Vashu Agarwal

E21CSEU0054

Lab5

In [22]:
```
# Naive Bayes
# Importing the libraries
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd

# Task 1 : Importing the dataset


dataset = pd.read_csv('/Users/vashuagarwal/Downloads/BENNETT  things
```

In [63]:
```
# Task 2 : print the names of the first 13 feature
dataset.head()
```

Out[63]:

	type	fixed acidity	volatile acidity	citric acid	residual sugar	chlorides	free sulfur dioxide	total sulfur dioxide	density	pH	sulpha
0	white	7.0	0.27	0.36	20.7	0.045	45.0	170.0	1.0010	3.00	0
1	white	6.3	0.30	0.34	1.6	0.049	14.0	132.0	0.9940	3.30	0
2	white	8.1	0.28	0.40	6.9	0.050	30.0	97.0	0.9951	3.26	0
3	white	7.2	0.23	0.32	8.5	0.058	47.0	186.0	0.9956	3.19	0
4	white	7.2	0.23	0.32	8.5	0.058	47.0	186.0	0.9956	3.19	0

In [64]:
```
dataset = dataset.dropna()
```

In [ ]:

In [ ]:

In [65]:
```
X = dataset.iloc[:, [2, 10]].values
y = dataset.iloc[:, 12].values
```

In [66]:
```
# Splitting the dataset into the Training set and Test set
from sklearn.model_selection import train_test_split


# Task 3:  train the model
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size

print(X_train.shape, X_test.shape, y_train.shape, y_test.shape)
```

```
(4847, 2) (1616, 2) (4847,) (1616,)
```

In [ ]:
```

```

In [67]:
```
# Feature Scaling
from sklearn.preprocessing import StandardScaler
sc = StandardScaler()

X_train = sc.fit_transform(X_train)
X_test = sc.transform(X_test)
```

In [68]:
```
# Task 4: Fitting Naive Bayes to the Training set

from sklearn.naive_bayes import GaussianNB
classifier = GaussianNB()
classifier.fit(X_train, y_train)
```

Out[68]:
```
GaussianNB()
```

In [69]:
```
# Task 5: Predicting the Test set results

y_pred = classifier.predict(X_test)
print(y_pred)
```

```
[6 6 6 ... 6 7 6]
```

In [70]:
```
# Task 6 : Making the Confusion Matrix
from sklearn.metrics import confusion_matrix


cm = confusion_matrix(y_test, y_pred)
print(cm)
```

```
[[  1   0   3   5   0   0   0]
 [  1   1  14  29   0   0   0]
 [  1  10 142 379   4   0   0]
 [  0   2 101 609  13   0   0]
 [  0   0  21 240   5   0   0]
 [  0   0   3  30   1   0   0]
 [  0   0   0   1   0   0   0]]
```

In [71]:
```
# Task 7 : Making the Classification report

from sklearn.metrics import classification_report
print(classification_report(y_test, y_pred))
```

```
              precision    recall  f1-score   support

           3       0.33      0.11      0.17         9
           4       0.08      0.02      0.03        45
           5       0.50      0.26      0.35       536
           6       0.47      0.84      0.60       725
           7       0.22      0.02      0.03       266
           8       0.00      0.00      0.00        34
           9       0.00      0.00      0.00         1

    accuracy                           0.47      1616
   macro avg       0.23      0.18      0.17      1616
weighted avg       0.42      0.47      0.39      1616


/Users/vashuagarwal/opt/anaconda3/lib/python3.8/site-packages/sklearn/metrics/_classification.py:1221: UndefinedMetricWarning: Precision and F-score are ill-defined and being set to 0.0 in labels with no predicted samples. Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
```

