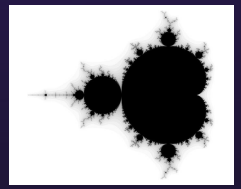
# Twitter Sentiment Analysis + IMDb recommendations

Robert Farzan Rodríguez

# 01

# LIBRARIES USED

Machine Learning models, Natural Language Processing, Twitter API and visualizations



## TextBlob

NLP Processing and classification

## SciKit-Learn

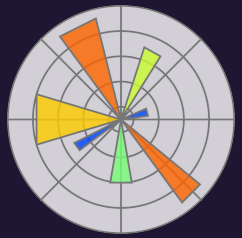
For training ML models

## NLTK

NLP: corpus, tokenization, lemmatization etc.

## Pandas

For data manipulation and analysis



## Matplotlib

Visualization library

## Tweepy

Twitter API for extracting tweets information

# TWO WAYS OF IMPLEMENTING SENTIMENT ANALYSIS

We can perform tweets sentiment analysis with the default models of TextBlob or developing our own models

## TextBlob approach

We use this library's **Naive Bayes analyzer**

We get a **sentiment *score*** as output

It takes **little effort**

It doesn't require text preprocessing before training

It's a **black box** and less efficient on testing

## Sklearn approach

We compare a **bunch of suitable models** for Sentiment Analysis

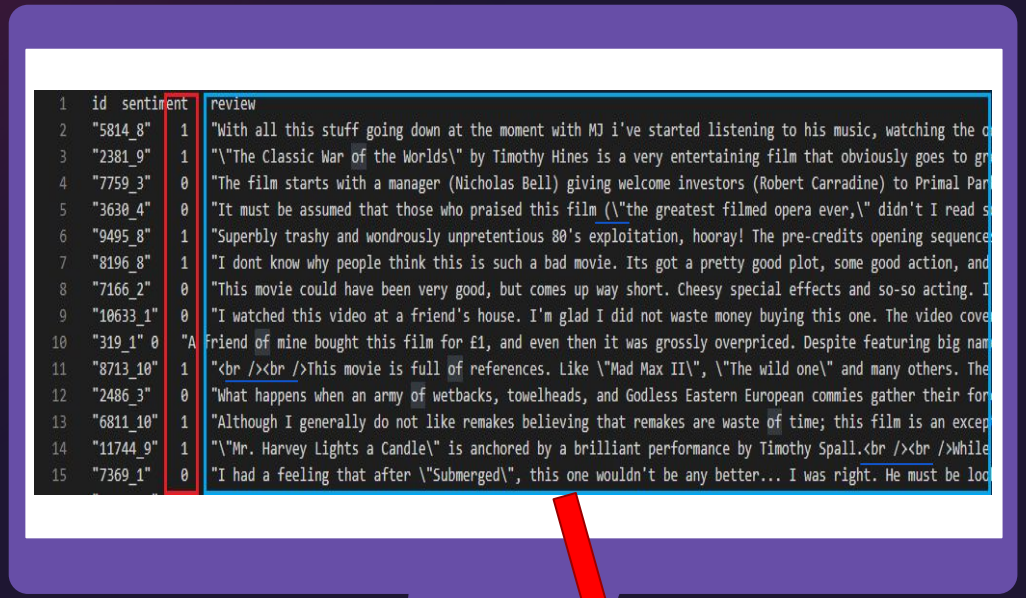
We get feedback from classifier's metrics

It's an exhaustive and harder approach

It does require text preprocessing

We can perform hyper-parameter tuning

# 02 ADVANCED SENTIMENT ANALYSIS



| | id | sentiment | review |
|---|---|---|---|
| 1 | "5814_8" | 1 | "With all this stuff going down at the moment with MJ i've started listening to his music, watching the o |
| 2 | "2381_9" | 1 | "\"The Classic War of the Worlds\" by Timothy Hines is a very entertaining film that obviously goes to gr |
| 3 | "7759_3" | 0 | "The film starts with a manager (Nicholas Bell) giving welcome investors (Robert Carradine) to Primal Par |
| 4 | "3630_4" | 0 | "It must be assumed that those who praised this film (\"the greatest filmed opera ever,\" didn't I read s |
| 5 | "9495_8" | 1 | "Superbly trashy and wondrously unpretentious 80's exploitation, hooray! The pre-credits opening sequence |
| 6 | "8196_8" | 1 | "I dont know why people think this is such a bad movie. Its got a pretty good plot, some good action, and |
| 7 | "7166_2" | 0 | "This movie could have been very good, but comes up way short. Cheesy special effects and so-so acting. I |
| 8 | "10633_1" | 0 | "I watched this video at a friend's house. I'm glad I did not waste money buying this one. The video cove |
| 9 | "319_1" | 0 | "A friend of mine bought this film for £1, and even then it was grossly overpriced. Despite featuring big nam |
| 10 | "8713_10" | 1 | "<br /><br />This movie is full of references. Like \"Mad Max II\", \"The wild one\" and many others. The |
| 11 | "2486_3" | 0 | "What happens when an army of wetbacks, towelheads, and Godless Eastern European commies gather their for |
| 12 | "6811_10" | 1 | "Although I generally do not like remakes believing that remakes are waste of time; this film is an excep |
| 13 | "11744_9" | 1 | "\"Mr. Harvey Lights a Candle\" is anchored by a brilliant performance by Timothy Spall.<br /><br />While |
| 14 | "7369_1" | 0 | "I had a feeling that after \"Submerged\", this one wouldn't be any better... I was right. He must be loo |

