

CLASSIFICATION OF FILM REVIEWS USING MACHINE LEARNING

A Mini project report submitted

To

MANIPAL UNIVERSITY JAIPUR

*For Partial Fulfillment of the Requirement for the
Award of the Degree*

of

BACHELOR OF TECHNOLOGY

in

COMPUTER SCIENCE AND ENGINEERING

by

AAKASH JAIN (120301005)

PUNIT CHHAJER (120301059)

Under the Guidance of

Mr. JAYA KRISHNA R

Assistant Professor,
Department of Computer Science and Engineering

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING



MANIPAL UNIVERSITY JAIPUR
JAIPUR - 303007, RAJASTHAN, INDIA

2015

Abstract

Machine learning techniques are a good fit for text analysis problems. We explore the applicability of various machine learning techniques to classifying film reviews on the basis of the star rating accompanying those reviews. This system could be used to quantify unrated film reviews, and also to gauge public opinion on films pre-release by analyzing comments on previews and trailers. Models explored are Naïve Bayes classification, Linear Support Vector Classification/Regression, Linear Regression, and Logistic Regression. We explore how the parameters affect the outcome and draw conclusions on which model works best. We create a webapp to demonstrate our work.

List of Figures

Fig. 2.1.1. SVM planes	7
Fig. 2.1.2. SVM plane distance	7
Fig. 3.3.1. IPython Notebook	14
Fig. 3.5.1. CLI use cases	18
Fig. 3.5.2. Webapp use cases	19
Fig. 3.6.1. Organizing the dataset	20
Fig. 3.6.2. Training and dumping a pipeline	20
Fig. 3.6.3. Predicting from a dumped pipeline	21
Fig. 4.2.1. Webapp (pos/neg)	27
Fig. 4.2.2. Webapp (stars)	28
Fig. 4.2.3. CLI (training)	29
Fig. 4.2.4. CLI (predicting)	30
Fig. 5.1.1. MultinomialNB (binary)	32
Fig. 5.1.2. MultinomialNB (stars)	32
Fig. 5.1.3. LinearSVC (binary)	33
Fig. 5.1.4. LinearSVC (stars)	34
Fig. 5.1.5. LogisticRegression (binary)	35
Fig. 5.1.6. LogisticRegression (stars)	36
Fig. 5.1.7. LinearSVR	37
Fig. 5.1.8. LinearRegression	38
Fig. 5.2.1. Binary set accuracy	39
Fig. 5.2.2. Stars set accuracy	39

Table of Contents

1	Introduction.....	3
1.1	Motivation.....	3
1.2	Statement.....	3
1.3	Organization of this Report.....	4
2	Background.....	5
2.1	Conceptual Overview	5
	Tf-Idf Score	5
	Naïve Bayes Classification	6
	Support Vector Machines.....	6
	Linear Regression	8
	Logistic Regression	8
2.2	Technologies Used	9
	Scikit-Learn	9
	Flask.....	11
3	Methodology.....	12
3.1	Overview	12
3.2	The ACLIMDB Dataset	12
3.3	Experimentation.....	14
3.4	Creating the prediction framework	17
3.5	Use Cases.....	17
	Command Line Usage.....	18
	Webapp Usage	19
3.6	Activity Diagrams.....	20

4	Implementation.....	22
4.1	Modules.....	22
	names	22
	organize_dataset.....	22
	data_loader	23
	model_presets.....	23
	tfidf_pipeline	24
	scorers	24
	train	24
	predict	25
	main.py (Webapp).....	25
	Other utilities	26
4.2	Screenshots	27
5	Results & Analysis.....	31
5.1	Results.....	31
	Naïve Bayes	31
	LinearSVC.....	33
	LogisticRegression	35
	LinearSVR.....	36
	LinearRegression	38
5.2	Conclusion	39
5.3	Future Work.....	40
	References	40

1 Introduction

1.1 Motivation

Text classification is a common problem with wide applications. Often, text classification problems have solutions that are domain-specific. A solution for texts from one domain cannot be expected to work for another domain without modification. Here, we try to find what machine learning techniques work best for the particular domain of classifying film reviews.

In recent times, it has become increasingly common for film reviews to not be accompanied by a star rating. In newspapers and blogs, for example, several famous movie reviewers have begun to leave out an explicit rating. An accurate classifier for film reviews could have many applications, such as recommending films to reviewers, analyzing trends in the sentiment of famous reviewers, etc.

It could also be used for analyzing public sentiment about a film from comments or blog posts about the film's preview or trailer, since such comments do not include a rating. This would help filmmakers quantify how the public feels about their film even before release.

1.2 Statement

This mini-project aims to find a good machine learning model for classifying text-based film reviews on the basis of how many stars the writer has rated the film.

To accomplish this, we test various machine learning models and try to find the best set of parameters for those models. We try the following:

- Naïve Bayes Classifier
- Linear Support Vector Classifier (LinearSVC)
- Linear Support Vector Regressor (LinearSVR)
- Logistic Regression
- Linear Regression

These models are trained and tested on the AclImdb dataset, a freely available academic dataset of 50,000 film reviews from the IMDB website, accompanied by star rating of 1 to 4 and 7 to 10.

We use the Scikit-Learn library for Python, which contains tools for using the above models, as well as utilities for working with the data, preprocessing text, testing, etc.

We first work towards classifying reviews as negative or positive sentiment, then work towards classifying as the number of stars.

We finally create a small Flask webapp to demonstrate our best classifier.

1.3 Organization of this Report

Chapter 2: Background deals with explanations of the machine learning concepts used in this project as well as the third-party libraries that have been used in it.

Chapter 3: Methodology explains our approach to this project and also contains an explanation of the usage workflow.

Chapter 4: Implementation deals with the organization of the code and how it is to be used. It also contains screenshots of the project in action.

Chapter 5: Results & Analysis presents the findings of our experiments with the various machine learning models, our conclusions, and our thoughts of possibilities for future work.

2 Background

2.1 Conceptual Overview

Tf-Idf Score

Simply representing a document as word counts is known as *bag of words* representation. However this does not tell us how important a word is in the corpus. Intuitively, the rarer a word, the more information it would give us when it does appear.

Tf-idf scores are a way of representing a test document as a vector of scores created from the word counts of the document. The tf-idf value increases proportionally to the number of times a word appears in the document, but is offset by the frequency of the word in the corpus, which helps to adjust for the fact that some words appear more frequently in general.

Term frequency $tf(t, d)$: It is the number of times a term t occurs in a document d . However, we also use *logarithmically scaled frequency* to account for non-linear scale of importance. A term that occurs 20 times is not likely to be twice as important as a term that occurs 10 times. However, a term that appears 100 times may be twice as important. So , we use $1 + \log(tf(t, d))$

Inverse document frequency $idf(t)$: It is the inverse of the fraction of documents in the corpus that contain term t . Again, it is logarithmically scaled. We also add 1 to the count to prevent division by 0 in case we encounter an unseen term.

$$idf(t) = \log\left(\frac{N}{1 + d}\right)$$

where N is the number of documents in the corpus, and d is the number of documents that contain term t .

Then the tf-idf score of a term t in a document d can be found as

$$tfidf(t, d) = tf(t, d) * idf(t)$$

Naïve Bayes Classification

Naive Bayes is a model that assign class labels to inputs, represented as vectors of feature values, where the class labels are drawn from some finite set. They are based on the Naïve Bayes assumption: that the value of a particular feature is independent of the value of any other feature, given the class variable.

Given a training set where each document is represented as a feature vector $\{x_1...x_n\}$, and a set of target classes $\{C_1..C_m\}$,

Naïve Bayes finds the probability that each class contains each feature value, ie, $P(X_j|C_i)$. It also finds the prior probability of a document belonging to each class, ie, $P(C_i)$.

Now given an unseen document X represented as vector $\{x_1...x_n\}$, we are able to find the probability that it belongs to each class using the formula:

$$P(C_i | X) = P(x_1 | C_i) * ... * P(x_n | C_i) * P(C_i)$$

We then classify X as the C_i for which $P(C_i/X)$ was found to be maximum.

Support Vector Machines

Usually used for classifying input vectors into two classes, SVM can also be extended to classifying into multiple classes, or even to a form of regression (ie, prediction in the form of a continuous variable instead of classes).

SVMs try to create a separation between inputs by forming a hyperplane that separates the inputs of different classes by the maximum possible distance.

Classification using the SVM assumes that inputs are linearly separable on a k -dimensional plane (where k is the size of the input vectors). This makes it unsuitable for many types of data.

Non-linear SVMs are also possible, but are usually extremely slow for high dimensional data such as text documents.

