

WEB SERVICE CLASSIFICATION

Comparing Modelling techniques and machine learning algorithms

Table of Contents

SERVICE MINING PROBLEM.....	1
Motivation and Benefit	1
EVALUATION PLAN	1
TOOLS USED	2
RESULTS COMPARISON	2
DATA PREPROCESSING AND PREPARATION:.....	2
IMPLEMENTATION AND RESULTS(with classifier algorithm comparison).....	4
Modelling Techniques:	4
1) TF-IDF	4
2) TOPIC MODELLIDNG	6
3) WORD EMBEDDING(Word2Vec)	7
OVERALL COMPARISON:	10
CONCLUSION	11
FUTURE WORK	11
REFERENCES	11

SERVICE MINING PROBLEM

Given the parsed_api data from assignment 3, I am performing text classification on it to identify the “Category” of an API given a service description, name, tags , title and summary. The goal is to compare and investigate the modelling techniques and machine learning algorithms to identify which combination performs the best for this data.

Motivation and Benefit

This aided in improving the current understanding of natural language processing techniques and helped to understand the underlying behavior of various text modelling techniques. We have discussed these service mining techniques in class and the assignment helps to provide deeper understanding of those concepts by means of implementation.

EVALUATION PLAN

Following Modeling techniques are used to get word vectors:

- 1) TF-IDF

- 2) Topic Modelling
- 3) WordEmbedding (WordToVec)

The machine learning algorithm ran on the generated vectors are:

- 1) Decision tree
- 2) Multinomial Naïve Bayes(because this is a multiclass classification problem)
- 3) K – Nearest neighbor
- 4) SVM
- 5) Logistic Regression

TOOLS USED

- 1) Python's scikit learn library is used to run machine learning algorithms, perform cross validation and to get the accuracy
- 2) Matplot lib and seaborn is used to create plots
- 3) Genism: used for word embedding
- 4) NLTK is used for data preprocessing such as tokenization, stopwords removal and stemming.
- 5) Pandas, Numpy is used to read and process data frame.

RESULTS COMPARISON

The data is prepared and the modeling techniques and algorithms are applied on it.

The results are then evaluated and compared as below:

- k – Cross fold validation is used with value of K = 10 to get a mean accuracy in order to avoid overfitting.
- I found the problem statement interesting and have tried another machine algorithm that was not mentioned in the assignment description ("Logistic Regression").
- As the number of class labels are more than 40, confusion matrix is a bit difficult to show for every case. I have shown it for some of the combinations.
- A comparison plot is generated for every modelling technique.
- I have compared the accuracies of TF_IDF and topic modelling for raw data and cleaned data as well.
- The machine learning algorithms accuracy for a particular modelling technique is compared (refer to [Implementation and results](#) section) and then an overall comparison of all modelling techniques is done (refer to [overall comparison](#) section)

DATA PREPROCESSING AND PREPARATION:

The api data is gathered from assignment 3 by exporting it in csv format from mongo db database.

- As can be seen from the histogram, Data is biased meaning the number of records are unbalanced. There are approximately 11199 rows and 66 categories in the raw data. As shown in figure below:



- Below is the distribution of cleaned data:



IMPLEMENTATION AND RESULTS(with classifier algorithm comparison)

Modelling Techniques:

The mining algorithms do not process the text directly. They expect the features in numerical form. Therefore, following modelling techniques are used to generate feature vectors:

1) TF-IDF

In this technique term frequency per document is calculated. I have used sk-learn's `sklearn.feature_extraction.text.TfidfVectorizer` to calculate tf-idf vector for the api text description.

```
tfidf = TfidfVectorizer(sublinear_tf=True, min_df=5, norm='l2', encoding='latin-1', ngram_range=(1, 2), stop_words='english')
features = tfidf.fit_transform(df.full_text).toarray()
labels = df.category_id
print(features.shape)
```

The shape of resultant features is:

(10557, 16931)


Meaning that each of 10557 api description is represented by 16931 features.

The count vectorizer is used to convert vector to numbers. The multinomial variant of Naïve Bayes is suitable in this case.

