classification

May 18, 2021

1 Classification

[1]:
```
from pretraitement import TextPreTraitement
from clean import clean_claimKG
from classification import trueVSfalse, trueFalseVSmixture,␣
↪trueVSfalseVSmixture, cut_data
from utilities import plot_confusion_matrix
```

[2]:
```
import pandas as pd
import matplotlib.pyplot as plt

from sklearn.model_selection import train_test_split, GridSearchCV
from sklearn.pipeline import Pipeline
from sklearn.metrics import accuracy_score, confusion_matrix,␣
↪classification_report

#vectorisation
from sklearn.feature_extraction.text import TfidfVectorizer

# classifier
from sklearn.neighbors import KNeighborsClassifier as KNN
from sklearn.linear_model import LogisticRegression
from sklearn.tree import DecisionTreeClassifier
from sklearn.naive_bayes import GaussianNB
from sklearn.svm import SVC
```

[3]:
```
file_name = "../data/claimKG.csv"
origin = pd.read_csv(file_name)

origin = clean_claimKG(origin, verbose=False, inplace=True)
```

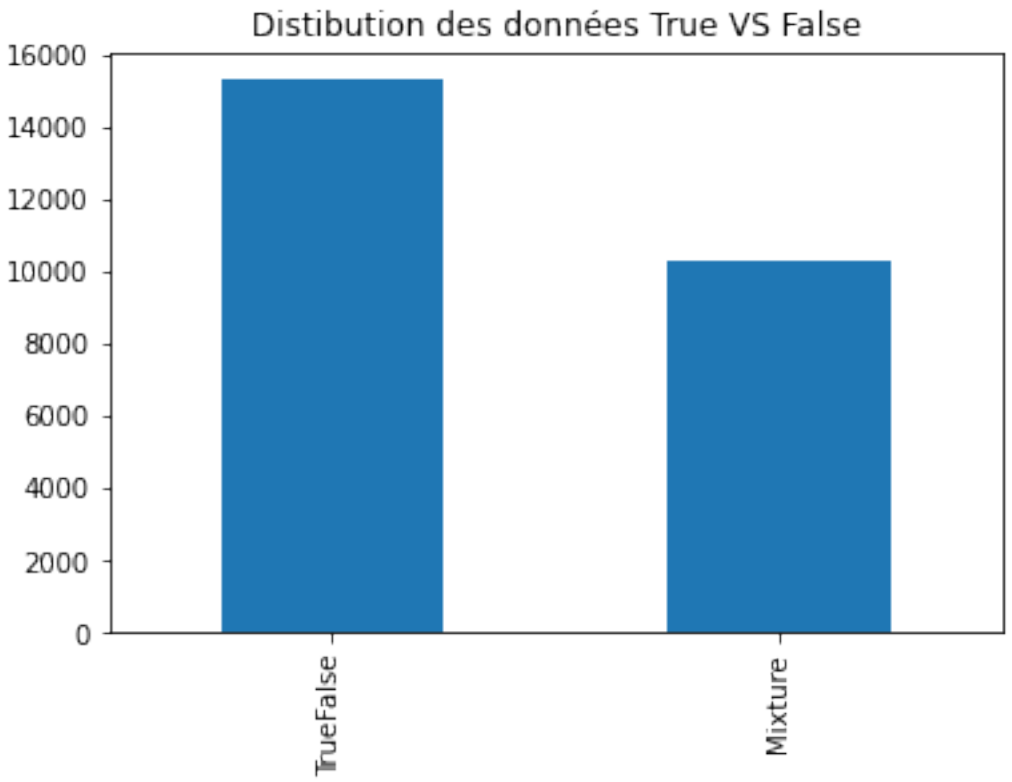
1.1 Découpage des données

Ici nous générons trois groupes de données

- `kg_tf` contient les deux classes **True** -> `1` et **False** -> `0`.
- `kg_tf_m` contient les deux classes **True-False** -> `0` et **Mixture** -> `1`.
- `kg_tfm` contient les trois classes **True** -> `1`, **False** -> `0` et **Mixture** -> `2`