# TRAINING DATASET FOR SENTIMENT ANALYSIS

- Movies reviews dataset
- TSV (Tab Separated Values)
- 1=positive / 0=negative

**HTML tags, special characters?? That is not okay**

# Before anything else...

## Cleaning special characters and HTML tags

**Removing punctuation marks and HTML tags**

```
def clean_text(raw_text):
    clean = re.compile("<.*?>|([^A-Za-z'])|('s)")
    cleantext = re.sub(clean, ' ', raw_text)
    cleantext = " ".join(re.split('[!?\., ]', cleantext))
    cleantext = re.sub(r'\s+', ' ', cleantext)
    cleantext = re.sub("\s\W+\s", ' ', cleantext)
    return cleantext

#apply cleansing to all of the texts
train_tuple = list(map(lambda x: (clean_text(x[0]), x[1]), train_tuple))

print(train_tuple[0:1]) #print example
```

```
[("I watched the first minutes thinking it was a real documentary with an irritatingly overly dramatic on camera prod
ucer When I realized it was all staged I thought why would I want to waste my time watching this junk So I turned it
off and came online to warn other people The characters don't act in a believable way too much immature emotion for a
guy to travel half way around the world into a war torn country he acted like a kid and I don't believe it was becaus
e his character was so upset about the trade center bombings very trite and stupid have you seen city of lost childre
n french dark fantasy film about a guy who kidnaps kids and steals their dreams I liked it ", 0)]
```

# TextBlob approach

## Reading and splitting the dataset

### Train-test split and text cleansing

In [4]:
```
dataset = pd.read_csv('./movie_reviews.tsv', delimiter='\t')[:2000]
```

In [5]:
```
from sklearn.model_selection import train_test_split

X, y = dataset['review'], dataset['sentiment']
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.25, random_state=2)

#this is only necessary for formatting into a tuple shape to train the TextBlob classifiers
X_train2 = X_train.values.reshape(len(X_train), 1)
y_train2 = y_train.values.reshape(len(y_train), 1)

#tuple form: (X_train, y_train)
train_tuple = list(map(lambda x: (x[0], x[1]), np.hstack((X_train2, y_train2))))
```

# TextBlob approach

## Training the Naive Bayes classifier

Training the model with NB Classifier from TextBlob

```
from textblob import TextBlob
from textblob.classifiers import NaiveBayesClassifier, DecisionTreeClassifier

cl1 = NaiveBayesClassifier(train_tuple)
```

## Getting the predictions

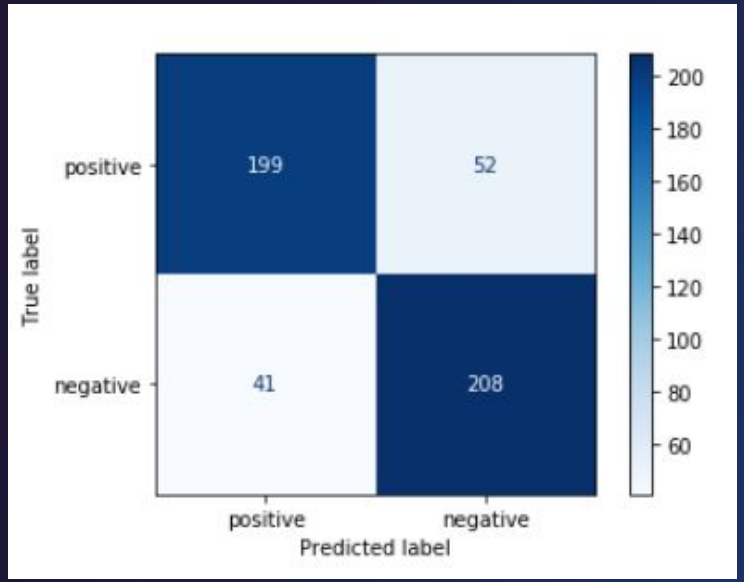
Classify each text and store the results in 1D-array to extract metrics afterwards

```
def classify(clf):
    y_pred = []
    for text in X_test:
        y_pred.append(clf.classify(text))

    return y_pred

y_pred1 = np.array(classify(cl1))
```

# TextBlob approach

## TextBlob model metrics

| | precision | recall | f1-score | support |
|---|---|---|---|---|
| positive | 0.83 | 0.79 | 0.81 | 251 |
| negative | 0.80 | 0.84 | 0.82 | 249 |
| accuracy | | | 0.81 | 500 |
| macro avg | 0.81 | 0.81 | 0.81 | 500 |
| weighted avg | 0.81 | 0.81 | 0.81 | 500 |



# TextBlob RESULTS

| | |
|---|---|
| ACCURACY | 81% |
| F1-SCORE | 81.5% |

# SKLearn approach!

**Can we beat TextBlob's model?**

Let's try a few models and compare them:

- Bayesian classifiers
- Non-bayesian classifiers: SVC, SGD Classifier, Random Forest and Logistic regression.

# THE (ALMOST) MOST IMPORTANT STEP

## TEXT PREPROCESSING

**Without cleaning and lemmatizing our texts, the performance of our models will DROP SUBSTANTIALLY!!**

# Text preprocessing

## Removing stopwords

```
def remove_stopwords(text):

    all_stopwords = stopwords.words('english')
    all_stopwords.remove('not')

    negation = False
    result = []
    delims = "?.,!:; "

    # no permitimos que las negaciones sean borradas de los textos
    for word in text.split():
        stripped = word.strip(delims).lower()
        negated = "not " + stripped if negation else stripped
        negated = re.sub("n\'t", " not", stripped)
        negated = re.sub("'ve", " have", stripped)
        result.append(negated)

        if any(neg in word for neg in ["not", "n't", "no"]):
            negation = not negation

        if any(c in word for c in delims):
            negation = False

    text = [word for word in result if not word in set(all_stopwords)]
    text = ' '.join(text)

    return text.lower()
```

**We allow stopwords like “not” and derivatives !!**

# Text preprocessing

Next step...

**Stem or Lemmatize? That is the question**

# Stemming vs Lemmatization

| Original | Stemming | Lemmatization |
|---|---|---|
| New | New | New |
| York | York | York |
| is | is | be |
| the | the | the |
| most | most | most |
| densely | dens | densely |
| populated | popul | populated |
| city | citi | city |
| in | in | in |
| the | the | the |
| United | Unite | United |
| States | State | States |

**Stemming:** it takes common prefixes and suffixes, and it cuts off the end or beginning of a word.

**Lemmatization:** takes into consideration the morphological analysis of the words. To do so, it needs detailed dictionaries.