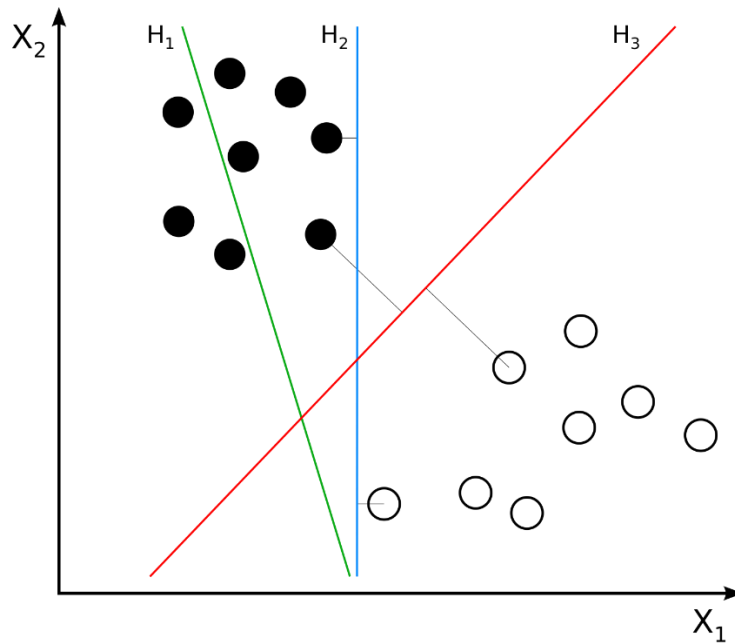


Fig. 2.1.1. Here we see 3 planes of separation for the input. H_1 does not correctly separate, while H_2 and H_3 do. However, H_3 separates them by the largest margin. The objective of SVM is to find such a plane.

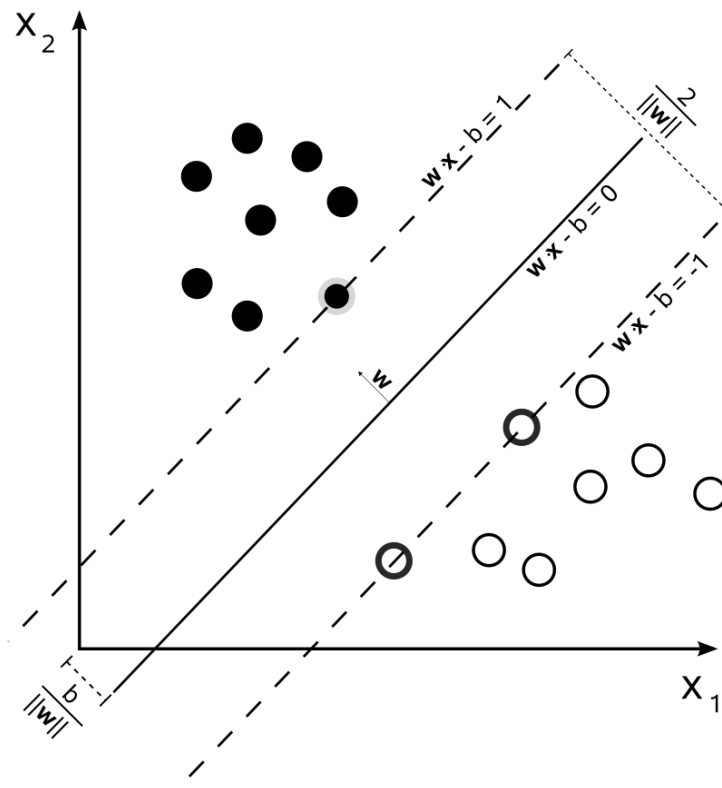


Fig. 2.1.2. SVMs work by trying to form a hyperplane that separates classes by the largest distance. In this figure, the value $2/\|w\|$ is to be maximized, given a plane represented as $w \cdot x - b = 0$.

Linear Regression

Linear regression attempts to formulate a linear equation that, given an input vector, predicts a continuous variable as output.

Given $X = \{x_1 \dots x_n\}$, and $F(X) = y$, linear regression would try to fit values B such that

$$\sum B_i * x_i + c = y$$

This fitting is done using the least squares approach, which leads to the formula:

$$B = (X^T X)^{-1} X^T y$$

This works well for input whose corresponding output is a linear combination of its features. Intuitively, it should work well for text.

Logistic Regression

Rather than predict the exact output value like in linear regression, logistic regression fits an equation that predicts the probability of input belonging to a particular class. It is different from Naïve Bayes since it does not simply calculate this probability using the likelihood of terms, instead it fits a curve that predicts probabilities.

Given an input vector X , we predict the probability of observing its output y as

$P(X, \alpha, \beta) = \frac{e^{\alpha + \beta X}}{1 + e^{\alpha + \beta X}}$ where α and β are to be found such that for all input vectors X_i , the product of their P is maximized.

This maximization is done using linear regression.

Usually it is modeled for 2 output classes but can be extended to multiple classes using various strategies.

2.2 Technologies Used

Scikit-Learn

The library used in this project is Scikit-Learn. It is an open source Python library entirely geared towards machine learning. It provides various types of customizable machine learning models as well as other tools often required in machine learning. Once a model has been created, it can be fit on training data and then predicted on testing data as:

```
model.fit(train.data, train.target)
predicted = model.predict(test.data)
```

`sklearn.naive_bayes.MultinomialNB:`

Naïve Bayes classifier for multinomial (more than 2 classes) data.

Relevant parameters:

- `alpha`: additive smoothing parameter

`sklearn.svm.LinearSVC:`

Linear Support Vector Classifier with several options.

Relevant parameters:

- `multi_class`: whether to use a one-vs-rest approach or the Crammer-Singer approach for more than 2 classes.
- `loss`: the loss function is the measure of quality of a solution. It usually measures error.
- `penalty`: the norm to apply to constrain a solution.
- `C`: the coefficient of the loss function.
- `dual`: whether to solve as a primal problem or dual problem.

`sklearn.svm.LinearSVR:`

Linear Support Vector Regression with several options.

Relevant parameters:

- loss: the loss function is the measure of quality of a solution. It usually measures error.
- C: the coefficient of the loss function.
- dual: whether to solve as a primal problem or dual problem.
- epsilon: another parameter used in the loss function.

`sklearn.linear_model.LinearRegression`:

Simple Linear Regression performed using least squares approach.

Relevant parameters: None.

`sklearn.linear_model.LogisticRegression`:

Logistic Regression using LinearRegression. Has several options.

Relevant parameters:

- solver: the method used for solving. We try Newton-CG and LibFGS since the other options are inaccurate for high dimensional multiclass data.
- loss: how to normalize penalties. Must be 'l2' for our chosen solvers.
- dual: whether to solve the primal problem or dual problem. Must be primal for our chosen solvers.
- multi_class: the strategy to use for more than 2 classes.
- C: inverse of regularization strength.

`sklearn.feature_extraction.text.TfidfVectorizer`:

Converts text documents to tf-idf score vectors. First converts to word counts, then finds tf-idf values.

Relevant parameters:

- decode_error: what to do when a character cannot be decoded.
- strip_accents: what to do with accented characters
- lowercase: whether to convert all letters to lowercase
- stop_words: whether to remove all standard English stop-words.
- analyzer: whether to treat character sequences or words as features.
- ngram_range: minimum and maximum size of ngrams to form as features.
- min_df: minimum document frequency for a term to be considered.
- use_idf: whether to use tf-idf or only tf.
- smooth_idf: whether to add 1 to idf denominator.

- `sublinear_tf`: whether to use log-scale tf.

`sklearn.model_selection.GridSearchCV`:

Used to automate evaluation of models with various parameter options. Has parallelization support. Can be used to find best scoring parameter combinations.

Relevant parameters:

- `model`: the model object.
- `parameters`: a dictionary of named parameters.
- `n_jobs`: number of parallel jobs to run at a time.

Flask

Flask is a lightweight server framework for Python. We use it to create the backend for our webapp. It was essential for the framework to be Python-based since otherwise we could not run our trained classifier.

Flask works very simply based on URL endpoints and HTTP methods.

```
app = Flask('foo')
@app.route('/', methods=['GET'])
def homepage():
    return 'Hello World!'
```

The above example is sufficient to create a server that shows “Hello World!” when we navigate to the root URL.

Flask can also return webpages and Jinja templates as well as content for those templates to create more complex webapps.

We simply load our classifier from disk when the server starts, then watch for POST requests containing reviews. The predicted result can be sent back to the user.

3 Methodology

3.1 Overview

We use interactive Python sessions for preliminary evaluation of models on various parameters.

Using IPython Notebooks, we are able to store and organize our findings with respect to which parameters work best, and the accuracy, precision, recall, etc. of our classifiers.

We simultaneously create a series of modules that form our framework for using models, working with the dataset, saving and loading models, etc.

Before detailing the workflow, it is essential to provide an overview of the dataset and how Scikit-Learn handles datasets.

3.2 The ACLIMDB Dataset

The ACL IMDB dataset contains two sets, train & test, intended for training and testing purposes respectively. Each of them contains 12500 positive and 12500 negative reviews respectively, for a total for 50000 reviews in the dataset. There are also 50000 unlabeled reviews, which are not useful for our purposes.

Filenames of each review indicate the number of stars that review was rated. We will need to process the dataset to split on the basis of stars. We call the positive/negative set the 'binary' set, and the star-rated set the 'stars' set.

