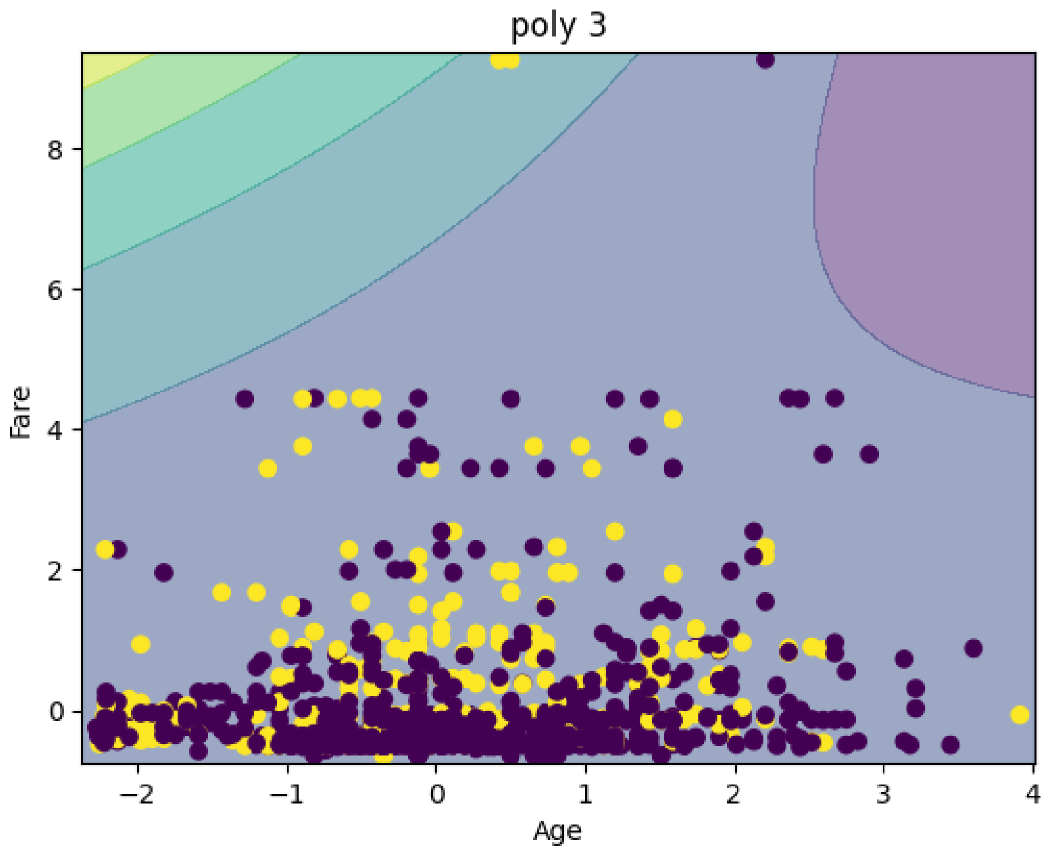
```
     kernel degree (only for poly)  accuracy  \
0    linear                   None  0.786260
1      poly                      2  0.786260
2      poly                      3  0.790076
3      poly                      5  0.767176
4       rbf                   None  0.790076
5   sigmoid                   None  0.675573

                                     support vectors
0  [[-0.5041107949194775, -0.3921192264701557, -0…
1  [[-0.6591458263607518, -0.4922550837903344, -0…
2  [[1.5888621295377254, 3.4457259384308845, 0.48…
3  [[1.5888621295377254, 3.4457259384308845, 0.48…
4  [[1.5888621295377254, 3.4457259384308845, 0.48…
5  [[1.5888621295377254, 3.4457259384308845, 0.48…
```