# Stemming vs Lemmatization

**Our final decision will be... Lemmatization**

**Why?**

- Although **stemming** is quicker and doesn't require downloading dictionaries, **lemmatization** is better for **NLP purposes**, as it takes morphology and grammar rules into account.

- This may result in a performance boost in our models, but it is not 100% guaranteed.

# Text preprocessing

## Lemmatization

```
49  def lemmatize_text(text):
50
51      lem = WordNetLemmatizer()
52      tag_text = nltk.pos_tag(nltk.word_tokenize(text))
53      text = map(lambda x: (x[0], wordnet_tag(x[1])), tag_text)
54      lemmatized_sentence = []
55      for word, tag in text:
56          if tag is None:
57              lemmatized_sentence.append(word)
58          else:
59              lemmatized_sentence.append(lem.lemmatize(word, tag))
60
61      return " ".join(lemmatized_sentence)
62
```

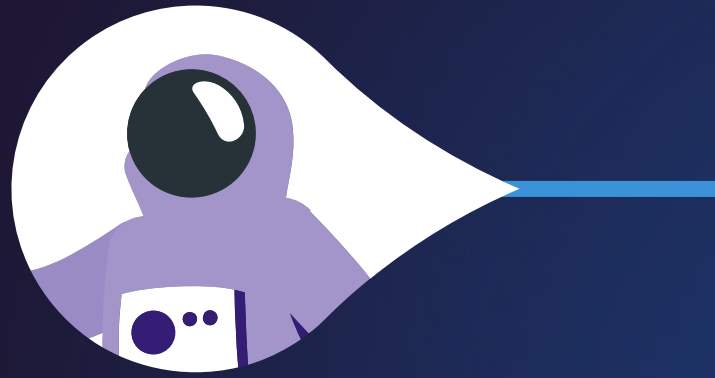
We lemmatize each word of the sentence and then join them together

# Text preprocessing

## Building the corpus

```
1 corpus = []
2 for rev in dataset['review']:
3     rev = preprocess_text(rev)
4     corpus.append(rev)
```

Join all the preprocessed sentences into a single dataset



# TF-IDF vs CountVectorizer

**TF-IDF is better for Sentiment Analysis. It gives a *score* to each term based on its importance in the corpus.**

**TF-IDF formula**

$$w_{i,j} = tf_{i,j} \times \log\left(\frac{N}{df_i}\right)$$

# TF-IDF

```
1  from sklearn.feature_extraction.text import TfidfVectorizer
2
3  tf = TfidfVectorizer()
4  X = tf.fit_transform(corpus).toarray()
5  y = dataset['sentiment'].values
6
7  X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.25, random_state=1)
8
9  print(len(tf.get_feature_names()))
```

```
21390
```

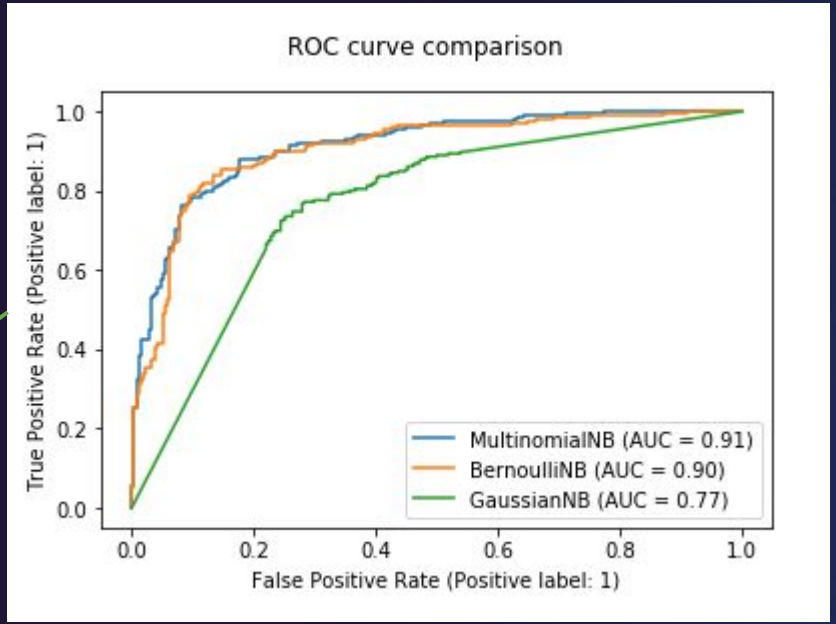
**It featured 21390 terms! An average person only uses around 3000 words. If we limit the *max_features*, we'll increase performance.**

# Bayesian algorithms

We will compare the following bayesian algorithms:

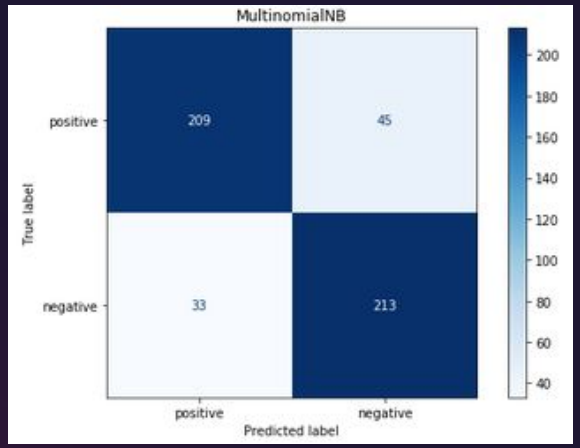
- MultinomialNB
- BernoulliNB
- GaussianNB

