Entendendo a base de dados.

O conjunto de dados fornecido contém dados de tweets nas alças de twitter de várias companhias aéreas.

Ele contém um total de 12 colunas, das quais uma coluna especifica o sentimento do tweet. Todas as outras colunas fornecem várias informações relacionadas ao que foi o tweet, de onde foi postado, quando foi postado, é retuitado; etc.

Data Description

Description of columns of the dataset is given below -

tweet_id -- Id of the tweet

airline_sentiment -- Sentiment of the tweet (Target variable)

airline_sentiment_confidence -- Confidence with which the given sentiment was determined

negativereason_confidence -- Confidence with which the negative reason of tweet was predicted

name -- Name of the person who tweeted

retweet_count -- Number of retweets

text -- Text of the tweet whose sentiment has to be predicted

tweet_created -- Time at which the tweet was created

tweet_location -- Location from where the tweet was posted

user_timezone -- Time zone from where the tweet was posted

negativereason -- Reason for which user posted a negative tweet

airline -- Airline for which the tweet was posted

In [1]:

```
import numpy as np
import pandas as pd
import re
import nltk
import matplotlib.pyplot as plt
%matplotlib inline
```

In [2]:

```
data_source_url = 'Tweets.csv'
airline_tweets = pd.read_csv(data_source_url)
```

Precisamos Fazer um pre-processamento desse texto, para isso vamos tirar pontos, virgulas, underlines e etc.

In [3]:

```
features = airline_tweets.iloc[:, 10].values#Pegando os textos
labels = airline_tweets.iloc[:, 1].values#classes
```

In [4]:

```
processed_features = []

for sentence in range(0, len(features)):
    # Remove todos os caracteres especiais
    processed_feature = re.sub(r'\W', ' ', str(features[sentence]))

    # remova todos os caracteres únicos
    processed_feature= re.sub(r'\s+[a-zA-Z]\s+', ' ', processed_feature)

    # Remova caracteres únicos desde o início
    processed_feature = re.sub(r'\^[a-zA-Z]\s+', ' ', processed_feature)

    # substitui multiplos espaços
    processed_feature = re.sub(r'\s+', ' ', processed_feature, flags=re.I)

    # Removing prefixed 'b'
    processed_feature = re.sub(r'^b\s+', '', processed_feature)

    # Converting to caixa baixa
    processed_feature = processed_feature.lower()

    processed_features.append(processed_feature)
```

In [5]:

```
#Doc1 = "I like to play football"
#Doc2 = "It is a good game"
#Doc3 = "I prefer football over rugby"

#feature vector

#Vocab = [I, like, to, play, football, it, is, a, good, game, prefer, over, rugb
y]
#[1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0]
```

In [6]:

```
#nltk.download("stopwords")
```

In [7]:

```
from nltk.corpus import stopwords


from sklearn.feature_extraction.text import TfidfVectorizer

vectorizer = TfidfVectorizer(max_features=2500, min_df=7, max_df=0.8, stop_words
=stopwords.words('english'))

processed_features = vectorizer.fit_transform(processed_features).toarray()
```

max_dffloat or int, default=1.0

When building the vocabulary ignore terms that have a document frequency strictly higher than the given threshold (corpus-specific stop words). If float in range [0.0, 1.0], the parameter represents a proportion of documents, integer absolute counts. This parameter is ignored if vocabulary is not None.

min_dffloat or int, default=1

When building the vocabulary ignore terms that have a document frequency strictly lower than the given threshold. This value is also called cut-off in the literature. If float in range of [0.0, 1.0], the parameter represents a proportion of documents, integer absolute counts. This parameter is ignored if vocabulary is not None.

max_features int, default=None

If not None, build a vocabulary that only consider the top max_features ordered by term frequency across the corpus. This parameter is ignored if vocabulary is not None.

In [8]:

```
# separando o modelo em treino e text
from sklearn.metrics import accuracy_score
from sklearn.metrics import confusion_matrix
from sklearn.metrics import accuracy_score
from sklearn.metrics import classification_report
```

In [9]:

```
from sklearn.model_selection import train_test_split

# Nessa situação, vamos separara train,test and validation.
X_train, X_test, y_train, y_test = train_test_split(processed_features, labels,
test_size=0.2, random_state=0)
```

In [10]:

```
from sklearn.linear_model import LogisticRegression
from sklearn.neighbors import KNeighborsClassifier
from sklearn.svm import SVC, LinearSVC, NuSVC
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier, AdaBoostClassifier, Gradien
tBoostingClassifier
from sklearn.naive_bayes import GaussianNB
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis
from sklearn.discriminant_analysis import QuadraticDiscriminantAnalysis
from sklearn.metrics import accuracy_score
```

In [11]:

```
from sklearn.metrics import confusion_matrix
from sklearn.metrics import accuracy_score
from sklearn.metrics import classification_report
```

In [12]:

```
def plot_confusion_matrix(cm, classes,
                          normalize=False,
                          title='Confusion matrix',
                          cmap=plt.cm.Blues):
    plt.imshow(cm, interpolation='nearest', cmap=cmap)
    plt.title(title)
    plt.colorbar()
    tick_marks = np.arange(len(classes))
    plt.xticks(tick_marks, classes, rotation=45)
    plt.yticks(tick_marks, classes)
    if normalize:
        cm = cm.astype('float') / cm.sum(axis=1)[:, np.newaxis]

    thresh = cm.max() / 2.
    for i, j in itertools.product(range(cm.shape[0]), range(cm.shape[1])):
        plt.text(j, i, cm[i, j],
                 horizontalalignment="center",
                 color="white" if cm[i, j] > thresh else "black")

    plt.tight_layout()
    plt.ylabel('True label')
    plt.xlabel('Predicted label')
```

In [13]:

```
## k-means
from sklearn.cluster import KMeans
kmeans = KMeans(n_clusters=3, random_state=0).fit(X_train)
```

In [14]:

```
print(__doc__)

from time import time
import numpy as np
import matplotlib.pyplot as plt

from sklearn import metrics
from sklearn.cluster import KMeans
from sklearn.datasets import load_digits
from sklearn.decomposition import PCA
from sklearn.preprocessing import scale

np.random.seed(42)

data = scale(X_train)

n_samples, n_features = data.shape
n_digits = len(np.unique(y_train))
labels = y_train

sample_size = 300

print("n_digits: %d, \t n_samples %d, \t n_features %d"
      % (n_digits, n_samples, n_features))


print(82 * '_')
print('init\t\ttime\tinertia\thomo\tcompl\tv-meas\tARI\tAMI\tsilhouette')


def bench_k_means(estimator, name, data):
    t0 = time()
    estimator.fit(data)
    print('%-9s\t%.2fs\t%i\t%.3f\t%.3f\t%.3f\t%.3f\t%.3f\t%.3f'
          % (name, (time() - t0), estimator.inertia_,
             metrics.homogeneity_score(labels, estimator.labels_),
             metrics.completeness_score(labels, estimator.labels_),
             metrics.v_measure_score(labels, estimator.labels_),
             metrics.adjusted_rand_score(labels, estimator.labels_),
             metrics.adjusted_mutual_info_score(labels,  estimator.labels_),
             metrics.silhouette_score(data, estimator.labels_,
                                      metric='euclidean',
                                      sample_size=sample_size)))

bench_k_means(KMeans(init='k-means++', n_clusters=3, n_init=10),
              name="k-means++", data=data)

bench_k_means(KMeans(init='random', n_clusters=3, n_init=10),
              name="random", data=data)


pca = PCA(n_components=n_digits).fit(data)
bench_k_means(KMeans(init=pca.components_, n_clusters=3, n_init=1),
              name="PCA-based",
              data=data)
```

```
Automatically created module for IPython interactive environment
n_digits: 3,     n_samples 11712,        n_features 2301
_____________________________________________________________________

______________
init            time    inertia homo    compl   v-meas  ARI     AMI
silhouette
k-means++       8.93s   26904994        0.008   0.016   0.011   -0.0
19      0.011   0.002
random          10.02s  26910058        0.067   0.107   0.083   -0.0
57      0.082   -0.025
PCA-based       2.93s   26901659        0.107   0.091   0.099   0.07
4       0.099   -0.004
```

In [ ]: