Problem Statement: -

In this case study, you have been given Twitter data collected from an anonymous twitter handle. With the help of a Naïve Bayes model, predict if a given tweet about a real disaster is real or fake.
1 = real tweet and 0 = fake tweet

Data Pre-processing

In [37]:
```
#Problem Statement: -
#In this case study, you have been given Twitter data collected from an anonymous
#1 = real tweet and 0 = fake tweet

import pandas as pd
import numpy as np
from sklearn.feature_extraction.text import CountVectorizer,TfidfTransformer

# Loading the data set
twitter_data = pd.read_csv("D:\\360Digi\\naive bayes\\Disaster_tweets_NB.csv")


import re
stop_words = []
# Load the custom built Stopwords
with open("D:/360Digi/Machine learning/Text mining/stopwords_en.txt","r") as sw:
    stop_words = sw.read()

stop_words = stop_words.split("\n")
```

In [36]:

```
def cleaning_text(i):
    i = re.sub("[^A-Za-z" "]+"," ",i).lower()
    i = re.sub("[0-9" "]+"," ",i)
    w = []
    for word in i.split(" "):
        if len(word)>3:
            w.append(word)
    return (" ".join(w))

# testing above function with sample text => removes punctuations, n

twitter_data.text = twitter_data.text.apply(cleaning_text)
twitter_data.text
```

Out[36]:

```
0               deeds reason this earthquake allah forgive
1                       forest fire near ronge sask canada
2        residents asked shelter place being notified o...
3        people receive wildfires evacuation orders cal...
4        just sent this photo from ruby alaska smoke fr...
                               ...
7608     giant cranes holding bridge collapse into near...
7609     aria ahrary thetawniest control wild fires cal...
7610                             volcano hawaii http zdtoyd
7611     police investigating after bike collided with ...
7612     latest more homes razed northern california wi...
Name: text, Length: 7613, dtype: object
```

In [33]:

```
# removing empty rows
twitter_data = twitter_data.loc[twitter_data.text != " ",:]

# CountVectorizer
# Convert a collection of text documents to a matrix of token counts

# splitting data into train and test data sets
from sklearn.model_selection import train_test_split

twitter_train, twitter_test = train_test_split(twitter_data, test_size = 0.2)
twitter_data.head()
```

Out[33]:

	id	keyword	location	text	target
0	1	NaN	NaN	deeds reason this earthquake allah forgive	1
1	4	NaN	NaN	forest fire near ronge sask canada	1
2	5	NaN	NaN	residents asked shelter place being notified o...	1
3	6	NaN	NaN	people receive wildfires evacuation orders cal...	1
4	7	NaN	NaN	just sent this photo from ruby alaska smoke fr...	1

In [28]:
```
y=twitter_data.iloc[:,4]
X=twitter_data.iloc[:,3]
```

In [29]:
```

# creating a matrix of token counts for the entire text document
def split_into_words(i):
    return [word for word in i.split(" ")]

# Defining the preparation of twiiter texts into word count matrix format - Bag o
twitter_bow = CountVectorizer(analyzer = split_into_words).fit(twitter_data.text)

# Defining BOW for all messages
all_twitter_matrix = twitter_bow.transform(twitter_data.text)

# For training messages
train_twitter_matrix = twitter_bow.transform(twitter_train.text)

# For testing messages
test_twitter_matrix = twitter_bow.transform(twitter_test.text)

# Learning Term weighting and normalizing on entire emails
tfidf_transformer = TfidfTransformer().fit(all_twitter_matrix)

# Preparing TFIDF for train emails
train_tfidf = tfidf_transformer.transform(train_twitter_matrix)
train_tfidf.shape # (row, column)

# Preparing TFIDF for test emails
test_tfidf = tfidf_transformer.transform(test_twitter_matrix)
test_tfidf.shape #  (row, column)
```

Out[29]: (1523, 19280)