We can already discard GaussianNB



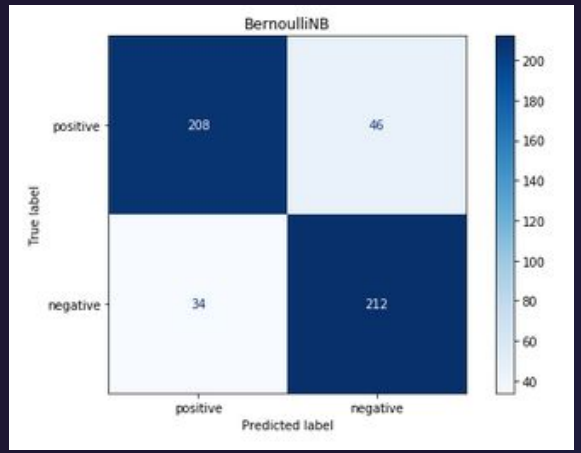
# Bayesian algorithms

## Metrics comparative



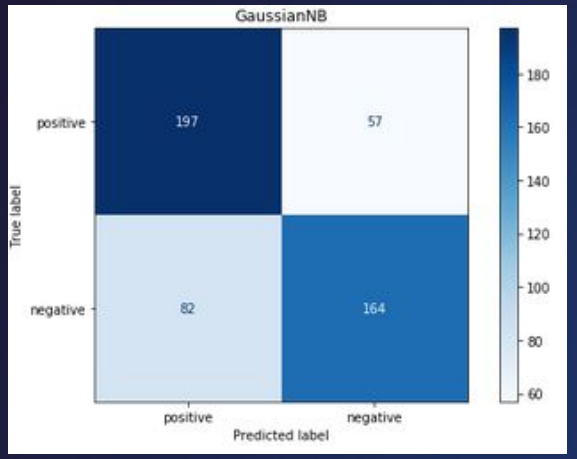

ACCURACY = 84%

F1-SCORE = 84,5%




ACCURACY = 84%

F1-SCORE = 84%




ACCURACY = 72%

F1-SCORE = 72%

# Bayesian algorithms

## How do we calculate those metrics?

```
from sklearn.metrics import classification_report

for cls in classifiers:
    y_pred = cls.predict(X_test)
    cr = classification_report(y_test, y_pred, target_names=['positive', 'negative'])
    print('********\t' + type(cls).__name__ + '\t********\n')
    print(cr)
```

=

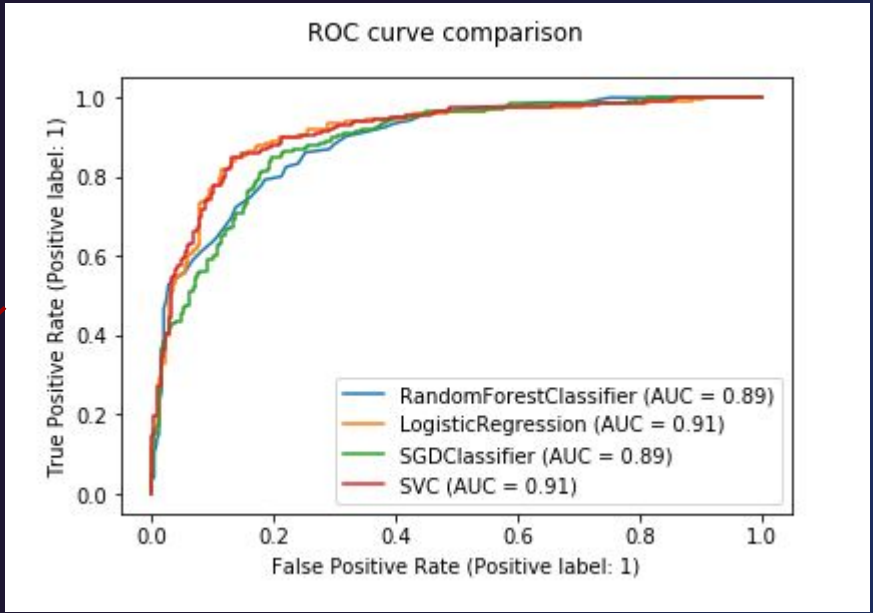
```
********        MultinomialNB    ********

              precision    recall  f1-score   support

    positive       0.86      0.82      0.84       254
    negative       0.83      0.87      0.85       246

    accuracy                           0.84       500
   macro avg       0.84      0.84      0.84       500
weighted avg       0.84      0.84      0.84       500
```

# Non-Bayesian algorithms

**We will compare the following non-bayesian classifiers:**

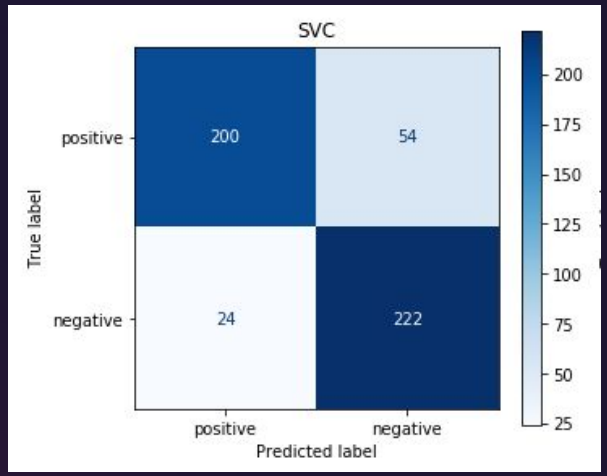
- Random Forest
- Logistic Regression
- Stochastic Gradient Descent
- Support Vector Machines

