obl2-part-a

October 13, 2019

0.0.1 IN4080 2019, Mandatory assignment 2, part A

[1]:
```
import nltk
import random
import numpy as np
import scipy as sp
import sklearn
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.feature_extraction import DictVectorizer
from sklearn.naive_bayes import MultinomialNB
from sklearn.naive_bayes import BernoulliNB
from sklearn.linear_model import LogisticRegression
import pandas as pd
from sklearn.preprocessing import OneHotEncoder
from sklearn.preprocessing import LabelEncoder
from sklearn.model_selection import train_test_split
from sklearn.model_selection import cross_val_score
from sklearn.pipeline import Pipeline
from sklearn.metrics import precision_score, recall_score, f1_score
from sklearn.metrics import confusion_matrix
import matplotlib.pyplot as plt
from nltk.corpus import movie_reviews
from sklearn.feature_extraction.text import TfidfVectorizer
import itertools
```

[2]:
```
def plot_confusion_matrix(cm,
                          target_names,
                          title='Confusion matrix',
                          cmap=None,
                          normalize=True):

    accuracy = np.trace(cm) / float(np.sum(cm))
    misclass = 1 - accuracy

    if cmap is None:
        cmap = plt.get_cmap('Blues')
```

```
    plt.figure(figsize=(8, 6))
    plt.imshow(cm, interpolation='nearest', cmap=cmap)
    plt.title(title)
    plt.colorbar()

    if target_names is not None:
        tick_marks = np.arange(len(target_names))
        plt.xticks(tick_marks, target_names, rotation=45)
        plt.yticks(tick_marks, target_names)

    if normalize:
        cm = cm.astype('float') / cm.sum(axis=1)[:, np.newaxis]


    thresh = cm.max() / 1.5 if normalize else cm.max() / 2
    for i, j in itertools.product(range(cm.shape[0]), range(cm.shape[1])):
        if normalize:
            plt.text(j, i, "{:0.4f}".format(cm[i, j]),
                     horizontalalignment="center",
                     color="white" if cm[i, j] > thresh else "black")
        else:
            plt.text(j, i, "{:,}".format(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\naccuracy={:0.4f}; misclass={:0.4f}'.
↪format(accuracy, misclass))
    plt.show()
```

0.0.2 Ex 1 First classifier

[3]:
```
def createSets(movie_dev):
    X = []
    y = []
    for movie in movie_dev:
        X.append(movie[0])
        y.append(movie[1])

    label_encoder = LabelEncoder()
    y_labels = label_encoder.fit_transform(y)
    return X, y
```

[4]:
```
raw_movie_docs = [(movie_reviews.raw(fileid), category) for
                    category in movie_reviews.categories() for fileid in
```

```
                    movie_reviews.fileids(category)]
```

[5]:
```
random.seed(2920)
random.shuffle(raw_movie_docs)
movie_test = raw_movie_docs[:200]
movie_dev = raw_movie_docs[200:]
```

[35]:
```
X, labels = createSets(movie_dev)
```

[7]:
```
X_train, X_validation, Y_train, Y_validation = train_test_split(X, labels,␣
↪test_size=0.125, random_state=42)
```

[8]:
```
v = CountVectorizer()
v.fit(X_train)

X_train_vectors = v.transform(X_train)
X_val_vectors = v.transform(X_validation)
```

