

Data Mining Capstone Task 6

Thomas. A.W. Tilli

Introduction

Overview of task 6: (see <https://class.coursera.org/dataminingcapstone-001/wiki/Task6>)

“Sometimes we make decisions beyond the rating of a restaurant. For example, if a restaurant has a high rating but it often fails to pass hygiene inspections, then this information can dissuade many people to eat there. Using this hygiene information could lead to a more informative system; however, it is often the case where we don’t have such information for all the restaurants, and we are left to make predictions based on the small sample of data points.

In this task, you are going to predict whether a set of restaurants will pass the public health inspection tests given the corresponding Yelp text reviews along with some additional information such as the locations and cuisines offered in these restaurants. Making a prediction about an unobserved attribute using data mining techniques represents a wide range of important applications of data mining. Through working on this task, you will gain direct experience with such an application. Due to the flexibility of using as many indicators for prediction as possible, this would also give you an opportunity to potentially combine many different algorithms you have learned from the courses in the Data Mining Specialization to solve a real world problem and experiment with different methods to understand what’s the most effective way of solving the problem.”

Very important (!!):

“As you have probably noticed, this task is similar to Task 4 in the programming assignment of the Text Mining and Analytics course; however, there are three major differences:

1. The training data is perfectly balanced, whereas the testing data is skewed, which creates a new challenge since the training and testing data have different distributions.
2. The main performance metric is the F1 score as opposed to the classification accuracy that was used in the Text Mining course. This means that a good classifier is expected to perform well on both classes.
3. Extra non-textual features such as the cuisines, locations, and average rating are given. This might help in further improving the prediction performance and provide an opportunity to experiment with many more strategies for solving the problem.”

The dataset contains 546 training instances with labels and 12753 (!!) test instances without labels.

This has the following consequences:

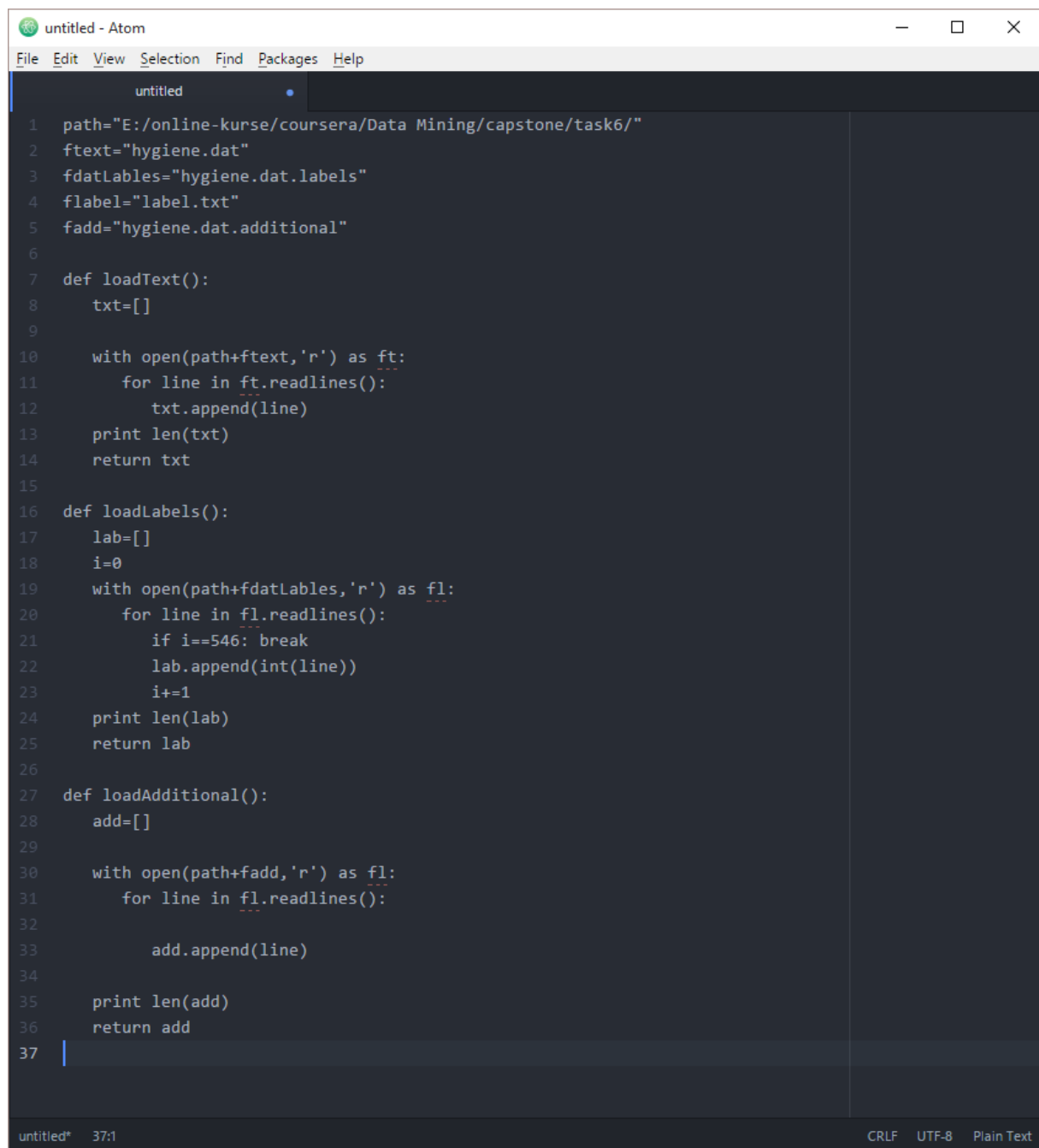
1. The training data set is very small (and balanced) compared with the testing data set (skewed). Therefore I decided not to split up the training data set into a new training data set and some small testing data set for classifier evaluation. I use the training data set completely for training and I use it to for an evaluation by the F1 metrics. The F1 values calculated in this way will only give some crude indication. If the F1 value calculated in this way is above 0.95 then it is an clear indication for overfitting, since

the classifier is then able to predict the training data set perfectly. In such a case the performance on the test data set will be quite poor.

2. The testing data set will be evaluated by the F1 metrics by submission on the Coursera web site. This means, that the evaluation of classifier can only be done by manually submitting the labeled test instances on the Coursera web site and then inspecting the F1 value after submission. Therefore an automatic grid search for optimal classificatory parameters cannot be done automatically but only manually.

Processing Pipeline

I used skikit learn for the classification tasks. First of all I loaded all datasets into three dictionaries with the following simple python code:



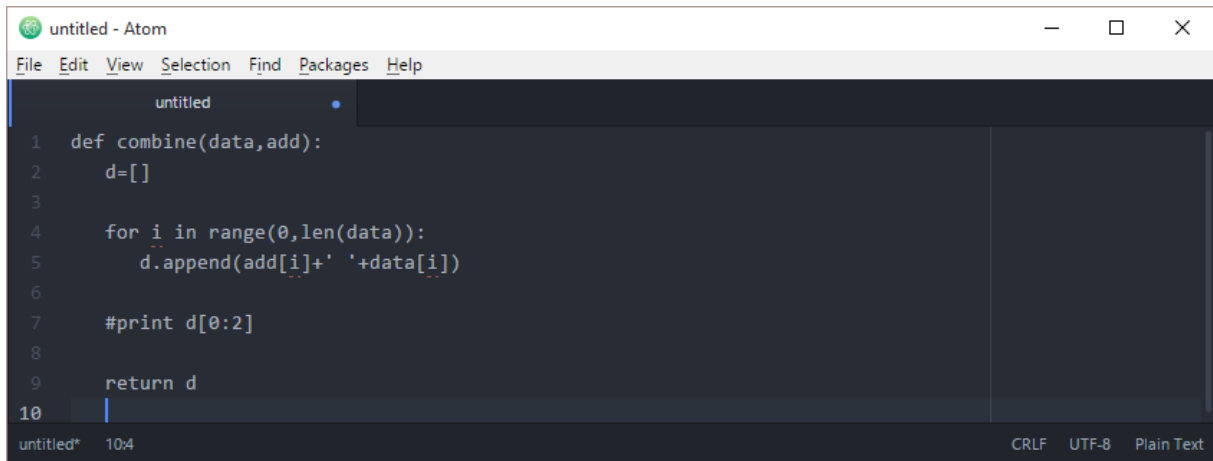
```
untitled - Atom
File Edit View Selection Find Packages Help

untitled
1 path="E:/online-kurse/coursera/Data Mining/capstone/task6/"
2 ftext="hygiene.dat"
3 fdatLables="hygiene.dat.labels"
4 flabel="label.txt"
5 fadd="hygiene.dat.additional"
6
7 def loadText():
8     txt=[]
9
10    with open(path+ftext,'r') as ft:
11        for line in ft.readlines():
12            txt.append(line)
13    print len(txt)
14    return txt
15
16 def loadLabels():
17     lab=[]
18     i=0
19     with open(path+fdatLables,'r') as fl:
20         for line in fl.readlines():
21             if i==546: break
22             lab.append(int(line))
23             i+=1
24     print len(lab)
25     return lab
26
27 def loadAdditional():
28     add=[]
29
30     with open(path+fadd,'r') as fl:
31         for line in fl.readlines():
32
33             add.append(line)
34
35     print len(add)
36     return add
37
```