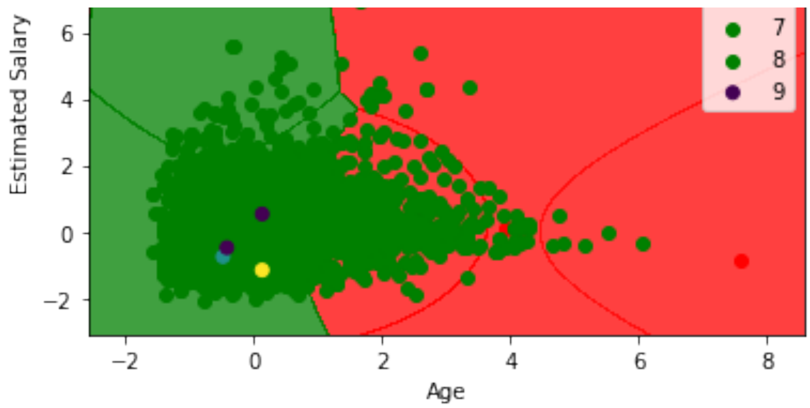
In [72]:
```
# Task 8 : Making the Classification accuracy score
from sklearn.metrics import accuracy_score
print ("Accuracy : ", accuracy_score(y_test, y_pred))
```

```
Accuracy :  0.46905940594059403
```

In [73]:

```
# Visualising the Training set results
from matplotlib.colors import ListedColormap
X_set, y_set = X_train, y_train
X1, X2 = np.meshgrid(np.arange(start = X_set[:, 0].min() - 1, stop
                     np.arange(start = X_set[:, 1].min() - 1, stop
plt.contourf(X1, X2, classifier.predict(np.array([X1.ravel(), X2.ra
             alpha = 0.75, cmap = ListedColormap(('red', 'green')))
plt.xlim(X1.min(), X1.max())
plt.ylim(X2.min(), X2.max())
for i, j in enumerate(np.unique(y_set)):
    plt.scatter(X_set[y_set == j, 0], X_set[y_set == j, 1],
                c = ListedColormap(('red', 'green'))(i), label = j)
plt.title('Naive Bayes (Training set)')
plt.xlabel('Age')
plt.ylabel('Estimated Salary')
plt.legend()
plt.show()
```

```
*c* argument looks like a single numeric RGB or RGBA sequence, whi
ch should be avoided as value-mapping will have precedence in case
its length matches with *x* & *y*.  Please use the *color* keyword
-argument or provide a 2D array with a single row if you intend to
specify the same RGB or RGBA value for all points.
*c* argument looks like a single numeric RGB or RGBA sequence, whi
ch should be avoided as value-mapping will have precedence in case
its length matches with *x* & *y*.  Please use the *color* keyword
-argument or provide a 2D array with a single row if you intend to
specify the same RGB or RGBA value for all points.
*c* argument looks like a single numeric RGB or RGBA sequence, whi
ch should be avoided as value-mapping will have precedence in case
its length matches with *x* & *y*.  Please use the *color* keyword
-argument or provide a 2D array with a single row if you intend to
specify the same RGB or RGBA value for all points.
*c* argument looks like a single numeric RGB or RGBA sequence, whi
ch should be avoided as value-mapping will have precedence in case
its length matches with *x* & *y*.  Please use the *color* keyword
-argument or provide a 2D array with a single row if you intend to
specify the same RGB or RGBA value for all points.
*c* argument looks like a single numeric RGB or RGBA sequence, whi
ch should be avoided as value-mapping will have precedence in case
its length matches with *x* & *y*.  Please use the *color* keyword
-argument or provide a 2D array with a single row if you intend to
specify the same RGB or RGBA value for all points.
*c* argument looks like a single numeric RGB or RGBA sequence, whi
ch should be avoided as value-mapping will have precedence in case
its length matches with *x* & *y*.  Please use the *color* keyword
-argument or provide a 2D array with a single row if you intend to
specify the same RGB or RGBA value for all points.
```