[6]:
```
kg_tf = trueVSfalse(origin)
kg_tf_m = trueFalseVSmixture(origin)
kg_tfm = trueVSfalseVSmixture(origin)

index_tf={0:'False', 1: 'True'}
index_tf_m={0:'TrueFalse', 1: 'Mixture'}
index_tfm={0:'False', 1: 'True', 2: 'Mixture'}

kg_tf_m['ratingValue'].value_counts().rename(index=index_tf_m).plot(
    kind='bar',title="Distibution des données True VS False")
```

[6]:
```
<AxesSubplot:title={'center':'Distibution des données True VS False'}>
```

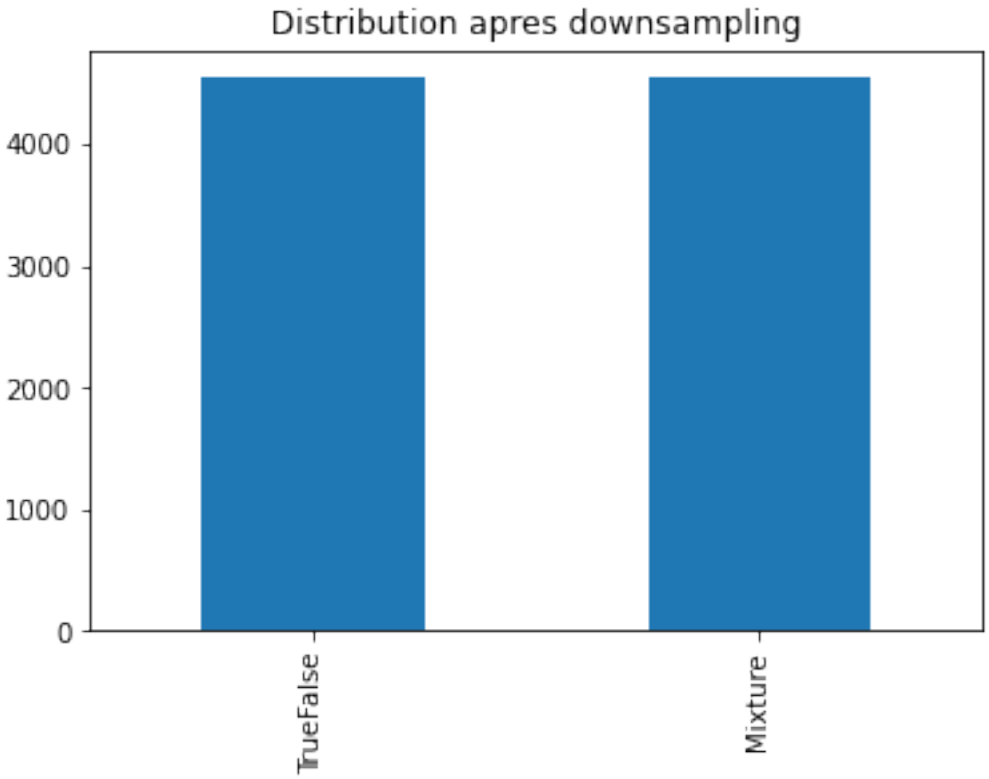


Comme afficher dans la cellule précedente, le nombre d'assertions pour chaque classe est mal balancé. Ce qui peut conduire à un mauvais apprentissage de la part des modèles. En effet, par exemple, dans la tâche **True** VS **False**, il y a 70% d'assertions **False**. le modèle va donc être biaisé et être moins performant pour la prédiction des assertions **False**.

On a donc mis en point une fonction de sous-échantillonnage (Downsampling) pour remédier à ce problème.