untitled* 37:1 CRLF UTF-8 Plain Text

I perform no preprocessing on the data, with one exception: I skipped all entries in the label file after line 546 because all further lines contains the entry “none”.

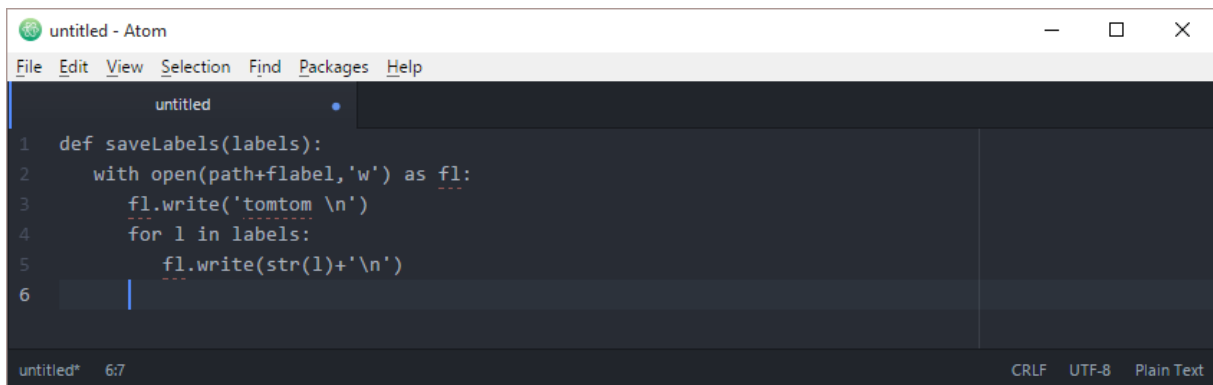
Furthermore I use the following simple function to combine the contents in the files “hygiene.dat” and “hygiene.dat.additional” line by line:



```
untitled - Atom
File Edit View Selection Find Packages Help
untitled
1 def combine(data,add):
2     d=[]
3
4     for i in range(0,len(data)):
5         d.append(add[i]+' '+data[i])
6
7     #print d[0:2]
8
9     return d
10
untitled* 10:4 CRLF UTF-8 Plain Text
```