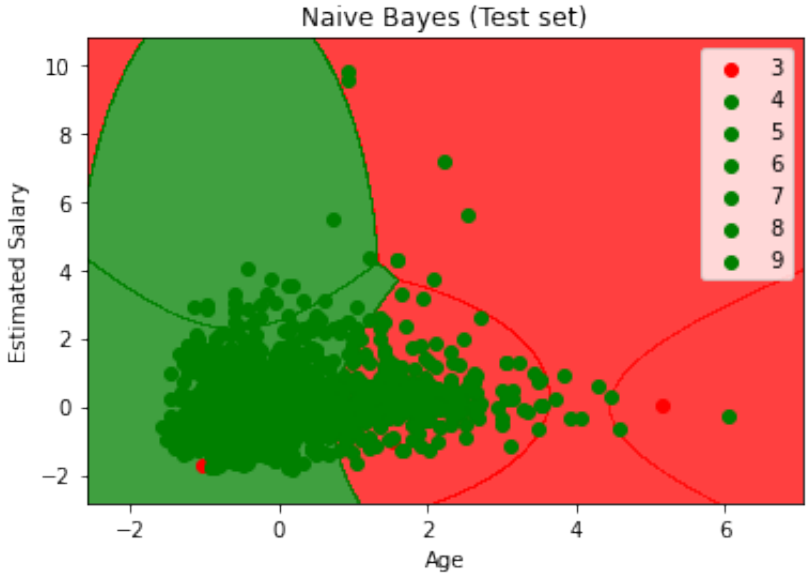


In [74]:

```
# Visualising the Test set results
from matplotlib.colors import ListedColormap
X_set, y_set = X_test, y_test
X1, X2 = np.meshgrid(np.arange(start = X_set[:, 0].min() - 1, stop
                     np.arange(start = X_set[:, 1].min() - 1, stop
plt.contourf(X1, X2, classifier.predict(np.array([X1.ravel(), X2.ra
             alpha = 0.75, cmap = ListedColormap(('red', 'green')))
plt.xlim(X1.min(), X1.max())
plt.ylim(X2.min(), X2.max())
for i, j in enumerate(np.unique(y_set)):
    plt.scatter(X_set[y_set == j, 0], X_set[y_set == j, 1],
                c = ListedColormap(('red', 'green'))(i), label = j)
plt.title('Naive Bayes (Test set)')
plt.xlabel('Age')
plt.ylabel('Estimated Salary')
plt.legend()
plt.show()
```

```
*c* argument looks like a single numeric RGB or RGBA sequence, whi
ch should be avoided as value-mapping will have precedence in case
its length matches with *x* & *y*.  Please use the *color* keyword
-argument or provide a 2D array with a single row if you intend to
specify the same RGB or RGBA value for all points.
*c* argument looks like a single numeric RGB or RGBA sequence, whi
ch should be avoided as value-mapping will have precedence in case
its length matches with *x* & *y*.  Please use the *color* keyword
-argument or provide a 2D array with a single row if you intend to
specify the same RGB or RGBA value for all points.
*c* argument looks like a single numeric RGB or RGBA sequence, whi
ch should be avoided as value-mapping will have precedence in case
its length matches with *x* & *y*.  Please use the *color* keyword
-argument or provide a 2D array with a single row if you intend to
specify the same RGB or RGBA value for all points.
*c* argument looks like a single numeric RGB or RGBA sequence, whi
ch should be avoided as value-mapping will have precedence in case
its length matches with *x* & *y*.  Please use the *color* keyword
-argument or provide a 2D array with a single row if you intend to
specify the same RGB or RGBA value for all points.
*c* argument looks like a single numeric RGB or RGBA sequence, whi
ch should be avoided as value-mapping will have precedence in case
its length matches with *x* & *y*.  Please use the *color* keyword
```

```
-argument or provide a 2D array with a single row if you intend to
specify the same RGB or RGBA value for all points.
*c* argument looks like a single numeric RGB or RGBA sequence, whi
ch should be avoided as value-mapping will have precedence in case
its length matches with *x* & *y*.  Please use the *color* keyword
-argument or provide a 2D array with a single row if you intend to
specify the same RGB or RGBA value for all points.
*c* argument looks like a single numeric RGB or RGBA sequence, whi
ch should be avoided as value-mapping will have precedence in case
its length matches with *x* & *y*.  Please use the *color* keyword
-argument or provide a 2D array with a single row if you intend to
specify the same RGB or RGBA value for all points.
```



In [ ]:

In [ ]:

In [ ]:

Maximum Likelihood Classification

Vashu Agarwal

E21CSEU0054 ¶

Lab 5

In [14]:
```
import numpy as np
import pandas as pd
```

In [15]:
```
# Task 1: Extract the dataset using panda / read the dataset


df=pd.read_csv('/Users/vashuagarwal/Downloads/train.csv')

df
```

Out[15]:

