IMDB Sentiment Analysis

With Logistic Regression, K-Nearest Neighbours and Support Vector Machine Model

Taylan Tatlı

Physics PhD student

taylan.tatl@gmail.com

CMPE 544 Term Project Report

Department of Computer Engineering,
Boğaziçi University, 08/01/2023

Contents

1 Introduction

In this project, first the dataset of the Internet Movie Database (IMDb) movie reviews will be studied on. Dataset contains huge number of reviews (approximately 50000) those are both positive and negative reviews. The negative reviews have a score lower than rating point 4 out of 10, and positive reviews have a score higher than 7. The main goal of the project is to classify whether a review is positive or negative with using at least two different architecture from the following set: Suppor Vector Machine, Logistic Regression and Naive Bayes.

Dataset to be used :
`https://www.kaggle.com/code/lakshmi25npathi/sentiment-analysis-of-imdb-movie-reviews/data`

Main references: [1], [2].

2 Data Pre-Processing

- Data preprocessing is important to see what kind of data we have to deal with and for preparing those data to be implemented into the models. Our data is a text-based data.
- The first part of data pre-process aims at reducing the complexity by manipulating our text-based data, which enables to increase its generalization capabilities and so accuracy.
- Main steps of text manipulations are Contractions Expansion, Removing HTML Tags, Lower Case Conversion, Removing Stopwords, Removing Numbers, Removing Frequent Words, Word Tokenization, Lemmatization, Removing Punctuation.
- The other part of the data pre-process is vectorization.

```
In [2]: data = pd.read_csv("IMDB Dataset.csv")
        data["sentiment"]=data["sentiment"].map({"positive":1,"negative":0})
        data.head()
```

Out[2]:

	review	sentiment
0	One of the other reviewers has mentioned that ...	1
1	A wonderful little production. The...	1
2	I thought this was a wonderful way to spend ti...	1
3	Basically there's a family where a little boy ...	0
4	Petter Mattei's "Love in the Time of Money" is...	1

Figure 1: Raw IMDB review data. The column 'review' contains 50,000 non-null entries with a datatype object. The column 'sentiment' has 50,000 non-null entries with a datatype int64. The 'sentiment' column contains class sentiment for each review. If it is 0, then it is a negative review. If it is 1, then it is positive.

2.1 Text Manupulation

Text pre-processing is an important part of any problem as we have, Natural Language Problems. Many functions for text manipulation are created to reduce complexity as said above.The operations are explained shortly below:

2.1.1 Contractions Expansion:

This operation changes some contracted word to the open form. Such as: 'aren't' will be 'are not'. So every same meaning contracted term and open term will give us just one or two dimension. This operation is also required for that the other operations going well.

2.1.2 Removing HTML Tags:

This removes HTML tags like $< div >< /div >$ and $< br/ >$from the text. The HTML tags has no meaning for sentiment.

2.1.3 Word Tokenization:

This turns the large text into separate tokens, basically single words.

2.1.4 Lower Case Conversion:

This changes all the upper case characters into lower case. Upper and lower case don't make any difference for our model, "A" and "a" are the same thing for our model.

2.1.5 Removing Numbers:

This removes all the numbers in the text. Numbers do not play a significant role for the analysis of movie reviews.

2.1.6 Removing Stopwords:

This removes english language stopwords like 'me', 'our', 'where', 'what'.. etc. from the text.

2.1.7 Removing Frequent Words

This removes the most frequently occurring words in data, because frequent words don't provide any significant difference among the inputs, like the word 'movie'.

2.1.8 Lemmatization:

This enables using the base forms of words. For example, 'writing' will be 'write' 'cats' will be 'cat'.

2.1.9 Removing Punctuation:

This removes all the punctuation like !'(), from the text.

2.1.10 Result of Text Manipulation

After and before text manipulations, the data looks like below. It clearly provides us less complex data.

```
data.iloc[0]["review"]
```

```
"One of the other reviewers has mentioned that after watching just 1 Oz episode you'll be hooked. They are right,
as this is exactly what happened with me.<br /><br />The first thing that struck me about Oz was its brutality and
unflinching scenes of violence, which set in right from the word GO. Trust me, this is not a show for the faint he
arted or timid. This show pulls no punches with regards to drugs, sex or violence. Its is hardcore, in the classic
use of the word.<br /><br />It is called OZ as that is the nickname given to the Oswald Maximum Security State Pen
itentary. It focuses mainly on Emerald City, an experimental section of the prison where all the cells have glass
fronts and face inwards, so privacy is not high on the agenda. Em City is home to many..Aryans, Muslims, gangstas,
Latinos, Christians, Italians, Irish and more....so scuffles, death stares, dodgy dealings and shady agreements ar
e never far away.<br /><br />I would say the main appeal of the show is due to the fact that it goes where other s
hows wouldn't dare. Forget pretty pictures painted for mainstream audiences, forget charm, forget romance...OZ doe
sn't mess around. The first episode I ever saw struck me as so nasty it was surreal, I couldn't say I was ready fo
r it, but as I watched more, I developed a taste for Oz, and got accustomed to the high levels of graphic violenc
e. Not just violence, but injustice (crooked guards who'll be sold out for a nickel, inmates who'll kill on order
and get away with it, well mannered, middle class inmates being turned into prison bitches due to their lack of st
reet skills or prison experience) Watching Oz, you may become comfortable with what is uncomfortable viewing....th
ats if you can get in touch with your darker side."
```