**They are so close!!**



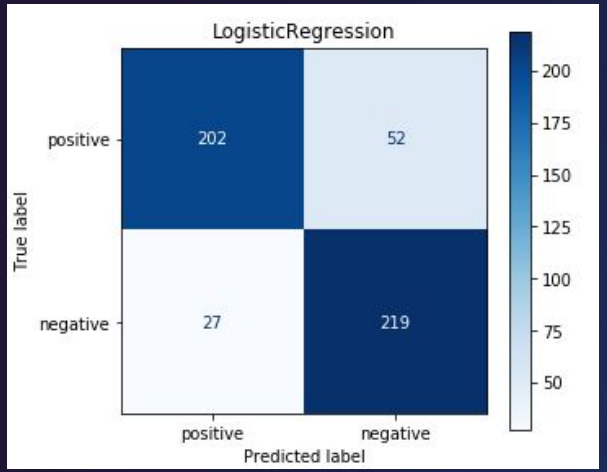
# Bayesian algorithms

## Metrics comparative




ACCURACY = 84%

F1-SCORE = 84,5%

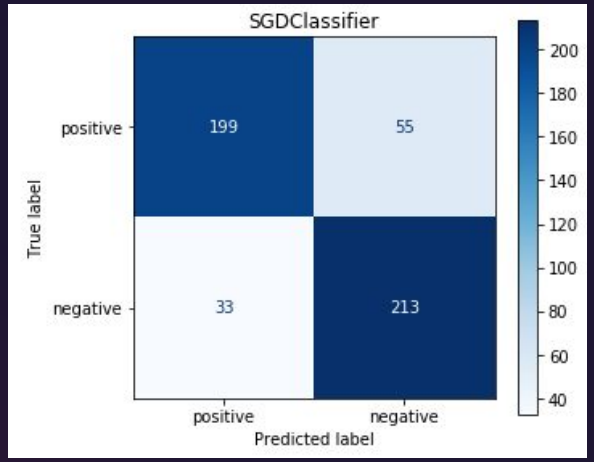



F1-SCORE = 84%

F1-SCORE = 84%

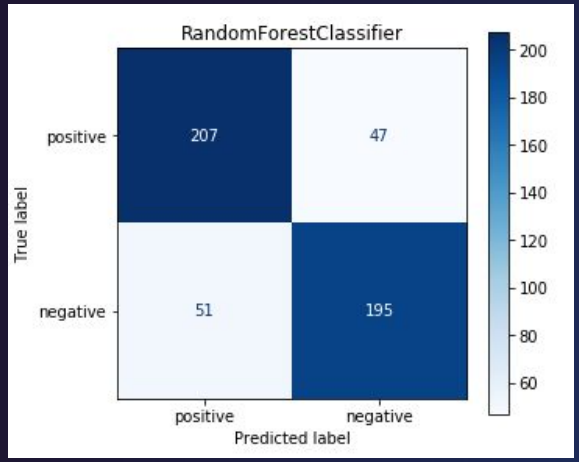
# Bayesian algorithms

## Metrics comparative




ACCURACY = 82%

F1-SCORE = 82,5%




ACCURACY = 80%

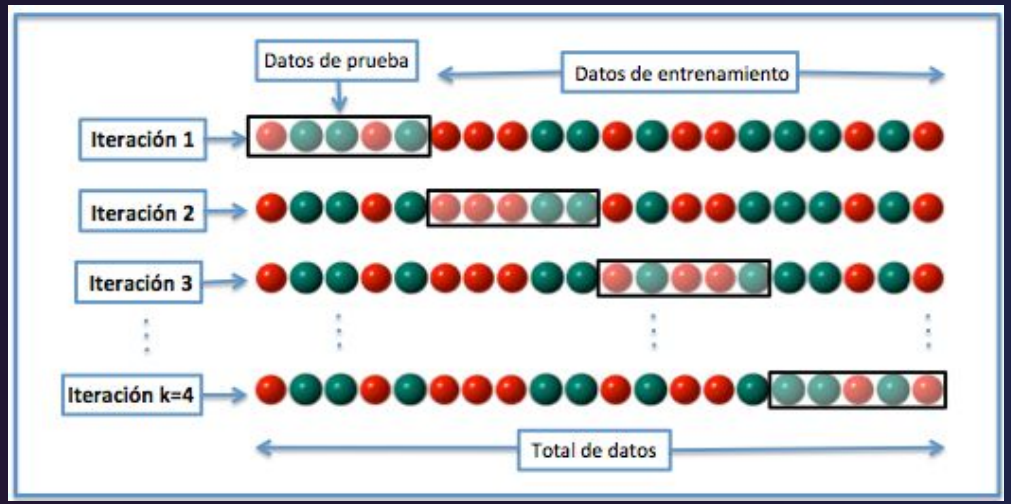
F1-SCORE = 80,5%

Are these metrics enough to choose our model?

**Not yet!**

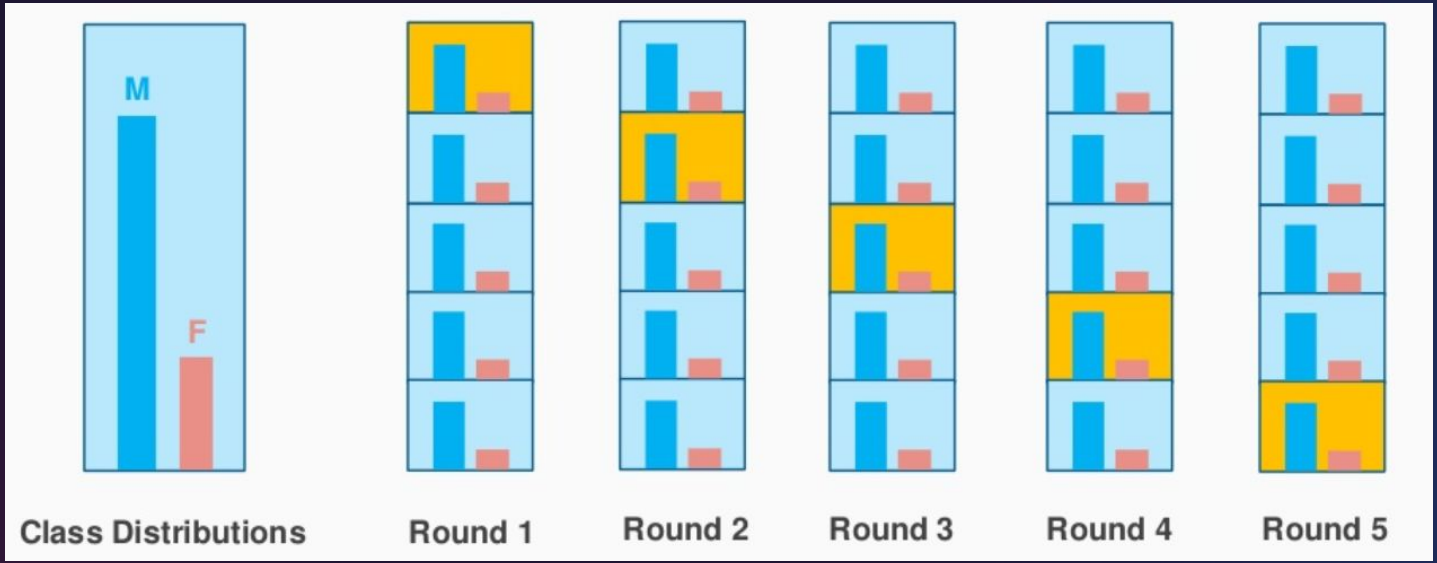
# Cross validation

**The goal of cross-validation is to test the model's ability to predict new data that was not used in estimating it and to give an insight on how the model will generalize to an independent dataset.**



# Cross validation

SkLearn provides stratified k-fold cross validation. This means each fold takes the same proportion of each class as there exists in the whole dataset.