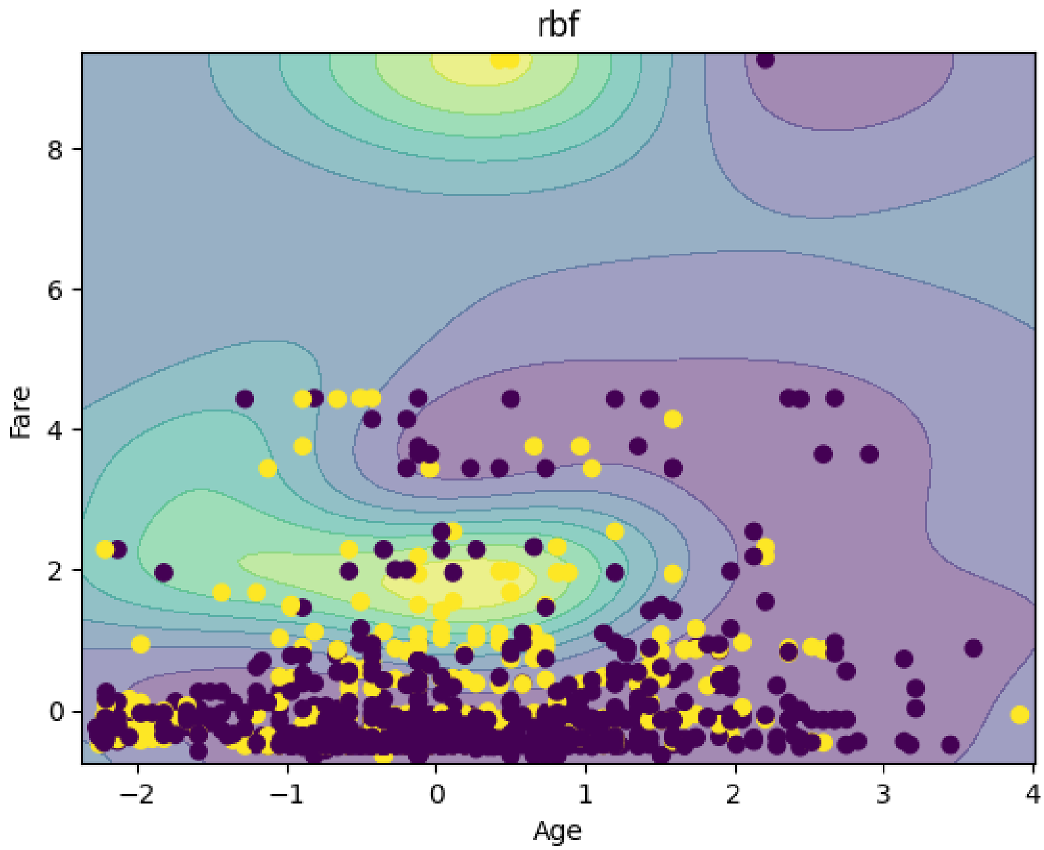
3. Take only two features from the dataset and train the models with the same parameters and plot the graphs to show the boundaries. Also, create a custom kernel function of your own using a mathematical function for suggestion Lograthmic or Tangent function.

[23]:

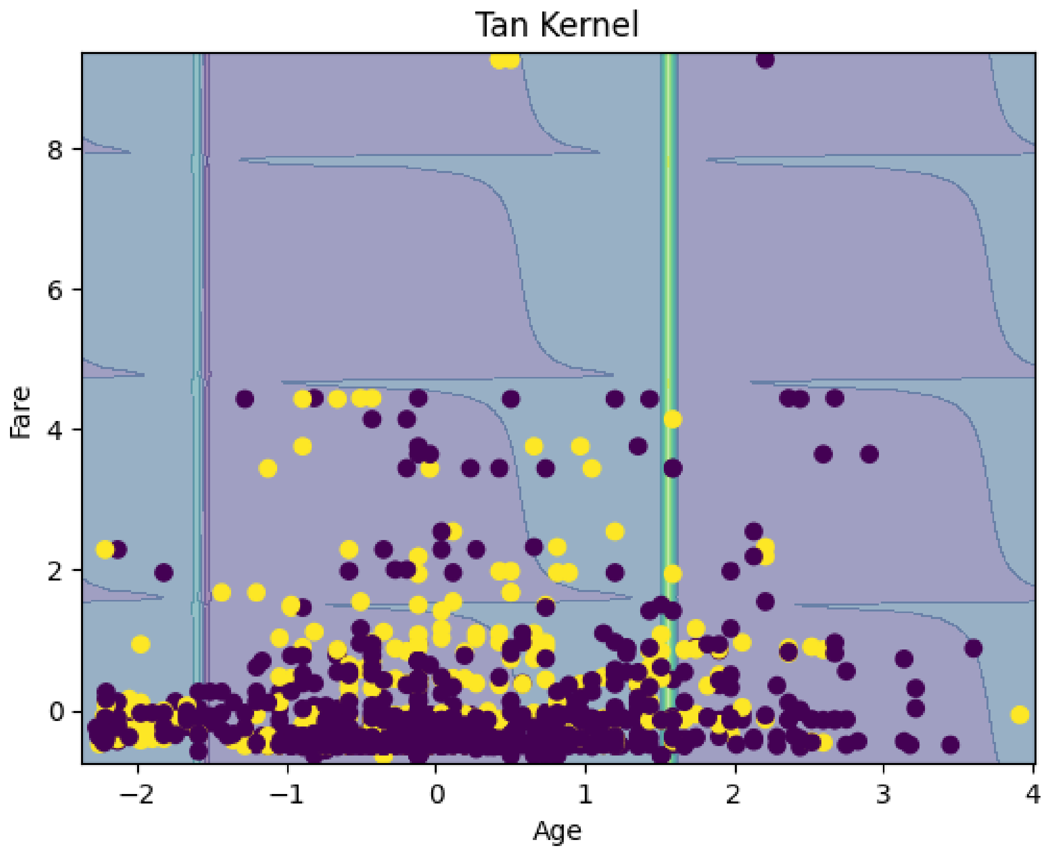
```python
from sklearn.inspection import DecisionBoundaryDisplay
import matplotlib.pyplot as plt

def plot_decision_boundary(kernel: str, degree=1):
    X = titanic_df.iloc[:, 0:2]
    model = SVC(kernel=kernel, degree=degree).fit(X, y.iloc[:, 0])
    disp = DecisionBoundaryDisplay.from_estimator(model, X, alpha=0.5, eps=0.1)
    disp.ax_.scatter(X.iloc[:, 0], X.iloc[:, 1], c=y.iloc[:, 0])

    if kernel == 'poly':
        plt.title(f"{kernel} {degree}")
    else:
        plt.title(f"{kernel}")

    plt.show()

configs = [("linear",), ("poly", 3), ("rbf",)]
for config in configs:
    plot_decision_boundary(*config)
```