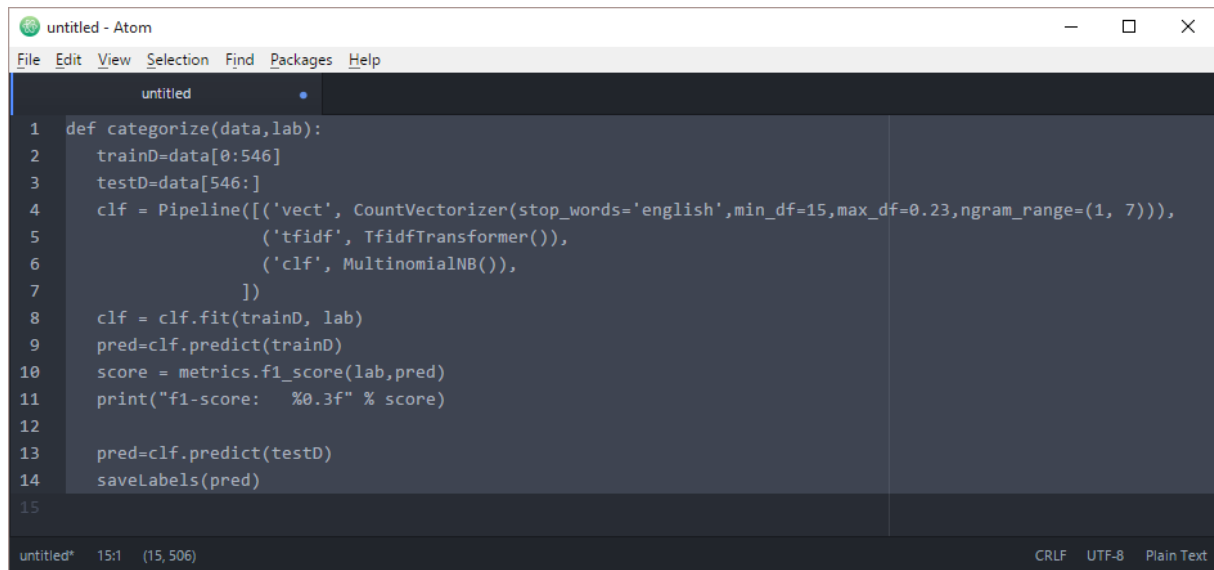
It is simply a concatenation.

Then I used the following simple helper function to save the predicted labels of the test data set:



```
untitled - Atom
File Edit View Selection Find Packages Help
untitled
1 def saveLabels(labels):
2     with open(path+flabel,'w') as fl:
3         fl.write('tomtom \n')
4         for l in labels:
5             fl.write(str(l)+'\n')
6
untitled* 6:7 CRLF UTF-8 Plain Text
```

The text processing, classifier learning and the calculation of the predicted labels is done with one python function using the skikit text processing pipeline:

The image shows a screenshot of the Atom code editor. The window title is "untitled - Atom". The menu bar includes "File", "Edit", "View", "Selection", "Find", "Packages", and "Help". The code is written in Python and defines a function named "categorize". The function takes two arguments: "data" and "lab". It splits the data into training and testing sets. A pipeline is created with a CountVectorizer, a TfidfTransformer, and a MultinomialNB classifier. The classifier is trained on the training data, and predictions are made on both training and testing data. The F1 score is calculated and printed. The status bar at the bottom shows "untitled*", "15:1", "(15, 506)", "CRLF", "UTF-8", and "Plain Text".

```
1 def categorize(data,lab):
2     trainD=data[0:546]
3     testD=data[546:]
4     clf = Pipeline([('vect', CountVectorizer(stop_words='english',min_df=15,max_df=0.23,ngram_range=(1, 7))),
5                     ('tfidf', TfidfTransformer()),
6                     ('clf', MultinomialNB()),
7                     ])
8     clf = clf.fit(trainD, lab)
9     pred=clf.predict(trainD)
10    score = metrics.f1_score(lab,pred)
11    print("f1-score:  %0.3f" % score)
12
13    pred=clf.predict(testD)
14    saveLabels(pred)
15
```

The first two lines splits the data set into the (rather small) trainings data set and in the large test data set. The lines 4 to 7 are for setting up the processing pipeline for the classifier. The first step is the vectorizer – here it shows the usage of a count vectorizer with some parameters: the `min_df` value, the `max_df` value and the `ngram_range` – here ngrams with 1 to 7 words are used. Instead of a count vectorizer a Tf Idf vectorizer could also be used.