Furthermore it would also be helpful to have an aggregate of the train and test sets so that we can split them as required (50% is a large fraction for a test set).

Scikit-Learn provides a *load_files* function that loads all files inside the folders in a path. The names of the folders are assumed as the labels or targets of the files inside them.

A loaded dataset contains three attributes: *.data*, *.target*, *.target_names*, which have the file contents, target indices, and target names respectively.

Initial dataset:

```

aclImdb/
  test/
    pos/
      0_10.txt
      1_10.txt
      ...
    neg/
      0_2.txt
      1_3.txt
      ...
  ...
  train/
    pos/

    neg/
      0_3.txt
      1_1.txt
      ...
    unsup/
      0_0.txt
      1_0.txt
      ...

```

Final dataset:

```

aclImdb/
  test/
    pos/
    neg/
    01/
      5_1.txt
      ...
    02/
      0_2.txt
      ...
  ...
  train/
    pos/
    neg/
    01/
    02/
    ...
  aggregate/
    pos/
    neg/
    01/
    02/
    ...

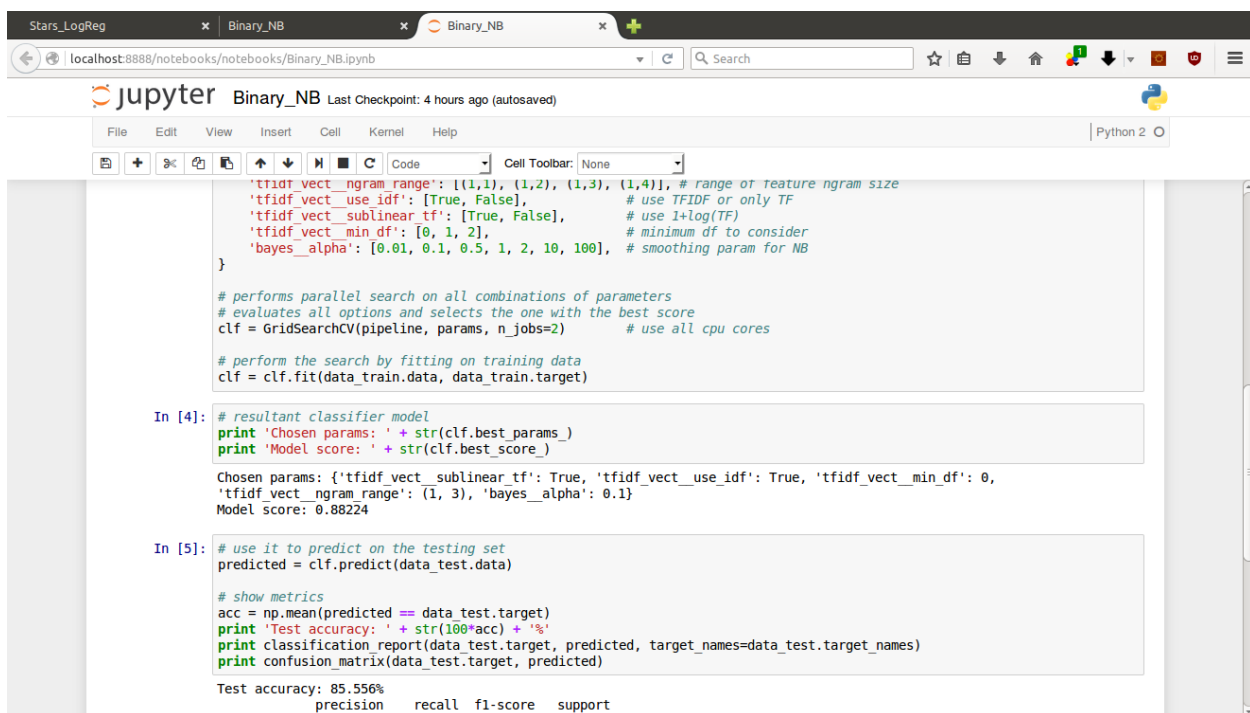
```

Example: train/pos/25_7.txt ('_7' in filename means 7 stars):

I never saw this when I was a kid, so this was seen with fresh eyes. I had never heard of it and rented it for my 5 year old daughter. Plus, the idea of Christopher Walken singing and dancing made me curious. The special fx are cheesy and the singing and dancing is mediocre. But the story is great. My daughter was entranced. I loved watching Walken in this role thinking about what the future held for him. Very amusing to see him dance! And if the songs weren't great, at least they weren't Disney over-produced saccharine sweetness. The ogre scene in the beginning was a little scary for her, and she was a little nervous when we saw him again at the end, but it was mostly benign. Interestingly, we had recently read "Puss in Boots", and I had wondered about the implausibility of the story. But while staying true to almost every aspect, Walken's acting made it believable. Great fun. I'd watch it again with my daughter.

3.3 Experimentation

1. In this phase we use IPython Notebooks to document our exploration of various parameters in the chosen models.
2. We use the test/train split rather than the aggregate set.
3. We use Grid Search to evaluate the effect of various parameters on the models, and use the best scoring set.
4. We store accuracy, confusion matrix, and classification report for each model's best version.



```
Stars_LogReg x Binary_NB x Binary_NB x +
localhost:8888/notebooks/notebooks/Binary_NB.ipynb
jupyter Binary_NB Last Checkpoint: 4 hours ago (autosaved)
Python 2
File Edit View Insert Cell Kernel Help
Code Cell Toolbar: None

'tfidf_vect_ngram_range': [(1,1), (1,2), (1,3), (1,4)], # range of feature ngram size
'tfidf_vect_use_idf': [True, False], # use TFIDF or only TF
'tfidf_vect_sublinear_tf': [True, False], # use 1+log(TF)
'tfidf_vect_min_df': [0, 1, 2], # minimum df to consider
'bayes_alpha': [0.01, 0.1, 0.5, 1, 2, 10, 100], # smoothing param for NB
}

# performs parallel search on all combinations of parameters
# evaluates all options and selects the one with the best score
clf = GridSearchCV(pipeline, params, n_jobs=2) # use all cpu cores

# perform the search by fitting on training data
clf = clf.fit(data_train.data, data_train.target)

In [4]: # resultant classifier model
print 'Chosen params: ' + str(clf.best_params_)
print 'Model score: ' + str(clf.best_score_)

Chosen params: {'tfidf_vect_sublinear_tf': True, 'tfidf_vect_use_idf': True, 'tfidf_vect_min_df': 0,
'tfidf_vect_ngram_range': (1, 3), 'bayes_alpha': 0.1}
Model score: 0.88224

In [5]: # use it to predict on the testing set
predicted = clf.predict(data_test.data)

# show metrics
acc = np.mean(predicted == data_test.target)
print 'Test accuracy: ' + str(100*acc) + '%'
print classification_report(data_test.target, predicted, target_names=data_test.target_names)
print confusion_matrix(data_test.target, predicted)

Test accuracy: 85.556%
precision recall f1-score support
```

Fig. 3.3.1. Experimenting on MultinomialNB in an IPython Notebook

At first, we work only with the binary dataset. This allows us to determine whether or not sentiment analysis is viable using our chosen techniques. For this classification, we use only MultinomialNB, LinearSVC, and LogisticRegression. The others need not be used since the problem is one of binary classification.

We can then move on to evaluating all 5 models with the stars dataset. In case of the regression models, we would need to convert the continuous predicted output to discrete values,

representing number of stars. It is necessary to allow some degree of error in the stars dataset. We may for example be comfortable with a deviation of 1 star from the actual value.

Algorithm for experimentation:

1. Import required libraries and functions
2. Load the default test/train data with either stars or binary categories, into *test* and *train*


```
test = load_files('aclImdb/test', categories=['neg', 'pos'])
```
3. Create a TfidfVectorizer *tfidf* with the best parameters
4. Fit *tfidf* on the train data and transform it


```
train_X = tfidf.fit_transform(train.data, train.target)
train_y = train.target
```
5. Transform the test data


```
test_X = tfidf.transform(test.data)
test_y = test.target
```
6. Create an instance of the model, and a list of the parameters to search through. Create a custom scoring function if required.


```
model = LogisticRegression()
params = {'solver': ['newton-cg', 'lbfgs'],
          'C': [0.1, 1.0, 5]}
def err1(true_y, pred_y):
    diff = abs(true_y - pred_y)
    return mean(diff <= ones(len(diff)))
```
7. Create a GridSearchCV instance and fit it on the train data. Find best parameters.


```
gs = GridSearchCV(model, params, scoring=make_scorer(err1))
gs.fit(train_X, train_y)
gs.best_params_
```
8. Test this best model by predicting on the test set. In case of continuous values, clip and round the predicted values.


```
predicted = gs.predict(test_X)
predicted = clip(round(predicted), 0, 7)
```

9. Print various accuracy measures if required. Print confusion matrix. Print classification report if required.

```
Exact accuracy: mean(test_y == predicted)
Off-by-one: mean(diff <= ones(len(diff)))
Off-by-two: mean(diff <= 2 * ones(len(diff)))
Classification report: classification_report(test_t, predicted)
Confusion matrix: confusion_matrix(test_y, predicted)
```

Below we explain some of the metrics used above and throughout this report:

Exact accuracy: It is the fraction of predicted values that exactly match the true values.