linear
Fare
8
6
4
2
0
−2
−1
0
1
2
3
4
Age


poly 3
Fare
8
6
4
2
0
−2
−1
0
1
2
3
4
Age



[24]:
```
import numpy as np

def my_kernel(X:"np.array", Y: "np.array") -> "np.array":
  M = np.array([[1, 0], [0, 1]])
  return np.dot(np.dot(np.tan(X), M), Y.T)
X_sub = titanic_df.iloc[:, 0:2]
model = SVC(kernel=my_kernel).fit(X_sub, y.iloc[:, 0])
disp = DecisionBoundaryDisplay.from_estimator(model, X_sub, alpha=0.5, eps=0.1)
disp.ax_.scatter(X_sub.iloc[:, 0], X_sub.iloc[:, 1], c=y.iloc[:, 0])
plt.title('Tan Kernel')
plt.show()
```



4. For RBF kernel vary the control parameter C with a binary search technique to reach an optimal C value. Plot the graph for validation accuracy. Using this, mention the situation of overfitting and underfitting. Set Gamma to 0.5. Create a function for the whole process. [Maximum 20 runs]

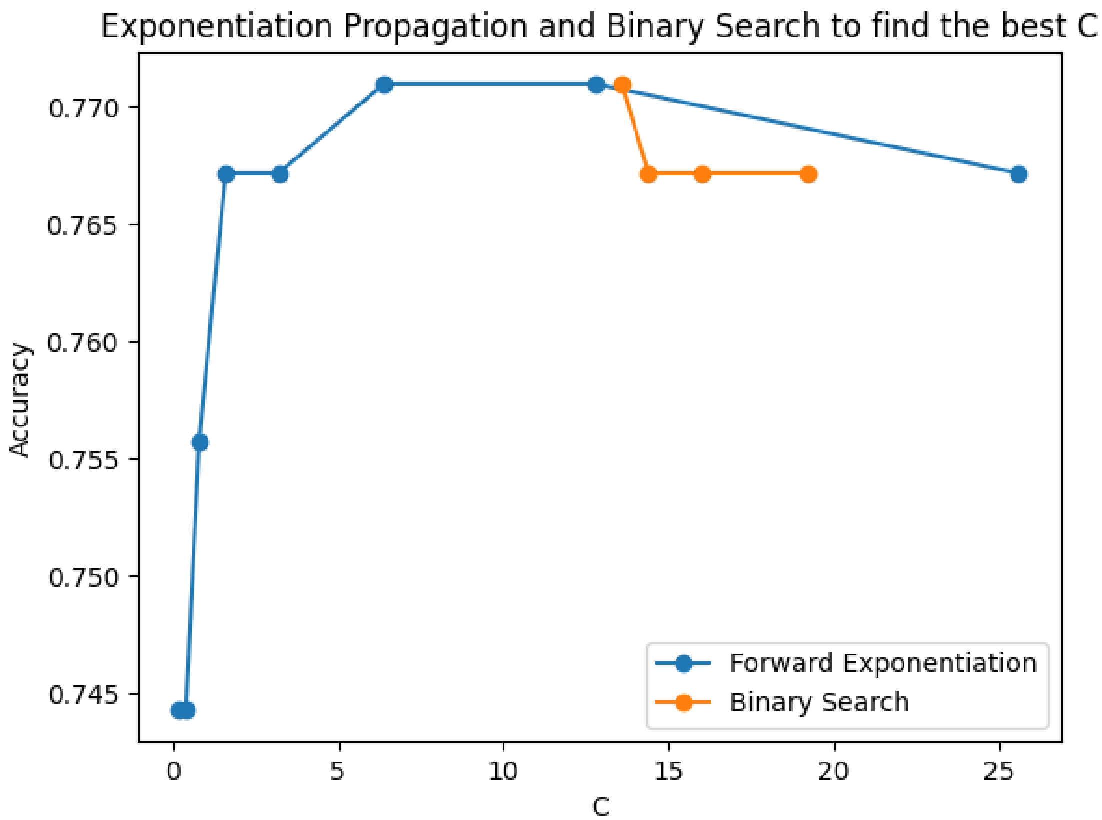
[48]:
```
def binary_search_C(X_train: "pd.DataFrame", X_val: "pd.DataFrame", y_train:␣
↪"pd.DataFrame", y_val: "pd.DataFrame"):
    right = 0.1
    left = 0
    max_acc = 0
    acc = 0.1
    accuracies = []
    Cs = []
    mid = 0  # Initialize mid

    while acc >= max_acc:
        left = right
        right *= 2
        model = SVC(kernel='rbf', gamma=0.5, C=right).fit(X_train, y_train.
↪iloc[:, 0])
```

```
        max_acc = max(max_acc, acc)
        acc = model.score(X_val, y_val)
        print(left, right, acc)
        accuracies.append(acc)
        Cs.append(right)

    plt.plot(Cs, accuracies, 'o-', label="Forward Exponentiation")
    Cs = []
    accuracies = []
    print("phase2")

    while left <= right:
        mid = (left + right) / 2
        model = SVC(kernel='rbf', C=mid, gamma=0.5).fit(X_train, y_train.iloc[:
↪, 0])
        acc = model.score(X_val, y_val)
        accuracies.append(acc)
        Cs.append(mid)
        print(left, mid, right, acc)
        if acc < max_acc:
            right = mid - 0.0001
        elif acc > max_acc:
            left = mid + 0.0001
            max_acc = acc
        else:
            break

    plt.plot(Cs, accuracies, 'o-', label="Binary Search")
    plt.title("Exponentiation Propagation and Binary Search to find the best C")
    plt.xlabel("C")
    plt.ylabel('Accuracy')
    plt.legend()
    plt.show()

    return mid
```

[50]:
```
X = titanic_df.iloc[:, 0:2]
X_train, X_val, y_train, y_val = train_test_split(X, y, test_size=0.2)
best_c = binary_search_C(X_train, X_val, y_train, y_val)
```

```
0.1 0.2 0.7442748091603053
0.2 0.4 0.7442748091603053
0.4 0.8 0.7557251908396947
0.8 1.6 0.767175572519084
1.6 3.2 0.767175572519084
3.2 6.4 0.7709923664122137
6.4 12.8 0.7709923664122137
```

```
12.8 25.6 0.767175572519084
phase2
12.8 19.200000000000003 25.6 0.767175572519084
12.8 15.999950000000002 19.199900000000003 0.767175572519084
12.8 14.399925000000001 15.999850000000002 0.767175572519084
12.8 13.599912500000002 14.399825000000002 0.7709923664122137
```



5. Using the above-created function now varies the Gamma parameter with the same binary search techniques as above for the C value which has maximum validation accuracy. Explain, whether the above calculated maximum test accuracy is the optimal test accuracy or there can be a better value of C and Gamma.

