Επεξεργασία Φυσικής Γλώσσας – 2022/20223 - Ο.Π.Α

Εργασία 2

Φοίβος-Αστέριος Νταντάμης (f3312204) (phivos93@yahoo.com)

Παναγιώτης Καραστατήρης () (pan.karastatiris@gmail.com)

Άσκηση 15

notebook url: https://colab.research.google.com/drive/19BDPgAe6xb88vuniNTEN017Ao5E0Bs6Q#scrollTo=DQJz-3ArZXIg

Περιγραφή Dataset (Φοίβος Νταντάμης)

Ζητείται να φτιαχτεί ένας sentiment classifier για ένα dataset με δύο κλάσεις. Το dataset που χρησιμοποιήθηκε είναι το Cornell Movie Review Data από το ομώνυμο πανεπιστήμιο. Το dataset αυτό περιέχει κριτικές ταινιών από το αρχείο του IMDb κατηγοριοποιημένες ως θετικές(pos) ή αρνητικές(neg). Το μέγεθος του συνόλου όλων των κριτικών είναι 2000, στις οποίες οι 1000 ανήκουν στην κατηγορία pos και οι υπόλοιπες 1000 στην κατηγορία neg. Ο μέσος όρος χαρακτήρων ανά κριτική είναι 3893.002, ενώ ο μέσος όρο των λέξεων είναι 746.3405.

```
char_sum = word_sum = n = 0

for i in x:
  char_sum += len(i)
  word_sum += len(i.split())
  n += 1

avg_chars = char_sum/n
avg_words = word_sum/n

print(f'Average characters per document (including whitespacs characeter): {avg_chars}')
print(f'Average words per document: {avg_words}')
```

```
Average characters per document (including whitespacs characeter): 3893.002
Average words per document: 746.3405
```

Προεπεξεργασία (Φοίβος Νταντάμης)

Το πρώτο βήμα που εφαρμόζουμε είναι να αφαιρέσουμε τα stopwords της αγγλικής γλώσσας, φιλτράροντας όλα τα κείμενα για τις λέξεις που θεωρεί το nltk ως stopwords.

```
for doc in x:

    # Remove all stopwords
    stop_words = stopwords.words('english')
    stopwords_dict = Counter(stop_words) # this is just for optimazation purposes
    document = ' '.join([word for word in doc.split() if word not in stopwords_dict])
```

Στη συνέχεια με τη βοήθεια του WordNetLemmatizer αντικαθιστούμε όλες τις λέξεις με τα λήμματά τους.

```
# Split the document based on whitespaces (--> List of words)
document = document.split()

# Lemmatization
document = [stemmer.lemmatize(word) for word in document]

# Reconstruct the document by joining the words on each whitespace
document = ' '.join(document)
```

Συνεχίζοντας, μετατρέπουμε όλους τους χαρακτήρες σε μικρούς.

```
# Convert to Lowercase
document = document.lower()
```

Σε αυτήν την φάση έχουν προκύψει λέξεις οι οποίες ανήκουν στα stopwords, οπότε θα ξαναφιλτράρουμε τα κείμενα.

```
# Remove stopwords that occurered after lemmatization
document = ' '.join([word for word in document.split() if word not in stopwords_dict])
```

Έπειτα αφαιρούμε τα σημεία στίξης και τα σύμβολα που ουσιαστικά δεν αποτελούν κάποια λέξη.

```
# Remove non-word (special) characters such as punctuation
document = re.sub(r'\W', ' ', document)
```

Αφαιρούμε όλες τις λέξεις που αποτελούνται από έναν χαρακτήρα μόνο, μιας και δεν προσφέρουν κάποια ιδιαίτερη πληροφορία για την κατηγοριοποίηση.

```
# Remove all single characters
document = re.sub(r'\s+[a-z]\s+', ' ', document)
```

Σε αυτό το σημείο και πάλι έχουν προκύψει κάποιες λέξεις που είναι stopwords οπότε φιλτράρουμε εκ νέου.

```
# Remove once again stopwords that occured
document = ' '.join([word for word in document.split() if word not in stopwords_dict])
```

Αφαιρούμε τώρα όλους τους αριθμούς

```
# Remove numbers
document = re.sub(r'\s+[0-9]+', ' ', document)
```

Αφαιρούμε τέλος τα honorifics

```
# Remove honorifics
document = re.sub(r'\s+mr\s+|\s+dr\s+|\s+jr\s+|\s+ms\s+|\s+miss\s+', ' ', document)
```

Διαχωρισμός Dataset (Φοίβος Νταντάμης)

Σπάσαμε το Dataset σε Train, Development και Test sets με ποσοστά 70%, 20%, 10% αντίστοιχα. Προκύπτουν έτσι το Train Dataset με 1400 κριτικές και μέγεθος λεξιλογίου 30635. Το Development Dataset με 400 κριτικές και μέγεθος λεξιλογίου 17726. Το Testing Dataset με 200 κριτικές και μέγεθος λεξιλογίου 12148.

```
x_train, x_temp, y_train, y_temp = train_test_split(docs, y, test_size=0.3, random_state=17)
x_dev, x_test, y_dev, y_test = train_test_split(x_temp, y_temp, test_size=0.333, random_state=25)

print('Sizes of train, dev and test sets are:')
print(len(y_train), len(y_dev), len(y_test))

train_vocab = set()
dev_vocab = set()
test_vocab = set()

for i in x_train:
  for j in i.split():
    train_vocab.add(j)

for i in x_dev:
  for j in i.split():
    dev_vocab.add(j)

for i in x_test:
  for j in i.split():
    test_vocab.add(j)

print(f'Train set vocabulary size is: {len(train_vocab)}')
print(f'Dev set vocabulary size is: {len(dev_vocab)}')
print(f'Test set vocabulary size is: {len(test_vocab)}')
```

```
Sizes of train, dev and test sets are:
1400 400 200
Train set vocabulary size is: 30635
Dev set vocabulary size is: 17726
Test set vocabulary size is: 12148
```

Feature Exctraction (Φοίβος Νταντάμης)

Προκειμένου να μετατρέψουμε τα κείμενα σε διανύσματα χρησιμοποιήσαμε δύο τρόπους. Ο πρώτος είναι δημιουργώντας Boolean Features και ο δεύτερος TF-IDF features.

Boolean Features:

χρησιμοποιούμε τον CountVectorizer με binary times. Κρατάμε μόνο τα μονογράμματα και μόνο αυτά με συχνότητα εμφάνισης τουλάχιστον 10. Τέλος κρατάμε τα 5000 πιο συχνά εμφανιζόμενα μονογράμματα.

```python
from sklearn.feature_extraction.text import CountVectorizer

# Use unigram & bi-gram tf*idf features
boolean_vectorizer = CountVectorizer(
    ngram_range = (1, 1),
    max_features = 5000,
    min_df = 10,
    binary = True
)

# Fit Vectorizer on train data
# Transform on all data (train - dev - test)
x_train_boolean = boolean_vectorizer.fit_transform(x_train)
x_dev_boolean = boolean_vectorizer.transform(x_dev)
x_test_boolean = boolean_vectorizer.transform(x_test)

print(boolean_vectorizer.get_feature_names())
```

```
['abandon', 'abandoned', 'ability', 'able', 'aboard', 'absence',
```

Στη συνέχεια με κριτήριο το Information Gain(Mutual Info) από τα 5000, κρατάμε τα 2500 καλύτερα features.

```python
# Number of features to keep
k = 2500

# Convert sparse matrix to np.array --> Slow
x_train_boolean_array = x_train_boolean.toarray()

# The function relies on nonparametric methods
# based on entropy estimation from k-nearest neighbors distances
mutual_information = mutual_info_classif(x_train_boolean_array, y_train, n_neighbors=3,
                                         discrete_features=True,
                                         random_state=42)

# Indexes of the feature columns
indexes = np.arange(len(mutual_information))

# Sort by mutual information values --> (value, idx)
sorted_indexes = sorted(zip(mutual_information, indexes), reverse=True)
print('Sorted indexes: {}'.format(sorted_indexes[:4]))

# keep the indexes of the best k features
best_idxs = [idx for val, idx in sorted_indexes[:k]]
print('Best indexes: {}\n'.format(best_idxs[:4]))

# Keep only the colums of the best features
x_train_best_boolean = x_train_boolean_array[:, best_idxs]
x_test_best_boolean = x_test_boolean[:, best_idxs]
```

TF-IDF Features:

χρησιμοποιούμε τον TfidfVectorizer. Κρατάμε τα μονογράμματα και τα διγράμματα και μόνο αυτά με συχνότητα εμφάνισης τουλάχιστον 10. Τέλος κρατάμε τα 5000 πιο συχνά εμφανιζόμενα μονογράμματα ή διγράμματα.

```
from sklearn.feature_extraction.text import TfidfVectorizer

# Use unigram & bi-gram tf*idf features
vectorizer = TfidfVectorizer(
    ngram_range = (1, 2),
    max_features = 5000,
    #Apply sublinear tf scaling, i.e. replace tf with 1 + log(tf).
    sublinear_tf = True,
)


# Fit Vectorizer on train data
# Transform on all data (train - test)
x_train_tfidf = vectorizer.fit_transform(x_train)
x_dev_tfidf = vectorizer.fit_transform(x_dev)
x_test_tfidf = vectorizer.transform(x_test)

print(boolean_vectorizer.get_feature_names())
```

Στη συνέχεια με κριτήριο το Information Gain(Mutual Info) από τα 5000, κρατάμε τα 2500 καλύτερα features με τον ίδιο ακριβώς τρόπο όπως και στα boolean features.

Classifiers (Φοίβος Νταντάμης)

Για την κατηγοριοποίηση των διανυσμάτων μας χρησιμοποιούμε αρχικά μόνο για σκοπούς σύγκρισης τον Dummy Classifier. Στη συνέχεια χρησιμοποιούμε τον LogisticRegression, μία φορά με τα boolean features και μία φορά με τα tfidf features. Υπολογίζουμε για κάθε περίπτωση τα precision, recall, f1, auc, macro precision, macro recall, macro f1 και macro auc για κάθε set από τα train, dev και test χωριστά.

Dummy Classifier:

Για τον DummyClassifier χρησιμοποιούμ την στρατηγική επιλογής της πιο συχνά εμφανιζόμενης κλάσης

```
# Train
# The dummy classifier always predicts the 'most frequent' class
baseline = DummyClassifier(strategy='most_frequent')
start_time = time.time()
baseline.fit(x_train_tfidf, y_train)
print("Training took: {} seconds \n".format(time.time() - start_time))

# Evaluate

# Train
predictions_train_dummy_boolean = baseline.predict(x_train_boolean)
precision_train_dummy_boolean, recall_train_dummy_boolean, f1_train_dummy_boolean, support_train_dummy_boolean = precision_recall_fscore_support(y_train, prediction
macro_precision_train_dummy_boolean = (precision_train_dummy_boolean[0] + precision_train_dummy_boolean[1])/2
macro_recall_train_dummy_boolean = (recall_train_dummy_boolean[0] + recall_train_dummy_boolean[1])/2
macro_f1_train_dummy_boolean = (f1_train_dummy_boolean[0] + f1_train_dummy_boolean[1])/2
probs_y_train_dummy_boolean = baseline.predict_proba(x_train_boolean)
precisions_train_pos_dummy_boolean, recalls_train_pos_dummy_boolean, thresholds_train_pos_dummy_boolean = precision_recall_curve(y_train, probs_y_train_dummy_boolea
precisions_train_neg_dummy_boolean, recalls_train_neg_dummy_boolean, thresholds_train_neg_dummy_boolean = precision_recall_curve(y_train, probs_y_train_dummy_boolea
auc_train_pos_dummy_boolean = auc(recalls_train_pos_dummy_boolean, precisions_train_pos_dummy_boolean)
auc_train_neg_dummy_boolean = auc(recalls_train_neg_dummy_boolean, precisions_train_neg_dummy_boolean)
macro_auc_train_dummy_boolean = (auc_train_pos_dummy_boolean + auc_train_neg_dummy_boolean)/2
```

Για τον LogisticRegression χρησιμοποιούμε τον saga solver για να επιλέξουμε το elasticnet ως penalty, ώστε να μπορούμε να ορίσουμε έναν ή και τους δύο τύπους regularization l1 και l2, ορίζοντας κατάλληλα το ratio μεταξύ τους.

```
from sklearn.linear_model import LogisticRegression


clf = LogisticRegression(solver="saga", penalty='elasticnet', l1_ratio=0.5, C=0.5)
start_time = time.time()
clf.fit(x_train_boolean, y_train)
print("Training took: {} seconds \n".format(time.time() - start_time))
```

Απεικόνιση αποτελεσμάτων (Φοίβος Νταντάμης)

Τελικά αποτυπώνουμε όλα τα παραπάνω σε έναν πίνακα. Στις στήλες των macro μεγεθών δεν υπάρχει νόημα στον διαχωρισμό βάση των κλάσεων και για αυτό τα αποτελέσματα είναι τα ίδια και στις δύο στήλες κάθε φορά για ευκολία υλοποίησης.

Scores for Test

```
df = pd.DataFrame([[precision_test_dummy_boolean[0], precision_test_dummy_boolean[1], recall_test_dummy_boolean[0], recall_test_dummy_boolean[1], f1_test_dummy_bool
                    [precision_test[0], precision_test[1], recall_test[0], recall_test[1], f1_test[0], f1_test[1], auc_test_pos, auc_test_neg, macro_precision_test,
                    [precision_test_tfidf[0], precision_test_tfidf[1], recall_test_tfidf[0], recall_test_tfidf[1], f1_test_tfidf[0], f1_test_tfidf[1], auc_test_pos_t
                   index=pd.Index(['Dummy Classifier', 'Logistic Regression (Boolean)', 'Logistic Regression (Tf-Idf)'], name='Model:'),
                   columns=pd.MultiIndex.from_product([['Precision', 'Recall', 'F1', 'AUC', 'Macro Precision', 'Macro Recall', 'Macro F1', 'Macro AUC'],['Positive',
df.style
```

Scores:	Precision		Recall		F1		AUC		Macro Precision		Macro Recall		Macro F1		Macro AUC	
Class:	Positive	Negative	Positive	Negative	Positive	Negative	Positive	Negative	Positive	Negative	Positive	Negative	Positive	Negative	Positive	Negative
Model:																
Dummy Classifier	0.000000	0.495000	0.000000	1.000000	0.000000	0.662207	0.747500	0.752500	0.247500	0.247500	0.500000	0.500000	0.331104	0.331104	0.750000	0.750000
Logistic Regression (Boolean)	0.849462	0.794393	0.782178	0.858586	0.814433	0.825243	0.909950	0.321752	0.821927	0.821927	0.820382	0.820382	0.819838	0.819838	0.615851	0.615851
Logistic Regression (Tf-Idf)	0.590909	0.537313	0.386139	0.727273	0.467066	0.618026	0.574792	0.445113	0.564111	0.564111	0.556706	0.556706	0.542546	0.542546	0.509953	0.509953

Learning Curves (Παναγιώτης Καραστατήρης)

Για να φτιάξουμε για κάθε classifier τις 3 καμπύλες εκμάθησης στα 3 ζητούμενα dataset χρησιμοποιήσαμε τη συνάρτηση plot_learning_curve. Την καλούμε 4 φορές, 2 φορές για κάθε classifier (linear και dummy) διότι έχουμε 2 εναλλακτικούς vectorizer, τον Boolean και τον TfidfVectorizer.

plot_learning_curve : Της δίνουμε σαν ορίσματα κάθε φορά τα x_tfidf ή τα x_boolean στο train,dev,test dataset με τα αντίστοιχα y.

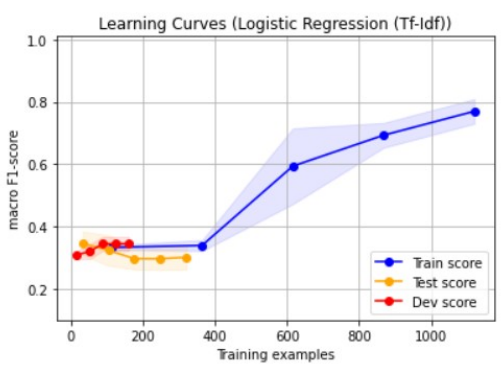
```
title = "Learning Curves (Logistic Regression (Tf-Idf))"
cv = ShuffleSplit(n_splits=5, test_size=0.2, random_state=0)
estimator = LogisticRegression(solver="saga", penalty='elasticnet', l1_ratio=0.5, C=0.5)
plot_learning_curve(estimator, title, x_train_tfidf, y_train, x_dev_tfidf, y_dev, x_test_tfidf, y_test,(0.1, 1.01), cv=cv, n_jobs=-1)



title = "Learning Curves (Logistic Regression (Boolean))"
cv = ShuffleSplit(n_splits=5, test_size=0.2, random_state=0)
estimator = LogisticRegression(solver="saga", penalty='elasticnet', l1_ratio=0.5, C=0.5)
plot_learning_curve(estimator, title, x_train_boolean, y_train, x_dev_boolean, y_dev, x_test_boolean, y_test,(0.1, 1.01), cv=cv, n_jobs=-1)


title = "Learning Curves (DummyClassifier (Tf-Idf))"
cv = ShuffleSplit(n_splits=5, test_size=0.2, random_state=0)
estimator = DummyClassifier(strategy='most_frequent')
plot_learning_curve(estimator, title, x_train_tfidf, y_train, x_dev_tfidf, y_dev, x_test_tfidf, y_test,(0.1, 1.01), cv=cv, n_jobs=-1)



title = "Learning Curves (DummyClassifier(Boolean))"
cv = ShuffleSplit(n_splits=5, test_size=0.2, random_state=0)
estimator = DummyClassifier(strategy='most_frequent')
plot_learning_curve(estimator, title, x_train_boolean, y_train, x_dev_boolean, y_dev, x_test_boolean, y_test,(0.1, 1.01), cv=cv, n_jobs=-1)
plt.show()
```

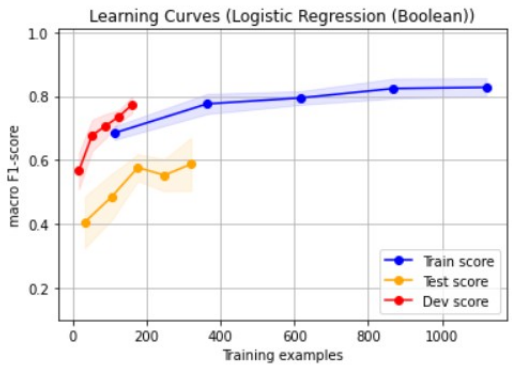
Εσωτερικά καλούμε 3 φορές την learning_curve μια φορά για κάθε dataset με τα αντίστοιχα ορίσματα και scoring='f1_macro' όπως και ζητείται.

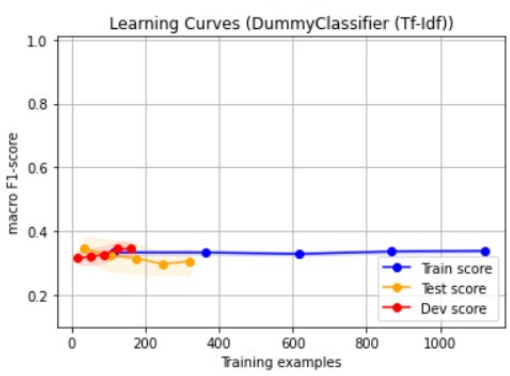
```
train_sizes1, train_scores, cv_train_scores = learning_curve(
    estimator, X, y, cv=cv, n_jobs=n_jobs,scoring='f1_macro', train_sizes=train_sizes)


train_sizes2, dev_scores, cv_dev_scores = learning_curve(
    estimator, dev_x, dev_y,cv=cv, n_jobs=n_jobs,scoring='f1_macro', train_sizes=train_sizes)

train_sizes3, test_scores, cv_test_scores = learning_curve(
    estimator, test_x, test_y,cv=cv, n_jobs=n_jobs,scoring='f1_macro', train_sizes=train_sizes)
```

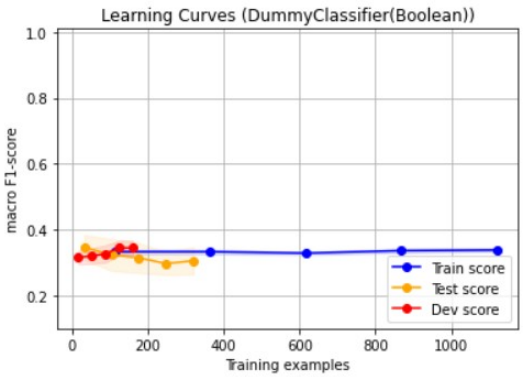
Με τα scores στη συνέχεια φτιάχνουμε τα αντίστοιχα mean και std scores τα οποία χρησιμοποιούν η plt.fill_between και η plt.plot.

```
plt.fill_between(train_sizes1, cv_train_scores_mean -  cv_train_scores_std,
                 cv_train_scores_mean +  cv_train_scores_std, alpha=0.1,
                 color="b")
plt.fill_between(train_sizes2, cv_test_scores_mean - cv_test_scores_std,
                 cv_test_scores_mean + cv_test_scores_std, alpha=0.1, color="orange")

plt.fill_between(train_sizes3, cv_dev_scores_mean - cv_dev_scores_std,
                 cv_dev_scores_mean + cv_dev_scores_std, alpha=0.1, color="red")

plt.plot(train_sizes1, cv_train_scores_mean, 'o-', color="b",
         label="Train score")
plt.plot(train_sizes2, cv_test_scores_mean, 'o-', color="orange",
         label="Test score")
plt.plot(train_sizes3, cv_dev_scores_mean, 'o-', color="red",
         label="Dev score")
```









Hypermaremeter tuning (Φοίβος Νταντάμης)

Το ερώτημα αυτό είναι ημιτελές.

Προκειμένου να διαλέξουμε τις τιμές των υπερπαραμέτρων c, l1_ratio αλλά και τον αριθμό των features, που κρατάμε θα έπρεπε να ψάξουμε σε όλο τις διαστάσεις του προβλήματος αυτού και να διαλέξουμε τις καλύτερες τιμές. Προκειμένου να βρούμε τις τιμές αυτές, αρχικά φτιάχνουμε ένα pipeline το οποίο θα τροφοδοτήσουμε σε κάποια συνάρτηση αναζήτησης βέλτιστης λύσης.

```
from sklearn.pipeline import Pipeline
from sklearn.linear_model import LogisticRegression
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.feature_selection import SelectKBest

reudce_dim = SelectFromModel(estimator=MutualInfoEstimator(random_state=42))


lr_pipeline = Pipeline([
    ('count_vectorizer', CountVectorizer()),
    ('lr', LogisticRegression())])
```

Στο pipeline θα θέλαμε εκτός από τη διαδικασία του vectorizing και του solving να έχουμε και τη διαδικασία του feature reduction μέσω mutual_information. Δεν υπάρχει αυτή η διαδικασία στο sklearn οπότε έπρεπε να φτιάξουμε έναν δικό μας estimator που θα χρησιμοποιεί το mutual_info_classif και θα επιστρέφει τα καλύτερα features στο επόμενο stage του pipeline

```python
from sklearn.base import BaseEstimator
from sklearn.feature_selection import SelectFromModel

class MutualInfoEstimator(BaseEstimator):
    def __init__(self, discrete_features=True, n_neighbors=3, copy=True, random_state=None, number_of_features=2500):
        self.discrete_features = discrete_features
        self.n_neighbors = n_neighbors
        self.copy = copy
        self.random_state = random_state
        self.number_of_features = number_of_features


    def fit(self, X, y):

       mutual_information = mutual_info_classif(X, y, discrete_features=self.discrete_features,
                                                n_neighbors=self.n_neighbors,
                                                copy=self.copy, random_state=self.random_state)
       indexes = np.arange(len(mutual_information))
       sorted_indexes = sorted(zip(mutual_information, indexes), reverse=True)
       best_idxs = [idx for val, idx in sorted_indexes[:self.number_of_features]]
       self.feature_importances_ = X[:, best_idxs]
```

Στη συνέχεια ο RandomSearch θα δεχόταν τα εύρη των παραμέτρων στα οποία αναζητάμε λύση και θα έβγαζε τον καλύτερο συνδυασμό. Δυστυχώς κάτι σκάει σε αυτήν τη διαδικασία.

```python
from sklearn.model_selection import RandomizedSearchCV


grid_params = {
  'count_vectorizer__min_df': np.linspace(0.01, 0.1, 10),
  'count_vectorizer__ngram_range': [(1,1)],
  'count_vectorizer__binary': [True],
  'lr__solver': ['saga'],
  'lr__C': np.linspace(0.1, 1, 10),
  'lr__max_iter': [100, 200, 300],
  'lr__penalty': ['elasticnet'],
  'lr__l1_ratio': np.linspace(0.1, 1, 10)
}

clf2 = RandomizedSearchCV(lr_pipeline, grid_params, scoring='f1', cv=3, n_jobs=-1, n_iter=100)
start_time = time.time()
clf2.fit(x_dev_boolean, y_dev)
print("RandomizedSearchCV took: {} seconds \n".format(time.time() - start_time))

print("Best Score: ", clf2.best_score_)
print("Best Params: ", clf2.best_params_)
```