Off-by-one: It is the fraction of predicted values that exactly match the true values, or are greater or smaller by 1. This is a good metric for stars classification.

Off-by-two: It is the fraction of predicted values that exactly match the true values, or are greater or smaller by up to 2.

Confusion matrix: It is a matrix where each row corresponds to actual values and each column corresponds to predicted values.

Practical:			Ideal:		
	predicted A	predicted B		predicted A	predicted B
actual A	400	100	actual A	500	0
actual B	50	450	actual B	0	500

The above practical example for 2 classes A and B shows that, out of 500 inputs that should have been classified as A, 400 were predicted as A and 100 were confused as B. Out of 500 inputs that should have been classified as B, 450 were predicted as B and 50 were confused as A.

Classification report: It is a format for reporting precision, recall, and f1 scores class-wise. It is unsuitable for large number of classes and when some amount of error is acceptable. This is why we don't use it for the stars dataset.

	precision	recall	f1-score	support
neg	0.83	0.89	0.86	12500
pos	0.88	0.82	0.85	12500
avg / total	0.86	0.86	0.86	25000

The above report shows precision, recall, and f1-score in each class and overall. Support means the number of inputs that actually belong to a particular class.

3.4 Creating the prediction framework

Following experimentation, we would need to code the following features, broadly:

1. To organize the dataset, by categorizing on basis of stars and by creating an aggregate dataset.
2. To load the data set, either aggregate or test/train, stars or binary, and also to split the data on a custom fraction.
3. Presets for the best models that we have found.
4. Presets for the best configuration of TfidfVectorizer, along with a way to pipeline vectorization with classification.
5. To train a chosen model on either stars or binary data, and possibly to dump a copy. Also to train and dump all possible classifiers.
6. To predict on any passed input using the dump of a chosen model.
7. Some other utilities such as scoring functions.

Further, the webapp requires the following features:

1. A server that watches for a review to be submitted, and returns the classification.
2. Frontend templates for the website
3. A script to create low size classifier dumps specially for usage in the webapp.

3.5 Use Cases

The products of this mini-project can be used either as a command-line application or in form of the webapp. We model use cases for both these forms of usage.