[9]:
```
clf = MultinomialNB()
clf.fit(X_train_vectors, Y_train)
```

[9]:
```
MultinomialNB(alpha=1.0, class_prior=None, fit_prior=True)
```

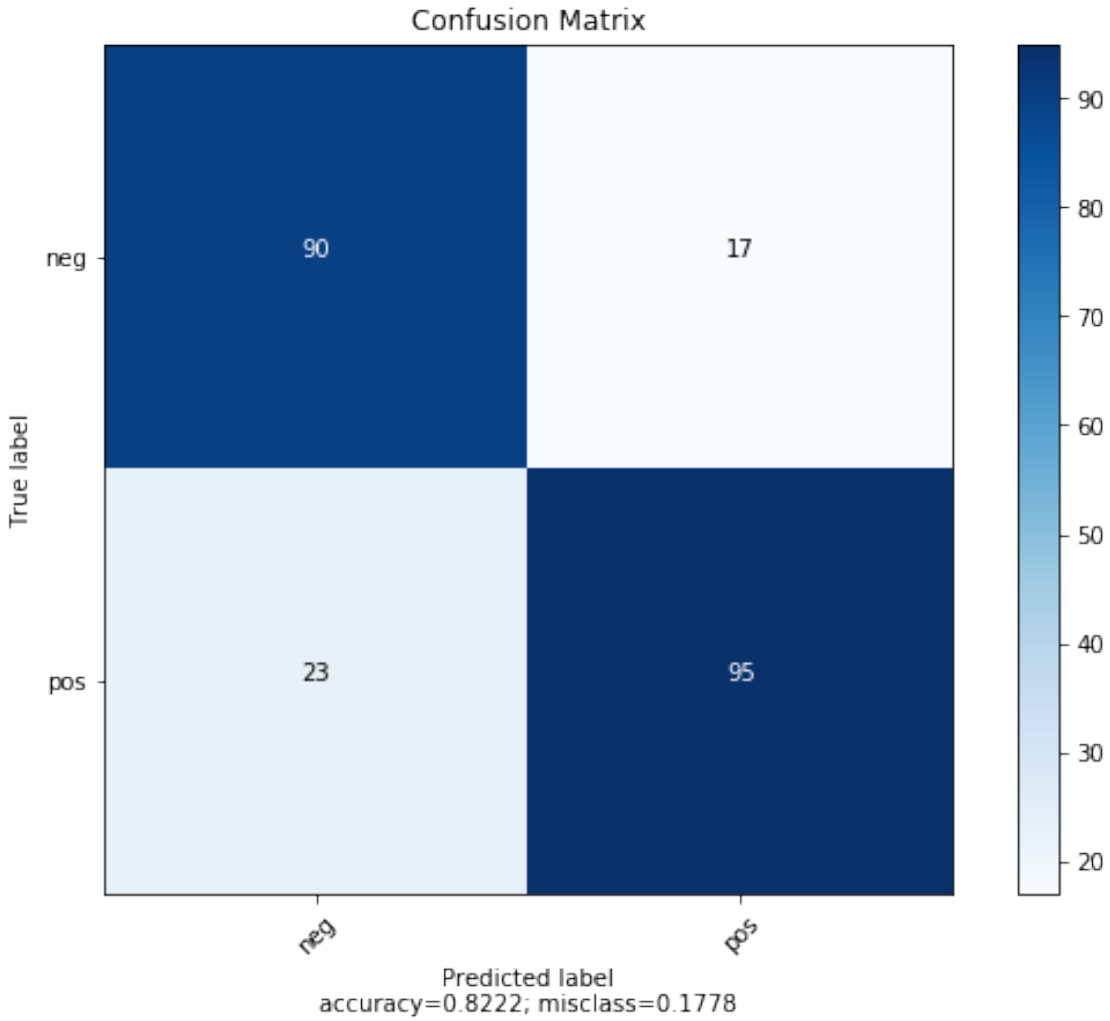
[10]:
```
pred = clf.predict(X_val_vectors)
```

[11]:
```
clf.score(X_val_vectors, Y_validation)
```

[11]:
```
0.8222222222222222
```

[12]:
```
f1 = f1_score(pred, Y_validation, labels=['neg','pos'], average='micro')
cm = confusion_matrix(Y_validation, pred)
```

[13]:
```
plot_confusion_matrix(cm,
                      normalize    = False,
                      target_names = ['neg', 'pos'],
                      title        = "Confusion Matrix")
```



[14]:
```
def multi_nb_exp(X, y):
    text_clf = Pipeline([
                         ('vect', CountVectorizer()),
                         ('clf', MultinomialNB())])
    X_train, X_validation, Y_train, Y_validation = train_test_split(X, y,␣
 ↪test_size=0.125, random_state=42)
    text_clf.fit(X_train, Y_train)
    predicted = text_clf.predict(X_validation)
    acc = text_clf.score(X_validation, Y_validation)
    return acc, predicted
```

[15]:
```
%%time
(acc, pred) = multi_nb_exp(X, labels)
print(acc)
```

```
0.8222222222222222
```

```
CPU times: user 1.92 s, sys: 13.7 ms, total: 1.94 s
Wall time: 776 ms
```