Figure 2: The first element of data before text manipulation.

```
data.iloc[0]["review"]
```

```
'one reviewer mentioned watching oz episode hooked right exactly happened methe first thing struck oz brutality un
flinching scene violence set right word go trust show faint hearted timid show pull punch regard drug sex violence
hardcore classic use wordit called oz nickname given oswald maximum security state penitentary focus mainly emeral
d city experimental section prison cell glass front face inwards privacy high agenda em city home manyaryans musli
m gangsta latino christian italian irish moreso scuffle death stare dodgy dealing shady agreement never far awayi
would say main appeal show due fact go show would dare forget pretty picture painted mainstream audience forget ch
arm forget romanceoz mess around first episode ever saw struck nasty surreal could say ready watched developed tas
te oz got accustomed high level graphic violence violence injustice crooked guard whofll sold nickel inmate whofll
kill order get away well mannered middle class inmate turned prison bitch due lack street skill prison experience
watching oz may become comfortable uncomfortable viewingthats get touch darker side'
```

Figure 3: The first element of data after text manipulation.

2.2 Vectorization (Feature Extraction)

There is no model that can understand the text-based data, so we need to transform our text-based data into the for of that the models can understand. There are two main approches for this, which are Count Vectorizer and Term Frequency Inverse Document Frequency(TF-IDF) Vectorizer. Count Vectorizer gives number of frequency with respect to index of vocabulary. On the other hand, Tf-Idf consider overall documents of weight of words. For more details you can visit page [3] [4]. Count vectorizer has some disadvantages as follows: inability in identifying more important and less important words for the analysis, consideration of words that are found mostly in a corpus as the most statistically significant word. On the other side, Tf-Idf is a statistic that is based on the frequency of a word in the corpus. It also provides a numerical representation of how important a word is for statistical analysis. TF-IDF is better than Count Vectorizers since it considers both the frequency of words present and the importance of the words in the text which provides us working with less input dimension and important words. After the vectorization(Feature Extraction) part, without any dimension reduction of Tf-Idf, so meaning just changing words into vectors without considering how important these words are, we have very high-dimensional data. Train shape is (40000, 208350).

```
# Vectorization and Train-Test Split
X = data['review']
y = data['sentiment']
tfidf = TfidfVectorizer()#max_features=10) --> No Dimensionality Reduction for now.
X = tfidf.fit_transform(X)
#y=tfidf.fit_transform(y)
x_train, x_test, y_train, y_test = train_test_split(X, y, test_size = 0.2, random_state=42)
```

We see very high dimensional case!!!

```
x_train.shape,x_test.shape,y_train.shape,y_test.shape
```

```
((40000, 208350), (10000, 208350), (40000,), (10000,))
```

Figure 4: Outputs of feature extraction (vectorization) part.

3 Model Implementation using Scikit Learn Libraries

In this section. Logistic Regression, K-Nearest Neighbours(KNN) and Support Vector Machine (SVM) models are created and implemented using Scikit-Learn Libraries. No dimension reduction or tunnin hyperparemeters are performed.

3.1 Test and Train Accuracy for Each Model

- **Logistic Regression Result**
  - Test Accuracy : 0.894
  - Train Accuracy: 0.932475
- **KNN Result**
  - k=3 : Test Accuracy: 0.7625 Train Accuracy: 0.882975
  - k=5 : Test Accuracy: 0.7725 Train Accuracy: 0.856125
  - k=10 : Test Accuracy: 0.7907 Train Accuracy: 0.8368
- **SVM Result**
  - No Result in approximately 1 hour!!

Model	**Test Accuracy**	**Train Accuracy**
Logistic Regression	0.894	0.932475
KNN k=3	0.7625	0.85995
KNN k=5	0.7358	0.856125
KNN k=10	0.7907	0.8368
SVM	none	none

There is an overfitting problem here !! Test accuracy's are bigger than train accuracy's. It is not a big overfitting problem and also logistic regression looks like perform satisfactory but still better model performance without overfitting and with high accuracy's are possible.