# Cross validation

SkLearn provides stratified k-fold cross validation. This means each fold takes the same proportion of each class as there exists in the whole dataset.

```
from sklearn.model_selection import cross_val_score

cv_scores = []
classifiers = [svc, lr, sgd, rf, mnb]

print('Cross validation accuracies (means)')
for cls in classifiers:
    scores = cross_val_score(cls, X_train, y_train, cv=10, scoring='accuracy', n_jobs = -1) #
    print('-> ' + type(cls).__name__ + ': {:.2f}%'.format(round(scores.mean()*100, 2)))
```

```
Cross validation accuracies (means)
-> SVC: 82.80%
-> LogisticRegression: 82.67%
-> SGDClassifier: 79.33%
-> RandomForestClassifier: 80.60%
-> MultinomialNB: 81.00%
```

# Hyper-parameter tuning

**Grid Search on SVC (best model)**

```
from sklearn.model_selection import GridSearchCV
from sklearn.svm import SVC

parameters = [{'C': [1, 10, 100], 'kernel': ['linear', 'poly']},
              {'C': [1, 10, 100], 'kernel': ['rbf'], 'gamma': ['scale', 'auto']}]

grid_search = GridSearchCV(SVC(), parameters, scoring='accuracy', cv = 10, n_jobs = -1)
grid_search.fit(X_train, y_train)

print("Best Accuracy: {:.2f} %".format(grid_search.best_score_*100))
print("Best Parameters:", grid_search.best_params_)
```

```
Best Accuracy: 82.80 %
Best Parameters: {'C': 1, 'gamma': 'scale', 'kernel': 'rbf'}
```

**It didn’t boost performance**

# Hyper-parameter tuning

## Let's try again...

**Grid Search on Logistic Regression**

```
1  from sklearn.model_selection import GridSearchCV
2  from sklearn.linear_model import LogisticRegression
3
4  parameters = {'solver': ['saga', 'liblinear'], 'penalty': ['l1', 'l2'], 'C': [0.5, 1.0, 2.0, 10.0]}
5
6  grid_search2 = GridSearchCV(LogisticRegression(), parameters, scoring='accuracy', cv = 10, n_jobs = -1)
7  grid_search2.fit(X_train, y_train)
8
9  print("Best Accuracy: {:.2f} %".format(grid_search2.best_score_*100))
10 print("Best Parameters:", grid_search2.best_params_)
```

```
Best Accuracy: 82.80 %
Best Parameters: {'C': 1.0, 'penalty': 'l2', 'solver': 'liblinear'}
```

**It raised from 82,67% to 82,80% !!**

# Exporting the model

We will use the exported model and TF-IDF to use them in the Recommender System

**Retraining and exporting the winner model with the best parameters**

```
1 from sklearn.svm import SVC
2 import joblib
3
4 svc = SVC(kernel='rbf', C=1.0, gamma='scale')
5 svc.fit(X_train, y_train)
6
7 joblib.dump(svc, 'svcmodel.joblib') #SAVE MODEL
8 joblib.dump(tf, 'tfidf.joblib')
```

```
['tfidf.joblib']
```

# 03

# RECOMMENDER SYSTEM

Twitter API and cosine similarity

# IMDb recommendations

## Importing the Movies Dataset

```
import json

# this function is used to convert the genres column (in JSON format) to a string separated by whitespaces
def parse_json(text):
        text = ast.literal_eval(text)

        r = []
        for i in text:
            i = str(i).replace("\'", "\"")
            movie = json.loads(i)
            r.append(movie['name'])

        return " ".join(r)

dataset = pd.read_csv('movies_metadata.csv', dtype=str).loc[:8000, ['original_title', 'genres']].dropna()
dataset['genres'] = dataset['genres'].apply(lambda x: parse_json(x))

print(dataset[1:5])
```

```
              original_title                        genres
1                    Jumanji  Adventure Fantasy Family
2           Grumpier Old Men              Romance Comedy
3          Waiting to Exhale        Comedy Drama Romance
4  Father of the Bride Part II                      Comedy
```

# IMDb recommendations

## Importing the SVC model and TF-IDF vectorizer

**SVC model**

```
import joblib
from sklearn.feature_extraction.text import TfidfVectorizer

classifier = joblib.load('svcmodel.joblib')
tfidf = joblib.load('tfidf.joblib') #it is necessary for the test cases to share
```

**TextBlob default model**

```
from textblob import TextBlob, Blobber
from textblob.sentiments import NaiveBayesAnalyzer

classifier2 = Blobber(analyzer=NaiveBayesAnalyzer())
```

# IMDb recommendations

## Setting up Twitter API (tweepy)

```
from tweepy import OAuthHandler

consumer_key = 'XXXX'
consumer_secret = 'YYYY'
access_token = 'ZZZZ'
access_token_secret = 'AAAA'

auth = OAuthHandler(consumer_key, consumer_secret)
auth.set_access_token(access_token, access_token_secret)
api = tweepy.API(auth)
```

# IMDb recommendations

## Fetching tweets


```
username = 'DevFarzan' #Replace by the desired username
count = 5 #Replace by the desired number of tweets

tweets = api.user_timeline(screen_name=username, count=count, include_rts = False, tweet_mode = 'extended')
tweets = list(map(lambda tw: tw.full_text, tweets))

for tweet in tweets:
    print(tweet, '\n')

```

```
Ace Ventura. Neither terrible, boring nor soporific, just not very funny.

Jumanji, with plenty of laughs, action-packed excitement, great music, spectacular sets, and inspirational themes, this fil
m is an absolutely winning adventure.

Die Hard. There are good performances from everyone in this long, often funny, very violent but exciting melodrama.

Meet Joe Black. I've never encountered such dramatic flatulence, never heard so many pregnant silences that don't deliver,
never watched so many close-ups that graze on actors' faces until every last trace of expression has been devoured.

Toy Story is a Pixar classic, one of the best kids' movies of all time.
```

# IMDb recommendations

## Text preprocessing

We have to clean and preprocess every tweet exactly as we did in the previous section to achieve the best results.