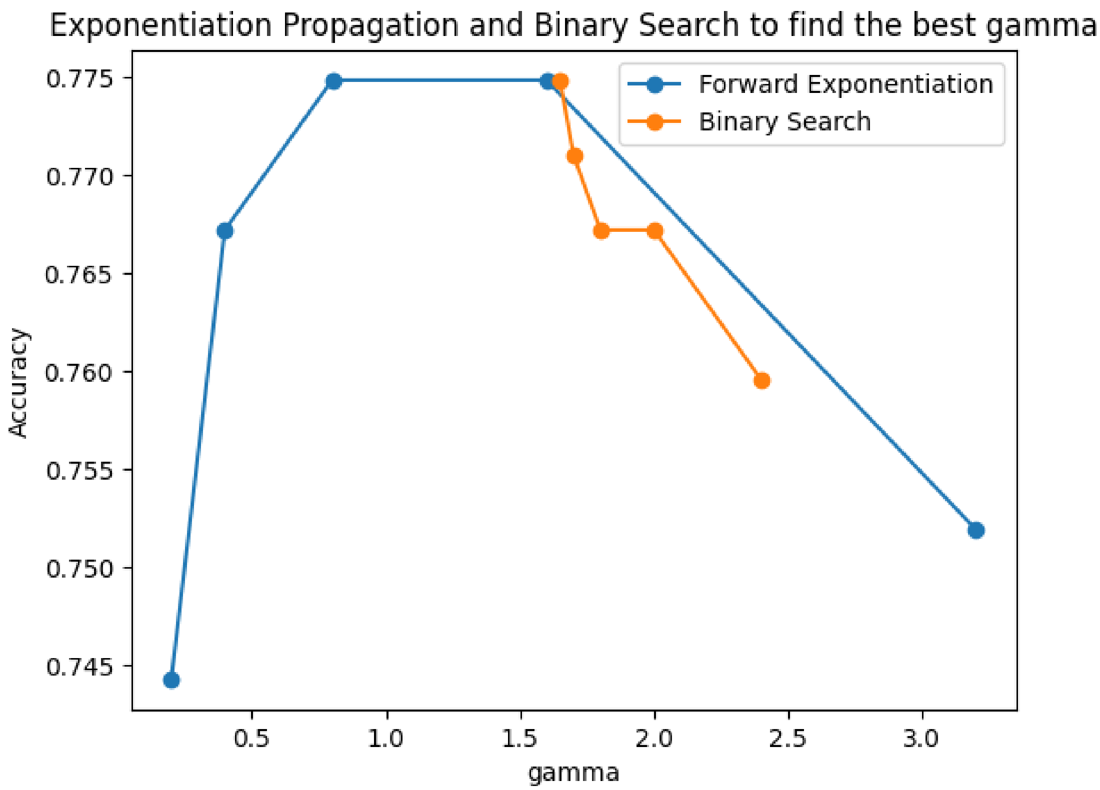
[53]:
```python
def binary_search_gamma(X_train: "pd.DataFrame", X_val: "pd.DataFrame", y_train:
 ↪ "pd.DataFrame", y_val: "pd.DataFrame", best_c: "float"):
    right = 0.1
    left = 0
    max_acc = 0
    acc = 0.1
    accuracies = []
    Cs = []
```

```
    while acc >= max_acc:
        left = right
        right *= 2
        model = SVC(kernel='rbf', gamma=right, C=best_c).fit(X_train, y_train.
↪iloc[:, 0])
        max_acc = max(max_acc, acc)
        acc = model.score(X_val, y_val)
        print(left, right, acc)
        accuracies.append(acc)
        Cs.append(right)

    plt.plot(Cs, accuracies, 'o-', label='Forward Exponentiation')
    Cs = []
    accuracies = []
    print("phase2")

    while left <= right:
        mid = (left + right) / 2
        model = SVC(kernel='rbf', C=best_c, gamma=mid).fit(X_train, y_train.
↪iloc[:, 0])
        acc = model.score(X_val, y_val)
        accuracies.append(acc)
        Cs.append(mid)
        print(left, mid, right, acc)
        if acc < max_acc:
            right = mid - 0.0001
        elif acc > max_acc:
            left = mid + 0.0001
            max_acc = acc
        else:
            break

    plt.plot(Cs, accuracies, 'o-', label='Binary Search')
    plt.title("Exponentiation Propagation and Binary Search to find the best␣
↪gamma")
    plt.xlabel("gamma")
    plt.ylabel('Accuracy')
    plt.legend()
    plt.show()

    return mid
```

[54]:
```
binary_search_gamma(X_train, X_val, y_train, y_val, best_c)
```

```
0.1 0.2 0.7442748091603053
0.2 0.4 0.767175572519084
0.4 0.8 0.7748091603053435
```

```
0.8 1.6 0.7748091603053435
1.6 3.2 0.7519083969465649
phase2
1.6 2.4000000000000004 3.2 0.7595419847328244
1.6 1.9999500000000001 2.3999 0.767175572519084
1.6 1.799925 1.9998500000000001 0.767175572519084
1.6 1.6999125 1.799825 0.7709923664122137
1.6 1.64990625 1.6998125 0.7748091603053435
```



[54]: 1.64990625