The next step is an `TfidfTransformer` (default is using inverse-document-frequency reweighting). Here a transformation of a count matrix to a normalized tf or tf-idf representation is done.

The next step defines the classifier itself. The code shows a Naive Bayes classifier for multinomial models for classification with discrete features, well suited for word counts for text classification.

Line 8 shows the training of the classifier, line 9 and 10 the prediction on the training data set and F1 evaluation, line 13 the prediction on the test data set.

Steps performed

I evaluated a lot of classifiers. In general I have done the following main tasks:

1. Classifier design and selection using only the data in file “hygiene.dat” with the labels provided and only using unigram models.
2. Classifier design and selection using the data in files “hygiene.dat” and “hygiene.dat.additional” with the labels provided and only using ngram models.

Task 1

In this task I used only data in file “hygiene.dat” with the labels provided and only using unigram models. First I used the classifier `MultinomialNB` (naive Bayes) and I tried to tune the parameters of the count vectorizer. The best results I get for the following parameters of the count vectorizer:

`min_df=3,max_df=0.249, stop_words='english'.`

For this parameter set and the classifier `MultinomialNB` I got the following results after submitting:

The F1 score of this submission is: 0.566072055142. Your highest F1 score is: 0.566072055142. At the time of this submission, you ranked: 15.

Then I tried to get better results by using the TfidfVectorizer, but in all tests the CountVectorizer performs slightly better. Therefore in all following steps I used only the CountVectorizer.

Then I evaluated different classifiers:

1. MultinomialNB – this is a Naive Bayes classifier for multinomial models for classification with discrete features.
2. BernoulliNB - this is Naive Bayes classifier for multivariate Bernoulli models suitable for discrete data (Boolean features).
3. GaussianNB – this is Gaussian Naive Bayes classifier.
4. KNeighborsClassifier - this is a K neighbours classifier
5. DecisionTreeClassifier -this is a decision tree classifier
6. RandomForestClassifier - this is an ensemble type classifier based on decision trees
7. AdaBoostClassifier - this is an ensemble type classifier is a meta-estimator that begins by fitting a classifier on the original dataset and then fits additional copies of the classifier on the same dataset but where the weights of incorrectly classified instances are adjusted such that subsequent classifiers focus more on difficult cases.
8. LDA – this is Linear Discriminant Analysis classifier.
9. QDA – this is Quadratic Discriminant Analysis classifier.
10. SGDClassifier - Linear classifiers (SVM, logistic regression, a.o.) with SGD training.
11. SVC – support vector machine classifier.

The results are:

- RandomForestClassifier , AdaBoostClassifier and SVC performs quite bad, mainly due to overfitting problems – in almost all cases the F1 value calculated on the training data was 1.00 – clearly a strong indication of overfitting. Maybe with a much larger training data set the results would be much better.
- LDA and QDA, KNeighborsClassifier and the DecisionTreeClassifier performed not well either. This is not a really surprise, they seldom performed best on text classification problems with a large feature set.
- GaussianNB performs worse than MultinomialNB, simply because this classifier is not well suited for discrete features.
- BernoulliNB performs slightly worse than MultinomialNB.
- SGDClassifier – this classifier gives also good results depending on the parameters and loss function used. Possible loss functions are:
 - ‘hinge’, ‘log’, ‘modified_huber’, ‘squared_hinge’, ‘perceptron’, or a regression loss: ‘squared_loss’, ‘huber’, ‘epsilon_insensitive’, or ‘squared_epsilon_insensitive’ The loss function to be used. Defaults to ‘hinge’, which gives a linear SVM. The ‘log’ loss gives logistic regression, a probabilistic classifier. ‘modified_huber’ is another smooth loss that brings tolerance to outliers as well as probability estimates. ‘squared_hinge’ is like hinge but is quadratically penalized. ‘perceptron’ is the linear loss used by the perceptron algorithm. The other losses are designed for regression but can be useful in classification as well; see SGDRegressor for a description.
 - As expected the “log” loss function performs best, since we have discrete labels and logistic regression is designed to deal with that.