The value of k in the KNN algorithm is related to the error rate of the model. A small value of k could lead to overfitting and on the other side big value of k can lead to

underfitting. Also high dimensionality can lead to weak performance of model. That's why Logistic model could perform an overfitting. So dimension reduction can be help to perform better and also it can help SVM to converge in an reasonable time. More dimension means more time for all models but, for SVM, it is significantly important.

3.2 Overfitting and High Dimension for Models

3.2.1 Logistic Regression

- For this case, LogisticRegression from Scikit learn is used (sklearn.linear_model.Logistic Regression deoesnt use Stochastic Gradient Descent algorithm.) It uses regularization and so it should be resistant to be overfit the data. But because of high dimensionality, the model may not perform so well and also in our case we observe this slightly.
- The model wants to predict y=1 or y=0. It can be done by drawing a line through data points so that y=1 points are at the right side and y=0 points are at the left side of the line or vice versa. If all points are separated like that then it is called perfect separation.
- To do separation, regression weight must be set as large as possible so that model could fit the data but software has some limits so instead of this, it iteratively tries higher and higher.
- The more predictors you have (the higher the dimensionality), the more likely it is that it will be possible to perfectly separate the two sets of values. As a result, overfitting becomes more of an issue [5].

3.2.2 KNN Model

- The kNN algorithm assumes similar items are near each other. It classifies a data point by considering its nearest neighbors.
- In high dimensional case, the kNN algorithm should expect some important difficulties to perform efficiently. The main one is to compute distance among data points and so to find the nearest neighbors and this, is getting harder and more

complex as the dimension meaning feature space increases. The more dimension the more time of model performance and less accurate results.

- Also, one of the inefficiency of performance is overfiting.
- The only parameter that can be adjusted for kNN is the number of negihbors (k) parameter. The larger k means the smooter decision boundary, and the less kNN complexity. However, difficult to set the most optimal value of k. Very high k can lead underfitting and low ones can lead overfitting. Choosing a large k leads to high bias by classifying everything as the more probable class. As a result, we have a smooth decision boundary between the classes, with less emphasis on data points. Small k values leads to high variance. Minor changes in the training set cause large changes in classification boundaries [6].

3.2.3 SVM Model

- The SVM approach is to map data to higher dimensional space than the dataset has, to achieve better separability. That's why, dimension of dataset is mainly important for SVM algorithms since it effect directly. Thats why, first case effects SVM time so much.
- SVM is normally overfitting resistant, since it uses regularization term like Logistic Regression model we used before.
- So , SVMs should be highly resistant to over-fitting, but in practice this depends on the careful choice of C (strength parameter of regularization penalty term.) and the kernel parameters. Over-fitting can also occur easily when tuning the hyper-parameters.

4 Dimensionality Reduction and Hyperparemeter Optimization for Models

- In this part again Scikit Learn Library will be used for model implementation.
- First Dimensionality Reduction teqniques are used and then model implementation and after Hyperparemeter Tunning.
- At the end, results containing comparisons.

4.1 Dimensionality Reduction

Three kind of techniques are used to reduce dimensions. Which are following:

- Tf-Idf basic dimension reduction by setting max_feature parameter.
- Using Principal Component Analysis(PCA)
- And very simple way, by reducing data samples.

,

4.1.1 Basic Dimension Reduction for Tf-Idf

Tf-Idf is a numerical statistic that reflects how important a word is to a document in a collection. It increases proportionally to the number of times a word appears in the document and weighting the word importance it creates a feature space. The purpose of max_features is to limit the number of features (words) from the dataset for which we want to calculate the TF-IDF scores. This is done by choosing the features based on term frequency across the corpus. When we set max_feauture=50 it means take top 50 important feature and so it reduces dimension of dataset. First, dimension is reducet to 300 by setting max_feature=300

```
from sklearn.feature_extraction.text import TfidfVectorizer
X = data['review']
y = data['sentiment']
tfidf = TfidfVectorizer(max_features=300)
X = tfidf.fit_transform(X)
```

```
X.shape,y.shape
```

```
((50000, 300), (50000,))
```

```
x_train, x_test, y_train, y_test = train_test_split(X, y, test_size = 0.2, random_state=42)
```

Figure 5: Train test split with Tf-Idf basic dimension reduction.

With an simple complexity reduction by directly reducing feauture number using paremeter max_feature of Tf-Idf Vectorizer as explained above. Our training input x_train.shape : (40000, 300) and result of each model results are shon in Table 1.