[16]:
```
def n_fold(clf, x, y, n=9):
    acc = cross_val_score(clf, x, y, cv=n, n_jobs=-1) # same as using K-fold␣
 ↪with splits.
    return acc

text_clf = Pipeline([
                     ('vect', CountVectorizer()),
                     ('clf', MultinomialNB())])

acc = n_fold(text_clf, X, labels)
#get the mean of each fold
print("Accuracy of Model with Cross Validation is:",acc.mean() * 100)
```

```
Accuracy of Model with Cross Validation is: 81.33396807142401
```

[17]:
```
def scores(model, X_validation, Y_validation):
    predicted = model.predict(X_validation)
    precision = precision_score(Y_validation, predicted)
    recall = recall_score(Y_validation, predicted)
    f1 = f1_score(Y_validation, predicted)
    print("Precision of Modelis:",precision * 100)
    print("Recall of Model is:",recall * 100)
    print("F1 score of Model is:",f1 * 100)
    return f1, precision, recall
```

[ ]:
```

```

0.0.3 1.2 Ex 2 Parameters of the vectorizer

[False, True] and ngram_range vary over [[1,1], [1,2], [1,3]]. Run the experiment with 9-fold cross-validation and report the accuracy with the 6 different settings.

[18]:
```
def arg_experiment():
    k = [True, False]
    ngram_range_results = []
    ngram_range = [(1,1), (1,2),(1,3)]

    results = []
    for binary in k:
        liste = []
        for ngram in ngram_range:
            count_experiment_clf = Pipeline([
                                ('vect', CountVectorizer(ngram_range=ngram,␣
 ↪binary=binary)),
                                ('clf', MultinomialNB())])
```

```
            acc = cross_val_score(count_experiment_clf, X, labels, cv=9,␣
 ↪n_jobs=-1)

            liste.append(acc.mean() * 100)
        ngram_range_results.append(liste)

    df = pd.DataFrame(ngram_range_results, index=['True','False'],␣
 ↪columns=['(1,1)','(1,2)','(1,3)'])
    return df
df = arg_experiment()
df
```

[18]:

	(1,1)	(1,2)	(1,3)
True	82.337017	85.615942	85.113711
False	81.333968	83.225367	82.557301

True and ngram_range(1,2) gives the best results.

0.1 Ex 3 Logistic Regression

[19]:
```
from warnings import simplefilter
# ignore all future warnings
simplefilter(action='ignore', category=FutureWarning)

log_reg_clf = Pipeline([
     ('vect', CountVectorizer()),
     ('clf', LogisticRegression())])

acc = cross_val_score(log_reg_clf, X, labels, cv=9, n_jobs=-1)
print("The average accuracy of Model with Cross Validation is:",acc.mean() *␣
 ↪100)
```

```
The average accuracy of Model with Cross Validation is: 83.33925570361482
```

[20]:
```
def logistic_reg():
    from warnings import simplefilter
    # ignore all future warnings
    simplefilter(action='ignore', category=FutureWarning)

    log_reg_clf = Pipeline([
         ('vect', CountVectorizer()),
         ('clf', LogisticRegression())])

    acc = cross_val_score(log_reg_clf, X, labels, cv=9, n_jobs=-1)
    return acc

acc = logistic_reg()
print("Accuracy of Model with Cross Validation is:",acc.mean() * 100)
```

```
Accuracy of Model with Cross Validation is: 83.33925570361482
```

0.1.1 Ex 4 The Bernoulli model

[21]:
```
all_words = nltk.FreqDist(w.lower() for w in movie_reviews.words())
def document_features(document,n):
    word_features = list(all_words)[:n]
    document_words = set(document)
    features = {}
    for word in word_features:
        features[word] = (word in document_words)
    return features
```

[22]:
```
documents = [(list(movie_reviews.words(fileid)), category)
             for category in movie_reviews.categories()
             for fileid in movie_reviews.fileids(category)]

random.shuffle(documents)
```

[23]:
```
feature_set = [(document_features(doc,2000), category) for (doc, category) in␣
 ↪documents]
```

[24]:
```
movie_feat_test = feature_set[:200]
movie_feat_dev = feature_set[200:]

x, y = createSets(movie_feat_dev)
```

[25]:
```
ber_clf = Pipeline([
     ('dict_vect', DictVectorizer()),
     ('clf', BernoulliNB())])
```

[26]:
```
acc = cross_val_score(ber_clf, x, y, cv=9, n_jobs=-1)
print("Accuracy of Model with Cross Validation is:",acc.mean() * 100)
```

```
Accuracy of Model with Cross Validation is: 79.27777777777779
```