# IMDb recommendations

## Analyzing the sentiment

**Get sentiment**

```
# vectorize test array into TF-IDF
tftest = tfidf.transform(test).toarray()

#analyze sentiment
sentiment = ['pos' if x == 1 else 'neg' for x in classifier.predict(tftest)]

print("SVC classifier: ", sentiment)

#try the textblob sentiment classifier
s = []
for x in tweets:
    s.append(classifier2(x).sentiment.classification)

print("TextBlob NB Classifier: ", s)

expected = ['neg', 'pos', 'pos', 'neg', 'pos']
print("Expected: ", expected)
```

```
SVC classifier:  ['neg', 'pos', 'pos', 'pos', 'pos']
TextBlob NB Classifier:  ['neg', 'pos', 'pos', 'pos', 'pos']
Expected:  ['neg', 'pos', 'pos', 'neg', 'pos']
```

# IMDb recommendations

## Analyzing the sentiment

```
#append sentiment to review

tw = np.array(tweets)
sent = np.array(sentiment)

reviews = np.hstack((tw.reshape((len(tw), 1)), sent.reshape((len(sent), 1))))

print(reviews)
```

```
[['Ace Ventura. Neither terrible, boring nor soporific, just not very funny.'
  'neg']
 ['Jumanji, with plenty of laughs, action-packed excitement, great music, spectacular sets, and inspirational themes, this
film is an absolutely winning adventure.'
  'pos']
 ['Die Hard. There are good performances from everyone in this long, often funny, very violent but exciting melodrama.'
  'pos']
 ["Meet Joe Black. I've never encountered such dramatic flatulence, never heard so many pregnant silences that don't delive
r, never watched so many close-ups that graze on actors' faces until every last trace of expression has been devoured."
  'pos']
 ["Toy Story is a Pixar classic, one of the best kids' movies of all time."
  'pos']]
```

# IMDb recommendations

## Content-based recommendations

We will base our recommendations on the positive reviews of a film in a tweet. We won't get recommendations on negative reviews or tweets that don't talk about movies.

We find the liked movie in the dataset, look up its genres, and make recommendations of other movies with the same genres using cosine similarity.

# IMDb recommendations

## Finding the mentioned movie in the dataset

```python
import re

#search movie titles in the dataset using regex, and return the movie index of the dataset
def is_movie_tweet(text):

    dataset['original_title'] = dataset.loc[:,'original_title'].str.lower()
    for series in dataset.iterrows():
        t = r"\b" + re.escape(series[1]['original_title']) + r"\b"
        if re.search(t, text.lower()) != None:
            return series

    return False

movies = list(filter(lambda x: x != False, [is_movie_tweet(text) for text in positive_reviews[:, 0]]))

print(movies)
```

```
[(1, original_title                    jumanji
genres            Adventure Fantasy Family
Name: 1, dtype: object), (1007, original_title           die hard
genres            Action Thriller
Name: 1007, dtype: object), (688, original_title    faces
genres            Drama
Name: 688, dtype: object), (0, original_title                toy story
genres            Animation Comedy Family
Name: 0, dtype: object)]
```

Meet Joe Black. I've never encountered such dramatic flatulence, never heard so many pregnant silences that don't deliver, never watched so many close-ups that graze on actors' **faces** until every last trace of expression has been devoured.

**Not too reliable!!**

# IMDb recommendations

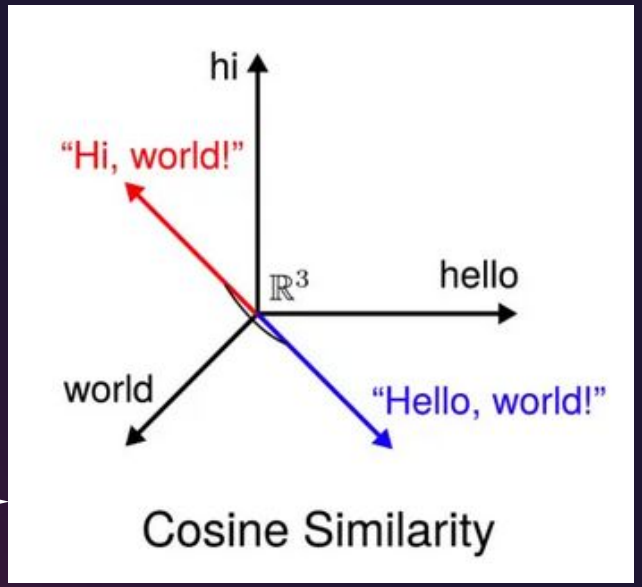
## Getting the recommendations

```
1  from sklearn.feature_extraction.text import CountVectorizer
2  from sklearn.metrics.pairwise import cosine_similarity
3
4  def similarContentBased(movieId):
5
6      cv = CountVectorizer()
7      X = cv.fit_transform(dataset['genres']).toarray()[:10000]
8
9      sim = cosine_similarity(X)
10     sim = pd.Series(sim[movieId]).sort_values(ascending=False)
11     indexes = list(sim.index)
12
```

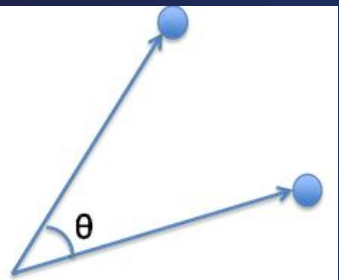
????