Command Line Usage

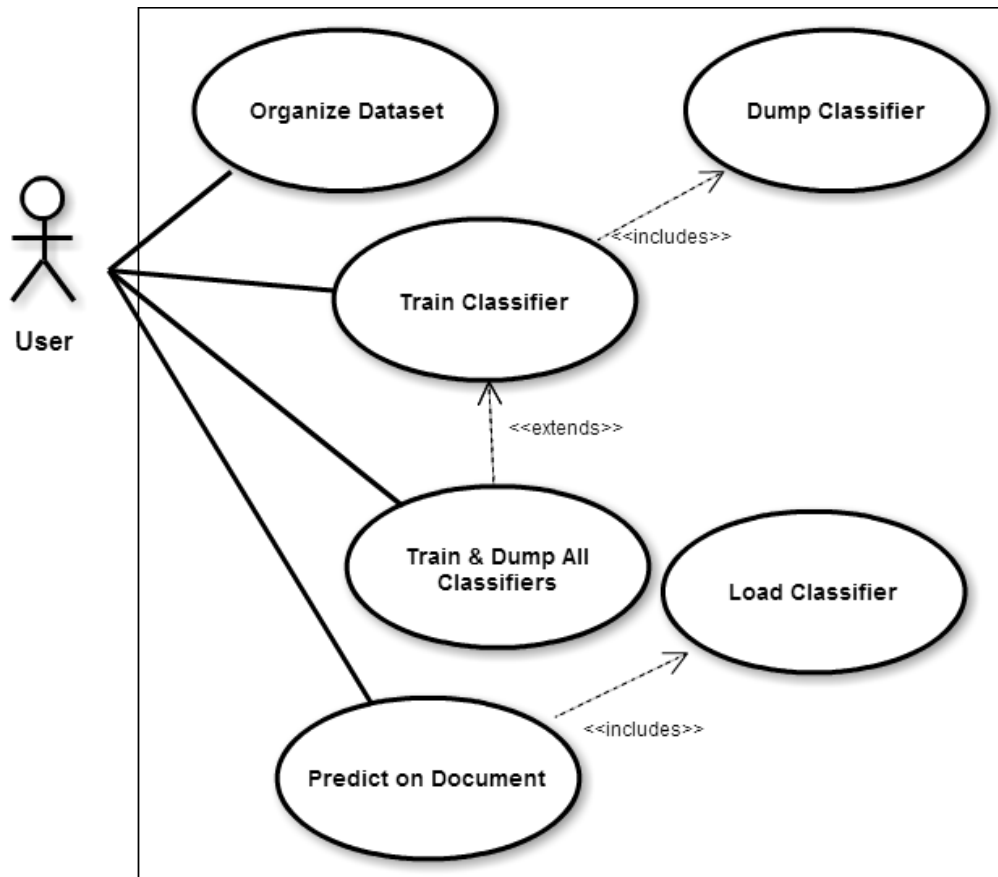


Fig. 3.5.1. CLI use cases

Webapp Usage

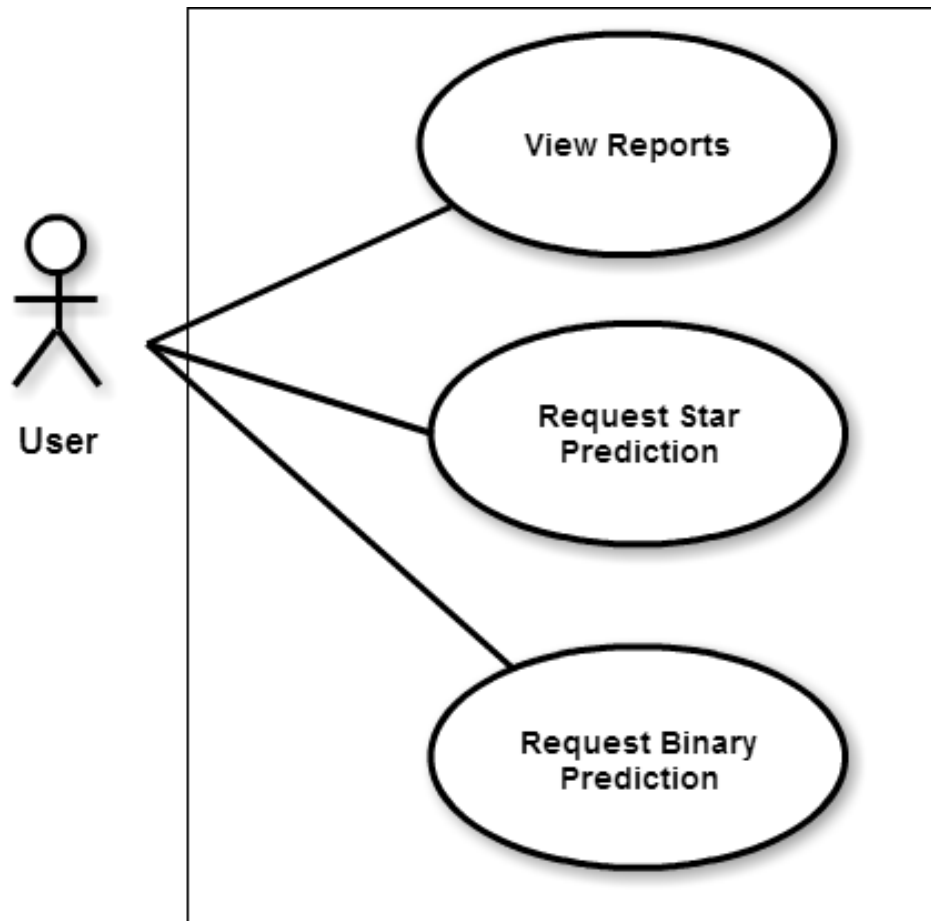


Fig. 3.5.2. Webapp use cases

3.6 Activity Diagrams

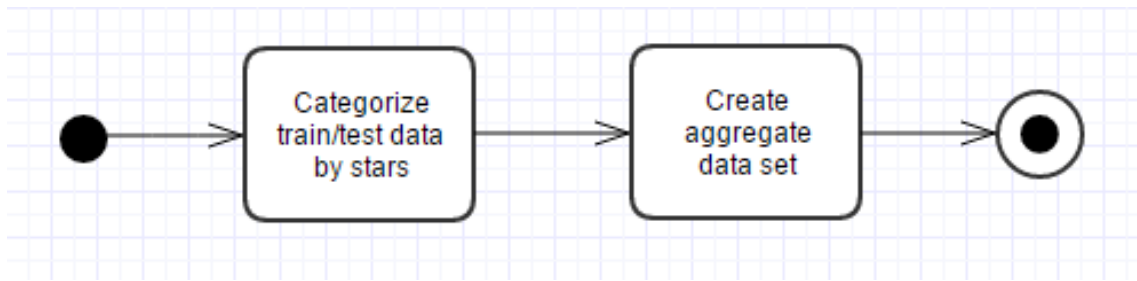


Fig. 3.6.1. Organizing the dataset

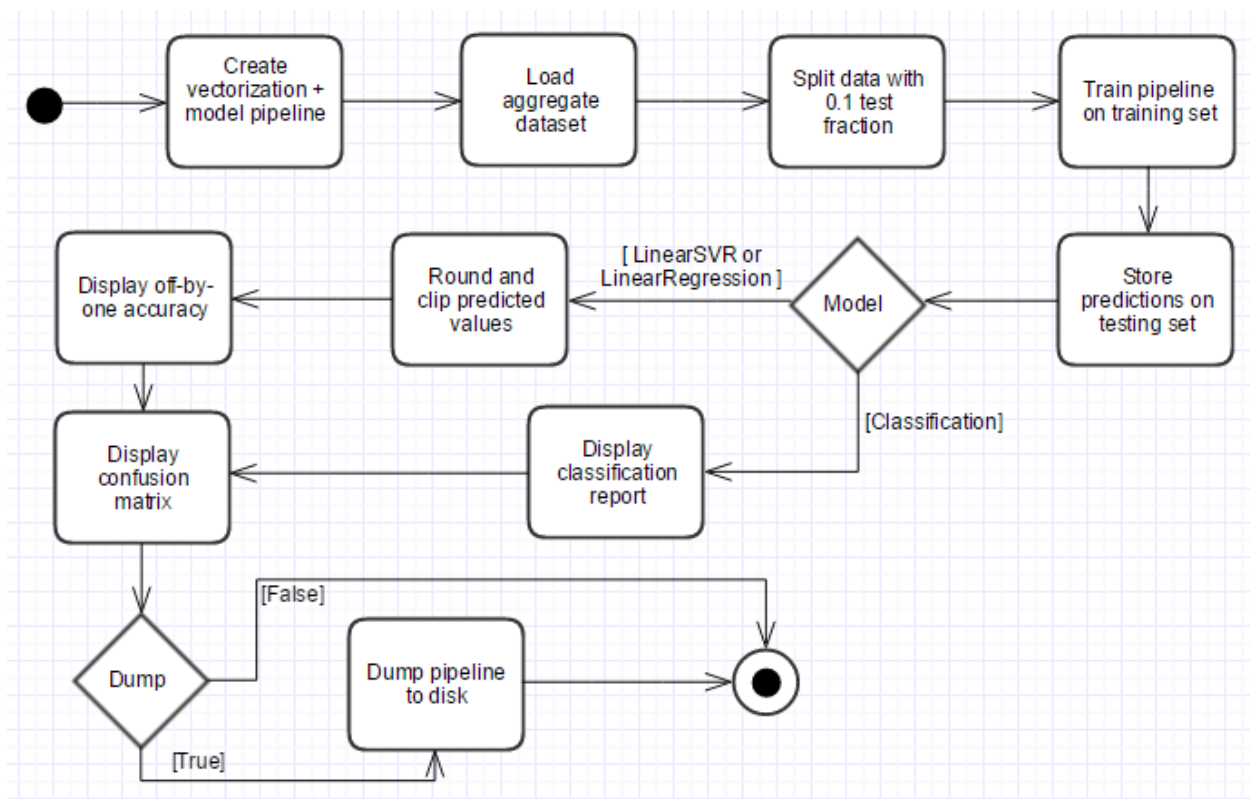


Fig. 3.6.2. Training and dumping a pipeline

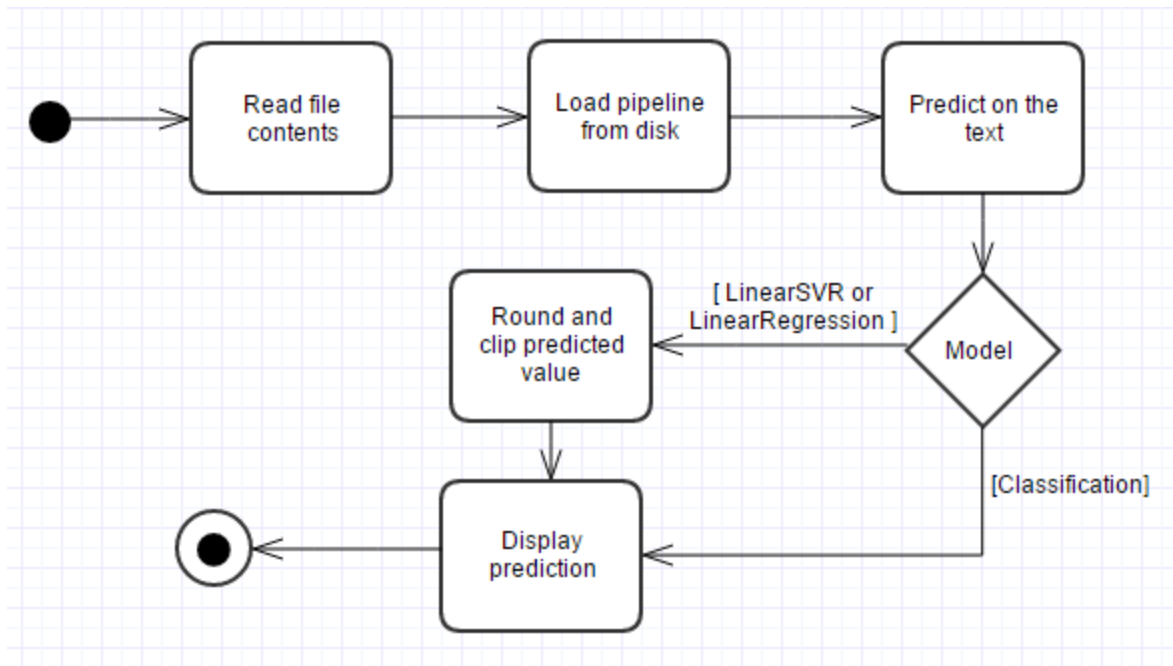


Fig. 3.6.3. Predicting from dumped pipeline

Above, we outline general workflows for using the command line application.

In addition, any of the actions in these activities can be performed in isolation by importing the corresponding module into the user's own Python code.

4 Implementation

4.1 Modules

names

This module contains strings used throughout the project as names for categories and models. It contains the following:

categories – contains class names for ‘binary’ and ‘stars’ dataset.

models – short names for all the models we are using.

classifiers – feasible model/category pairs.

organize_dataset

This module contains functions for organizing the AclImdb dataset to the point required for our project to work. It can be run directly from the command line. Contains:

*categorize_by_stars(*paths)* – Takes any number of path (intended to be the paths for *test* and *train*) and copies the files in them into folders on the basis of filename. It is assumed files are named as *id_stars.txt*

```
categorize_by_stars(path):
    for star in [1,2,3,4,7,8,9,10]
        make star folder
    for category in [pos, neg]
        for file in category folder
            extract star rating from filename
            copy file to corresponding star folder
```

create_aggregate_set(path) – Takes a path, and concatenates the contents of *train* and *test* subfolders into a new *aggregate* folder.

```
create_aggregate_set(path):
    for category in [neg,pos,1,2,3,4,7,8,9,10]
        make aggregate folder for category
        for set in [train, test]
            for file in set
                copy file to aggregate folder
```

data_loader

This module is used to load data for training & testing. Contains:

load(dataset, categories) – takes a dataset argument (either ‘full’ or ‘split’) and a list of category names. In case of ‘full’, it returns an sklearn dataset object containing the aggregate dataset. In case of ‘split’, it returns two sklearn dataset objects containing the train and test sets respectively.

```
load(dataset, categories[])
    if dataset == full
        train = load(aggregate set, categories)
        return train
    if dataset == split
        train = load(train set, categories)
        test = load(test set, categories)
        return test, train
```

split(dataset, test_size) – takes a dataset object and splits it, depending on a float between 0 and 1 that indicates what fraction of it should be the test set. This split is done in a manner that preserves the ratio of the different classes between the resulting split sets.

```
split(dataset, test_size)
    splitter = StratifiedShuffleSplit(test_size)
    //provided in sklearn, split indices preserve class ratios
    train, test = splitter.split(dataset.data, dataset.target)
    return train, test
```

model_presets

This module contains the presets for the best configuration of each of our models that we have found through experimentation.

get(model_name) – takes the name of the required model and returns an instance of the classifier with the best parameters in place.

```
get(model_name)
    return {'nb': MultinomialNB(...),
            'svc': LinearSVC(...),
            'logreg': LogisticRegression(...),
            'linreg': LinearRegression(...),
            'svr': LinearSVR(...)}[model_name]
```

tfidf_pipeline

This module is used to build a pipeline for performing tf-idf vectorization and training or predicting with a model.

`__tfidf()` – returns an instance of `TfidfVectorizer` with the best parameters in place.

`make(model_name)` – returns a pipeline that first performs vectorization, and then predicts or trains with the model corresponding to the passed name.

```
make(model_name)
    return Pipeline([('tfidf', __tfidf),
                     ('model', model_presets.get(model_name))])
```

scorers

This module contains scoring functions for checking exact accuracy, off-by-one accuracy (shown below), and off-by-two accuracy. This can be used for diagnostics and during grid searching.

```
err1(true, pred) //off-by-one accuracy
    pred = clip(round(pred), 0, 7)
    accuracy = mean(|true - pred| <= [1]*len(pred))
    return accuracy
```

train

This module is for training (and optionally dumping) any of the feasible models. It can be run directly from the command line. Contains:

`train(model_name, category_type, dump)` – Trains the specified model on the specified categories. Uses the aggregate dataset, with a test fraction of 0.1. Dumps the pipeline to disk if `dump` is `True`.

```
train(model, category, dump)
    data = data_loader.load(full, category)
    train, test = data_loader.split(data, 0.1)
    clf = tfidf_pipeline.get(model)
    clf.fit(train.data, train.target)
    predicted = clf.predict(test.data, test.target)
    if regression model
        clip & round predicted
        print scorers.err1
    else
```

```
print scorers.err0
print classification report
print confusion matrix
plot normalized confusion matrix
```

predict

This module is for predicting on a review, using a dumped pipeline. Can be run directly from the command line by passing it a file containing a review.

predict(text, model_name, category_type) – loads the pipeline corresponding to the specified model and categories, then predicts and prints the output.

```
predict(text, model_name, category_type) //text is read from a file
clf = load dumped classifier
predicted = clf.predict(text)
if regression model
    clip & round predicted
print labels corresponding to predicted values
```

main.py (Webapp)

A Flask webserver that watches on the '/' URL for HTTP POST requests containing form submissions. The form should contain review text and category type (stars or pos/neg). It then runs the LinearRegression pipeline and renders the prediction.

The prediction pipeline is loaded up when the server starts.

```
load linear regression (stars) and svc (binary) models
watch on route / for requests
if request method == GET
    render the template with empty prediction
if method == POST
    get text and category from request
    predict with appropriate model
    render the template with predicted value
```

Other utilities

Below are some miscellaneous scripts used in this project.

dump_classifiers.py – runs *train* on all feasible model/category combinations and dumps them to a folder. This is an extremely heavy process may take several hours while causing the computer to become unresponsive.

dump_web_classifiers.py - used to create small size classifiers for use in the webapp. It trains and dumps LinearRegression for stars and LinearSVC for binary, using the test/train set rather than the aggregate set.

plot.py – contains a function *plot_confusion_matrix* that normalizes a confusion matrix and creates a grid plot that helps in visualization.

4.2 Screenshots

Home

Comments


Bit of a mess all around this. Never really got off the ground and the script was a load of nonsense. Acting was only ever tepid and the sets were not good at all.. Michael Caine was just in it for a few quid and Van Diesel can look forward to the money he earned for it..maybe it could be used for acting lessons. Bad effects and all the extras were uninspired the whole way through. Brought nothing at all to the genre and is an easily forgettable pile of bad movie making.

☐ Stars ☒ Positive/Negative

Submit

🤔

Fig. 4.2.1. Pos/Neg classification using the webapp. This review for The Last Witch Hunter is correctly classified as negative.

Home

Comments

When I first heard that Nolan was preparing a sci-fi movie, I felt like a kid again, waiting for his Christmas gift under the tree. I knew it would become a classic. And I'm sure it will.

☐ Stars ☒ Positive/Negative

Submit




Fig. 4.2.2. Star classification using the webapp. This review for Interstellar was actually rated 8, but is classified as 7.


```
aakash@aakash-HP-Pavilion-15-Notebook-PC: ~/code/project
aakash@aakash-HP-Pavilion-15-Notebook-PC:~/code/project$ ipython train.py linreg
stars dump
Loading data...
Done.
Training...
Done.
Testing...
Accuracy (off-by-one): 0.6916
[[537 181 138 101 40 11 4 0]
 [160 111 86 64 23 13 0 2]
 [135 89 107 90 52 14 8 1]
 [ 62 88 108 132 97 34 11 1]
 [ 1 3 26 51 119 127 102 51]
 [ 0 3 11 33 102 154 130 153]
 [ 0 1 7 20 51 103 103 176]
 [ 1 3 12 20 81 161 219 476]]
Saving classifier...
Done.
aakash@aakash-HP-Pavilion-15-Notebook-PC:~/code/project$
```

Fig. 4.2.3. Using the CLI to train and dump the LinearRegression model for stars classification. The confusion matrix shows fairly good results. Despite quite a lot of misclassified cases, the separation of positive and negative sentiment remains mostly intact.

```
aakash@aakash-HP-Pavilion-15-Notebook-PC: ~/code/project
aakash@aakash-HP-Pavilion-15-Notebook-PC:~/code/project$ ipython predict.py linr
eg stars 60_8.txt
Reading file...
Done.
Loading classifier...
Done.
Predicting...
08
aakash@aakash-HP-Pavilion-15-Notebook-PC:~/code/project$
```

Fig. 4.2.4. Using the CLI to classify a review with the dump from Fig 4.2.3. The review is correctly classified as 8 stars.

5 Results & Analysis

5.1 Results

Here, we present our findings with respect to the best configurations and the classification results they give. **Convention:** *Best parameters are bold and underlined.*

Naïve Bayes

MultinomialNB has a single parameter, the alpha value. We first test with the binary dataset. We also used this phase to test various options for the TfidfVectorizer. We tested the following:

```
ngram_range: [(1,1), (1,2), (1,3), (1,4)]
Upper and lower limit on the size of ngrams used as features.

use_idf: [True, False]
Whether to use tf-idf or only tf.

sublinear_tf: [True, False]
Whether to use tf or 1+log(tf).

min_df: [0, 1, 2]
Minimum df for a term to be considered in the vocabulary.

alpha: [0.01, 0.1, 0.5, 1, 2, 10, 100]
Prior probability value for classes.
```

The results for TfidfVectorizer were found to be applicable for all models.

Upon testing, the results were:

```
Exact accuracy: 0.85556
Classification report:
              precision    recall  f1-score   support

     neg         0.83         0.89         0.86       12500
     pos         0.88         0.82         0.85       12500
avg / total         0.86         0.86         0.86       25000

Confusion matrix:
[[11132  1368]
 [ 2243 10257]]
```

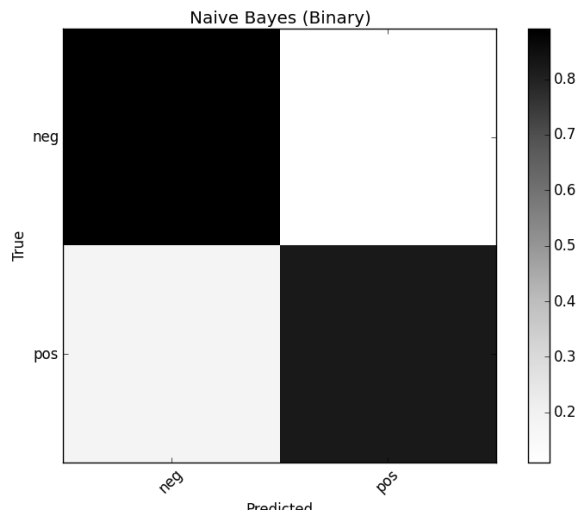


Fig 5.1.1 The results are decent. However, for such a trivial classification, we can probably do better.

We also tried MultinomialNB with the stars dataset. The best value for alpha was again found to 0.1. This time we used an off-by-one scorer while grid searching. On the test set, the results were:

Exact accuracy: 0.35772
Off-by-one accuracy: 0.51604
Off-by-two accuracy: 0.68612
Confusion matrix:
`[[4905 0 0 0 0 0 0 117]`
`[2169 15 0 0 0 0 0 118]`
`[2292 0 24 0 0 0 0 225]`
`[2175 0 0 23 1 3 0 433]`
`[967 0 0 1 3 7 0 1329]`
`[874 0 0 0 1 19 0 1956]`
`[556 0 0 0 0 1 9 1778]`
`[1052 0 0 0 1 1 0 3945]]`

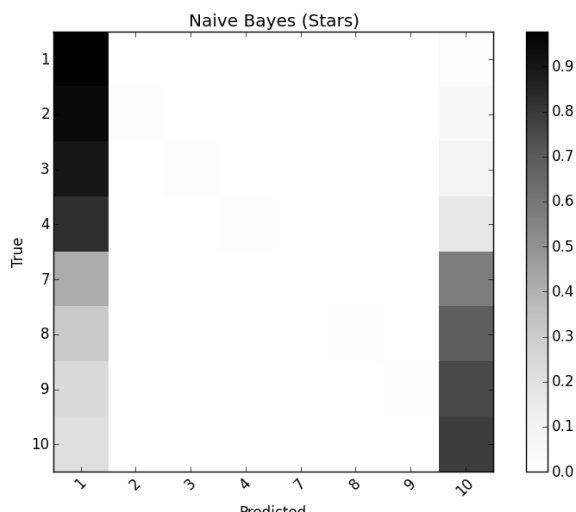


Fig. 5.1.2. It can clearly be seen that MultinomialNB fails entirely here. It classifies most of the reviews as either 1 or 10 stars, and even then does not maintain the positive/negative split well.

LinearSVC

We explored the following parameters on the binary dataset:

loss: ['hinge', 'squared hinge']

This is the measure of how good a solution is. Hinge is calculated as $\sum \max(0, 1 - y_i * f(X_i))$ while squared hinge is the same with squared terms

penalty: ['l1', 'l2']

The type of constraint to apply to the coefficients that form the solution of the model. L1 is $\sum |C_i|$ while L2 is $\sqrt{\sum |C_i|^2}$

C: [0.1, 0.5, 1.0, 2, 4, 8, 12, 16, 32]

Coefficient that is multiplied with the loss.

dual: [True, False]

Solve either the primal or the dual form of the problem.

On the test set, the results were:

Exact accuracy: 0.89032

Confusion matrix:

```
[[11116 1384]
 [ 1358 11142]]
```

Classification report:

	precision	recall	f1-score	support
neg	0.89	0.89	0.89	12500
pos	0.89	0.89	0.89	12500
avg / total	0.89	0.89	0.89	25000

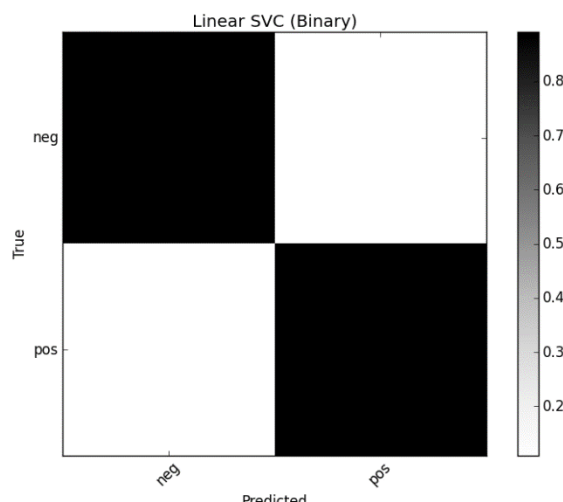


Fig 5.1.3. These results are noticeably better than MultinomialNB.

We also tried it on the stars dataset, testing the same parameters as before with the addition of:

```
multi_class: ['ovr', 'crammer-singer']
```

This is the strategy applied to compare classes in case of more than 2 classes. One-vs-rest treats each class' opponents as a single combined class while Crammer-Singer takes a consolidated approach.

It is surprising that one-vs-rest performs better here despite such a large number of classes. The results on the test set were:

Exact accuracy: 0.42484

Off-by-one accuracy: 0.68876

Off-by-two accuracy: 0.85386

Confusion matrix:

```
[[4220  111  178  216   55   56   26  160]
 [1412  136  225  283   85   46   23   92]
 [1070  163  382  523  154   85   31  133]
 [ 712  114  389  750  308  190   44  128]
 [ 138   42  101  246  675  517  134  454]
 [ 136   22   83  147  534  697  214 1017]
 [ 118   19   36   65  256  454  195 1201]
 [ 242   25   58   87  249  463  259 3616]]
```

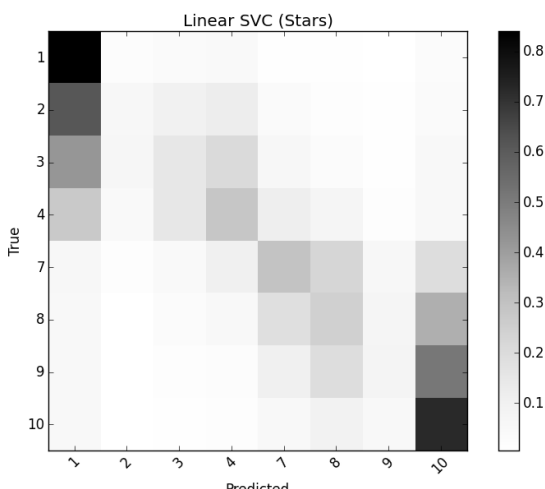


Fig 5.1.4. While this model is far better than MultinomialNB and is more accurate, we see that prediction is still very skewed towards 1 and 10 stars, and there is too much misclassification between the positive and negative sides.

LogisticRegression

We tested this model on the binary dataset with the following parameters:

`solver: ['newton-cg', 'lbfgs']`

The implementation of the solution. These two options are optimized by Scikit-learn to be accurate for high dimensional data with more than 2 classes.

Both of these solvers force usage of `penalty='l2'` and `dual=False`. These parameters have the same meanings as in `LinearSVC`.

`multi_class: ['multinomial', 'ovr']`

This is the strategy applied to compare classes in case of more than 2 classes. One-vs-rest treats each class' opponents as a single combined class while multinomial takes a consolidated approach.

`C: [0.1, 1.0, 5, 10, 50, 100, 500, 1000]`

Coefficient that is multiplied with the loss.

The results on the test set were:

Exact accuracy: 0.88788

Confusion matrix:

```
[[11090  1410]
 [ 1393 11107]]
```

Classification report:

	precision	recall	f1-score	support
neg	0.89	0.89	0.89	12500
pos	0.89	0.89	0.89	12500
avg / total	0.89	0.89	0.89	25000

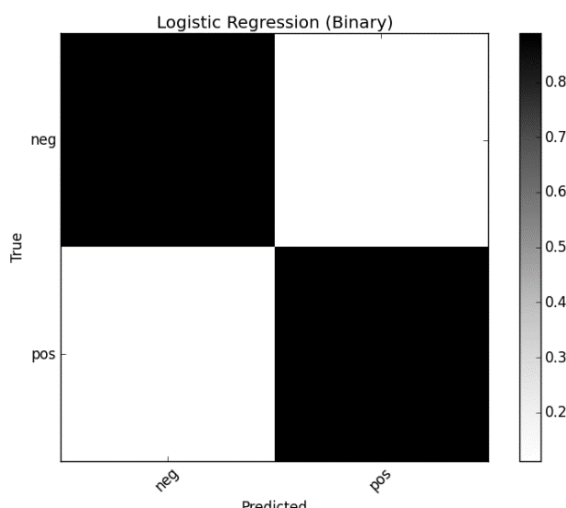


Fig 5.1.5. As per the accuracy and confusion matrix, this model is slightly worse than `LinearSVC`. It is also much slower to train.

We also tested it on the stars dataset:

Exact accuracy: 0.43056
 Off-by-one accuracy: 0.69288
 Off-by-two accuracy: 0.85532
 Confusion matrix:
 [[4256 78 181 225 62 53 17 150]
 [1424 114 240 288 77 47 13 99]
 [1069 127 388 574 146 70 27 140]
 [720 81 378 793 312 190 26 135]
 [148 28 93 246 729 540 90 433]
 [149 17 68 145 575 736 179 981]
 [119 10 38 65 272 491 165 1184]
 [250 19 59 86 267 525 210 3583]]

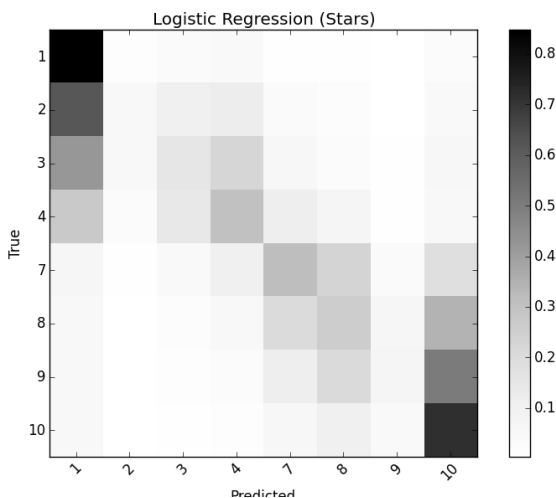


Fig 5.1.6. These results have the same problems as LinearSVC. There is a skew towards 1 & 10 stars and too much misclassification.

LinearSVR

Since this model outputs a continuous variable, it is unsuitable for the binary dataset. We tested the following parameters on the stars dataset:

loss: ['epsilon insensitive', 'squared_epsilon_insensitive']
 These loss functions ignore errors if they are within ϵ (epsilon) distance of the true value. $\sum 0$ if $|y_i - f(X_i)| \leq \epsilon$ else $|y_i - f(X_i)|$. The squared function squares each term.
 epsilon: [0.0001, 0.001, 0.01, 0.1, 1, 10, 100]
 The value of ϵ for the loss function.

C: [0.1, 1, 5, 10, 50, 100, 500, 1000]
Coefficient that is multiplied with the loss.

dual: [True, False]
Whether to solve the primal or dual form of the problem.

The results on the test set were:

Exact accuracy: 0.32412
Off-by-one accuracy: 0.67856
Off-by-two accuracy: 0.88612
Confusion matrix:
[[2565 965 784 451 189 50 8 10]
[815 521 450 307 142 54 11 2]
[547 524 614 531 222 77 19 7]
[301 411 640 650 408 178 38 9]
[7 41 141 320 593 630 382 193]
[3 15 77 265 547 783 618 542]
[4 10 40 144 376 581 544 645]
[3 28 67 245 602 1034 1187 1833]]

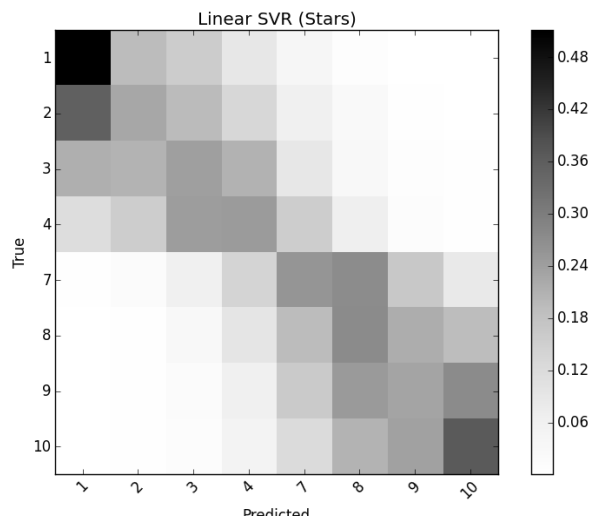


Fig 5.1.7. These results are much better than any of the previous ones. The confusion matrix shows a good separation of positive and negative sentiment and eliminates skew. But accuracy still needs to be improved.

Note: It seems surprising that the exact and off-by-one accuracy of Linear SVR is less than that of Logistic Regression and Linear SVC, despite the confusion matrix clearly showing that Linear SVR is superior.

1 & 10 star inputs each account for 20% of the dataset, with every other class at 10%. When we consider the skewed nature of Linear SVC and Logistic Regression's predictions, this explains why they are able to reach over 60% off-by-one accuracy despite results that are actually poor.

LinearRegression

This model is also unsuitable for the binary set. It does not have any parameters since it solves by simply performing a least squares minimization.

We obtained the following results on test data:

Exact accuracy: 0.32528

Off-by-one accuracy: 0.67944

Off-by-two accuracy: 0.88644

Confusion matrix:

```
[ [2495  973  805  474  198   56   11   10]
 [ 781  522  453  316  156   60   11    3]
 [ 509  499  642  532  247   83   21    8]
 [ 280  394  629  661  422  191   45   13]
 [    5   33  129  310  581  628  402  219]
 [    3   15   68  240  531  762  654  577]
 [    3   11   38  130  358  571  551  682]
 [    2   24   66  228  565 1007 1189 1918]]
```

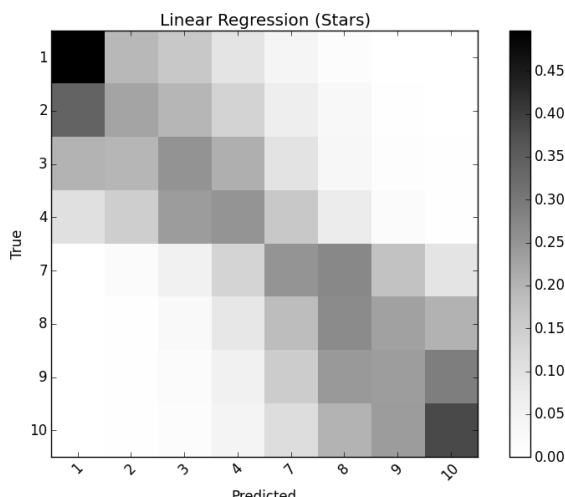


Fig. 5.1.8. While these results appear identical to LinearSVR, the confusion matrix shows slight improvements. Furthermore LinearRegression is much faster at both training and prediction. Accuracy still needs to be improved.

Once again, the exact and off-by-one accuracy is lower than that of Logistic Regression, but the confusion matrix shows that Linear Regression is superior. The reason is the same as stated in Linear SVR.

5.2 Conclusion

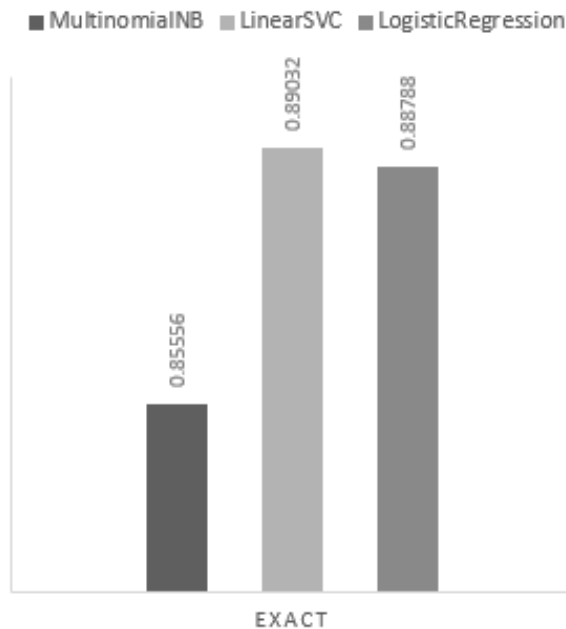


Fig.5.2.1 Accuracy comparison on the binary dataset

We reach the following conclusions:

1. Multinomial Naïve Bayes, Linear SVC, and Logistic Regression are all suitable for the binary classification problem. They are unsuitable for the stars classification problem.
2. **Linear SVC** offers the best results for binary classification at approximately **89%** accuracy.
3. Linear Regression and Linear SVR may be considered suitable for the stars classification if some room for error is given.
4. **Linear Regression** offers the better results for star classification at approximately **68%** accuracy, if off-by-one errors are permissible.
5. The results for the stars classification problem are not very satisfactory. While the positive/negative separation is maintained well, confusion within these categories is somewhat high.

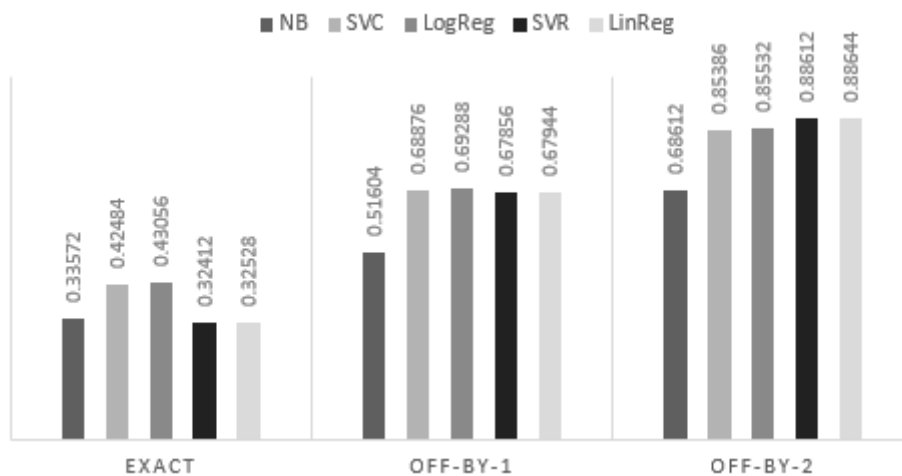


Fig. 5.2.2. Accuracy comparison on the stars dataset.

(See SVR results for an explanation of the unexpected accuracy in SVC and LogReg)

5.3 Future Work

With this project yielding unsatisfactory results, we have a lot of room for future work.

1. The primary concern is to increase the accuracy on the stars classification problem. Other, more complicated, machine learning models could be explored for the same.
 2. A greater reliance on preprocessing could also be explored. We have laid minimal reliance on preprocessing. Various options such as other scoring systems or other tokenizing strategies could be researched.
 3. On a tangent, it could be explored how effective these classifiers are when applied to reviews from other domains. Intuitively, such a large corpus should have a large enough vocabulary to make it effective for any domain, but this may not be true in practice.
 4. The size of classifier dumps needs to be reduced drastically to make usage on the web practical. The bloated size is mainly due to our usage of TfidfVectorizer, which stores the actual text of each one of the ngram features.
-

References

1. *ACL IMDB Dataset*: Potts, Christopher. 2011. On the negativity of negation. In Nan Li and David Lutz, eds., *Proceedings of Semantics and Linguistic Theory 20*, 636-659.
 2. *Scikit-learn: Machine Learning in Python*, Pedregosa *et al.*, *JMLR* 12, pp. 2825-2830, 2011.
 3. Domingos, P. (2012). A few useful things to know about machine learning. *Communications of the ACM*, 55(10), 78-87.
 4. *Predicting Star Rating based on Yelp User Review Text*:
<http://zhongyaonan.com/predict-rating-of-yelp-user-review-text/>
-