0.1.2 Ex 5 Logistic regression

[27]:
```
def freq_experiment(pipeline, n):
    feat_set = [(document_features(doc,n), category) for (doc, category) in␣
 ↪documents]
    movie_feat_test = feat_set[:200]
    dev = feat_set[200:]
    x, y = createSets(dev)
    acc = cross_val_score(pipeline, x, y, cv=9, n_jobs=-1)
    return acc
```

[28]:
```
def log_clf_freq_experiment(n):
    log_clf = Pipeline([
         ('vect', DictVectorizer()),
```

```
        ('clf', LogisticRegression(n_jobs=-1))])
    log_clf_freq = []
    for i in n:
        acc = freq_experiment(log_clf, i)
        log_clf_freq.append(acc)
    return log_clf_freq

def ber_clf_experiment(n):
    ber_clf = Pipeline([
     ('dict_vect', DictVectorizer()),
     ('clf', BernoulliNB())])
    ber_clf_freq = []
    for i in n:
        acc = freq_experiment(ber_clf, i)
        ber_clf_freq.append(acc)
    return ber_clf_freq
```

[29]:
```
def ex5_partb():
    n = [1000, 2000, 3000, 4000, 5000]
    accs_log = log_clf_freq_experiment(n)
    accs_ber = ber_clf_experiment(n)
    dict1 = {}
    for i in range(len(accs_ber)):
        dict1[str(n[i])] = [accs_log[i].mean()*100, accs_ber[i].mean()*100]

    df = pd.DataFrame(dict1)
    df.index = ['LogisticRegression', 'BernoulliNB']
    return df
df = ex5_partb()
df
```

[29]:

	1000	2000	3000	4000	5000
LogisticRegression	75.111111	78.666667	80.444444	82.055556	83.222222
BernoulliNB	76.888889	79.277778	79.944444	80.333333	81.222222

0.1.3 Bringing on the test set and the best model

Exercise 6

[30]:
```
feat_set = [(document_features(doc,5000), category) for (doc, category) in␣
 ↪raw_movie_docs]
```

[37]:
```
log_reg_clf = Pipeline([
     ('vect', CountVectorizer(ngram_range=(1,2), binary=True)),
     ('clf', LogisticRegression(n_jobs=-1, solver='lbfgs'))])
```

[38]:
```
#X, y = createSets(feat_set)
log_reg_clf.fit(X, labels)
```

[38]:
```
Pipeline(memory=None,
         steps=[('vect',
                 CountVectorizer(analyzer='word', binary=True,
                                 decode_error='strict',
                                 dtype=<class 'numpy.int64'>, encoding='utf-8',
                                 input='content', lowercase=True, max_df=1.0,
                                 max_features=None, min_df=1,
                                 ngram_range=(1, 2), preprocessor=None,
                                 stop_words=None, strip_accents=None,
                                 token_pattern='(?u)\\b\\w\\w+\\b',
                                 tokenizer=None, vocabulary=None)),
                ('clf',
                 LogisticRegression(C=1.0, class_weight=None, dual=False,
                                    fit_intercept=True, intercept_scaling=1,
                                    l1_ratio=None, max_iter=100,
                                    multi_class='warn', n_jobs=-1, penalty='l2',
                                    random_state=None, solver='lbfgs',
                                    tol=0.0001, verbose=0, warm_start=False))],
         verbose=False)
```

[39]:
```
# prepareing movie_test set
y_test = []
X_test = []
for movie in movie_test:
    X_test.append(movie[0])
    y_test.append(movie[1])

label_encoder = LabelEncoder()
y_test_labels = label_encoder.fit_transform(y_test)
```

[40]:
```
pred_test = log_reg_clf.predict(X_test)
pred_test = label_encoder.fit_transform(pred_test)
```

[41]:
```
precision = precision_score(y_test_labels, pred_test)
recall = recall_score(y_test_labels, pred_test)
f1 = f1_score(y_test_labels, pred_test)
tn, fp, fn, tp = confusion_matrix(y_test_labels, pred_test).ravel()
acc = ((tp + tn) / (tp+tn+fp+fn))
print("Accuracy of Model is:",acc*100)
print("Precision of Model is:",precision * 100)
print("Recall of Model is:",recall * 100)
print("F1 score of Model is:",f1 * 100)
```