Model Building

In [30]:

```
# Preparing a naive bayes model on training data set

from sklearn.naive_bayes import MultinomialNB as MB

# Multinomial Naive Bayes
classifier_mb = MB()
classifier_mb.fit(train_tfidf, twitter_train.target)

# Evaluation on Test Data
test_pred_m = classifier_mb.predict(test_tfidf)
test_pred_m
accuracy_test_m = np.mean(test_pred_m == twitter_test.target)
accuracy_test_m

from sklearn.metrics import accuracy_score
accuracy_score(test_pred_m, twitter_test.target)

pd.crosstab(test_pred_m, twitter_test.target)

# Training Data accuracy
train_pred_m = classifier_mb.predict(train_tfidf)
accuracy_train_m = np.mean(train_pred_m == twitter_train.target)
accuracy_train_m


print("accuracy_test", accuracy_test_m)
print("Crosstab",pd.crosstab(test_pred_m, twitter_test.target))
print("accuracy_train",accuracy_train_m)
print("crosstab train",pd.crosstab(train_pred_m, twitter_train.target))

# Multinomial Naive Bayes changing default alpha for laplace smoothing
# if alpha = 0 then no smoothing is applied and the default alpha parameter is 1
# the smoothing process mainly solves the emergence of zero probability problem i
```

```
accuracy_test 0.8023637557452397
Crosstab target    0    1
row_0
0        780  228
1         73  442
accuracy_train 0.9044334975369458
crosstab train target     0     1
row_0
0        3422   515
1          67  2086
```

In [31]:

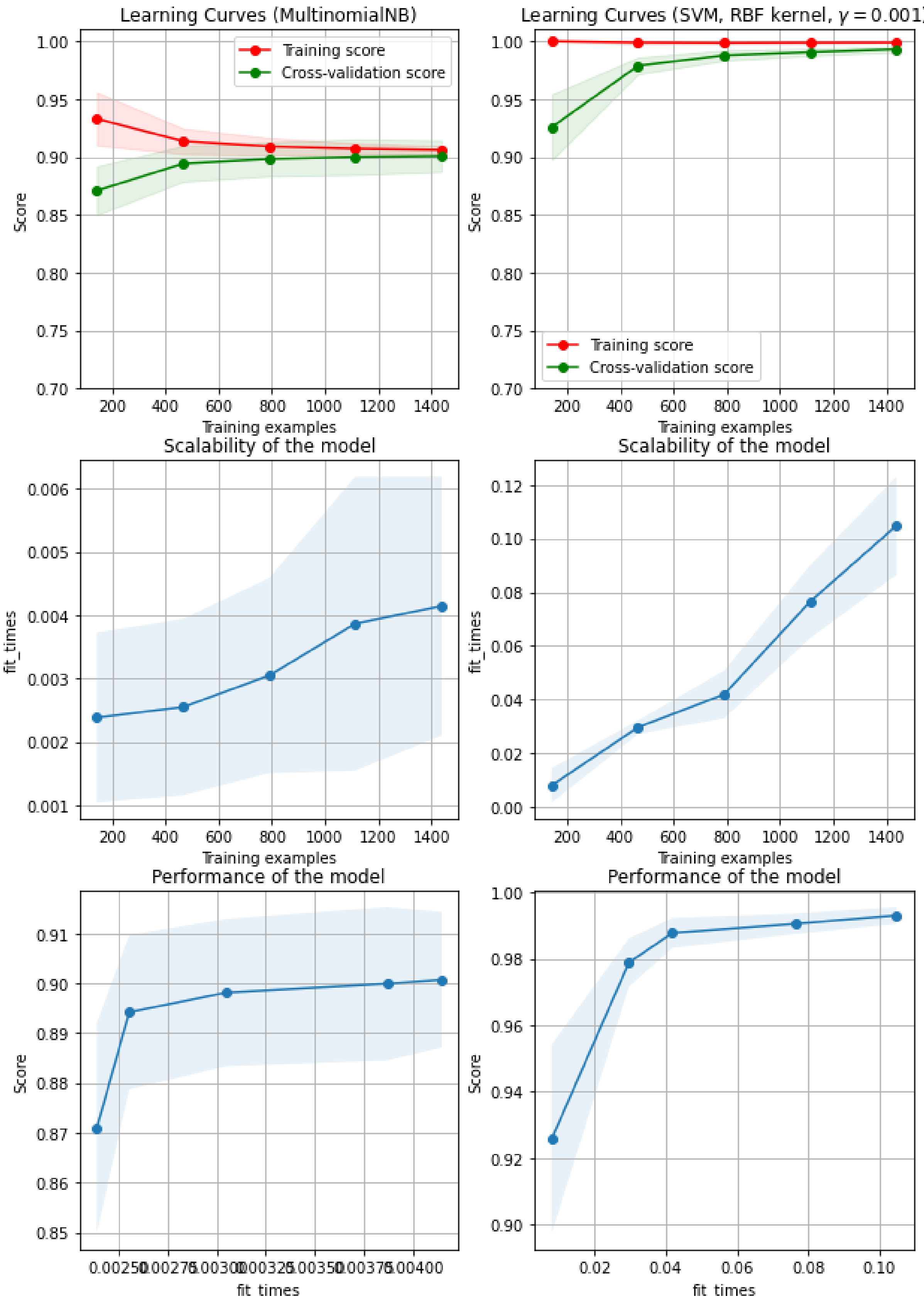
```
classifier_mb_lap = MB(alpha = 13)
classifier_mb_lap.fit(train_tfidf, twitter_train.target)

# Evaluation on Test Data after applying laplace
test_pred_lap = classifier_mb_lap.predict(test_tfidf)
accuracy_test_lap = np.mean(test_pred_lap == twitter_test.target)
accuracy_test_lap

from sklearn.metrics import accuracy_score
accuracy_score(test_pred_lap, twitter_test.target)

pd.crosstab(test_pred_lap, twitter_test.target)

# Training Data accuracy
train_pred_lap = classifier_mb_lap.predict(train_tfidf)
accuracy_train_lap = np.mean(train_pred_lap == twitter_train.target)
accuracy_train_lap

print("accuracy_test", accuracy_test_lap)
print("Crosstab test",pd.crosstab(test_pred_lap, twitter_test.target))
print("accuracy_train",accuracy_train_lap)
print("crosstab train",pd.crosstab(train_pred_lap, twitter_train.target))
```

```
accuracy_test 0.7045305318450427
Crosstab test target    0    1
row_0
0        841  438
1         12  232
accuracy_train 0.7408866995073892
crosstab train target     0     1
row_0
0        3479  1568
1          10  1033
```

In [32]:
```
import matplotlib.pyplot as plt
from matplotlib.colors import ListedColormap
import matplotlib.pyplot as mtp
import numpy as nm
from sklearn.metrics import confusion_matrix, classification_report, accuracy_sc
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import ShuffleSplit
from sklearn.model_selection import learning_curve
from sklearn.datasets import load_digits
from sklearn.svm import SVC
from sklearn.naive_bayes import GaussianNB
import matplotlib.pyplot as plt
from sklearn.naive_bayes import BernoulliNB
from sklearn.model_selection import train_test_split
import pandas as pd
import numpy as np
from sklearn.feature_extraction.text import CountVectorizer, TfidfTransformer


def plot_learning_curve(estimator, title, X, y, axes=None, ylim=None, cv=None,
                        n_jobs=None, train_sizes=np.linspace(.1, 1.0, 5)):
    """
    Generate 3 plots: the test and training learning curve, the training
    samples vs fit times curve, the fit times vs score curve.

    Parameters
    ----------
    estimator : estimator instance
        An estimator instance implementing `fit` and `predict` methods which
        will be cloned for each validation.

    title : str
        Title for the chart.

    X : array-like of shape (n_samples, n_features)
        Training vector, where ``n_samples`` is the number of samples and
        ``n_features`` is the number of features.

    y : array-like of shape (n_samples) or (n_samples, n_features)
        Target relative to ``X`` for classification or regression;
        None for unsupervised learning.

    axes : array-like of shape (3,), default=None
        Axes to use for plotting the curves.

    ylim : tuple of shape (2,), default=None
        Defines minimum and maximum y-values plotted, e.g. (ymin, ymax).

    cv : int, cross-validation generator or an iterable, default=None
        Determines the cross-validation splitting strategy.
        Possible inputs for cv are:

          - None, to use the default 5-fold cross-validation,
          - integer, to specify the number of folds.
          - :term:`CV splitter`,
          - An iterable yielding (train, test) splits as arrays of indices.
```

```
        For integer/None inputs, if ``y`` is binary or multiclass,
        :class:`StratifiedKFold` used. If the estimator is not a classifier
        or if ``y`` is neither binary nor multiclass, :class:`KFold` is used.

        Refer :ref:`User Guide <cross_validation>` for the various
        cross-validators that can be used here.

    n_jobs : int or None, default=None
        Number of jobs to run in parallel.
        ``None`` means 1 unless in a :obj:`joblib.parallel_backend` context.
        ``-1`` means using all processors. See :term:`Glossary <n_jobs>`
        for more details.

    train_sizes : array-like of shape (n_ticks,)
        Relative or absolute numbers of training examples that will be used to
        generate the learning curve. If the ``dtype`` is float, it is regarded
        as a fraction of the maximum size of the training set (that is
        determined by the selected validation method), i.e. it has to be within
        (0, 1]. Otherwise it is interpreted as absolute sizes of the training
        sets. Note that for classification the number of samples usually have
        to be big enough to contain at least one sample from each class.
        (default: np.linspace(0.1, 1.0, 5))
    """
    if axes is None:
        _, axes = plt.subplots(1, 3, figsize=(20, 5))

    axes[0].set_title(title)
    if ylim is not None:
        axes[0].set_ylim(*ylim)
    axes[0].set_xlabel("Training examples")
    axes[0].set_ylabel("Score")

    train_sizes, train_scores, test_scores, fit_times, _ = \
        learning_curve(estimator, X, y, cv=cv, n_jobs=n_jobs,
                       train_sizes=train_sizes,
                       return_times=True)
    train_scores_mean = np.mean(train_scores, axis=1)
    train_scores_std = np.std(train_scores, axis=1)
    test_scores_mean = np.mean(test_scores, axis=1)
    test_scores_std = np.std(test_scores, axis=1)
    fit_times_mean = np.mean(fit_times, axis=1)
    fit_times_std = np.std(fit_times, axis=1)

    # Plot learning curve
    axes[0].grid()
    axes[0].fill_between(train_sizes, train_scores_mean - train_scores_std,
                         train_scores_mean + train_scores_std, alpha=0.1,
                         color="r")
    axes[0].fill_between(train_sizes, test_scores_mean - test_scores_std,
                         test_scores_mean + test_scores_std, alpha=0.1,
                         color="g")
    axes[0].plot(train_sizes, train_scores_mean, 'o-', color="r",
                 label="Training score")
    axes[0].plot(train_sizes, test_scores_mean, 'o-', color="g",
                 label="Cross-validation score")
    axes[0].legend(loc="best")
```

```
    # Plot n_samples vs fit_times
    axes[1].grid()
    axes[1].plot(train_sizes, fit_times_mean, 'o-')
    axes[1].fill_between(train_sizes, fit_times_mean - fit_times_std,
                         fit_times_mean + fit_times_std, alpha=0.1)
    axes[1].set_xlabel("Training examples")
    axes[1].set_ylabel("fit_times")
    axes[1].set_title("Scalability of the model")

    # Plot fit_time vs score
    axes[2].grid()
    axes[2].plot(fit_times_mean, test_scores_mean, 'o-')
    axes[2].fill_between(fit_times_mean, test_scores_mean - test_scores_std,
                         test_scores_mean + test_scores_std, alpha=0.1)
    axes[2].set_xlabel("fit_times")
    axes[2].set_ylabel("Score")
    axes[2].set_title("Performance of the model")

    return plt


fig, axes = plt.subplots(3, 2, figsize=(10, 15))

X, y = load_digits(return_X_y=True)

title = "Learning Curves (MultinomialNB)"
# Cross validation with 100 iterations to get smoother mean test and train
# score curves, each time with 20% data randomly selected as a validation set.
cv = ShuffleSplit(n_splits=100, test_size=0.2, random_state=0)

estimator = MB()
plot_learning_curve(estimator, title, X, y, axes=axes[:, 0], ylim=(0.7, 1.01),
                    cv=cv, n_jobs=4)

title = r"Learning Curves (SVM, RBF kernel, $\gamma=0.001$)"
# SVC is more expensive so we do a lower number of CV iterations:
cv = ShuffleSplit(n_splits=10, test_size=0.2, random_state=0)
estimator = SVC(gamma=0.001)
plot_learning_curve(estimator, title, X, y, axes=axes[:, 1], ylim=(0.7, 1.01),
                    cv=cv, n_jobs=4)

plt.show()


####
```


Learning Curves (MultinomialNB)
Training score
Cross-validation score
Score
Training examples
Learning Curves (SVM, RBF kernel, $\gamma = 0.001$)
Training score
Cross-validation score
Score
Training examples
Scalability of the model
fit_times
Training examples
Scalability of the model
fit_times
Training examples
Performance of the model
Score
fit_times
Performance of the model
Score
fit_times

Summary:

```
The accuaracy of the test is good 70% as false negative values is less.

The accuarcy of the training is also good 74% as false negative values i
s 10 after tunning the parameter with alpha as before it was overfittin
g.

Multinomial Naïve Bayes uses term frequency i.e. the number of times a g
iven term appears in a document. ... After normalization, term frequency
can be used to compute maximum likelihood estimates based on the trainin
g data to estimate the conditional probability
```

In [ ]:

In [ ]: