DATA IN MOTION

One-Class Text Classification

21.07.2023

One-Class Text Classification

Part Contents

1 Suggester for Privacy-related Model Fields

Section Contents

1 Suggester for Privacy-related Model Fields
Introduction
One-Class Text Classification

Introduction

- Our first use case is, given a meta-model (e.g. person.ecore) provide warnings and suggestions on fields and/or combination of fields that might constitute a privacy risk under a certain regulation (e.g. GDPR) or that have to obey to certain open data standards;
- We can see the problem as a one-class text classification problem, where our only class is the category of privacy-related terms, while all the other terms have to be identified as outliers;
- We can then train a model on a set of terms which we consider privacy-related, and then test it over some other examples.

Section Contents

1 Suggester for Privacy-related Model Fields

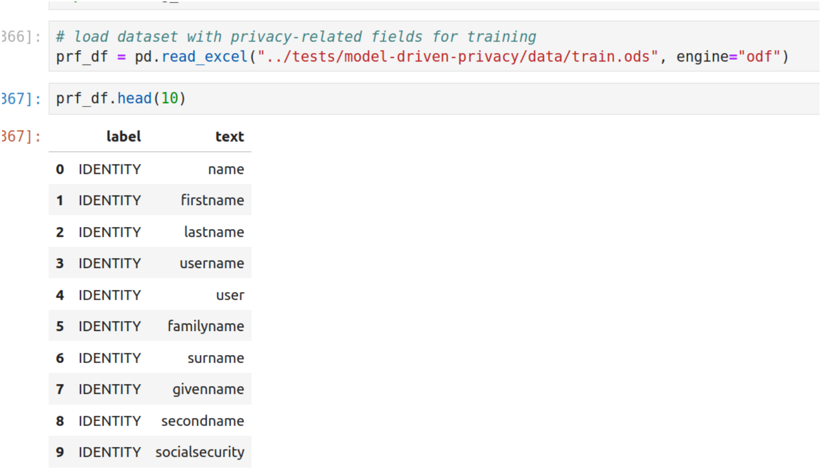
Introduction

One-Class Text Classification

Training Set

- I built an initial training set, with all the terms I could think of and that may constitute a privacy concern, namely terms which can potentially be associated with personal information, as so, that might result in identifying an individual;
- The terms are written as I would expect to encounter them as a model field (e.g. “first name” is firstname), all in lower case and singular, to avoid useless repetitions.
- I assigned these terms a *label* IDENTITY, meaning that they are terms that might be associated with the identity of an individual.

Training Set



```
366]: # load dataset with privacy-related fields for training
      prf_df = pd.read_excel("../tests/model-driven-privacy/data/train.ods", engine="odf")
```

```
367]: prf_df.head(10)
```

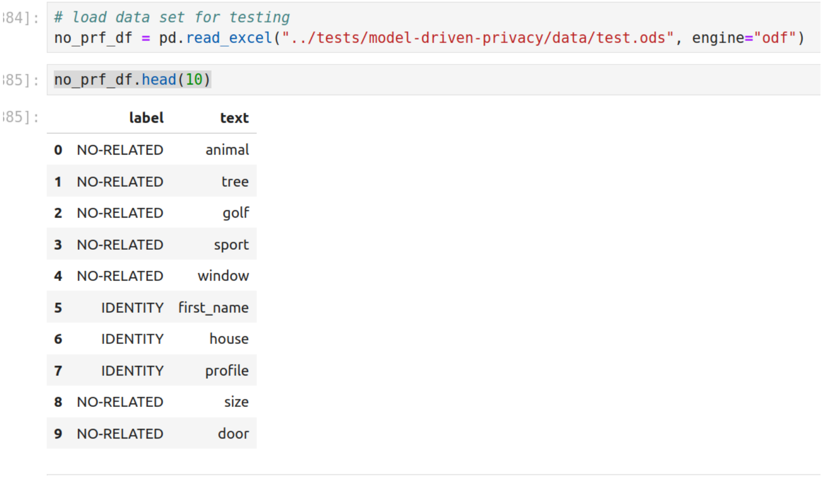
367]:

	label	text
0	IDENTITY	name
1	IDENTITY	firstname
2	IDENTITY	lastname
3	IDENTITY	username
4	IDENTITY	user
5	IDENTITY	familyname
6	IDENTITY	surname
7	IDENTITY	givenname
8	IDENTITY	secondname
9	IDENTITY	socialsecurity

Test Set

- For testing I build another set of terms, this time containing both terms that I would expect to be suggested as privacy-related and terms that should have nothing to do with privacy-related matters;
- I also used some of the terms I put for training, but written in a slightly different manner (plural, snake_case, synonyms);
- It would be the task of the preprocessing to get rid of such minor differences and of the model itself to identify similar terms as well.

Test Set



```
84]: # load data set for testing
     no_prf_df = pd.read_excel("../tests/model-driven-privacy/data/test.ods", engine="odf")
```

```
85]: no_prf_df.head(10)
```

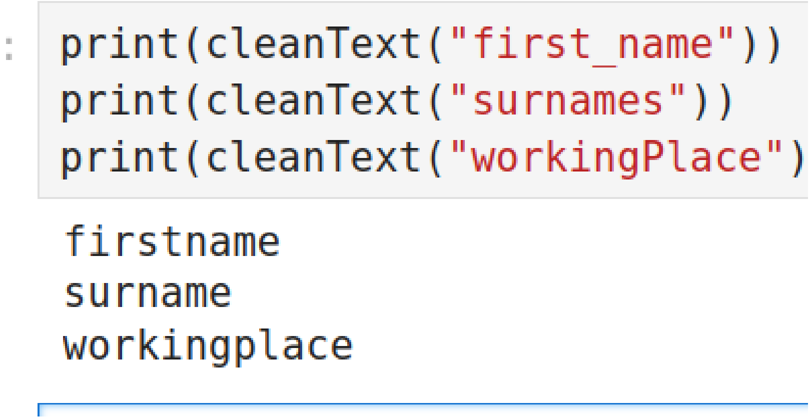
85]:

	label	text
0	NO-RELATED	animal
1	NO-RELATED	tree
2	NO-RELATED	golf
3	NO-RELATED	sport
4	NO-RELATED	window
5	IDENTITY	first_name
6	IDENTITY	house
7	IDENTITY	profile
8	NO-RELATED	size
9	NO-RELATED	door

Data Pre-processing

- The input text is first stripped, all the new lines and carriage return characters are removed;
- Then, if text is in snake_case, it is converted in camelCase;
- Special symbols are removed;
- The text is divided into lower case parts based on the camelCase definition (e.g. firstName > first , name);
- Each part is then lemmatized (e.g. names > name);
- The parts are then put back together to form just a single term.

Data Pre-processing



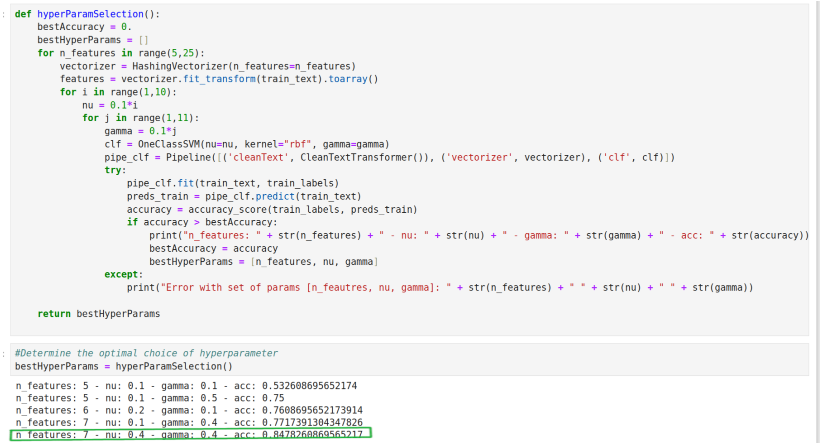
```
print(cleanText("first_name"))
print(cleanText("surnames"))
print(cleanText("workingPlace")
```

```
firstname
surname
workingplace
```

The Model

- Then the data has to be “vectorized”, meaning from text we have to pass to a mathematical vector that the algorithm is able to understand;
- For that I used an HashingVectorizer (there are others, so more research here might be needed), which has one hyper-parameter (n_features) to set;
- I then trained a OneClassSVM model, using the scikit□learn python library, which requires other 3 hyper-parameters;
- I built a function that loops over a set of values for each hyper-parameter, and for each combination fits the model and computes the accuracy on the training set;
- I then picked the set of hyper-parameters that gave me the best accuracy.

The Model



```
def hyperParamSelection():
    bestAccuracy = 0.
    bestHyperParams = []
    for n_features in range(5,25):
        vectorizer = HashingVectorizer(n_features=n_features)
        features = vectorizer.fit_transform(train_text).toarray()
        for i in range(1,10):
            nu = 0.1*i
            for j in range(1,11):
                gamma = 0.1*j
                clf = OneClassSVM(nu=nu, kernel="rbf", gamma=gamma)
                pipe_clf = Pipeline([('cleanText', CleanTextTransformer()), ('vectorizer', vectorizer), ('clf', clf)])
                try:
                    pipe_clf.fit(train_text, train_labels)
                    preds_train = pipe_clf.predict(train_text)
                    accuracy = accuracy_score(train_labels, preds_train)
                    if accuracy > bestAccuracy:
                        print("n_features: " + str(n_features) + " - nu: " + str(nu) + " - gamma: " + str(gamma) + " - acc: " + str(accuracy))
                        bestAccuracy = accuracy
                        bestHyperParams = [n_features, nu, gamma]
                except:
                    print("Error with set of params [n_feautres, nu, gamma]: " + str(n_features) + " " + str(nu) + " " + str(gamma))

    return bestHyperParams
```

```
#Determine the optimal choice of hyperparameter
bestHyperParams = hyperParamSelection()
```

```
n_features: 5 - nu: 0.1 - gamma: 0.1 - acc: 0.532608695652174
n_features: 5 - nu: 0.1 - gamma: 0.5 - acc: 0.75
n_features: 6 - nu: 0.2 - gamma: 0.1 - acc: 0.7608695652173914
n_features: 7 - nu: 0.1 - gamma: 0.4 - acc: 0.7717391304347826
n_features: 7 - nu: 0.4 - gamma: 0.4 - acc: 0.8478260869565217
```

The Results

- After the model was trained I tested it on the test set, and compute precision and recall for both categories (privacy-related terms and non);
- *Precision* is computed as the ratio between the relevant retrieved instances and all the retrieved instances (e.g. how many privacy-related terms are correctly identified over the total number of terms identified as privacy-related terms);
- *Recall* is computed as the ratio between the relevant retrieved instances and all the relevant instances (e.g. how many privacy-related terms are correctly identified over the total number of privacy-related terms).

The Results

Category	Precision	Recall
Privacy-related terms	0.32	1.00
NON privacy-related terms	1.00	0.23

The Results

- All privacy-related terms are properly identified as such (recall 1.0 for that category);
- Of course, also some non-related terms are mis-identified as privacy-related;
- This is not too bad, as we just want a suggestion mechanism, so it's better that we find all the relevant ones, plus some more, than the other way around.

Further Steps

- There are some additional steps for pre-processing data that I would like to explore (e.g. stemming);
- I have not spent too much time into the different ways of vectorizing text, so it might be I am not using the fancier one for our purposes;
- The training set could probably be enriched with more terms.

Conclusion

Useful Links

OSGi Working Group

Working Group: www.osgi.org
WG Blog: www.osgi.org/blog
Twitter: @osgiwg
Bndtools: bndtools.org

Data In Motion

Web: www.datainmotion.com
Blog: datainmotion.com/blog
Twitter: @motion_data

Jürgen Albert

Email: j.albert@data-in-motion.biz

Mark Hoffmann

Email:
m.hoffmann@data-in-motion.biz