[11]:
```
kg = kg_tf_m[:15000]
index = index_tf_m

downsampling = True
if downsampling:
    kg = cut_data(kg)
    print(kg['ratingValue'].value_counts().rename(index=index).plot(
        kind='bar',title="Distribution apres downsampling"))
```

```
AxesSubplot(0.125,0.125;0.775x0.755)
```



1.2 Classification

1.2.1 Jeu d'apprentissage et de test

Dans cette étape nous procédant à la création d'un jeu d'apprentissage et de test en utilsant la fonction `train_test_split` de la librairie `sklearn`.

Nous avons choisis de reserver 70% des données pour l'apprentissage et les 30% restantes pour l'étape de validation.

[12]:
```
X_origin=kg['claimReview_claimReviewed'] # Les assertions
y_origin=kg['ratingValue'] # les labels
```

```
X = X_origin
y = y_origin

train_size=0.7

X_train,X_test,y_train,y_test=train_test_split(X, y,
                                               train_size=train_size ,
                                               random_state=30,stratify=y)
```

Dans ce qui suit, nous utiliserons la classe `TextPreTraitement` vue précédemment (Notebook pretraitement) pour le prétraitement des données.

[10]:
```
pretraitement = TextPreTraitement(stopword=True,
                                  lem=True,
                                  lowercase=True,
                                  number2words=True)

X = pretraitement.fit_transform(X)

X_train = pretraitement.fit_transform(X_train)

X_test = pretraitement.fit_transform(X_test)
```

1.2.2 Vectorisation

L'objectif ici est de transformer l'ensemble des assertions (composées de mots) en un ensemble de vecteurs pour que les modèles puissent effectuer des opérations mathématique dessus. Il existe trois approches principales :

- Bag of Word
- TF-IDF
- Word2Vec

Nous utiliserons principalement TF-IDF pour la vectorisation des données.

[11]:
```
ngram=(1,2)

vectorizer = TfidfVectorizer(
            lowercase=False,
            ngram_range=ngram,
            preprocessor=lambda x:x,
            min_df=0.005,
            max_df=0.7)
```

[12]:
```
vectorizer.fit(X_train)
vectorizer.fit(X_test)
vectorizer.fit(X)
```

```
vector = pd.DataFrame(
        data=vectorizer.transform(X).toarray(),
        columns=vectorizer.get_feature_names()
    )

vector_train = pd.DataFrame(
        data=vectorizer.transform(X_train).toarray(),
        columns=vectorizer.get_feature_names()
    )
vector_test = pd.DataFrame(
        data=vectorizer.transform(X_test).toarray(),
        columns=vectorizer.get_feature_names()
    )
```