Table 1: Train and Test accuracies, with Tf-Idf max_feature=300.

Model	**Test Accuracy**	**Train Accuracy**
Logistic Regression	0.8131	0.816875
KNN k=3	0.7293	0.85995
KNN k=5	0.7358	0.830275
KNN k=10	0.7614	0.81395
SVM	0.885	0.976

Lets compare with result of without any dimensionality reduction. Dimensionality reduction leads to models perform faster, SVM converges but there is an overfitting problem even though test accuracy results are satifactory. Just for Logistic Regression model, we can say there is no overfitting, but test score decreased at this time. For KNN, almost no change, little bit worse scores but reduced overfitting.

4.1.2 Principal Component Analysis and Tf-Idf

PCA is a very common technique used for dimensional reduction while keeping the patterns in the data. The goal of PCA is to maximize the variance. PCA can help only reducing dimensionality. This is realized by linearly transforming the data into a new coordinate system. In this new coordinate system, most of the variation in the data are described with fewer dimensions than the initial data. It uses a Singular Value Decomposition of the data to transform the data into a lower dimension space, principal components. These principal components summarize the information of the original features, which are linear combinations of the original variables [7].

```
vectorizer = TfidfVectorizer(max_features=5000) # for default Unable to allocate 77.6 GiB for an array wit
reviews = list(data['review'])                  # so set max_feature
labels = data['sentiment']

tfidf_reviews = vectorizer.fit_transform(reviews)
```

```
tfidf_array = tfidf_reviews.toarray()
print("Shape of the array:",tfidf_array.shape)
```

```
Shape of the array: (50000, 5000)
```

```
print("Percentage of zeros:",
      np.count_nonzero(tfidf_array==0)/(tfidf_array.shape[0]*tfidf_array.shape[1])*100)
#only 0.078% of the array elements are non zero. Such a waste of space!
```

```
Percentage of zeros: 98.4765016
```

```
from sklearn.decomposition import PCA
NUM_COMPONENTS = 300
pca = PCA(NUM_COMPONENTS)
reduced = pca.fit_transform(tfidf_array)
```

```
reduced.shape
```

```
(50000, 300)
```

```
x_train, x_test, y_train, y_test = train_test_split(reduced, labels, test_size = 0.2, random_state=42)
```

Figure 6: Train test split with Tf-Idf and PCA.

PCA keep patterns of vectorizer data after Tf-Idf applied so better result expected. Simple dimension reduction of Tf-Idf with max_features=300 gives us most of the important pattern. But what if the computer capacity, hardware spesifications can allow so much lower dimension which leads to loss of important patterns. If this was tthe case then PCA would create a dramaitc change since it keeps patterns. It would possible to keep patterns of max_feature=300 and working dimension much lower. But still, for our case it is expected that dimensionality reduction with PCA can be better.

Model implementations example and results are given below, after PCA operation on Tf-Idf vector space, in Figure 7 and Table 2 recursively.

```
logistic_model = LogisticRegression(max_iter=200)
logistic_model.fit(x_train, y_train)
logistic_model.score(x_test,y_test),logistic_model.score(x_train,y_train)
```

```
(0.8692, 0.87235)
```

```
knn=KNeighborsClassifier(n_neighbors=10)
knn.fit(x_train,y_train)
#y_pred=knn.predict(x_test)
#confusion_matrix(y_test,y_pred)
knn.score(x_test,y_test), knn.score(x_train,y_train)
```

```
(0.714, 0.783025)
```

```
from sklearn import svm
clf = svm.SVC()
clf.fit(x_train, y_train)
#y_pred = clf.predict(x_test)
#confusion_matrix(y_test,y_pred)
clf.score(x_train,y_train),clf.score(x_test,y_test)
```

```
(0.94865, 0.8787)
```

Figure 7: An example of basic model implementations after PCA.

Table 2: Train and Test accuracies, with PCA

Model	**Test Accuracy**	**Train Accuracy**
Logistic Regression	0.8692	0.87235
KNN k=10	0.714	0.783025
SVM	0.8787	0.94865

If we compare the results in Table 1, it looks like the results in Table 2 more satisfactory, it has narrower range between train and test scores and higher test scores for the models except KNN k=10 case.