	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	F
0	1	0	3	Braund, Mr. Owen Harris	male	22.0	1	0	A/5 21171	7.2
1	2	1	1	Cumings, Mrs. John Bradley (Florence Briggs Th...	female	38.0	1	0	PC 17599	71.2
2	3	1	3	Heikkinen, Miss. Laina	female	26.0	0	0	STON/O2. 3101282	7.9
3	4	1	1	Futrelle, Mrs. Jacques Heath (Lily May Peel)	female	35.0	1	0	113803	53.1
4	5	0	3	Allen, Mr. William Henry	male	35.0	0	0	373450	8.0
...	...	...	...	...	...	...	...	...	...	
886	887	0	2	Montvila, Rev. Juozas	male	27.0	0	0	211536	13.0

887	888	1	1	Graham, Miss. Margaret Edith	female	19.0	0	0	112053	30.0
888	889	0	3	Johnston, Miss. Catherine Helen "Carrie"	female	NaN	1	2	W./C. 6607	23.4
889	890	1	1	Behr, Mr. Karl Howell	male	26.0	0	0	111369	30.0
890	891	0	3	Dooley, Mr. Patrick	male	32.0	0	0	370376	7.7

891 rows × 12 columns

In [16]:
```
# Task 2: Generate descriptive statistics of df


df.describe()
#print(description)


# O/P is shown below
```

Out[16]:

	PassengerId	Survived	Pclass	Age	SibSp	Parch	Far
count	891.000000	891.000000	891.000000	714.000000	891.000000	891.000000	891.00000
mean	446.000000	0.383838	2.308642	29.699118	0.523008	0.381594	32.20420
std	257.353842	0.486592	0.836071	14.526497	1.102743	0.806057	49.69342
min	1.000000	0.000000	1.000000	0.420000	0.000000	0.000000	0.00000
25%	223.500000	0.000000	2.000000	20.125000	0.000000	0.000000	7.91040
50%	446.000000	0.000000	3.000000	28.000000	0.000000	0.000000	14.45420
75%	668.500000	1.000000	3.000000	38.000000	1.000000	0.000000	31.00000
max	891.000000	1.000000	3.000000	80.000000	8.000000	6.000000	512.32920

Simplifying our data

In [17]:
```
def simplify(df: pd.DataFrame):
    del df['PassengerId']
    del df['Name']
    del df['Pclass']
    df['Sex'] = (df['Sex'].values == 'male').astype(int)
    mean_age = np.mean(df['Age'].values[~np.isnan(df['Age'].values)
    df['Age'] = [mean_age if np.isnan(age) else age for age in df['
    del df['Ticket']
    del df['Cabin']
    mean_fare = np.mean(df['Fare'].values[~np.isnan(df['Fare'].valu
    df['Fare'] = [mean_fare if np.isnan(fare) else fare for fare in
    df['S'] = (df['Embarked'].values == 'S').astype(int) + df['Emba
    del df['Embarked']
```

In [18]:
```
labels = df['Survived'].values
del df['Survived']

simplify(df)
```

In [19]:
```
df
```

Out[19]:

	Sex	Age	SibSp	Parch	Fare	S
0	1	22.000000	1	0	7.2500	1
1	0	38.000000	1	0	71.2833	0
2	0	26.000000	0	0	7.9250	1
3	0	35.000000	1	0	53.1000	1
4	1	35.000000	0	0	8.0500	1
...	...	...	...	...	...	...
886	1	27.000000	0	0	13.0000	1
887	0	19.000000	0	0	30.0000	1
888	0	29.699118	1	2	23.4500	1
889	1	26.000000	0	0	30.0000	0
890	1	32.000000	0	0	7.7500	0

891 rows × 6 columns

In [20]:
```
df.describe()
```

Out[20]:

	Sex	Age	SibSp	Parch	Fare	S
count	891.000000	891.000000	891.000000	891.000000	891.000000	891.000000
mean	0.647587	29.699118	0.523008	0.381594	32.204208	0.725028
std	0.477990	13.002015	1.102743	0.806057	49.693429	0.446751
min	0.000000	0.420000	0.000000	0.000000	0.000000	0.000000
25%	0.000000	22.000000	0.000000	0.000000	7.910400	0.000000
50%	1.000000	29.699118	0.000000	0.000000	14.454200	1.000000
75%	1.000000	35.000000	1.000000	0.000000	31.000000	1.000000
max	1.000000	80.000000	8.000000	6.000000	512.329200	1.000000

