# Vectorisation

Les algorithmes d'apprentissage automatique ne peuvent pas fonctionner correctement avec du texte brut, le texte doit être converti en chiffres. Dans cette étape on utilise la technique **BOW** (Bag of Words) qui est un moyen d'extraire des caractéristiques d'un texte à utiliser dans la modélisation qui décrit l'occurrence de mots dans un document. Donc on utilise BOW avec un TfidfVectorizer()pour obtenir la matrice des fréquences des termes.

On choisit ces Paramètres :- **min_df** (ignorer les termes qui ont une fréquence de document strictement inférieure à 12), et en appliquant l'algorithme n-grams (**ngram_range** : tuple (min_n, max_n)) pour 1 et 2 termes

```
vectorizer = TfidfVectorizer(min_df=12, ngram_range=(1,2))
```

# Cross-validation scores

Apes que la vectorisation effectuée, on utilise le cross validation pour éviter le problème essentiel que le modèle est appris sur un seul jeu de données et qu'en fonction de la sélection les résultats peuvent être très différents. Dans notre cas, nous allons utiliser une 10-fold cross validation pour évaluer la qualité, et pour comprendre le résultat de la classification on a utilise la métrique **Accuracy** pour évaluer les performances.

Maintenant, on va se concentrer sur la famille d'algorithme de classification (LogisticRegression , DecisionTreeClassifier ,RandomForestClassifier , Gaussian Naive Bayes Classifier, Stochastic Gradient Descent Classifier, K Nearest Neighbors Classifier, Linear SVM Classifier (SVM with a linear kernel)) . Ces algorithmes permettent de catégoriser le sentiment d'un texte (positif ou négatif). On obtient ces résultats de la validation croisée sur les deux partitions de DataSet :-

| Modèle | Score moyen | Déviation standard |
|---|---|---|
| LinearSVC | 92% | 1% |
| SGDClassifier | 92% | 1% |
| LogisticRegression | 91% | 0.8% |
| GaussianNB | 84% | 1% |
| RandomForestClassifier | 81% | 1% |
| KNeighborsClassifier | 79% | 1% |
| DecisionTreeClassifier | 75% | 0.8% |

In [44]:
```
#CROSS VALIDATION RESULTS
# Cross validation started at 2019-04-18 12:38:23.624361
#
#     Time taken to complete cross validation of LogisticRegression: 6.682977199554443 seconds
#     Accuracy scores over 10 evaluations: [0.914 0.911 0.916 0.922 0.932 0.905 0.933 0.924 0.911 0.921]
#     Mean score: 0.9189
#     Standard deviation of scores: 0.008722958213817153
#
# Cross validation started at 2019-04-18 12:39:27.854962
#
#     Time taken to complete cross validation of SGDClassifier: 14.247564554214478 seconds
#     Accuracy scores over 10 evaluations: [0.9   0.909 0.92  0.934 0.94  0.907 0.936 0.925 0.913 0.925]
#     Mean score: 0.9209000000000002
#     Standard deviation of scores: 0.01277849756426787
#
# Cross validation started at 2019-04-18 12:40:48.950553
#
#     Time taken to complete cross validation of DecisionTreeClassifier: 796.7198390960693 seconds
#     Accuracy scores over 10 evaluations: [0.757 0.75  0.763 0.755 0.76  0.739 0.755 0.732 0.751 0.753]
#     Mean score: 0.7515000000000002
#     Standard deviation of scores: 0.008947066558375441
#
# Cross validation started at 2019-04-18 12:59:27.524247
#
#     Time taken to complete cross validation of RandomForestClassifier: 53.930598735809326 seconds
#     Accuracy scores over 10 evaluations: [0.803 0.815 0.813 0.823 0.819 0.833 0.835 0.814 0.799 0.826]
#     Mean score: 0.818
#     Standard deviation of scores: 0.011135528725660019
#
# Cross validation started at 2019-04-18 12:56:18.964747
#
#     Time taken to complete cross validation of GaussianNB: 23.15500497817993 seconds
#     Accuracy scores over 10 evaluations: [0.817 0.841 0.829 0.853 0.836 0.838 0.85  0.847 0.833 0.85 ]
#     Mean score: 0.8394
#     Standard deviation of scores: 0.010650821564555487
#
# Cross validation started at 2019-04-18 19:47:55.137561
#
#     Time taken to complete cross validation of KNeighborsClassifier: 2430.809322834015 seconds
#     Accuracy scores over 10 evaluations: [0.795 0.793 0.805 0.796 0.799 0.778 0.815 0.775 0.817 0.78 ]
#     Mean score: 0.7953
#     Standard deviation of scores: 0.013849548729110253
#
# Cross validation started at 2019-04-18 12:56:42.120080
#
#     Time taken to complete cross validation of LinearSVC: 6.601022243499756 seconds
#     Accuracy scores over 10 evaluations: [0.913 0.915 0.918 0.926 0.944 0.908 0.938 0.92  0.923 0.925]
#     Mean score: 0.923
#     Standard deviation of scores: 0.010497618777608539
```

Les modèles LogisticRegression et LinearSVC sont les mieux adaptés au travail. Alors on les utilisé avec GridSearchCV (dans notre cas nous avons privilégié l'accuracy), et On obtient ces résulté :-

| Modèle | Score moyen | Meilleurs Résultats des hyperparamètres |
|---|---|---|
| LogisticRegression | 90% | C = 11.288378916846883 ; penalty = $L_2$ |
| LinearSVC | 90% | C = 1 |

In [ ]:
```
#GRIDSEARCHCV RESULTS
# Grid search started at 2019-04-19 09:48:26.619439
#
# Time taken to complete Grid search of LogisticRegression: 68.54148316383362 seconds
#
#         Model: LogisticRegression
#         Best Accuracy Score: 0.9073333333333333
#         Best Estimator: LogisticRegression(C=11.288378916846883, class_weight=None, dual=False,
#           fit_intercept=True, intercept_scaling=1, max_iter=100,
#           multi_class='warn', n_jobs=None, penalty='l2', random_state=None,
#           solver='warn', tol=0.0001, verbose=0, warm_start=False)
#
# Grid search started at 2019-04-18 20:59:28.571292
#
# Time taken to complete Grid search of LinearSVC: 16.8770534992218 seconds
#
#         Model: LinearSVC
#         Best Accuracy Score: 0.9053333333333333
#         Best Estimator: LinearSVC(C=1, class_weight=None, dual=True, fit_intercept=True,
#      intercept_scaling=1, loss='squared_hinge', max_iter=1000,
#      multi_class='ovr', penalty='l2', random_state=None, tol=0.0001,
#      verbose=0)
#
# The best model with the best hyperparameters:
#
#         Model: LogisticRegression
#         Best Accuracy Score: 0.9073333333333333
#         Best Estimator: LogisticRegression(C=11.288378916846883, class_weight=None, dual=False,
#           fit_intercept=True, intercept_scaling=1, max_iter=100,
#           multi_class='warn', n_jobs=None, penalty='l2', random_state=None,
#           solver='warn', tol=0.0001, verbose=0, warm_start=False)
```

# Sauvegarde du modèle

---

Les Pipelines sont très importants lorsque l'on sauvegarde un modèle,  comme ils prennent en compte les pré-traitements tout est sauvegardé. Cela veut dire que dans le cas de nouvelles données à évaluer avec un modèle lors de la prédiction les données seront automatiquement transformées. Donc on crée d'un Pipeline pour le modèle Logistic Regression et pour le modèle Gaussian Naive Bayes

Pour sérialiser et sauvegarder les modèles on utilise Pickle (la librairie Python standard pour sérialiser-désérialiser des objets).