[13]:
```
display(vector_train.sample(5))
display(vector_test.sample(5))
```

```
      000  abortion  accident  account  across  act  actor  actually  \
4420  0.0       0.0       0.0      0.0     0.0  0.0    0.0       0.0
506   0.0       0.0       0.0      0.0     0.0  0.0    0.0       0.0
1723  0.0       0.0       0.0      0.0     0.0  0.0    0.0       0.0
5637  0.0       0.0       0.0      0.0     0.0  0.0    0.0       0.0
5618  0.0       0.0       0.0      0.0     0.0  0.0    0.0       0.0

      administration  admit  …  word  work  worker  world     would  write  \
4420             0.0    0.0  …   0.0   0.0     0.0    0.0  0.000000    0.0
506              0.0    0.0  …   0.0   0.0     0.0    0.0  0.414533    0.0
1723             0.0    0.0  …   0.0   0.0     0.0    0.0  0.000000    0.0
5637             0.0    0.0  …   0.0   0.0     0.0    0.0  0.000000    0.0
5618             0.0    0.0  …   0.0   0.0     0.0    0.0  0.000000    0.0

      year  year old  york  young
4420   0.0       0.0   0.0    0.0
506    0.0       0.0   0.0    0.0
1723   0.0       0.0   0.0    0.0
5637   0.0       0.0   0.0    0.0
5618   0.0       0.0   0.0    0.0

[5 rows x 416 columns]
```

```
      000  abortion  accident  account  across  act  actor  actually  \
2254  0.0       0.0  0.000000      0.0     0.0  0.0    0.0       0.0
701   0.0       0.0  0.000000      0.0     0.0  0.0    0.0       0.0
418   0.0       0.0  0.000000      0.0     0.0  0.0    0.0       0.0
2192  0.0       0.0  0.371433      0.0     0.0  0.0    0.0       0.0
617   0.0       0.0  0.000000      0.0     0.0  0.0    0.0       0.0
```

```
      administration  admit  …  word  work  worker  world  would  write  \
2254             0.0    0.0  …   0.0   0.0     0.0    0.0    0.0    0.0
701              0.0    0.0  …   0.0   0.0     0.0    0.0    0.0    0.0
418              0.0    0.0  …   0.0   0.0     0.0    0.0    0.0    0.0
2192             0.0    0.0  …   0.0   0.0     0.0    0.0    0.0    0.0
617              0.0    0.0  …   0.0   0.0     0.0    0.0    0.0    0.0

          year  year old  york  young
2254  0.186778       0.0   0.0    0.0
701   0.000000       0.0   0.0    0.0
418   0.000000       0.0   0.0    0.0
2192  0.000000       0.0   0.0    0.0
617   0.000000       0.0   0.0    0.0

[5 rows x 416 columns]
```

1.2.3 Classifieur

Comme l'indique le *No free lunch theorem*, il n'existe pas de classifieur universel. Il est donc toujours nécessaire d'en évaluer plusieurs afin de trouver le plus pertinent pour notre situation.

Les algorithmes testés sont les suivants:

- SVM
- Naive Bayes
- Decision Tree
- K-nearest neighbors
- Logistic Regression

[13]:
```python
def gridSearch(X, y, verbose=False):
    models = dict()
    param = dict()

    models['dt'] = DecisionTreeClassifier()
    param['dt'] = {'max_depth': [15],#[5,10,15],
        'criterion': ['gini'],#['gini', 'entropy'],
        'min_samples_leaf': [6]} #[5,6,7]}

    models['svm'] = SVC()
    param['svm'] = {'C': [0.1],
        'gamma' : ['auto'],
        'kernel': ['linear']}

    models['nb'] = GaussianNB()
    param['nb'] = {'priors': [None]}

    models['knn'] = KNN()
    param['knn'] = {'n_neighbors': [7],
        'metric': ['euclidean']}
```

```
    models['lr'] = LogisticRegression()
    param['lr'] = {'solver': ['newton-cg']}

    predictions = []

    for name, model in models.items():
        if verbose:
            print("Searching best param for "+name)
        gd_sr = GridSearchCV(estimator=model,
                             param_grid=param[name],
                             scoring='accuracy',
                             cv=2,
                             n_jobs=-1)

        gd_sr.fit(X, y)
        if verbose:
            print("Best score: "+str(gd_sr.best_score_))
        predictions.append((name,gd_sr.best_score_,gd_sr.best_estimator_,gd_sr.
↪best_params_))

        del gd_sr
    return predictions

#gridSearch(vector, y)
```