Below are the results of experimenting all machine learning algorithms on the resultant vectors:

Code:

A list of models is generated and the features are passed on the models one by one with 10-fold cross validation. The average accuracy from k-fold validation is displayed.

```
models = [
     tree.DecisionTreeClassifier(max_depth = 5),
    KNeighborsClassifier(),
    LinearSVC(),
    MultinomialNB(),
    LogisticRegression()
]
```

```

CV = 10
cv_df = pd.DataFrame(index=range(CV * len(models)))
entries = []
for model in models:
    model_name = model.__class__.__name__
    accuracies = cross_val_score(model, features, labels, scoring='accuracy', cv=CV)
    for fold_idx, accuracy in enumerate(accuracies):
        entries.append((model_name, fold_idx, accuracy))
cv_df = pd.DataFrame(entries, columns=['model_name', 'fold_idx', 'accuracy'])
print(cv_df.groupby('model_name').accuracy.mean())

```

Results:

SVM performs best with TF-IDF giving an accuracy of 80%, followed by Logistic regression with 76% accuracy. The Decision tree classifier till 5 depth performs the worst. The circles on the plot show accuracy for every k of k-fold cross validation.

TF-IDF	
model generated	
model_name	
DecisionTreeClassifier	27.326426
KNN	32.000000
LinearSVC	80.730936
LogisticRegression	76.544823
MultinomialNB	44.462833

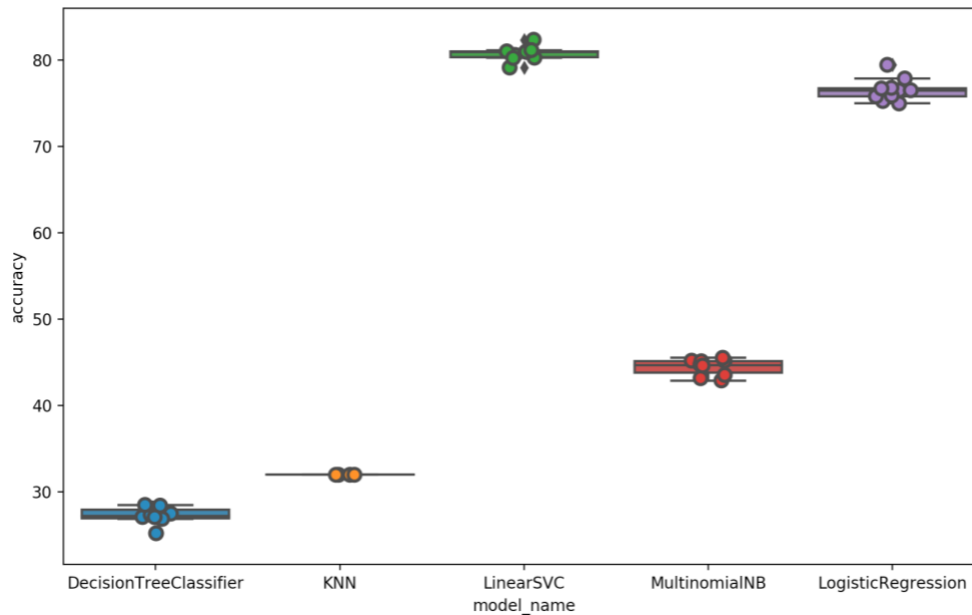


FIGURE: 3

2) TOPIC MODELLING

LDA is used to perform topic modelling to get feature vectors.

I have experimented with different number of topics to identify what gives the best accuracies.

40 topics	130 topics
----- LDA -----	----- LDA -----
model_name	model_name
DecisionTreeClassifier 21.198791	DecisionTreeClassifier 21.870314
KNeighborsClassifier 29.041398	KNN 12.250000
LinearSVC 33.693366	LinearSVC 41.068507
LogisticRegression 28.909663	MultinomialNB 13.227261
MultinomialNB 14.046375	Name: accuracy, dtype: float64
Name: accuracy, dtype: float64	

LDA performs best with 100 topics.

```
from sklearn.decomposition import LatentDirichletAllocation
number_of_topics = 100
LDAModel = LatentDirichletAllocation(n_components=number_of_topics, random_state=0)
lda_x = LDAModel.fit_transform(All_trainCount)
```

Below are the respective accuracies for 5 algorithms:

One would think that LDA should perform better but the result states otherwise for this problem. SVC again outperforms other machine learning algorithms with accuracy of 42%. However, the accuracies are less than TF-IDF modelling approach.

----- LDA -----
model_name
DecisionTreeClassifier 21.390576
KNeighborsClassifier 32.183873
LinearSVC 42.319380
LogisticRegression 32.898491
MultinomialNB 14.582402

RESULT OF LDA BY USING 100 TOPICS

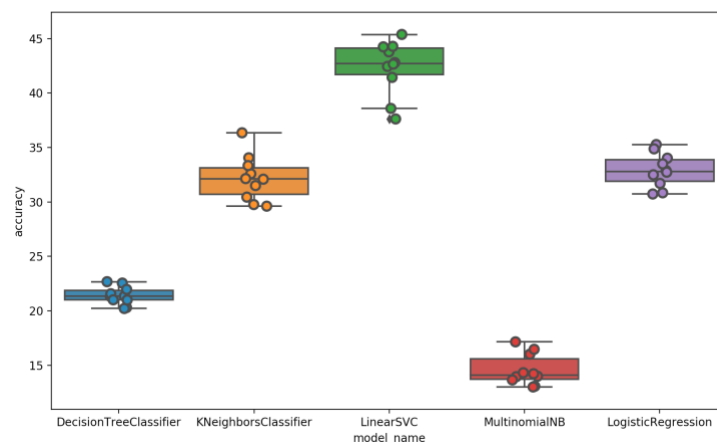


FIGURE: 4

3) WORD EMBEDDING(Word2Vec)

Using the same data word embedding is applied.

This part of the problem is most interesting to me as I have never worked with word embedding before. Word2vec is used for this.

Word2vec mapping gives similar vector representation to words with similar meaning. With the help of Neural network, surrounding words are used to generate a target word. The word representation is encoded by neural nets hidden layer.

The gensim library is used to train the model on api data and is stored in a text file:

```
import gensim
model = gensim.models.Word2Vec(sentences=review_lines, size=EMBEDDING_DIM, window=5, workers=4, min_count=1)
words = list(model.wv.vocab)
filename = "embedding_wordToVec.txt"
model.wv.save_word2vec_format(filename, binary=False)

wv = gensim.models.KeyedVectors.load_word2vec_format("embedding_wordToVec.txt", binary=False)
wv.init_sims(replace=True)
```

- Google pre-trained model which is trained on 100 billion word google news corpus[4] could have been used here for better results, however I wanted to experiment by training the model on our data. Moreover, the pretrained file is of over 10GB. My system's memory does not allow me to try that.
- Word2Vec provides word embeddings only. To characterize documents by embeddings, need averaging/summing/max operation on embeddings of all words from each document should be performed to have a D-dimensional vector that can be used for classification.[2]
- I have performed averaging[2].

```
def word_averaging(wv, words):
    all_words, mean = set(), []

    for word in words:
        if isinstance(word, np.ndarray):
            mean.append(word)
        elif word in wv.vocab:
            mean.append(wv.syn0norm[wv.vocab[word].index])
            all_words.add(wv.vocab[word].index)

    if not mean:
        print('warning(cannot compute similarity with no input %s', words)
        return np.zeros(wv.vector_size, )

    mean = gensim.matutils.unitvec(np.array(mean).mean(axis=0)).astype(np.float32)
    return mean

def word_averaging_list(wv, text_list):
    return np.vstack([word_averaging(wv, post) for post in text_list])
```

- These words averaging document features are then sent in machine learning algorithms.

Results: Logistic regression performs best with an accuracy of 62%. Decision tree performs the worst with accuracy of 23%.

```
----- Logistic regression -----  
accuracy 62.784090909091  
----- SVM -----  
accuracy 53.156565656566  
----- Decision Tree -----  
accuracy 23.421717171717  
----- KNN -----  
accuracy 42.266414141415  
----- Naive Bayes -----  
accuracy 27.176890823
```

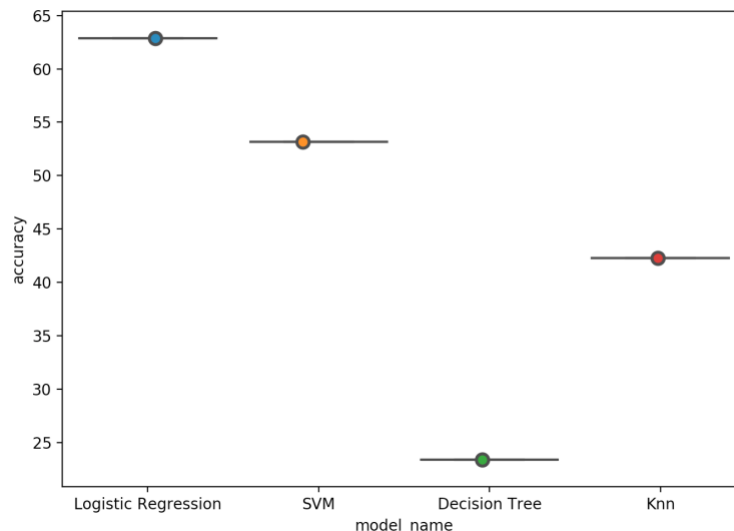


FIGURE: 5

I was interested in comparing the confusion matrix generated for algorithms with highest and lowest accuracy to observe for which categories model gets confused.

Logistic Regression:

The diagonals are prominent in the confusion matrix for logistic regression as compared to decision tree. As can be seen, Model is confused between Real Estate and Security. 26 security records are misclassified as real estate. 25 records of enterprise are misclassified as security.

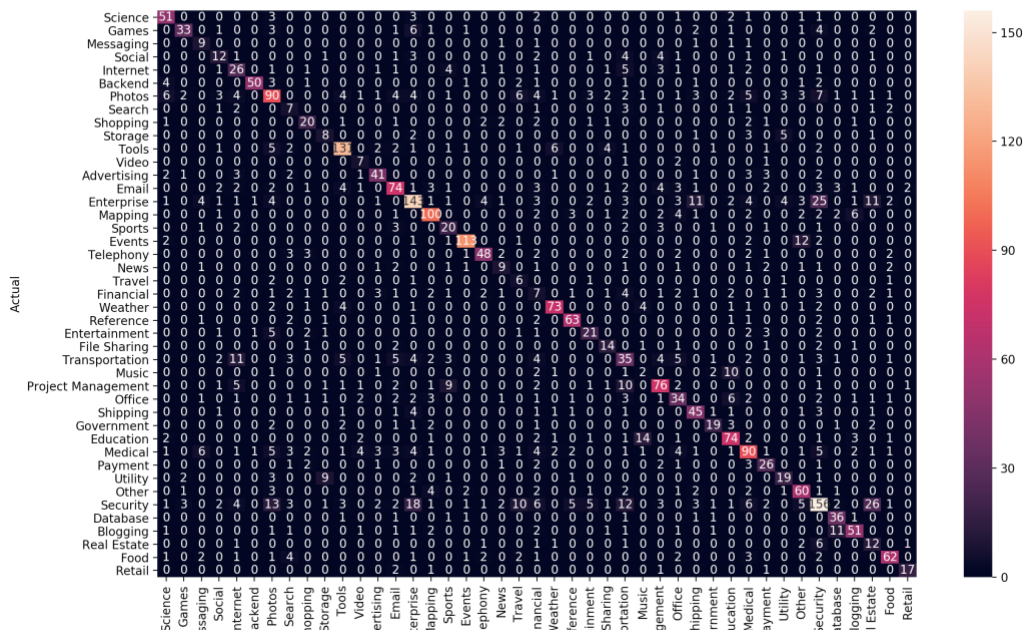


FIGURE: 6

Decision Tree:

OVERALL COMPARISON:

TF-IDF	TOPIC MODELLING	WORD EMBEDDING
----- TF-IDF	----- LDA	----- Logistic regression
model generated	model_name	accuracy 62.784090909091
model_name	DecisionTreeClassifier	-----SVM-----
DecisionTreeClassifier	21.390576	accuracy 53.156565656566
KNN	KNeighborsClassifier	-----Decision Tree-----
27.326426	32.183873	accuracy 23.421717171717
LinearSVC	LinearSVC	-----KNN-----
80.730936	42.319380	accuracy 42.266414141414
LogisticRegression	LogisticRegression	-----Naive Bayes-----
76.544823	32.898491	accuracy 27.176890823
MultinomialNB	MultinomialNB	
44.462833	14.582402	

FIGURE: 8

Model Comparison:

As can be seen from the results above, TF-IDF performed the best, followed by word-embedding for all machine learning algorithms. LinearSVC or SVM is giving good accuracies in all 3, with TF-IDF accuracy 38 higher than Topic modelling and 27 higher than word embedding. KNN accuracy remains almost the same for TF-IDF and topic modeling. Best accuracy of 80% is achieved by TF-IDF SVM.

Classification Method comparison:

SVM outperforms all other classifiers for TF-IDF and Topic modelling. Linear regression also has good accuracies for all modeling approaches.

K-nn, decision tree and naïve bayes have less accuracies. Decision tree is ran till depth 5 in all cases. Naïve bayes and decision tree gives best accuracy in Tf-IDF.

Conceptual Comparison:

- TF-IDF is done at the word level. Whereas topic modelling look for co-occurrences and semantics. For example, a document about helmets can be far from a document about bikes when using TF_IDF. Whereas, because helmets and bicycles frequently co-occur in articles they will come under same topic for LDA.
- TF-IDF simply creates a word count, topic modelling creates word vectors.
- Word2vec generates one vector for a word, TF-IDF however, produces a score.
- Word2vec is better when we need to go deeper into the documents to understand the content, subsets of content and their hierarchies. This is achieved using neural networks

- Word embedding can be used to enhance topic modelling since word embeddings identify synonym word for different languages as well.

RAW DATA ACCURACIES:

I tried performing TF-IDF and topic modelling on raw data as well to see the difference in results. The accuracy has improved by 7 for SVM in TF-IDF and by 6 for topic modelling by using cleaned and more balanced data. Similarly, for decision tree, the accuracy is improved by 2% for both modelling techniques.

More differenced can be seen by comparing Figure 8 and 9.

Raw_data_Tf-IDF		Raw_data_Topic Modelling	
model_name		model_name	
DecisionTreeClassifier	25.729640	DecisionTreeClassifier	19.994506
KNN	22.000000	KNeighborsClassifier	27.536614
LinearSVC	79.231575	LinearSVC	35.783869
LogisticRegression	73.883862	LogisticRegression	27.367245
MultinomialNB	41.624756	MultinomialNB	13.715391

FIGURE: 9

CONCLUSION

In this report, various modeling techniques and machine learning algorithms are compared. In this case TF- IDF modelling looks like a better way to go, however the modelling technique and machine algorithm to be selected highly depends on the type of the data.

FUTURE WORK

This assignment was interesting, there were many more things that I wanted to try but the time and computer's less memory didn't allow me to do so. I plan to work on the google news pretrained model and then perform word embedding for my api data. I believe that results will improve significantly.

REFERENCES

- [1] <http://nadbordrozd.github.io/blog/2016/05/20/text-classification-with-word2vec/>
- [2] <https://towardsdatascience.com/multi-class-text-classification-model-comparison-and-selection-5eb066197568>
- [3] <https://www.ritchieng.com/machine-learning-multinomial-naive-bayes-vectorization/>
- [4] <https://drive.google.com/file/d/0B7XkCwpl5KDYNINUTtISS21pQmM/edit>