Task 2

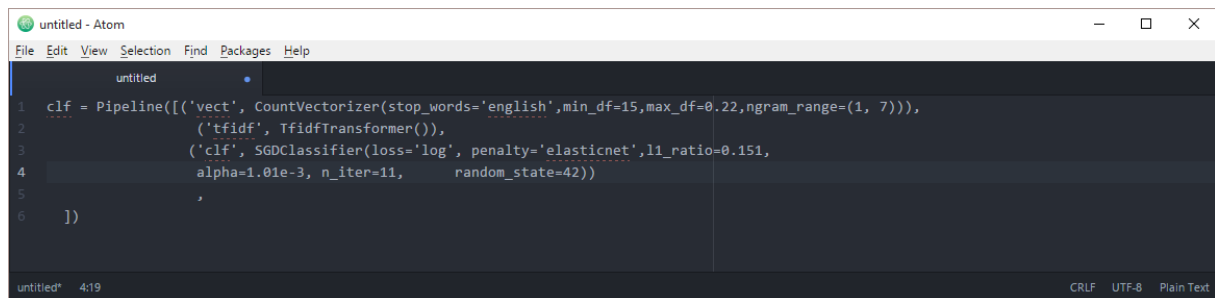
To get a better F1 performance I used now the additional information and made classifier design and selection using the data in files “hygiene.dat” and “hygiene.dat.additional” with the labels provided and only using unigram models. In this task I used first the MultinomialNB classifier. After some parameter tuning of the CountVectorizer I got the best results for the following parameter set:

- min_df=15,max_df=0.23,ngram_range=(1, 7)

and after submission I got the following results:

The F1 score of this submission is: 0.572304742229. Your highest F1 score is: 0.572304742229. At the time of this submission, you ranked: 2.

Further tuning was not possible, therefore I switched to the SGDClassifier. Here are the processing pipeline I used to get the best results:

A screenshot of the Atom code editor. The window title is 'untitled - Atom'. The menu bar includes File, Edit, View, Selection, Find, Packages, and Help. The code is written in a dark-themed editor and defines a Pipeline with two steps: a CountVectorizer and an SGDClassifier. The CountVectorizer parameters are stop_words='english', min_df=15, max_df=0.22, and ngram_range=(1, 7). The SGDClassifier parameters are loss='log', penalty='elasticnet', l1_ratio=0.151, alpha=1.01e-3, n_iter=11, and random_state=42. The status bar at the bottom shows 'untitled*' 4:19 and encoding options CRLF, UTF-8, and Plain Text.

```
1 clf = Pipeline([('vect', CountVectorizer(stop_words='english', min_df=15, max_df=0.22, ngram_range=(1, 7))),
2               ('tfidf', TfidfTransformer()),
3               ('clf', SGDClassifier(loss='log', penalty='elasticnet', l1_ratio=0.151,
4               alpha=1.01e-3, n_iter=11, random_state=42))
5               ])
6 ])
```

A CountVectorizer is used, with min_df=15, max_df=0.22 and ngrams from 1 to 7. As classifier an SGDClassifier with logistic regression is used. The penalty is elasticnet with parameter l1_ratio=0.151 (the Elastic Net mixing parameter, with $0 \leq l1_ratio \leq 1$. l1_ratio=0 corresponds to L2 penalty, l1_ratio=1 to L1. Defaults to 0.15). Alpha is a constant that multiplies the regularization term. n_iter = 11, the number of passes over the training data (aka epochs). The number of iterations is set to 1 if using partial_fit. Defaults to 5.

With this classifier I got after submission the following result:

The F1 score of this submission is: 0.574579584507. Your highest F1 score is: 0.574579584507. At the time of this submission, you ranked: 1.

Some Notes

The problem was difficult due to the small training data set and the large skewed test data set and the tedious manual evaluation through submission on the Coursera web site. The small training data set and the large skewed test data set explains the rather low F1 score achieved. The sklearn python library is well suited for text classification tasks with medium sized data sets (maybe up to some GBs) and is very easy to use as the code provided shows.

