

Machine Learning File Manager



NUI Galway
OÉ Gaillimh

James Mealy

13453662

B.Sc. Electronic and Computer Engineering

Project Supervisor: Dr. Matthias Nickles

30/03/17

Abstract

The aim of this project is to create an application for performing a range of machine learning tasks on a group of files, primarily sentiment analysis. The application will include a graphical user interface to allow non-technical users to be able to use it.

Acknowledgements

I would like to thank my supervisor Matthias Nickles for all the advice and guidance he provided over the course of this project.

I would also like to thank all the dedicated lecturers and staff that who have helped to make my time at NUIG an enjoyable and worthwhile experience.

Finally I would like to thank my family and friends, without whose support I'm sure I wouldn't have gotten this far.

Table Of Contents

1. Introduction and Literary Review	6
1.1. Structure of Report	6
1.2. Project Requirements	6
1.3. Machine Learning and its Uses	6
1.4. Project Overview	7
1.5. Sentiment Analysis	7
1.6. Topic Modelling	9
1.7. Context of Project	10
2. Technical Review	11
2.1. Pre-processing of Datasets	11
2.1.1. The Bag of Words Model	11
2.1.2. Document Term Weighting	12
2.1.3. Removal of Stop words	12
2.1.4. Stemming	13
2.2. Machine Learning Algorithms	13
2.2.1. Linear Support Vector Machine	13
2.2.2. Latent Dirichlet Allocation	13
2.3. Technologies Used	14
2.3.1. Linux Mint	14
2.3.2. Python 2.7.12	14
2.3.3. Scikit-learn	14
2.3.4. Gensim	14
2.3.5. NLTK	14
2.3.6. WxPython	14
3. Implementation and Design	15
3.1. Overall Program Flow	15
3.1.1. Importing Files	15
3.1.2. Classify Files	16
3.1.3. Exporting Files	16
3.2. Graphical User Interface Design	17
3.2.1. Main Window	17
3.2.2. Classify Window	18
3.3. Sentiment Classification Implementation	21
3.4. Topic Modelling Classification Implementation	23
4. Tests and Results	26
4.1. Repeated Random Sub-Sampling Validation	26
4.2. Supervised Classification Accuracy in the GUI	27
4.3. Testing Accuracy Using Different Algorithms and Datasets	29
4.4. Topic Model Accuracy for Two Topics	30

4.5. Topic Model Accuracy for Higher Numbers of Topics	31
5. Conclusions and Future Work	32
5.1. Conclusions	32
5.2. Future Work	32

1. Introduction and Literary Review

1.1. Structure of Report

This report consists of five chapters. **Chapter 1** outlines the high-level topics related to this project and gives an overview of the project. **Chapter 2** explains the technical concepts required for understanding this project and also gives an overview of the technologies used in its implementation. **Chapter 3** gives of a detailed explanation of the project's implementation, design, and use. **Chapter 4** contains the results of the project's implementation. **Chapter 5** contains the conclusions drawn from the project's implementation and results, and also outlines potential future work that could be done to improve the project.

1.2. Project Requirements

The primary requirement of this project was to create a GUI file manager application which allows a user to classify a directory of files according to their sentiment.

There were several possible secondary requirements for this project. The first secondary requirement achieved was to extend the application to be able to classify files using any labels, not just sentiment, as long as an appropriate set of training data is provided.

The next secondary requirement achieved in this project was to be able to use an unsupervised topic model to partition a group of files according to their topic.

1.3. Machine Learning and its Uses

Machine learning is an important and exciting field of computer science. It allows computers to learn without being explicitly programmed. This has many applications in almost every field of science and technology. It is used in medical diagnoses, recommendation systems for retailers, artificial intelligence, and too many other examples to mention.

The two sub-categories of machine learning used in this project are *supervised* machine learning and *unsupervised* machine learning. Supervised machine learning uses a set of labelled training examples to extract meaningful information. Each example in the training data consists of features. The goal of supervised learning is to learn how these features affect the label of the example. Once the relationship between the features of the training

examples and their labels has been learned, it is possible to take unlabelled examples and give them labels based on their features.

As a simple example of this, say supervised learning was used to classify pieces of fruits into apples, cherries, or bananas. The training examples in this case would be labelled pieces of fruit. The features could be color, shape and size. Using these training examples, a classifier can learn which features relate to which label (apples are large, round, and green; cherries are small, round, and red; bananas are large, long, and yellow). Then, if given an object that is large, long, and yellow, the classifier can deduce that it is probably a banana.

Unsupervised learning does not use labelled training data, and therefore cannot classify datasets based on any labels. An example of unsupervised learning is to group instances of a dataset based on their similarity, a process called clustering. In the example of the fruits, a clustering algorithm would group together the pieces of fruit that are similar, ie. ones that share features. Ideally each fruit would have its own group.

1.4. Project Overview

In this project, supervised learning is used to predict the sentiment of a given text by learning from a labelled dataset of movie reviews (labelled as either positive or negative). This is explained further in section 1.4. This process is also used to classify text based on language and topic.

Unsupervised learning is used in this project to separate text documents into similar topics. This category of unsupervised learning is called *topic modelling* and is explained further in section 1.6.

A GUI application was developed for the project to allow a user to easily perform these tasks, without needing to write any code.

1.5. Sentiment Analysis

Sentiment analysis, also known as opinion mining, is concerned with finding the attitude of the author of a document towards the subject matter they are writing about. This is done using machine learning, natural language processing, and other automated methods. In other words, it is the process of determining whether an author feels positively or negatively towards the topic they are discussing.

For example, imagine you wanted to determine whether past customers of a particular cafe thought it was good or bad. You could take tweets that mention that cafe, and perform sentiment analysis on them to see how many people thought it was good, and how many thought it was bad.

Although sentiment analysis is a relatively young domain, interest in it has exploded in recent years with the rise of social media and online consumer sites such as amazon. Data is being extracted from comments, tweets, and customer reviews in order to determine how people feel about products and businesses. This information can then be used to influence future product development and marketing.

Interest in sentiment analysis research started in the early 2000s. Bo Pang and Lillian Lee published a paper on the subject in 2002, making it one of the earliest papers on the topic to receive significant attention [1]. The paper tested the effectiveness of using machine learning, and some natural language processing, techniques to classify movie reviews collected from IMDb.com into positive and negative sentiment. It was found that using machine learning algorithms alone can produce excellent accuracy rates. This paper informed many of the decisions made throughout this project.

Another early influential paper, by Peter D. Turney, used unsupervised machine learning techniques to classify reviews of automobiles, banks, movies, and travel destinations into positive and negative sentiment [2]. To do so, phrases were extracted from the reviews. The phrases centered around adjectives and adverbs, as these words are found to be particularly descriptive in terms of sentiment. These phrases were assigned a *semantic orientation* value, which is a measure of how positive or negative the connotations associated with a phrase are. The semantic orientation was calculated by comparing the phrase to two reference words: “excellent”, representing positive semantic orientation, and “poor”, representing negative.

These two papers represent two different ways of approaching sentiment analysis. The first emphasises machine learning techniques. It relies on using training data (movie reviews with their associated rating) to inform machine learning algorithms. The second focusses more on natural language processing techniques, which rely on having information about the meaning and structure of the language used. Since the former method was found to produce similar results and is relatively simple to implement using machine learning libraries, it was the method I chose for this project.

1.6. Topic Modelling

Topic Modeling is used to describe topics or subjects present in a text. For example, it could be used to separate a large number of research papers according to the field of research they belong to.

Many topic models are unsupervised, meaning they do not need to be trained by documents which are labeled according to their topic. For this reason, they have become very popular in analysing the large amounts of unlabeled data that are increasingly being produced in recent years.

Topic models are used for sorting the articles in newspaper archives or digital libraries to make them more accessible. Another application is in advertising. Companies such as Google are able to automatically learn what topics a user is interested in, and use this information to provide targeted ads.

One of the earliest topic models was called Latent Semantic Analysis (LSA). LSA works by constructing a “low-dimensional spatial model wherein similar documents are placed near one another” [3]. More robust, statistics-based models have since been developed [4]. Probabilistic Latent Semantic Analysis (PLSA) improved on some of the limitations of LSA. For example, it added the ability to disambiguate words which have the same spelling but different meanings.

Latent Dirichlet Analysis (LDA) introduced further improvements in Topic modelling. LDA is a generative topic model, ie. it assumes documents are generated in the following way: [5]

1. Choose the length of the document (number of words).
2. For each word:
 - a. Choose a topic. Each topic is chosen with a certain probability, defined in the documents *Dirichlet Distribution*.
 - b. Choose a word from that topic. Each Topic consists of a list of words associated with it. Each word is chosen with a certain probability. Words with a higher probability of being chosen are more representative of that topic.

The model can then use statistical inference to work backwards and determine the topics that make up a document. For example, it might determine that a document is made up of 70% Topic 1, 20% Topic 2, and 10% Topic 3.

1.7. Context of Project

Some online tools exist for the purposes of searching and sorting data based on sentiment. The majority are used for the purposes of research by companies, who want to know how their brand or products are viewed by the public. Many of the tools use twitter as their source of information. For example, a tool called sentiment140 searches for tweets containing a specified keyword, and classifies them as positive, negative or indifferent [6]. Thereby it is possible to get a consensus of how the users of twitter think about a topic. Other similar online tools include rankspeed [7], socialmention [8] and whatdoestheinternetthink [9].

This project will differ from the previously mentioned tools in that it will be capable of searching files offline using sentiment analysis tools.

2. Technical Review

This chapter will outline the machine learning, natural language processing, and data pre-processing techniques used in the project. It will explain any underlying concepts to the level of detail necessary to be able to effectively use the tools and techniques that are based on them.

This chapter will also give an overview of the tools, programming languages, and libraries used in the project.

2.1. Pre-processing of Datasets

To create classifiers and perform machine learning tasks in the scikit-learn library, the datasets used must be processed into the required input format. This section outlines these necessary steps, and other pre-processing techniques which are used to increase the accuracy of classifiers.

2.1.1. The Bag of Words Model

The bag of words model is a way of representing a dataset of text documents as a vector of the words contained in those documents [10].

Take the following two strings (taken from the 20 newsgroups dataset used in this project [11]) as a trivial example of a dataset:

Text 1 = "This text is the first text"

Text 2 = "This is text two"

The bag of words representation of this dataset would be the following:

['text', 'first', 'is', 'the', 'this', 'two']

Punctuation is removed, so we are left with a list of the words making up the dataset.

The bag of words representation can be extended to include the number of occurrences of each word. One way of doing this is a Document Term Matrix (DTM). This is simply a matrix of integer values where each row in the matrix represents a document in the dataset, and each column represents a word. The integer values in the matrix represent the number of occurrences of a particular word in a particular document. For the example dataset given above, the DTM would look like this:

	text	first	is	the	this	two
Text 1	2	1	1	1	1	0
Text 2	1	0	1	0	1	1

In python, the data would be represented as the following NumPy matrix:

```
[[2 1 1 1 1 0]
 [1 0 1 0 1 1]]
```

2.1.2. Document Term Weighting

Document Term weighting is the process of applying weights to words in a document, based on their importance to the meaning of the document. In this project, a method called tf-idf is used. Tf-idf stands for “term frequency by inverse document frequency”, which refers to how the weight for each word in a document is calculated:

$$\text{tf-idf}(t,d) = \text{tf}(t,d) \times \text{idf}(t) \quad [12]$$

Term Frequency: The more frequent a word is in a document, the more likely it is to be significant to the meaning of that document. Term frequency in the scikit-learn implementation of tf-idf is a simple count of the occurrences of a term t in a document d .

Inverse Document Frequency: When a word is common throughout the whole dataset, it will have less importance to the meaning of any single document. More common terms will have a lower idf value, resulting in a lower tf-idf weight value.

2.1.3. Removal of Stop words

Stop words are simply common words of a given language that usually do not contribute significantly to the meaning of a text. In machine learning and natural language processing applications they are removed from a dataset. The list of stop words used in this project was taken from scikit-learn and nltk.

For the example dataset given in section 2.1.1., the resulting bag of words after the removal of stop words would be as follows:

```
['text', 'first', 'two']
```

The words 'is' and 'the', and 'this' were removed.

2.1.4. Stemming

Often in a dataset, there can be several forms of the same word. By default these are treated as separate words. Because the words effectively carry the same meaning, they are changed to be represented by their stem.

Below is an example dataset before stemming:

```
['fast', 'faster', 'fastest', 'open', 'opener', 'opened', 'opens', 'opening']
```

After stemming the dataset becomes the following:

```
['fast', 'open']
```

2.2. Machine Learning Algorithms

2.2.1. Linear Support Vector Machine

Linear support vector machine (Linear SVM) is a supervised classification algorithm.

According to the scikit-learn documentation it is “widely regarded as one of the best text classification algorithms” [13]. A scikit-learn implementation of this algorithm, called LinearSVC, is used in this project to perform the supervised classification tasks.

2.2.2. Latent Dirichlet Allocation

Latent dirichlet allocation (LDA) is an unsupervised topic modelling algorithm. Its inputs are a list of text documents and the number of topics to find in the documents. The output, in this project, is a list of the topics that correspond to the input files. LDA is explained in more detail in section 1.6.

2.3. Technologies Used

2.3.1. Linux Mint

The project was developed using Linux because I am familiar with bash scripting, which was useful for preparation of datasets. For example I used bash whenever I needed to rename files in bulk. Mint was chosen because that is the Linux distribution I am most familiar with.

2.3.2. Python 2.7.12

Python is a very popular programming language for machine learning and data mining problems. It has many useful libraries for these purposes, and there is a wealth of documentation and tutorials online to help get started. The main python libraries used in this project were scikit-learn, nltk and gensim.

2.3.3. Scikit-learn

Scikit-learn is a very popular machine learning library in python, build on the NumPy and SciPy libraries. It is a beginner friendly library with excellent documentation and examples provided on its website.

2.3.4. Gensim

Gensim is a machine learning library. It includes a good implementation of the LDA algorithm used in this project. Scikit-learn introduced an implementation of LDA in 2015, but it does not support the ability to classify files once the LDA model is created [14].

2.3.5. NLTK

NLTK is a natural language processing library. It was used in this project to pre-process datasets for the LDA algorithm. Scikit-learn pre-processing techniques could not be used as they require datasets to be in a format which is incompatible with Gensim's LDA algorithm.

2.3.6. WxPython

This is one of the most popular GUI building libraries for python. It was chosen due to its look and feel, and also because it is very beginner friendly.

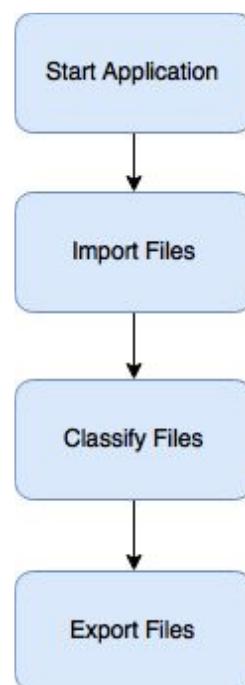
3. Implementation and Design

Chapter 2 outlined the main methodologies used in this project. This chapter detail how those methodologies are used in the project's implemented. It will also give an overview of the structure, flow, and design of the application. The most important sections of code are included.

3.1. Overall Program Flow

The purpose of the application is to allow a user to classify files according to some property such as sentiment. This is done in three steps. First the files to be classified are imported to the application. Then the files are classified, and finally exported.

The flowchart below shows the high-level program flow.



3.1.1. Importing Files

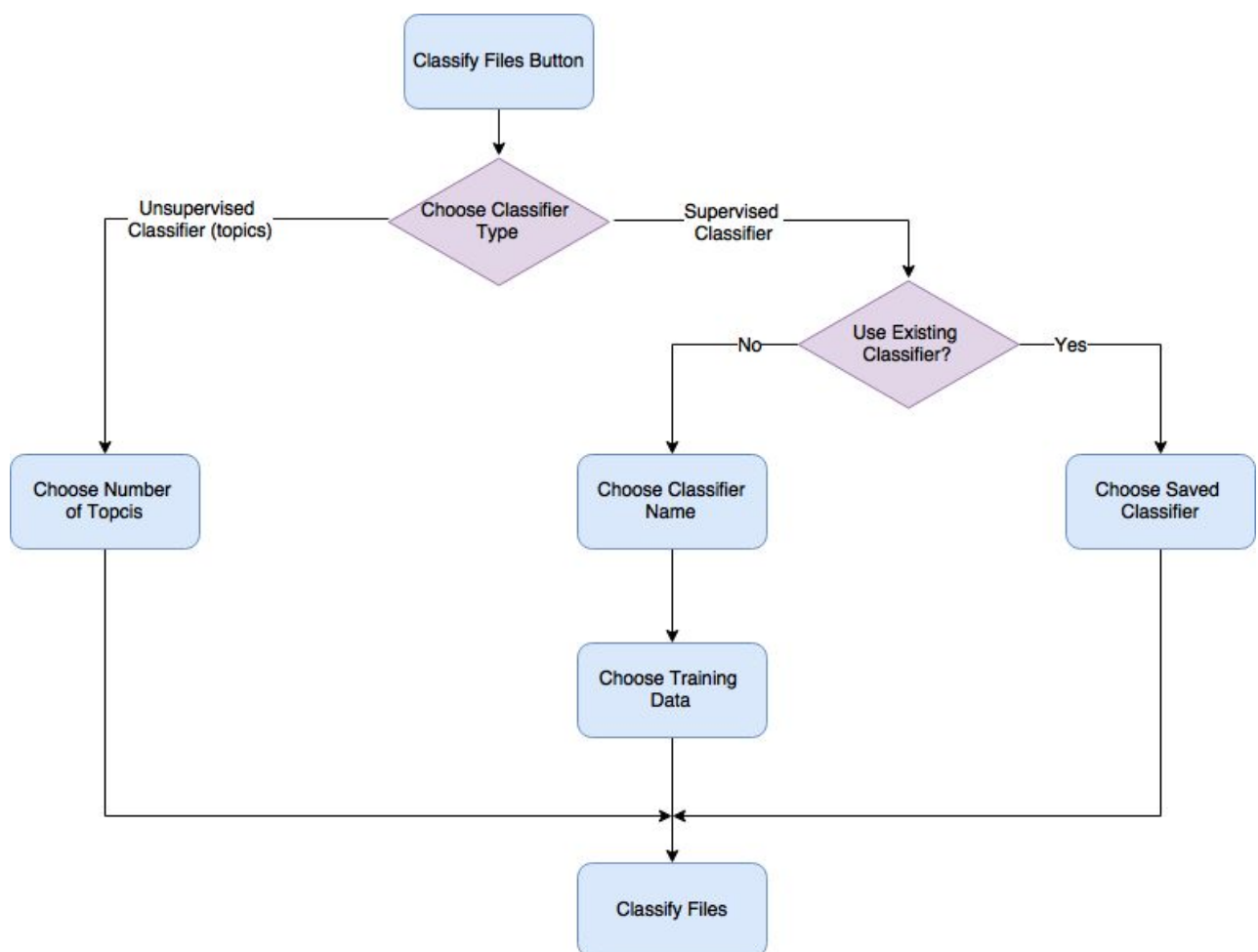
When the import button is pressed, a new window will open prompting the user to select a directory of files. These are the files the user wishes to classify. Once the files are selected, they will show up in the main program window.

3.1.2. Classify Files

There are two types of classifier to choose from when after pressing the Classify Files button. The first is a supervised classifier. Taking this option, the user can then decide to use a previously created classifier, or create a new one and use that.

When creating an unsupervised classifier, the user must choose the number of topics to partition the files into, and then the files can be classified.

The flowchart below outlines these options.



3.1.3. Exporting Files

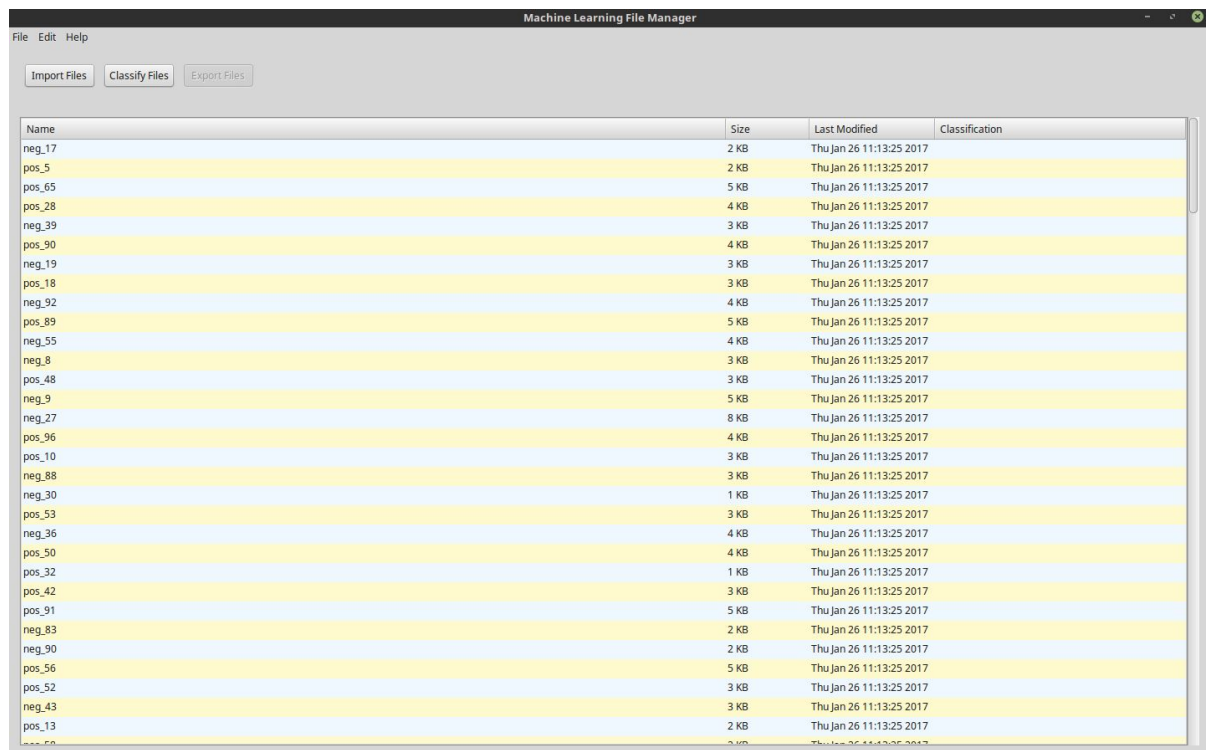
Once the files have been classified, they can be exported to a directory. This is done such that each label or classification has its own directory. All files classified as that label will be exported into that directory.

3.2. Graphical User Interface Design

The graphical user interface was designed and built using a python library called wxPython. The code skeletons on ZetCode.com were very useful in developing the GUI [15].

3.2.1. Main Window

The main elements in the window are a list showing the files and buttons for importing, classifying, and exporting files.



There are three buttons. One for importing files into the application for classification, one for creating and using a classifier, and one for exporting the classified files for use in a file manager. The export files button is disabled until the files have been classified. The buttons are shown below, before and after classification.



The list of files consists of 4 columns: Name, showing the name of each file; Size, showing the size of each file; Last Modified, showing the date and time of the last time the file was

modified; and Classification, showing the predicted label of the file. The last column is empty until the user performs classification on the files.

The file list before classification:

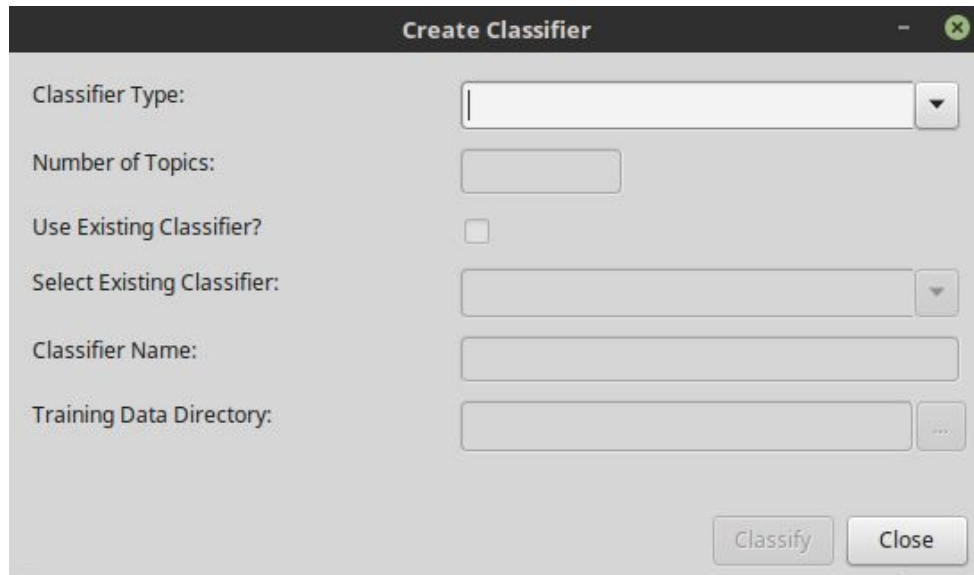
Name	Size	Last Modified	Classification
neg_17	2 KB	Thu Jan 26 11:13:25 2017	
pos_5	2 KB	Thu Jan 26 11:13:25 2017	
pos_65	5 KB	Thu Jan 26 11:13:25 2017	
pos_28	4 KB	Thu Jan 26 11:13:25 2017	
neg_39	3 KB	Thu Jan 26 11:13:25 2017	
pos_90	4 KB	Thu Jan 26 11:13:25 2017	
neg_19	3 KB	Thu Jan 26 11:13:25 2017	
pos_18	3 KB	Thu Jan 26 11:13:25 2017	
neg_92	4 KB	Thu Jan 26 11:13:25 2017	
pos_89	5 KB	Thu Jan 26 11:13:25 2017	
neg_55	4 KB	Thu Jan 26 11:13:25 2017	
neg_8	3 KB	Thu Jan 26 11:13:25 2017	
pos_48	3 KB	Thu Jan 26 11:13:25 2017	
neg_9	5 KB	Thu Jan 26 11:13:25 2017	

The file list after classification:

Name	Size	Last Modified	Classification
neg_17	2 KB	Thu Jan 26 11:13:25 2017	neg
pos_5	2 KB	Thu Jan 26 11:13:25 2017	pos
pos_65	5 KB	Thu Jan 26 11:13:25 2017	pos
pos_28	4 KB	Thu Jan 26 11:13:25 2017	pos
neg_39	3 KB	Thu Jan 26 11:13:25 2017	neg
pos_90	4 KB	Thu Jan 26 11:13:25 2017	pos
neg_19	3 KB	Thu Jan 26 11:13:25 2017	neg
pos_18	3 KB	Thu Jan 26 11:13:25 2017	pos
neg_92	4 KB	Thu Jan 26 11:13:25 2017	neg
pos_89	5 KB	Thu Jan 26 11:13:25 2017	pos
neg_55	4 KB	Thu Jan 26 11:13:25 2017	neg
neg_8	3 KB	Thu Jan 26 11:13:25 2017	neg
pos_48	3 KB	Thu Jan 26 11:13:25 2017	pos
neg_9	5 KB	Thu Jan 26 11:13:25 2017	neg

3.2.2. Classify Window

This is the window that pops up when the user clicks the 'Classify Files' button. It consist of 6 fields which are used to specify the type of classifier required. The first option for 'Classifier Type' is 'Text Classifier (Supervised)' for sentiment, language, topic and other supervised classifiers. The second option is 'Topic Modelling (Unsupervised)' for unsupervised topic classification. The other fields are used to define the parameters and options for the two types of classifier. These fields are disabled until they are required to be filled.

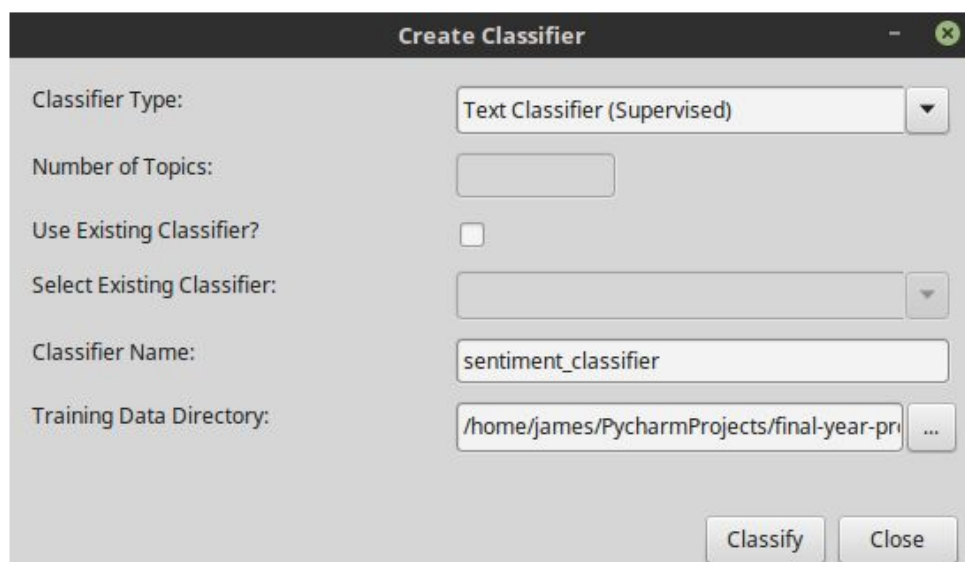


The image shows a 'Create Classifier' dialog box with a dark title bar. It contains several input fields: 'Classifier Type' (a dropdown menu), 'Number of Topics' (a text box), 'Use Existing Classifier?' (a checkbox), 'Select Existing Classifier:' (a dropdown menu), 'Classifier Name:' (a text box), and 'Training Data Directory:' (a text box with a file explorer button). At the bottom right, there are 'Classify' and 'Close' buttons.

Classifier Type:	
Number of Topics:	
Use Existing Classifier?	<input type="checkbox"/>
Select Existing Classifier:	
Classifier Name:	
Training Data Directory:	

Classify Close

When creating a supervised classifier (the first of the two options), the user can choose to create a new classifier by selecting the appropriate training data, or they can use a classifier they have already created. Once all required fields have been filled, the classify button is enabled. Below, the creation of a new classifier is shown.

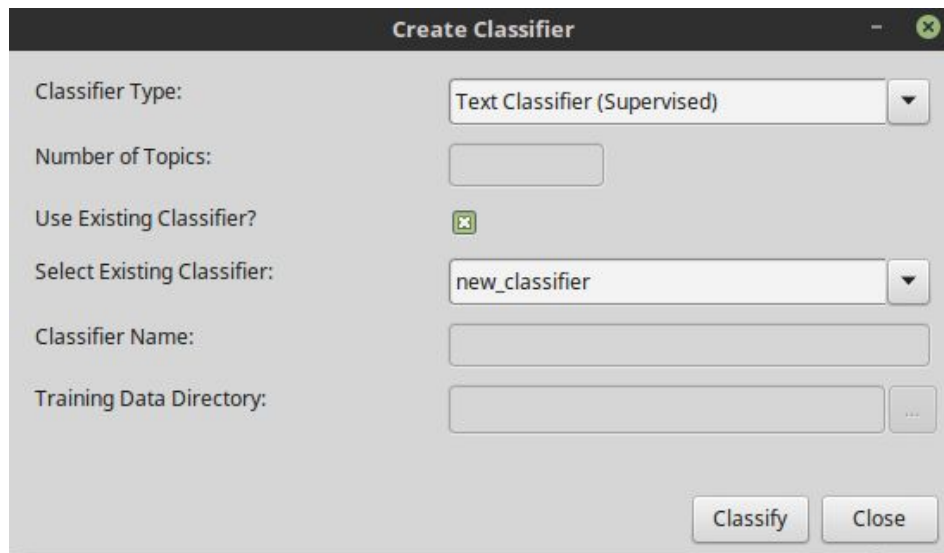


This image shows the same 'Create Classifier' dialog box, but with the fields filled out. 'Classifier Type' is set to 'Text Classifier (Supervised)', 'Classifier Name' is 'sentiment_classifier', and 'Training Data Directory' is '/home/james/PycharmProjects/final-year-pr'. The 'Classify' button is now enabled.

Classifier Type:	Text Classifier (Supervised)
Number of Topics:	
Use Existing Classifier?	<input type="checkbox"/>
Select Existing Classifier:	
Classifier Name:	sentiment_classifier
Training Data Directory:	/home/james/PycharmProjects/final-year-pr

Classify Close

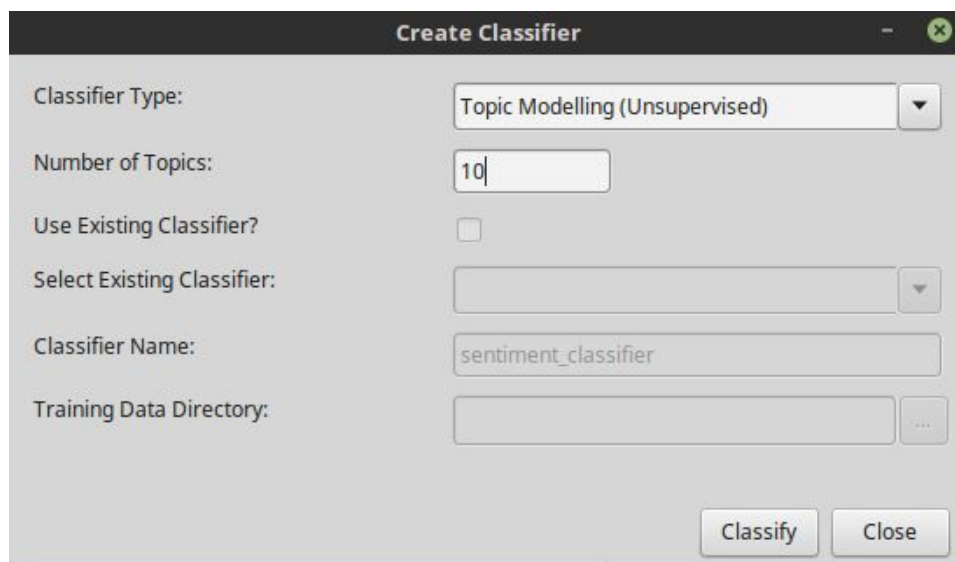
The screenshot below shows the option for choosing an existing classifier.



The screenshot shows a dialog box titled "Create Classifier". It contains the following fields and controls:

- Classifier Type:** A dropdown menu set to "Text Classifier (Supervised)".
- Number of Topics:** An empty text input field.
- Use Existing Classifier?:** A checkbox that is checked.
- Select Existing Classifier:** A dropdown menu set to "new_classifier".
- Classifier Name:** An empty text input field.
- Training Data Directory:** An empty text input field with a browse button (three dots) to its right.
- Buttons:** "Classify" and "Close" buttons at the bottom right.

The only option for the topic model classifier is the number of topics to partition the files into. This is shown below.



The screenshot shows a dialog box titled "Create Classifier". It contains the following fields and controls:

- Classifier Type:** A dropdown menu set to "Topic Modelling (Unsupervised)".
- Number of Topics:** A text input field containing the value "10".
- Use Existing Classifier?:** An unchecked checkbox.
- Select Existing Classifier:** An empty dropdown menu.
- Classifier Name:** A text input field containing the value "sentiment_classifier".
- Training Data Directory:** An empty text input field with a browse button (three dots) to its right.
- Buttons:** "Classify" and "Close" buttons at the bottom right.

3.3. Sentiment Classification Implementation

This section outlines the pre-processing steps taken to allow training data to be used by the scikit-learn library to build a classifier.

The first step taken is to remove any files that use encoding not supported by scikit-learn, such as 'latin1' encoding. This is done by attempting to perform a 'fit_transform' operation on each document in the dataset. Any documents with incompatible encoding will raise an exception and are collected.

```
incompatible_files = []
for i in range(len(files filenames)):
    try:
        count_vector.fit_transform(files.data[i:i + 1])
    except UnicodeDecodeError:
        incompatible_files.append(files.filenames[i])
    except ValueError:
        pass
```

These incompatible files can then be deleted:

```
# remove the incompatible files.
for f in incompatible_files:
    print 'deleting incompatible files'
    os.remove(f)
```

The next step is to separate the contents of the documents in the training data from their labels. This is done using a scikit-learn function called train_test_split(). This will give a list of documents and a corresponding list of labels.

The next steps are to convert the documents to a bag of words model (section 2.1.1.), perform documents term weighting (section 2.1.2.), remove common words (section 2.1.3), and create a classifier object. These are all done in one step using a scikit-learn pipeline. The pipeline defines the operations to be performed. The operations are performed by 'fitting' the dataset to the pipeline.

```
# Build a vectorizer / classifier pipeline that filters out tokens
# that are too rare or too frequent
pipeline = Pipeline([
    ('vect', TfidfVectorizer(min_df=3, max_df=0.95)),
    ('clf', LinearSVC(C=1000)),
])
```

The parameters for the TfidfVectorizer object are `min_df=3` and `max_df=0.95`. This means that a word must occur at least 3 times to be kept, and must not occur in more than 95% of the documents, as these words will carry less significance. This step effectively removes the stop words in the documents.

This classifier is wrapped in a SupervisedClassifier object. This object contains a mapping of label indices to the strings that they represent. Files can now be passed to this SupervisedClassifier object, and it will return the predicted labels for each file. The class implementation is shown below:

```
class SupervisedClassifier:
    """
    """
    def __init__(self, cls, groups):
        self.cls = cls
        self.label_names = groups

    def get_predictions(self, files):
        """Returns list of predicted labels for the files"""
        # Use dict with target names to get back the actual class values.
        return map(self.get_labels(), self.cls.predict(files))

    def get_labels(self):
        """Returns a list of the categories form the training data"""
        return self.label_names.get
```

3.4. Topic Modelling Classification Implementation

This section outlines the procedure for building the topic model classifier using Nltk and Gensim. The implementation was based mainly on the tutorial by Jordan Barber [16].

Files that are incompatible with nltk must first be removed, similar to the method described in the previous section. Once this is done, the documents to be classified are read into the program and stored in a list:

```
documents = []
for fil in os.listdir(data_path):
    with file(data_path + fil) as f:
        documents.append(f.read())
```

Now some further preprocessing must be done. As with the supervised classifier, the documents must be converted to a bag of words, and be stripped of stop words. The words are also stemmed (section 2.1.4.).

Some additional steps are taken to remove irrelevant data from the corpus of documents and improve the accuracy of the classifier. Words that are smaller than 3 characters, and words containing digits are removed.

The loop which performs all these tasks is shown below. The `tokenized_documents` list contains the stripped bag of words version of the corpus.

```
tokenized_documents = []
# loop through document list
for i in documents:
    # clean and tokenize document string
    raw = i.lower()
    tokens = tokenizer.tokenize(raw)
    # remove stop words from tokens
    tokens = [i for i in tokens if not i in en_stop]
    # stem tokens
    tokens = [p_stemmer.stem(i) for i in tokens]
    # remove small words
    tokens = [i for i in tokens if len(i) > 2]
    # remove digits
    tokens = [x for x in tokens if not any(c.isdigit() for c in x)]
    # add tokens to list
    tokenized_documents.append(tokens)
```

The methods for converting words to tokens (bag of words), removing stop words, and stemming are from the nltk library.


```
# tokenizer
tokenizer = RegexpTokenizer(r'\w+')
# create English stop words list
en_stop = get_stop_words('en')
# Create p_stemmer of class PorterStemmer
p_stemmer = PorterStemmer()
```

Using a 'dictionary' from the Gensim library, it is now possible to convert the bag of words to a document term matrix. A dictionary assigns each word in the bag of words model an id. This dictionary can be used to map the id values of the words used in the document term matrix back to their string values.

```
# turn our tokenized documents into a id <-> term dictionary
dictionary = corpora.Dictionary(tokenized_documents)
# convert tokenized documents into a document-term matrix
corpus = [dictionary.doc2bow(text) for text in tokenized_documents]
```

Using the dictionary and document term matrix, the topic model classifier is created.

```
# generate LDA model
ldamodel = gensim.models.ldamodel.LdaModel(corpus, num_topics=num_topics,
                                             id2word=dictionary, passes=20)
```

The corpus is converted to a tf-idf model, and passed to the topic model. For each document in the corpus it will return a topic distribution. The topic distribution describes what percentage of the document is made from each of the topics found.

```
tfidf = gensim.models.TfidfModel(corpus)
corpus_tfidf = tfidf[corpus]
topic_distributions = ldamodel[corpus_tfidf]
```

The most significant topic from each document is chosen as that document's classification.

```
topic_classifications = []
# For each document, record the topic that best describes it.
for i in topic_distributions:
    best_topic_index = max(i, key=lambda x:x[1])[0]
    topic_classifications.append(topic_names[best_topic_index])
```

Because this method is unsupervised, there are no labels to assign to the documents as classifications. The classifications are simply named Topic1, Topic2, ..., TopicN, where N is the number of topics. In order to make these labels more descriptive, and therefore more useful, the five most significant words of that topic are appended to the label name. An

example for a topic describing articles about space is: “Topic1: space, nasa, orbit, landing, astronaut”. The code to implement this is shown below.

```
topic_names = []
for i, topic in enumerate(ldamodel.show_topics(num_topics=num_topics,
                                                num_words=5, formatted=False)):
    # extract the topic words from the list of tuples
    topic_words = [el[0] for el in topic[1]]
    topic_names.append('Topic' + str(i) + ': ' + ' '.join(topic_words))
print topic_names
```

4. Tests and Results

4.1. Repeated Random Sub-Sampling Validation

The dataset used to assess the accuracy of the sentiment classification is a corpus of movie reviews taken from imdb.com, with labels of either 'pos' for positive and 'neg' for negative [17].

To assess the accuracy, the dataset of movie reviews is randomly split into two parts, a training set consisting of 75% of the dataset, and a test set consisting of 25%. Once the classifier has been created from the training data, the test set is classified, and the accuracy is measured, the data is randomly split again using the same proportions. This is repeated 10 times and the accuracy is averaged over all the runs. The test code is shown below.

```
avg = 0
runs = 10
for i in range(0, runs):
    docs_train, docs_test, y_train, y_test = train_test_split(
        dataset.data, dataset.target, test_size=0.25)

    # create classifier and predict
    clf = pipeline.fit(docs_train, y_train)
    y_predicted = clf.predict(docs_test)

    # mean accuracy and classification report.
    accuracy = np.mean(y_predicted == y_test)
    print i, accuracy
    avg += accuracy

print avg/runs
```

Results

Run	1	2	3	4	5	6	7	8	9	10
Accuracy	0.856	0.838	0.878	0.844	0.844	0.866	0.85	0.866	0.844	0.832

Average: 85.2%

4.2. Supervised Classification Accuracy in the GUI

Testing the accuracy through the GUI is done to ensure its performance is consistent with the previous test.

Because there is no functionality for testing built into the application, an indirect approach must be taken. Once again the movie review dataset is split into a test set and a validation set, with 1800 reviews for both positive and negative sentiment in the training set, and 200 in the test set. The test files are named with a prefix, denoting their sentiment. For example, the 10th file of positive sentiment is called 'pos_10'. The naming of the files in this way was done using a bash script, shown below.

```
#!/bin/bash
# loop through subdirectories and rename files thusly: <directoryname>_<integer>
# integer is the index of the file in the directory
path="/home/james/PycharmProjects/final-year-project/training_data/movie_reviews_test"
cd $path

for dir in */ ; do
    a=1
    cd $dir
    for fname in *
    do
        new_name="${dir%}/_${a}"
        echo $new_name
        mv $fname $new_name
        let a=a+1
    done
    cd ..
done
```

Through the application, the test files are imported, classified, and exported. Files of each label are saved in their own directory, and the directory is named after the label of the files it contains. So files of negative sentiment are saved in a directory called 'neg'.

Another bash script is then used to compare the names of the files (their actual sentiment label), with the name of the directory they are in (their predicted label). The accuracy is a count of the number of files that were correctly classified, over the total number of files.

```
#!/bin/bash
# Compare the name of each file with the name of its directory, to see if it has been correctly
# classified. Calculate the accuracy.
path="/home/james/Documents/college/FYP/test_results"
cd $path

correct=0
total=0

for dir in */ ; do
    cd $dir
    for fname in *
    do
        if [ ${fname%_*} == ${dir%/} ]; then
            let correct+=1
        fi
        let total+=1
    done
    cd ..
done

echo $(echo "$correct/$total" | bc -l)
```

Results

Accuracy: 86%

Since the accuracy is consistent with that of the previous test, the sentiment classification through the GUI application has been proven to be working successfully.

4.3. Testing Accuracy Using Different Algorithms and Datasets

The accuracies in this section were found using the same method as in section 6.1.1.

The language dataset is a corpus of paragraphs taken from wikipedia pages of eleven different languages. The script used to collect the data was taken from a scikit-learn tutorial [3]. The Topics dataset is called the 20 newsgroups dataset [11]. It is made up of entries in online newsgroups, across 20 topics.

Results

The Support Vector Machine algorithm was confirmed to be the best all-round classifier for text datasets, although Gradient Descent is also an excellent choice, especially for language classification.

Language classification proved to be the most successful. This is probably because the difference between languages is larger than differences of sentiment and topic, which are much more subtle.

The highest accuracy for each dataset is highlighted.

	Support Vector Machine	Gradient Descent	Perceptron	Naive Bayes
Sentiment	85.18%	84.54%	82.10%	81.24%
Language	95.78%	97.18%	95.60%	90.29%
Topics	90.10%	87.41%	86.29%	82.87%

4.4. Topic Model Accuracy for Two Topics

The aim of this test is to get an initial idea of the accuracy of the LDA topic model used in this project. The test dataset used was the 20 newsgroups dataset [11]. Four random combinations of the 20 topics were combined to make 4 test datasets. The topic combinations were: Macintosh computers & guns, atheism & medicine, christianity & motorcycles, and automobiles & guns.

For this test we want to know how well the groupings assigned by the algorithm match up with the actual topics. Since LDA is unsupervised, there is no way to directly test if the algorithm splits the data correctly. The approach taken was the following.

First, the test documents were renamed so that their names contain their topic name, using the same method as in section 4.1.2. Then the documents were run through the application so that they were stored in a directory denoting to their predicted group. For example if a document is predicted as being in Topic 0, it is stored in a directory of that name, with all the other files predicted as Topic 0.

Then, the directories were renamed according to the most common document topic that they contain. If the directory called Topic 0 contained a majority of documents which were of the guns topic (called "talk.politics.guns"), that directory would be renamed to "talk.politics.guns". Then the directory names could be compared with the file names it contains, to see how well the documents were partitioned. This was done using the same method as section 4.1.2.

Results

The accuracies for each of the four dataset are shown below.

```
james@jm ~/scripts $ ./accuracy.sh /home/james/Documents/Result/2topics/autos_space
Accuracy: .74367622259696458684
james@jm ~/scripts $ ./accuracy.sh /home/james/Documents/Result/2topics/chrts_motorb
Accuracy: .91979949874686716791
james@jm ~/scripts $ ./accuracy.sh /home/james/Documents/Result/2topics/macs_guns
Accuracy: .90267857142857142857
james@jm ~/scripts $ ./accuracy.sh /home/james/Documents/Result/2topics/medicine_atheism
Accuracy: .85461323392357875116
james@jm ~/scripts $
```

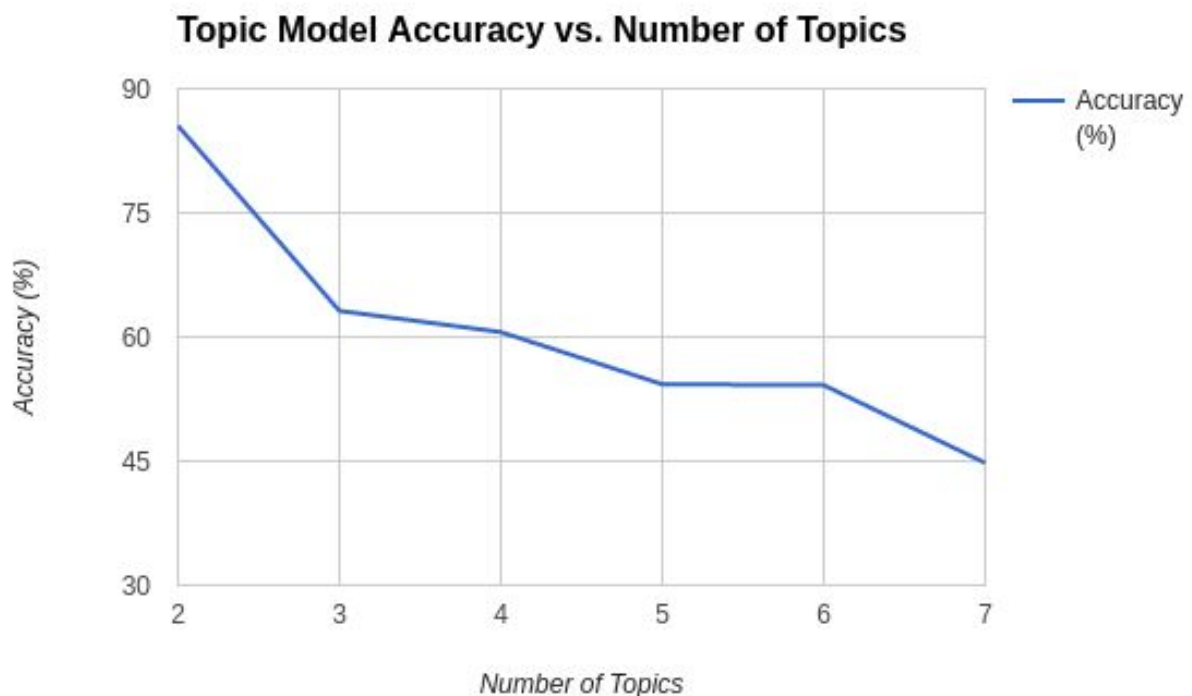
The average accuracy for splitting datasets of two topics was: **85.52%**

4.5. Topic Model Accuracy for Higher Numbers of Topics

This aim of this test is to see how increasing the number of topics in the dataset affects the accuracy. The procedure for calculating the accuracy is the same as for the previous test. The test datasets will contain between 2 and 7 topics. The corresponding accuracy will be recorded and graphed.

Results

As we can see in the graph below, the performance drops significantly for datasets of more than 2 topics.



One cause for the decreased accuracy was that the algorithm grouped documents from two or more topics into one group, especially topics that use similar vocabulary. Some examples of topics that were grouped together are Christianity & Atheism, Automobiles & Motorcycles and Mac Computers & Computer Graphics. The more topics in the dataset, the more likely it was to contain similar topics. After 7 topics it became difficult to determine any dominant topics in any of the predicted groups, making it difficult to test the accuracy for higher numbers of topics.

5. Conclusions and Future Work

5.1. Conclusions

As was demonstrated, the supervised classification of text documents was a success. A high accuracy was achieved across a range of different types of datasets. The classification worked well for all the datasets tested, so it can be assumed that it will also work for any dataset of the right format.

The topic model classification proved to be successful also but only for low numbers of topics. With an increase in topics, there was a corresponding decrease in accuracy. There is plenty of room for improvement here to maintain a high accuracy for higher numbers of topics.

5.2. Future Work

Given the opportunity to put more work into this project, the first thing I would do would be to attempt to improve the accuracy of the topic model for higher numbers of topics. Initially the accuracy was low, even with two topics, but I was able to improve it by removing irrelevant and noisy data from the documents and tweaking the parameters of the lda model. I think that with further experimentation I would be able to get the model to work well for higher numbers of topics also. I would also like to test the topic model algorithm on more datasets to test the robustness of the model.

Another potential area for improvement is the GUI application. Although it works and is easy to use, there is room for improvement in the user experience of the application. For example, when the dataset being fed into the unsupervised topic model classification is large, the classification process can take several minutes. Adding an estimated time of completion for the algorithm would improve the user experience. Other small changes like this could be implemented to improve the usability of the application.

There were many potential features of the application that I did not have time to implement over the course of this project. There is endless room for additional features, including different types of text classification and classification of images or audio files, that I could add to this application given more time.

References

- [1] Pang, B. and Lillian L. (2002). [Thumbs up? Sentiment Classification using Machine Learning Techniques](#).
- [2] Turney, P. (2002). [Thumbs Up or Thumbs Down? Semantic Orientation Applied to Unsupervised Classification of Reviews](#)
- [3] Deerwester et al. (1990) *Indexing by Latent Semantic Analysis*.
- [4] Alghamdi, R. and Alfalqi, K. (2015). A Survey of Topic Modeling in Text Mining.
- [5] Introduction to Latent Dirichlet Allocation. 2017. Introduction to Latent Dirichlet Allocation. [ONLINE] Available at: <http://blog.echen.me/2011/08/22/introduction-to-latent-dirichlet-allocation/>. [Accessed 30 March 2017].
- [6] Sentiment140 - A Twitter Sentiment Analysis Tool. 2017. Sentiment140 - A Twitter Sentiment Analysis Tool. [ONLINE] Available at: <http://www.sentiment140.com>. [Accessed 30 March 2017].
- [7] RankSpeed: Listen to the Crowd. 2017. RankSpeed: Listen to the Crowd. [ONLINE] Available at: <http://www.rankspeed.com>. [Accessed 30 March 2017].
- [8] Real Time Search - Social Mention. 2017. Real Time Search - Social Mention. [ONLINE] Available at: <http://www.socialmention.com>. [Accessed 30 March 2017].
- [9] hnldesign.nl. 2017. Whatdoestheinternetthink.net - What does the internet think?. [ONLINE] Available at: <http://www.whatdoestheinternetthink.net>. [Accessed 30 March 2017].
- [10] 4.2. Feature extraction - bag of words representation— scikit-learn 0.18.1 documentation. 2017. 4.2. Feature extraction — scikit-learn 0.18.1 documentation. [ONLINE] Available at: http://scikit-learn.org/stable/modules/feature_extraction.html#the-bag-of-words-representation. [Accessed 30 March 2017].
- [11] Home Page for 20 Newsgroups Data Set. 2017. Home Page for 20 Newsgroups Data Set. [ONLINE] Available at: <http://qwone.com/~jason/20Newsgroups/>. [Accessed 30 March 2017].
- [12] 4.2. Feature extraction - tfidf-term-weighting — scikit-learn 0.18.1 documentation. 2017. 4.2. Feature extraction — scikit-learn 0.18.1 documentation. [ONLINE] Available at: http://scikit-learn.org/stable/modules/feature_extraction.html#tfidf-term-weighting. [Accessed 30 March 2017].
- [13] Working With Text Data — scikit-learn 0.18.1 documentation. 2017. Working With Text Data — scikit-learn 0.18.1 documentation. [ONLINE] Available at: http://scikit-learn.org/stable/tutorial/text_analytics/working_with_text_data.html#evaluation-of-the-performance-on-the-test-set. [Accessed 30 March 2017].
- [14] sklearn.decomposition.LatentDirichletAllocation — scikit-learn 0.18.1 documentation. 2017. sklearn.decomposition.LatentDirichletAllocation — scikit-learn 0.18.1 documentation. [ONLINE] Available at: <http://scikit-learn.org/stable/modules/generated/sklearn.decomposition.LatentDirichletAllocation.html>. [Accessed 30 March 2017].
- [15] Jan Bodnar. 2017. Application skeletons in wxPython. [ONLINE] Available at: <http://zetcode.com/wxpython/skeletons>. [Accessed 30 March 2017].

- [16] Jordan Barber. 2017. Latent Dirichlet Allocation (LDA) with Python. [ONLINE] Available at:
https://rstudio-pubs-static.s3.amazonaws.com/79360_850b2a69980c4488b1db95987a24867a.html. [Accessed 30 March 2017].
- [17] Data. 2017. Data. [ONLINE] Available at:
<https://www.cs.cornell.edu/people/pabo/movie-review-data/>. [Accessed 30 March 2017].