4.1.3 Very Simple Way:Reducing Training Samples

Reducing training samples is tried so that it can prevent model to learn excessively. The half of the samples are taken as an input and then PCA applied on it. Codes and input

train shape are shown in Figure 8. It converges faster as expected but it didn't give better results, train and test accuracy's are shown in Table 3.

```
from sklearn.feature_extraction.text import TfidfVectorizer
reviews = data['review'].iloc[:25000]
label = data['sentiment'].iloc[:25000]
tfidf = TfidfVectorizer(max_features=5000)
tfidf_reviews = tfidf.fit_transform(reviews)
```

```
tfidf_array = tfidf_reviews.toarray()
print("Shape of the array:",tfidf_array.shape)
```

```
Shape of the array: (25000, 5000)
```

```
from sklearn.decomposition import PCA
NUM_COMPONENTS = 300
pca = PCA(NUM_COMPONENTS)
reduced = pca.fit_transform(tfidf_array)
```

```
x_train, x_test, y_train, y_test = train_test_split(reduced, label, test_size = 0.2, random_state=42)
```

```
x_train.shape, x_test.shape,y_train.shape
```

```
((20000, 300), (5000, 300), (20000,))
```

Figure 8: Reducing data samples to half and PCA applied.

Table 3: Train and Test accuracies, with PCA

Model	**Test Accuracy**	**Train Accuracy**
Logistic Regression	0.8598	0.87155
KNN k=10	0.707	0.78825
SVM	0.8632	0.9531

5 Hyperparemeter Optimizationor Tuning for Models

Hyperparameter tuning (or hyperparameter optimization) is the process of determining the right combination of hyperparameters that maximizes the model performance. For each model different parameters are considered. Data samples are reduced to get result faster. Hyperparemeters are applied to normal dataset after all hyperparameter optimization is finished.

- Logistic Regression: Different types of regularization, dual formulation, maximum iteration number and C values are considered.
- KNN : k number is considered.
- SVM : Kernel Functions, C values and Gamma parameter are considered.

5.1 Logistic Regression Hyperparameter Tunning

Regulurization is important to prune overfitting, as default Scikit Learn Logistic model has regulurization. So changing regularization types and parameters can effect model performance. Also dual formulation and maximum number of iteration are considered.

- penalty'l1', 'l2', 'elasticnet', None, default='l2' Specify the norm of the penalty:
  - None: no penalty is added;
  - 'l2': add a L2 penalty term and it is the default choice;
  - 'l1': add a L1 penalty term;
  - 'elasticnet': both L1 and L2 penalty terms are added.
- C, default=1.0 Inverse of regularization strength; must be a positive float. Like in support vector machines, smaller values specify stronger regularization.
- dualbool, default=False Dual or primal formulation. Dual formulation is only implemented for l2 penalty with liblinear solver. Prefer dual=False when n_samples ¿ n_features.
- max_iter : maximum number of iteration.

Results of changing regularization types and dual formulation change is given in Figure 9. Almost no change! Also maximum number of iterations parameter doesn't effect for between 50 and 300 iteration number. The codes of max_iter optmization given in Figure 10 to show how to optimize parameters generally. For the other methods, optmizations are handled in a similar way and the result is given Figure 11.

```
In [55]:
logistic_model = LogisticRegression(max_iter=200, C=1000) # default l2 ridge regurulization
logistic_model.fit(x_train, y_train)
first = logistic_model.score(x_test,y_test),logistic_model.score(x_train,y_train)

logistic_model = LogisticRegression(max_iter=200, C=1000000) #make weaker penalty term
logistic_model.fit(x_train, y_train)
second = logistic_model.score(x_test,y_test),logistic_model.score(x_train,y_train)

logistic_model = LogisticRegression(max_iter=200, penalty="none") # no penalty term
logistic_model.fit(x_train, y_train)
third = logistic_model.score(x_test,y_test),logistic_model.score(x_train,y_train)

logistic_model = LogisticRegression(max_iter=200, penalty="l1",solver="liblinear") # with L1 regularization
logistic_model.fit(x_train, y_train)
fourth = logistic_model.score(x_test,y_test),logistic_model.score(x_train,y_train)

first,second,third,fourth

Out[55]: ((0.8616, 0.87645), (0.8616, 0.87645), (0.8616, 0.87645), (0.8616, 0.87225))

In [56]:
# With Dual Formulation
logistic_model = LogisticRegression(max_iter=200, penalty="l2",solver="liblinear",dual=True)
logistic_model.fit(x_train, y_train)
logistic_model.score(x_test,y_test),logistic_model.score(x_train,y_train)

Out[56]: (0.8598, 0.87155)
```

Figure 9: Reducing data samples to half and PCA applied.

```
def optimize_max_iter(upper,lower):
    test_score=[]
    train_score=[]
    iterations=[]
    optimum_iter=0
    key_score=0
    for i in range(upper,lower):
        if i%20==0:
            logistic_model = LogisticRegression(max_iter=i)
            logistic_model.fit(x_train, y_train)
            test_accuracy=logistic_model.score(x_test,y_test)
            train_accuracy=logistic_model.score(x_train,y_train)
            test_score.append(logistic_model.score(x_test,y_test))
            train_score.append(logistic_model.score(x_train,y_train))
            iterations.append(i)
            if test_accuracy>key_score:
                optimum_iter=i
                key_score=test_accuracy
        else:
            continue
    train_test_frame = pd.DataFrame({"Test Scores": test_score,"Train Scores":train_score})
    #train_test_frame.plot()
    Main_frame = pd.DataFrame({"Iterations":iterations,"Test Scores": test_score,"Train Scores":train_score})
    plt.plot(iterations,test_score,label="Test Scores")
    plt.plot(iterations,train_score,label="Train Scores")
    plt.xlabel="Iterations"
    plt.ylabel="Accuracies"
    print("Optimum Score:",key_score,"The Iteration Number:",optimum_iter)
    return Main_frame,key_score,optimum_iter
```

Figure 10: Code example for parameter optmization. Model is logistic regression.

After that c values are optimized and the resul is given in Figure 12. The best one is C=10.

	Iterations	Test Scores	Train Scores
0	60	0.8346	0.8472
1	80	0.8346	0.8472
2	100	0.8346	0.8472
3	120	0.8346	0.8472
4	140	0.8346	0.8472
5	160	0.8346	0.8472
6	180	0.8346	0.8472
7	200	0.8346	0.8472
8	220	0.8346	0.8472
9	240	0.8346	0.8472
10	260	0.8346	0.8472
11	280	0.8346	0.8472

Figure 11: Results of max_iter optmization.

	C_Values	Test Scores	Train Scores
3	10.00	0.834	0.87425
4	1.00	0.834	0.86325
0	100000.00	0.825	0.87725
1	1000.00	0.825	0.87775
2	100.00	0.825	0.87650
5	0.10	0.813	0.83950
6	0.01	0.787	0.81300

Figure 12: Results of C values optmization.

5.2 KNN Hyperparemeter Optimization

- To work faster, less data is used (1/10 of samples).
- Choosing a large k value leads to high bias by classifying everything as the more probable class. As a result, we have a smooth decision boundary between the classes, with less emphasis on individual points. Large k can lead to underfitting.
- Conversely, small k values cause high variance and an unstable output. Minor changes in the training set cause large changes in classification boundaries. Small k can lead to overfitting.

The result is given in Figure 13.

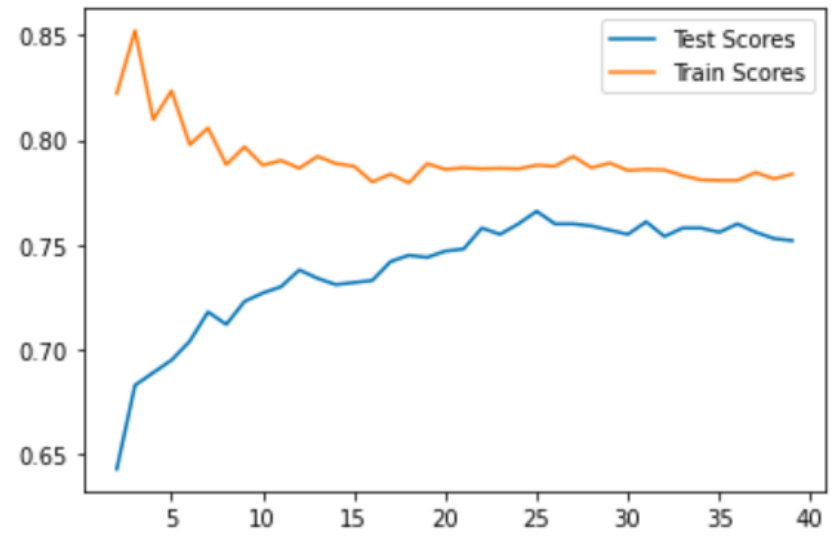

Figure 13: Results of k values optimization.

5.3 SVM Hyperparemeter Optmization

- Kernels: The main function of the kernel is to take low dimensional input space and transform it into a higher-dimensional space.
- C (Regularisation): C is the penalty parameter, inverse of regularization strength; must be a positive float.
- Gamma: It defines how far influences the calculation of plausible line of separation. when gamma is higher, nearby points will have high influence.
- To work faster, less data is used (1/10 of samples).

Tunning Hyperparemeters for SVM is handled with using GridSearch method of Scikit Learn Library. Code example is given in Figure 14.

```
In [83]: from sklearn.model_selection import GridSearchCV
         parameters = {'C': [0.1,1, 10, 100], 'gamma': [1,0.1,0.01,0.001],'kernel': ['rbf', 'poly', 'sigmoid']}

In [85]: #from sklearn.svm import SVC
         grid = GridSearchCV(clf,parameters,refit=True,verbose=2)
         grid.fit(x_train,y_train)
         Fitting 5 folds for each of 48 candidates, totalling 240 fits
         [CV] END .....................C=0.1, gamma=1, kernel=rbf; total time=   2.6s
         [CV] END .....................C=0.1, gamma=1, kernel=rbf; total time=   2.6s
         [CV] END .....................C=0.1, gamma=1, kernel=rbf; total time=   2.6s
         [CV] END .....................C=0.1, gamma=1, kernel=rbf; total time=   2.6s
```

Figure 14: SVM hyperparemeter optimization.

Result is optmized parameters are SVC(C=1, gamma=1,kernel=rbf), rbf is default.

5.4 Results after Hyperparemeter Optimizations

First Tf-Idf is applied with max_features=5000, and after PCA applied with number of components=300 and finally hyperparemetersare applied. Code is given in Figure 15. Results are given in Table 4.

Table 4: Train and Test accuracies, with PCA

Model	**Test Accuracy**	**Train Accuracy**
Logistic Regression	0.8703	0.874375
KNN k=25	0.7511	0.7885
SVM	0.8736	0.889075

The best result are obtained with hyperparemeter optmiziations. Almost no overfitting!! Satisfactory test scores for esspecially SVM and Logistic Regression.

6 Logistic Regression with Hand-Coded Algorithm

In this part, a very simple Logistic Regression model using Gradient Descent algorithm is created step by step by hand. It is mainly based on the matrix multiplication so data dimensionality reduced, 1/50 of the samples are used. **I couldnt do handcoded part without any reference, for a long time i tried but always it fails in a way.** This handcoded study is mainly inspired from following notebook:

`https://www.kaggle.com/code/kanncaa1/deep-learning-tutorial-for-beginners`

As a vectorizer, count vectorizer is used since we want each word as a vector giving only 1 or 0. Data sample is given in Figure 14, every column represent a word. What is done basically?

- First initialize parameters, weights and bias.
- Do matrix multiplication with x_train, meaning multiply each feature with an initialized weight and add bias term
- Put all matrix multiplication values into sigmoid function. Sigmoid function gives probabilistic value. If values ¿ 0.5 then it says positive, lower than ¡ 0.5 says negative.
- Compare with true y_train values and find the cost using loss function.
- Update weight and bias so that the costs will decrease.
- Updating parameters part need so much work, defining loss function, finding derivative, determining learning rate, iterations etc..
- Do this steps with a specified number of iterations.
- Create a training function to use all together and train your model with your input data.
- Create a predict function to see how your model work, calculate accuracy.

For the steps for creating model, many functions are written. All is given below. The result is given in the Figure 16. Train accuracy 0.91, test accuracy:0.81

	0	1	2	3	4	5	6	7	8	9	...	19098	19099	19100	19101	19102	19103	19104	19105	19106	19107
0	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	0
2	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	0
3	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	0
4	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	0
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
995	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	0
996	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	0
997	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	0
998	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	0
999	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	0

1000 rows × 19108 columns

Figure 15: Data for model. 1000 reviews. If reviews contains a word it gives 1 in that word column.

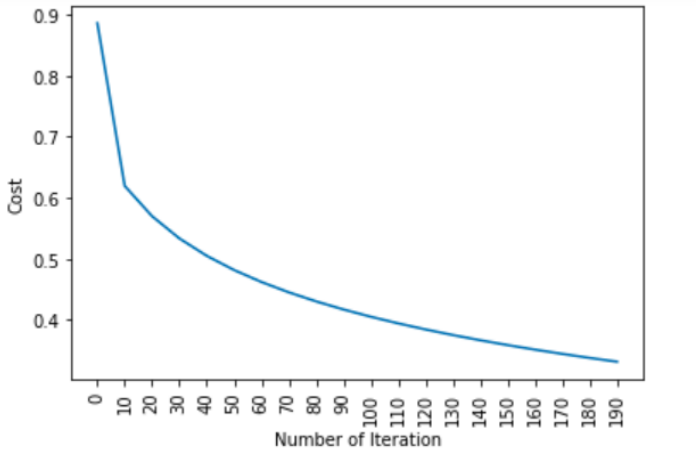

Figure 16: Hand-coded result

```python
def initialize_weight_and_bias(dimension):
    #dimension is the number of features. We need the same number of weights as the features are.
    w = np.full((dimension,1),0.01)
    b=0.0
    return w,b
```

```python
def sigmoid(z):
    y_head = 1/(1+np.exp(-z))
    return y_head
```

```python
def loss_cost(y_head,y_train):
    loss = -y_train*np.log(y_head)-(1-y_train)*np.log(1-y_head)
    cost = (np.sum(loss))/x_train.shape[1]      # x_train.shape[1]  is for scaling
    return cost
```

```python
def forward_propagate(w,b,x_train,y_train):
    z = np.dot(w.T,x_train)+b
    y_head = sigmoid(z)
    cost = loss_cost(y_head,y_train)
    return cost,y_head,z
```

```python
def calculate_gradients_after_forward(w,b,x_train,y_train):# BACKWARD PROPAGATION
    cost,y_head,z = forward_propagate(w,b,x_train,y_train)

    derivative_weight = (np.dot(x_train,((y_head-y_train).T)))/x_train.shape[1] # x_train.shape[1]
    derivative_bias = np.sum(y_head-y_train)/x_train.shape[1]                 # x_train.shape[1]  i
    gradients = {"derivative_weight": derivative_weight,"derivative_bias": derivative_bias}
    return cost,gradients
```

```python
def update_wegihts_bias(w,b,x_train,y_train,learning_rate):
    cost,gradients=calculate_gradients_after_forward(w,b,x_train,y_train)
    # lets update
    w = w - learning_rate * gradients["derivative_weight"]
    b = b - learning_rate * gradients["derivative_bias"]
    return w,b,cost
```

```
def do_recursively(w,b,x_train,y_train,learning_rate,num_of_iteration):
    cost_list=[]
    cost_list2=[]
    index = []
    for i in range(num_of_iteration):
        w,b,cost = update_wegihts_bias(w,b,x_train,y_train,learning_rate)
        cost_list.append(cost)
        if i % 10 == 0:
            cost_list2.append(cost)
            index.append(i)
            print ("Cost after iteration %i: %f" %(i, cost))

    parameters = {"weight": w,"bias": b}
    plt.plot(index,cost_list2)
    plt.xticks(index,rotation='vertical')
    plt.xlabel("Number of Iteration")
    plt.ylabel("Cost")
    plt.show()
    return parameters,cost_list
```

```
 # prediction
def predict(w,b,x_test):

    z = sigmoid(np.dot(w.T,x_test)+b)
    Y_prediction = np.zeros((1,x_test.shape[1]))

    for i in range(z.shape[1]):
        if z[0,i]<= 0.5:
            Y_prediction[0,i] = 0
        else:
            Y_prediction[0,i] = 1

    return Y_prediction
```

```
def logistic_regression(x_train, y_train, x_test, y_test, learning_rate, num_iterations):

    dimension =  x_train.shape[0]  # that is 19000
    w,b = initialize_weight_and_bias(dimension) #initialize_weights_and_bias(dimension)

    parameters, cost_list = do_recursively(w, b, x_train, y_train, learning_rate,num_iterations)

    y_prediction_test = predict(parameters["weight"],parameters["bias"],x_test)
    y_prediction_train = predict(parameters["weight"],parameters["bias"],x_train)


    print("train accuracy: {} %".format(100 - np.mean(np.abs(y_prediction_train - y_train)) * 100))
    print("test accuracy: {} %".format(100 - np.mean(np.abs(y_prediction_test - y_test)) * 100))

logistic_regression(x_train, y_train, x_test, y_test,learning_rate = 0.1, num_iterations = 200)
```

References

[1] N. G. Ramadhan and T. I. Ramadhan, "Analysis sentiment based on imdb aspects from movie reviews using svm," *Sinkron : jurnal dan penelitian teknik informatika*, vol. 7, pp. 39–45, Jan. 2022.

[2] L. Dey, S. Chakraborty, A. Biswas, B. Bose, and S. Tiwari, "Sentiment analysis of review datasets using naïve bayes' and k-nn classifier," *International Journal of Information Engineering and Electronic Business*, vol. 8, pp. 54–62, 07 2016.

[3] M. Chaudhary, "Tf-idf vectorizer scikit-learn." `https://medium.com/`

@cmukesh8688/tf-idf-vectorizer-scikit-learn-dbc0244a911a.

[4] Sklearn, "sklearn.feature_extraction.text.tfidfvectorizer." https://scikitlearn.org/stable/modules/generated/sklearn.feature_extraction.text.TfidfVectorizer.html.

[5] "Why does logistic regression overfit in high-dimensions?." http://eointravers.com/post/logistic-overfit/.

[6] "k-nearest neighbors and high dimensional data." https://www.baeldung.com/cs/k-nearest-neighbors.

[7] "Working with high dimensional data." https://medium.com/working-with-high-dimensional-data/working-with-high-dimensional-data-9e556b07cf99.