In [21]:
```
def train_test_split(x: np.ndarray, y: np.ndarray, train_ratio: flo
    '''
    Returns: tuple of form (x_train, y_train, x_test, y_test)
    '''
    n = x.shape[0]
    train_size = int(n * train_ratio)

    train_indices = np.random.choice(n, train_size)
    test_indices = [i for i in np.arange(n) if i not in train_indic

    x_train = np.array([x[i] for i in train_indices])
    y_train = np.array([y[i] for i in train_indices])
    x_test = np.array([x[i] for i in test_indices])
    y_test = np.array([y[i] for i in test_indices])

    return (x_train, y_train, x_test, y_test)
```

Implementing Maximum Likelihood Classification (MLClassifier)

I will use here a maximum likelihood classifier that assumes each observation is a random vector with a Multivariate Gaussian Distributuin:

$$f(x) = \frac{1}{\sqrt{(2\pi)^d \det \Sigma}} \cdot e^{-\frac{1}{2}(x-\mu)^T \Sigma^{-1} (x-\mu)}$$

where:

$$\begin{aligned} x &= \text{a column vector with data from one observation} \\ d &= \text{dimension of x (x is a } d \times 1 \text{ vector)} \\ \mu &= \text{mean of x (also } d \times 1) \\ \Sigma &= \text{covariance matrix of x (} d \times d \text{)} \end{aligned}$$

I will make the assumption that each class in our dataset (Survived / Not Survived) has different mean μ and variance Σ.

Training this model will consist mainly in the following:

- first split the dataset into Survived, Not Survived
- compute μ and Σ for each of these two classes

When making a prediction:

- plug input x and the computed μ and Σ into the Gaussian PDF (the formula above) for each class
- output y for the class with the highest value for PDF computed at previous step (y that maximizes the likelihood of our data vector x)

For this method to work the covariance matrix Σ should be **positive definite**. We will check in our code and show a warning if it is not.

In [22]:
```
class MLClassifier:
    def fit(self, x: np.ndarray, y: np.ndarray):
        self.d = x.shape[1] # no. of variables / dimensions
        self.nclasses = len(set(y))

        self.mu_list = []
        self.sigma_list = []

        n = x.shape[0] # no. of observations
        for i in range(self.nclasses):
            cls_x = np.array([x[j] for j in range(n) if y[j] == i])
            mu = np.mean(cls_x, axis=0)
            sigma = np.cov(cls_x, rowvar=False)
            self.mu_list.append(mu)
            self.sigma_list.append(sigma)

    def _class_likelihood(self, x: np.ndarray, cls: int) -> float:
        mu = self.mu_list[cls]
        sigma = self.sigma_list[cls]
        if np.sum(np.linalg.eigvals(sigma) <= 0) != 0:
            print(f'Warning! Covariance matrix for label {cls} is n
            print('The predicted likelihood will be 0.')
            return 0.0
        d = self.d

        #Task 3: Compute function f(x) given in description above

        exp = (-1/2)*np.dot(np.matmul(x-mu, sigma), x-mu)
        s_val = np.linalg.inv(sigma)
        c =  c = 1/np.sqrt(((2*np.pi)**self.d)*np.linalg.det(sigma)




        return c * (np.e**exp)

    def predict(self, x: np.ndarray) -> int:
        likelihoods = [self._class_likelihood(x, i) for i in range(
        return np.argmax(likelihoods)

    def score(self, x: np.ndarray, y: np.ndarray):
        n = x.shape[0]
        predicted_y = np.array([self.predict(x[i]) for i in range(n
        n_correct = np.sum(predicted_y == y)
        return n_correct/n
```

In [23]:
```
(x_train, y_train, x_test, y_test) = train_test_split(df.values, la
```

In [24]:
```
# Task 4: Call Maximum Likelihood classifier function

mlc = MLClassifier()



#Task 5: fit the Maximum Likelihood classifier for train test data

mlc.fit(x_train,y_train)
```

In [25]:
```
score = mlc.score(x_test,y_test)# pass first parameter, # pass seco

print(score)

# o/ p is shown below
```

```
0.6414634146341464
```