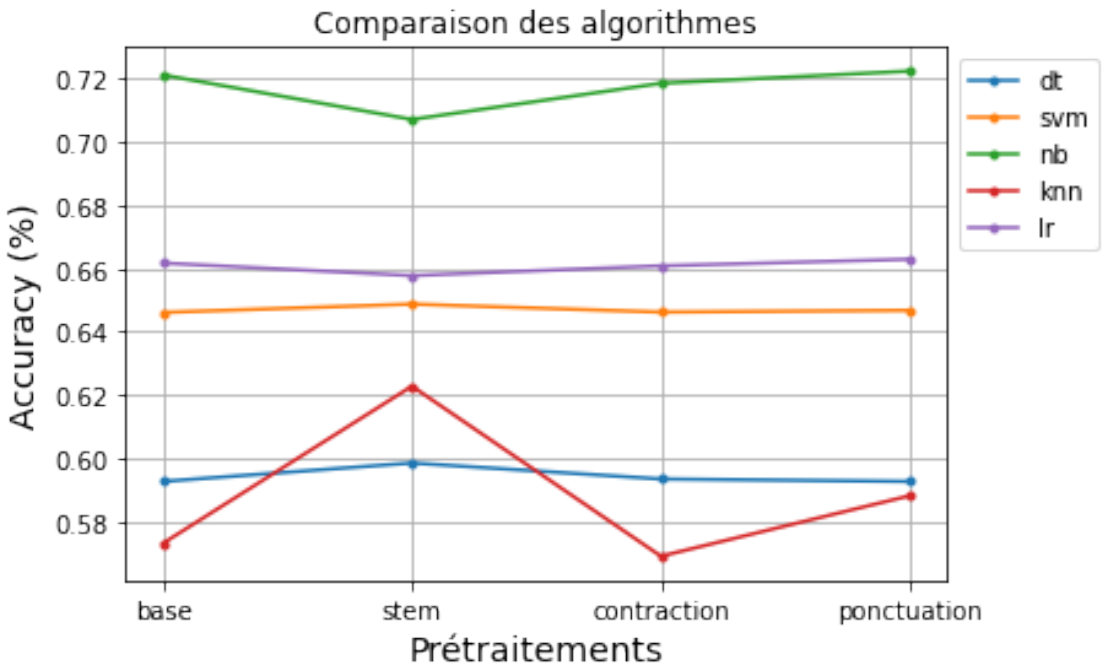
[30]:
```
def pretraitmentSearch(X_origin, y, verbose=False):
    options = ['base', 'stem', 'contraction','ponctuation']
    base_options = {'lowercase': True}
    predictions = []

    ngram=(1,2)
    vectorizer = TfidfVectorizer(
            lowercase=False,
            ngram_range=ngram,
            preprocessor=lambda x:x,
            min_df=0.005,
            max_df=0.7)

    for option in options:
        if verbose:
            print("Calcul des résultats pour "+option+ ": True")
        pretraitement = TextPreTraitement()

        for op, v in base_options.items():
            setattr(pretraitement, '_'+op, v)
        if option != 'base':
```

```
            setattr(pretraitement, '_'+option, True)

        X = pretraitement.fit_transform(X_origin)

        vectorizer.fit(X)

        vector = pd.DataFrame(
                data=vectorizer.transform(X).toarray(),
                columns=vectorizer.get_feature_names()
            )

        predictions.append(gridSearch(vector, y,verbose))

        del vector, X, pretraitement

    models = {}

    for algo in predictions[0]:
        models[algo[0]] = pd.DataFrame([], columns=['option', 'score'])

    for option, pred in zip(options, predictions):
        for algo in pred:
            models[algo[0]] = models[algo[0]].append({'option': option, 'score':
↪ algo[1]}, ignore_index=True)

    return models
```

[31]:
```
models_result = pretraitmentSearch(X_origin, y, verbose=False)
```

[16]:
```
def plot_models_lines(models):
    for name, model in models.items():
        plt.plot(model['option'], model['score'], label=name,marker='.')

    plt.title('Comparaison des algorithmes')
    plt.xlabel('Prétraitements',fontsize=14)
    plt.ylabel('Accuracy (%)',fontsize=14)
    plt.grid()
    plt.legend(loc=1,bbox_to_anchor=(1.20, 1))
    plt.show()
```

[32]:
```
plot_models_lines(models_result)
```


Comparaison des algorithmes
Accuracy (%)
0.72
0.70
0.68
0.66
0.64
0.62
0.60
0.58
base
stem
contraction
ponctuation
Prétraitements
dt
svm
nb
knn
lr