# IMDb recommendations

**What is cosine similarity?**




$$sim(A,B) = \cos(\theta) = \frac{A \cdot B}{\|A\|\|B\|}$$



# IMDb recommendations

## Getting the recommendations

```
recommendations = [similarContentBased(r[0]) for r in movies]

print(recommendations)
```

```
[[('return to oz', 'Adventure Family Fantasy', 1.0000000000000002), ('peter pan', 'Adventure Fantasy Family', 1.0000000000000002), ('harry potter and the prisoner of azkaban', 'Adventure Fantasy Family', 1.0000000000000002), ('jason and the argonauts', 'Adventure Family Fantasy', 1.0000000000000002), ('clash of the titans', 'Adventure Fantasy Family', 1.0000000000000002)], [('on deadly ground', 'Action Thriller', 0.9999999999999998), ('air force one', 'Action Thriller', 0.9999999999999998), ('iron eagle iii', 'Action Thriller', 0.9999999999999998), ('the peacemaker', 'Action Thriller', 0.9999999999999998), ('d-tox', 'Action Thriller', 0.9999999999999998)], [('the hours', 'Drama', 1.0), ('querelle', 'Drama', 1.0), ('dead poets society', 'Drama', 1.0), ('the graduate', 'Drama', 1.0), ('coming apart', 'Drama', 1.0)], [('the great mouse detective', 'Comedy Animation Family', 1.0000000000000002), ('the wrong trousers', 'Animation Comedy Family', 1.0000000000000002), ("bon voyage, charlie brown (and don't come back!)", 'Animation Comedy Family', 1.0000000000000002), ('creature comforts', 'Animation Comedy Family', 1.0000000000000002), ('a close shave', 'Family Animation Comedy', 1.0000000000000002)]]
```

# IMDb recommendations

## Final recommendations

-- Movie: **Jumanji** | Genres: Adventure Fantasy Family | Index: 1

*Recommendation: **Return To Oz** | Genres: Adventure Family Fantasy | Similarity: 1.0

*Recommendation: **Peter Pan** | Genres: Adventure Fantasy Family | Similarity: 1.0

*Recommendation: **Harry Potter And The Prisoner Of Azkaban** | Genres: Adventure Fantasy Family | Similarity: 1.0

*Recommendation: **Jason And The Argonauts** | Genres: Adventure Family Fantasy | Similarity: 1.0

*Recommendation: **Clash Of The Titans** | Genres: Adventure Fantasy Family | Similarity: 1.0

# IMDb recommendations

## Final recommendations

```
-- Movie: Die Hard | Genres: Action Thriller | Index: 1007

        *Recommendation: On Deadly Ground | Genres: Action Thriller | Similarity: 1.0

        *Recommendation: Air Force One | Genres: Action Thriller | Similarity: 1.0

        *Recommendation: Iron Eagle Iii | Genres: Action Thriller | Similarity: 1.0

        *Recommendation: The Peacemaker | Genres: Action Thriller | Similarity: 1.0

        *Recommendation: D-Tox | Genres: Action Thriller | Similarity: 1.0
```

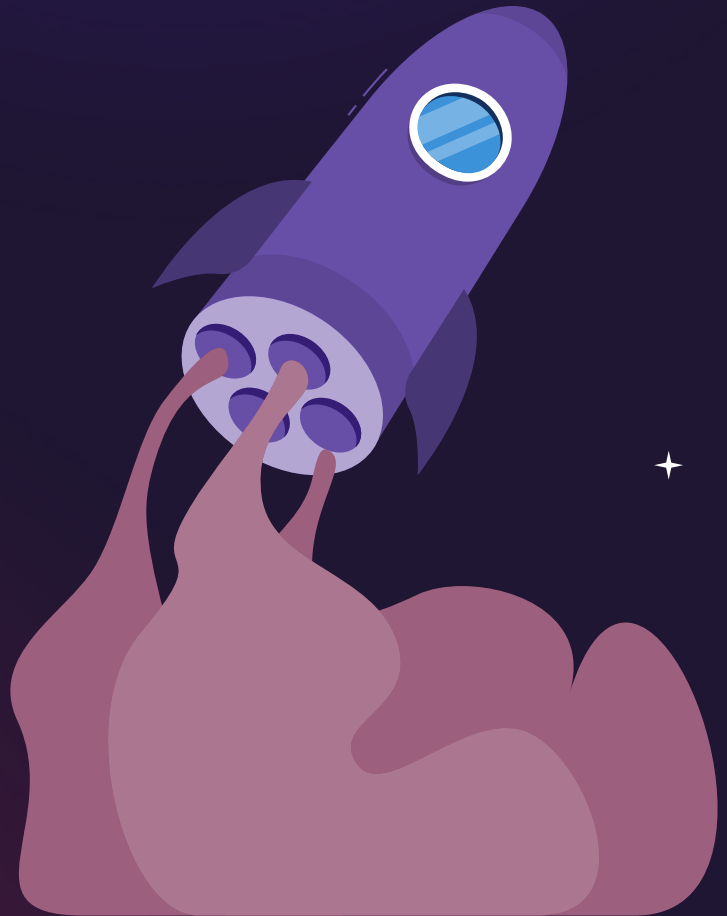
# IMDb recommendations

## Final recommendations

```
-- Movie: Toy Story | Genres: Animation Comedy Family | Index: 0

        *Recommendation: The Great Mouse Detective | Genres: Comedy Animation Family | Similarity: 1.0

        *Recommendation: The Wrong Trousers | Genres: Animation Comedy Family | Similarity: 1.0

        *Recommendation: Bon Voyage, Charlie Brown (And Don'T Come Back!) | Genres: Animation Comedy Family | Similarity:
1.0

        *Recommendation: Creature Comforts | Genres: Animation Comedy Family | Similarity: 1.0

        *Recommendation: A Close Shave | Genres: Family Animation Comedy | Similarity: 1.0
```

# WHAT TO IMPROVE

- Take all the available data (training & movies)
- More thorough parameter tuning
- Better text preprocessing
- Better movie title recognition **(Named-Entity Recognition)**



# THANKS FOR LISTENING!

Do you have any questions?