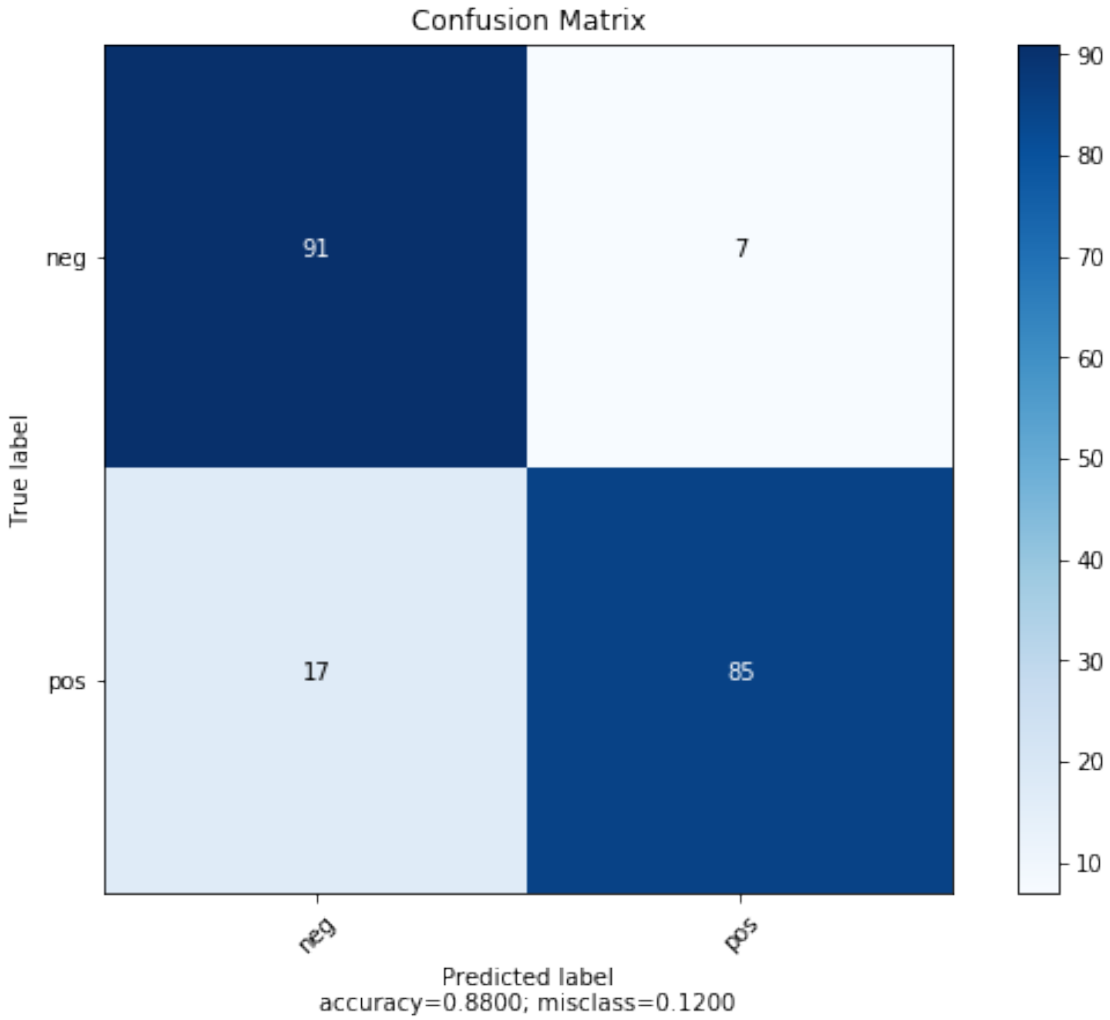
```
Accuracy of Model is: 88.0
Precision of Model is: 92.3913043478261
Recall of Model is: 83.33333333333334
F1 score of Model is: 87.62886597938144
```

The best classifier is LogisticRegression with tweeked settings for countVectorizer. I also tried

tf-idf but I did not get any better results.

[42]:
```
cm = confusion_matrix(y_test_labels, pred_test)
```

[43]:
```
plot_confusion_matrix(cm,
                      normalize    = False,
                      target_names = ['neg', 'pos'],
                      title        = "Confusion Matrix")
```



[ ]: