Final Assignment – Natural Language Processing & Machine Learning

Student number: 20324412

Module: CS7CS4/CSU44061

Part 1.1 Prediction of 'review polarity' from 'review text'

To determine if the review text (X) could be used to predict the review polarity (y), the data was preprocessed, a support vector classifier (svc) and a k nearest neighbour (knn) classifier were fit to the data and the subsequent predictive performance was inspected.

Data processing & feature extraction

The review texts were firstly translated to English using the function *google_translator* from the library *google_trans_new*, this was applied as follows;

```
X['text'] = X['text'].apply(translator.translate, lang_tgt='en')
```

The pipeline method from sklearn.pipeline was then implemented to apply various natural language processing (nlp) transforms. The pipeline sequentially applies a list of transforms and a final estimator. A `CountVectorizer()` was applied to convert the list of review texts into a sparse matrix of token (word) counts, similar to one-hot encoding. The stop-words were removed as they are considered to contribute little predictive power to the model. A `TfidfTransformer()` was then applied. This stands for *term frequency times inverse document frequency* and is used to reduce the weightage of more common words. Instead of using the raw frequencies of occurrence of a token in a given document (a given review in this case e.g X[1]) it scales down the impact of tokens that occur very frequently in a given corpus and that are therefore less informative than features that occur in a small fraction of the training corpus which may have a higher predictive power for a specific class. The classifier is the last step applied in the pipeline, be it either the svc or knn classifier. For example, the pipeline for the svc classifier was follows;

```
model = Pipeline( [('vect', CountVectorizer(stop_words =
nltnltk.corpus.stopwords.words('english'))), ('tfidf', TfidfTransformer()),
('clf', LinearSVC())])
```

To determine the optimal hyper-parameters of the model 5-Fold cross validation was used.

Hyperparameters – Choice of C in SVC via Cross Validation

The optimal value of the penalty term C in the svc model was determined in the range (0.001, 1000). Decreasing the value of C places more importance on the penalty. The optimal value was determined based on the value of C that corresponded to the greatest F1 score as shown in Figure 1. The F1 score is the harmonic mean of the precision and the recall and so is a good overall metric for comparison. The plots were obtained using the function `SVM_choose_c` and 5-fold cross validation was used to train and test the model. As shown in Figure 1 the F1 score increases dramatically as C is increased from 0.001, reaching a peak at C = 1 of 0.835 from which it decreases slightly and remains at a relatively constant level. Thus for the final model, the penalty term was left at the default value of 1.

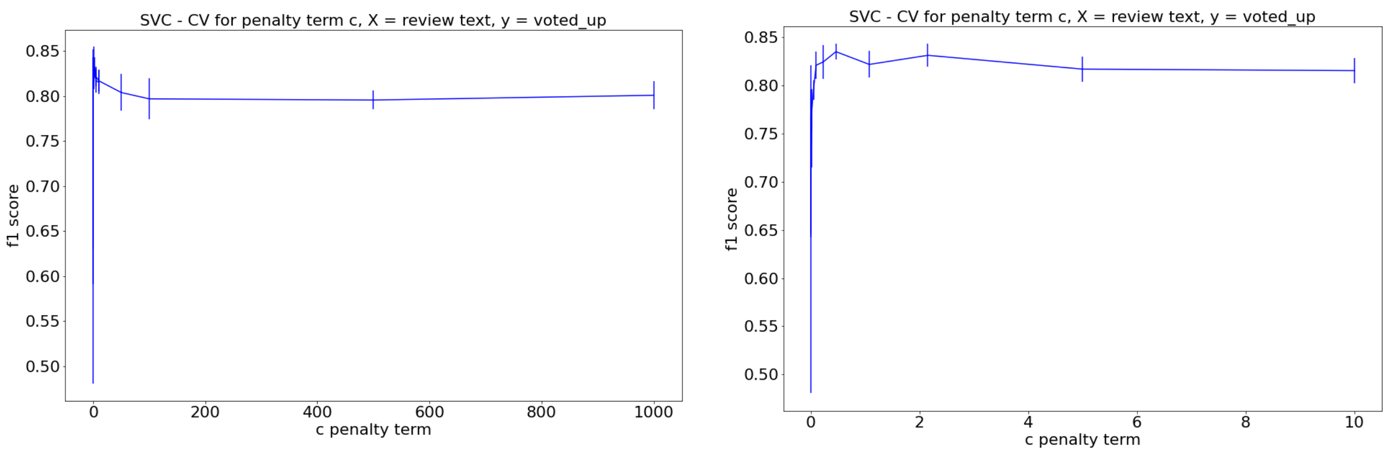

Figure 1: Penalty term C in SVC vs corresponding F1 score for C in the range (0.001, 1000) on the left and (0.001, 10) on the right. Model results were obtained using 5-fold cross validation.

Function – choice of c in SVC

```
def SVC_choose_c(X, y, c_range, num_splits, y_label, plot_color):

    '''Support Vector Classification - Implement 5 fold cross validation to
determinine optimal C'''

    #Param setup

    kf = KFold(n_splits = num_splits, shuffle = True)

    mean_f1 =[]; std_f1 =[];

    X = np.array(X); y = np.array(y)

    #Loop through each value of c

    for c_param in c_range

        f1_temp = []

        model = Pipeline([('vect', CountVectorizer(stop_words =
nltk.corpus.stopwords.words('english'))),

                          ('tfidf', TfidfTransformer()),

                          ('clf', LinearSVC(C = c_param))])

        for train_index, test_index in kf.split(X):

            model.fit(X[train_index], y[train_index])

            ypred = model.predict(X[test_index])

            f1_temp.append(f1_score(y[test_index],ypred))

        #Get mean & variance
        mean_f1.append(np.array(f1_temp).mean())

        std_f1.append(np.array(f1_temp).std())

    #Plot

    #plt.figure()

    plt.errorbar(c_range, mean_f1, yerr=std_f1, color = plot_color)

    plt.xlabel('c penalty term'); plt.ylabel('f1 score')

    plt.title('SVC - CV for penalty term c, X = review text, y = 
{}'.format(y_label))

    plt.savefig('./svm_{}'.format(y_label)); plt.show()
```

Hyperparameters – Choice of k in knn via Cross Validation

The value of k, i.e the number of neighbours in the knn model, was chosen in a similar manner via cross validation in the range (2, 300). The F1 score increases as the number of neighbours k is increased towards 150 where it reaches a peak (see figure 2) and subsequently declines and so 150 was chosen as the optimal value. This was achieved with the function `knn_choose_c` similar to the function `SVC_choose_c` above albeit with the model defined as follows;

```
Model = Pipeline([('vect', CountVectorizer(stop_words = 
nltk.corpus.stopwords.words('english'))),('tfidf', TfidfTransformer()),
('clf', KNeighborsClassifier(n_neighbors = k, weights= 'uniform'))])
```

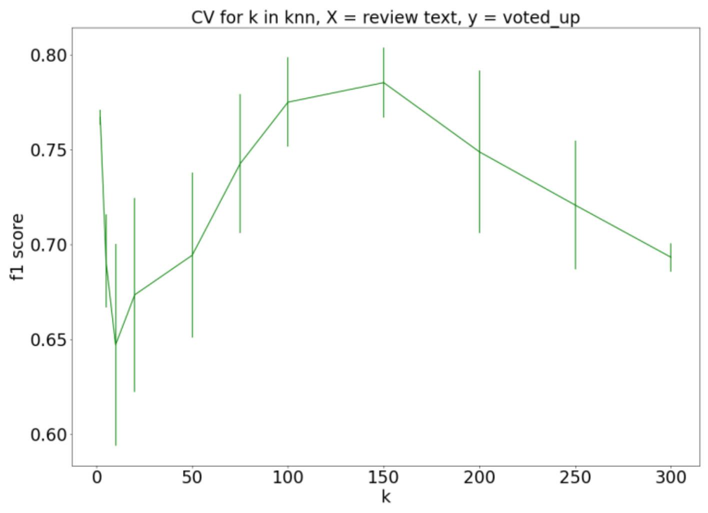

Figure 2. knn - p*lot of the number of neighbours k vs the F1 score obtained via cross validation.*

Grid search to determine optimal natural language transform

A grid search was used to determine the optimal variations of the nlp transforms; whether to use the tfidf transform aswell as the optimal *ngram* range. An n-gram is a sequence of n words, i.e a 2-gram (bigram) is a two-word sequence. In the grid search, unigram and bigrams were tested. It transpired that the best performing model was one that used the transform tfidf aswell as bigrams. The grid search is as below, which was fit to both the svc and knn model (svc model shown below)

```
parameters = {'vect__ngram_range': [(1, 1), (1, 2)], 'tfidf__use_idf':
(True, False)}

#Grid Search

svm_gs = GridSearchCV(svm_model, parameters, n_jobs=-1)
```

These optimal models were then fit to a train set (training size = 67%) and tested on the test set (test size = 33%) and the performance metrics were obtained. This was achieved using the function `optimal_models_performance` (see Appendix). For example, the svc was fit as follows and the resulting performance metrics iand ROC curve were determined. The same approach was used to assess the knn model as well as a baseline model or dummy classifier that always predicts the most common class of y the review polarity.

Code to train & test the optimal svc model (found within the function `optimal_models_performance`*)*

```
Xtrain, Xtest, ytrain, ytest = train_test_split(X, y, test_size= testSizeX,
random_state=42)

svm_gs = svm_gs.fit(Xtrain, ytrain)

print(svm_gs.best_params_)

predicted = svm_gs.predict(Xtest)

print(confusion_matrix(ytest, predicted))

print(classification_report(ytest, predicted))

#ROC plots

scores = svm_gs.decision_function(Xtest)

fpr, tpr, _= roc_curve(ytest, scores)

plt.plot(fpr,tpr, label = 'SVM')

print('SVM AUC = {}'.format(auc(fpr, tpr)))
```

Performance Metrics

The confusion matrices for the models are shown in Table 1, the performance metrics obtained from the classification report are shown below in Table 2 and the ROC plots are shown Figure 3.

Confusion Matrices

Table 1: Confusion matrices of the three models for X = review text, y = review polarity.

SVM			**Knn**			**Baseline model**		
	Predicted			**Predicted**			**Predicted**	
True	1 (Voted Up)	0	**True**	1 (Voted Up)	0	**True**	1 (Voted Up)	0
1 (Voted Up)	750	100	1 (Voted Up)	650	200	1 (Voted Up)	0	850
0	147	653	0	201	599	0	0	800

Classification Report

Table 2. Performance metrics of the three models for X = review text, y = review polarity.

y	**Model**	**Precision**	**Recall**	**F1 score**	**Accuracy**	**AUC**
X Voted up	SVM	0.83 False, 0.86 True	0.80 False, 0.83 True	0.82 False, 0.85 True	0.815	0.85
	Knn	0.75 False, 0.73 True	0.72 False, 0.75 True	0.73 False, 0.77 True	0.74	0.80
	Dummy (most freq)	0.48 False, 0.00 True	1.0 False, 0.00 True	0.65 False, 0.00 True	0.48	0.5

ROC curves

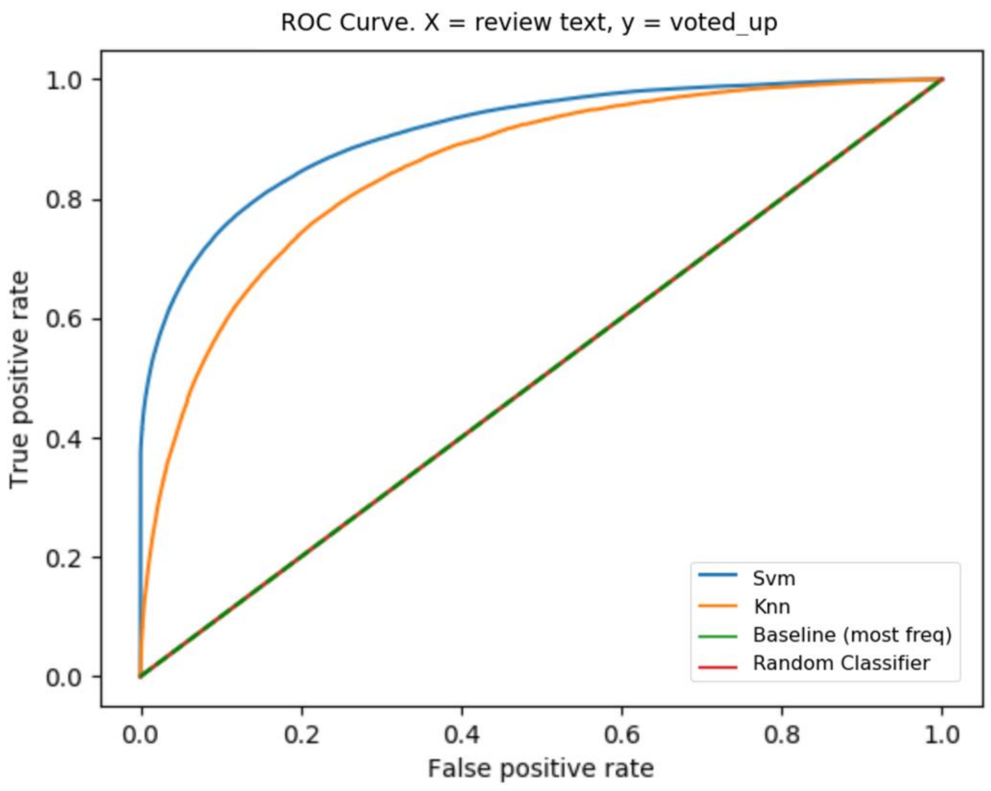

Figure 3. ROC Curve of all models for X = review text, y = review polarity.

Results – Discuss

The classifier models were able to predict the review polarity (y) from the review text (X). Both the SVC and knn model performed better than a dummy classifier baseline model which naively picked the most common class. Overall, the SVC was deemed the best performing model according to the performance metrics. It has the best receiver operator curve (ROC) as shown in Figure 3. A perfect classifier (100% true positives, 0% false positives) would produce a point in the top-left corner of the ROC plot and the SVC plot is closest to this point. Corresponding to this it's AUC is highest at 0.85, compared to the knn model at 0.8. Both models perform significantly better than the random and baseline classifiers, i.e at the optimal threshold, their True Positive Rate is higher and their False Positive rate is lower than the baseline models. The latter models cannot discriminate between classes and predict a random class in the case of the former and a constant class (the majority class) for all observations in the case of the latter. Their ROC Curves thus lie on the 45 degree line, which can be interpreted as the 'flip a coin' line and aligning with this the model produces an AUC of 0.5. (See Table 2)

These results also correspond with the confusion matrices in Table 1. It can be quickly observed that the logistic regression model and the SVM model are performing well, given that the diagonal entries are both high, showing the models' abilities to detect the true positives and true negatives.

Conversely, the off-diagonals are lower which shows that there is a low number of false positives and false negatives.

A classification report of all models was generated using sklearn which outputs the precision, recall, f1 score and accuracy as shown in Table 2. Again based on these results the SVC model performs best. It's F1 score, the harmonic mean of the precision and recall is in the range [0.82 - 0.85], compared to [0.73-0.77] for the knn model. As described the dummy classifier predicts the majority class for all cases, i.e that the review was not voted up (y = False). Thus it has a recall of 100% for the negative class and 0% for the positive class.

Part 1.2. Prediction of 'early access' from 'review text'

It was then determined whether the review text (X) could be used to predict the review polarity whether the review was for an "early access" version of a game or not (z). The same framework and functions were used as in part one; the data was preprocessed and transformed using the pipeline, a support vector classifier (svc) and a k nearest neighbour (knn) model were fit to the data and the subsequent predictive performance was inspected.

Hyperparameters – Choice of C in SVC via Cross Validation

The optimal value of the penalty term C in the SVM classifier was determined in the range (0.001, 1000) again using 5 fold cross validation (See Figure 4). The F1 score decreases significantly as C is increased from 1 towards 1000 and for the final model the penalty term was left at the default value of 1.

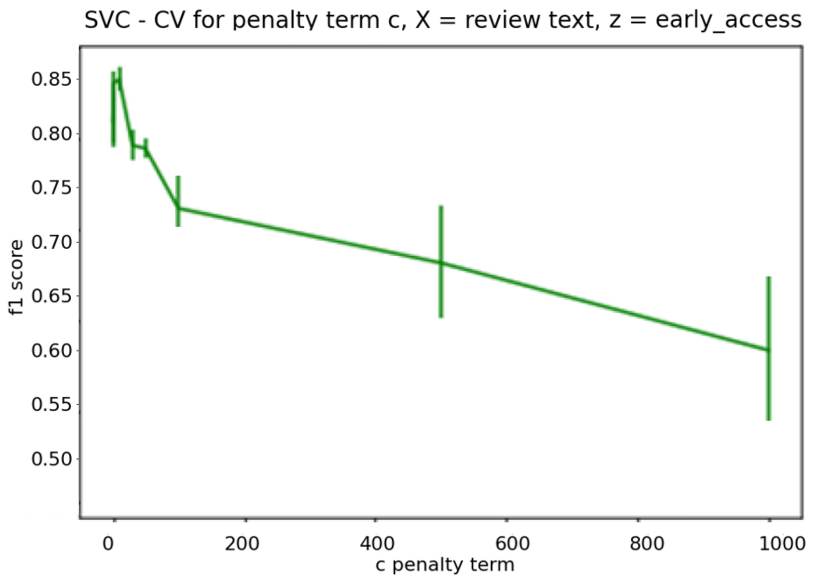

Figure 4: Penalty term C in SVC vs corresponding F1 score for C in the range (0.001, 1000).

Hyperparameters – Choice of k in knn via Cross Validation

The value of k was again chosen via cross validation in the range (2, 300). The F1 score increases as the number of neighbours k is increased towards 150 where it reaches a peak (Figure 5) and subsequently declines and so 150 was chosen as the optimal value.

Grid search to determine optimal natural language transform

A grid search was again used to determine the optimal variations of the nlp transforms and again the best performing model was one that used the tfidf transform aswell as bigrams.

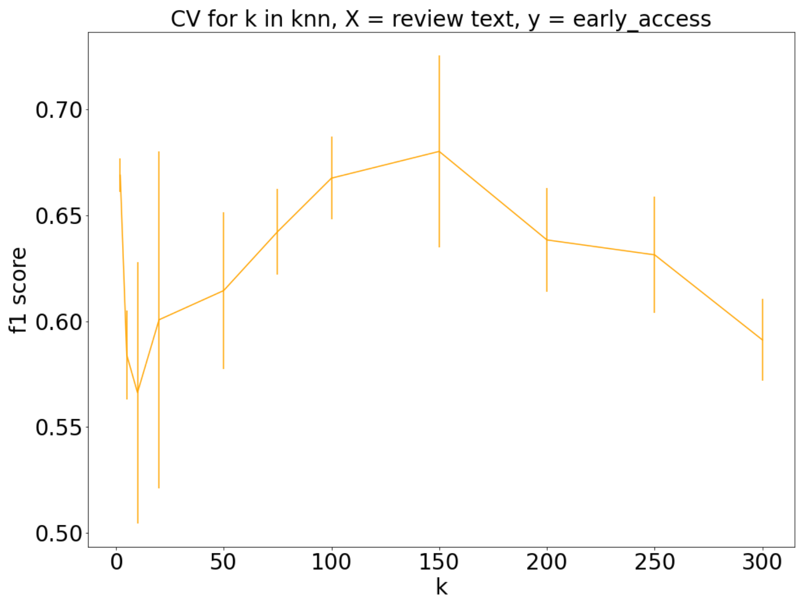

Figure 5. knn - plot of the number of neighbours k vs the F1 score obtained via cross validation for X = review text, z = early access.

Performance Metrics

The confusion matrices of the optimal models are shown in Table 3, the performance metrics are shown below in Table 4 and the ROC plots are shown Figure 4.

Confusion Matrices

Table 3: Confusion matrices of the three models for X = review text, z = 'early access'

SVM			**Knn**			**Baseline model**		
	Predicted			**Predicted**			**Predicted**	
True	1 (early access)	0	**True**	1 (early access)	0	**True**	1 (early access)	0
1 (early access)	117	51	1 (early access)	90	78	1 (early access)	0	168
0	21	1461	0	31	1451	0	0	1482

Classification Report

Table 4: Performance metrics for the three models for X = review text, z = 'early access'

Y label	**Model**	**Precision**	**Recall**	**F1 score**	**Accuracy**	**AUC**
Voted up	SVM	0.96 False, 0.85 True	0.98 False, 0.69 True	0.97 False, 0.76 True	0.92	0.77
	Knn	0.95 False, 0.74 True	0.97 False, 0.54 True	0.96 False, 0.64 True	0.90	0.69
	Dummy (most freq)	0.9 False, 0.00 True	1.0 False, 0.00 True	0.95 False, 0.00 True	0.90	0.5

ROC curves

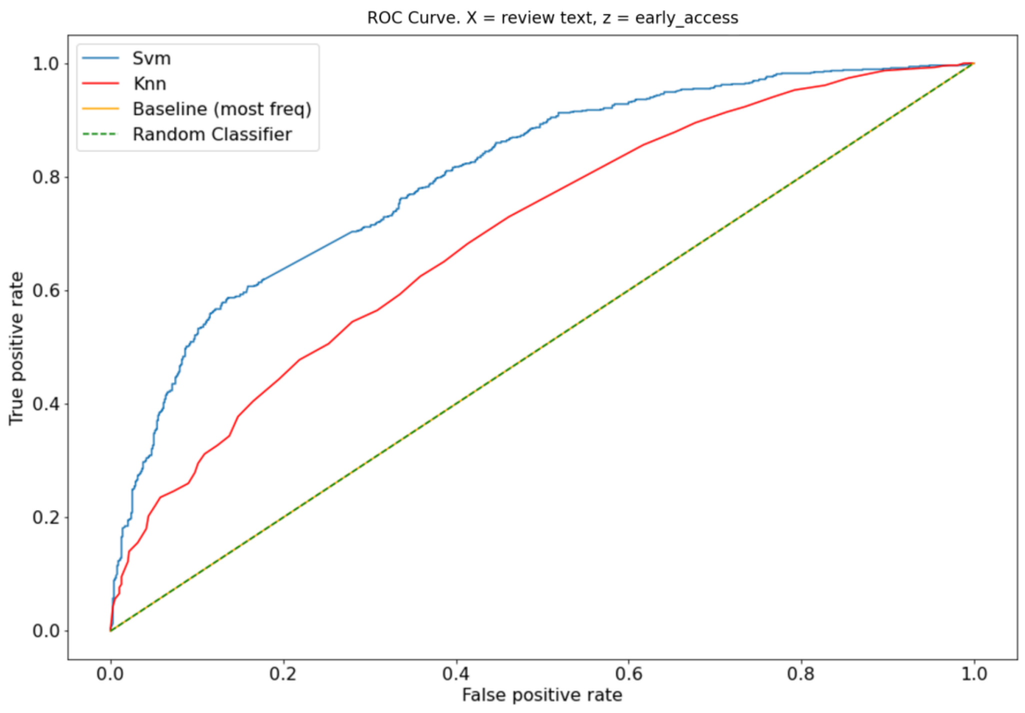

Figure 6. ROC Curve of all models for X = review text, z = early access

Results – Discuss

The classifier models were able to predict the early access label (z) from the review text (X) up to a certain degree. The data were highly imbalanced in that of the 5000 data points only 10.5% of the reviews corresponded to an early or 'beta' version of the game, while 89.5% did not. There was therefore less signal for the models to discern the positive class and as a result both the SVC classifier and the knn model performed relatively poorly on the positive class in comparison to the negative class. However given the imbalance in the dataset, the accuracy results are high, 0.92 for the SVC and 0.90 for the knn model, as they are calculated across all observations. This can be misleading in regard to the model's predictive capability and so it is necessary to look at the individual class results aswell. The SVC and knn model had an F1 score of 0.97 and 0.96 for the negative class respectively, reducing to 0.76 and 0.64 for the positive class. The recall was particularly poor for the positive class, i.e the model's ability to detect positive cases; 0.69 for the SVC and 0.54 for the knn model. These results brought down the overall F1 score. Again the SVC was the best performing model according to the performance metrics. It has the best receiver operator curve (ROC) as shown in Figure 3 and the highest AUC at 0.77, in comparison to the knn model (AUC; 0.7) and the dummy classifier (AUC; 0.5).

Part 2

Q I. Under-fitting & Over-fitting

Under fitting refers to when a predictive model fails to capture the underlying trend of the data, usually occurring if the model is too simple. The model performs poorly on both the training data and generalizes poorly to unseen data. Such a model often shows low variance but high bias. An example of under fitting would be if a linear model was fit to non-linear data (see Figure I). Such a model would tend to have poor predictive performance.

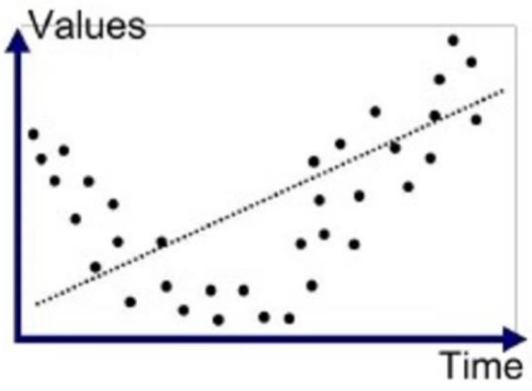

Figure I. Example of underfitting – a linear model fit to non-linear data.

Overfitting occurs when a predictive model captures the noise in the data and fits the training data too precisely. This often occurs if the model is too complex. As a result the model provides a very good fit of the training data but can perform poorly on unseen data given that it has captured the noise in the specific training set. Such a model often shows low bias but high variance. An example of over fitting would be if a 3rd order polynomial model was fit to data with an underlying linear trend (see Figure II). Such a model would also tend to have poor predictive performance.

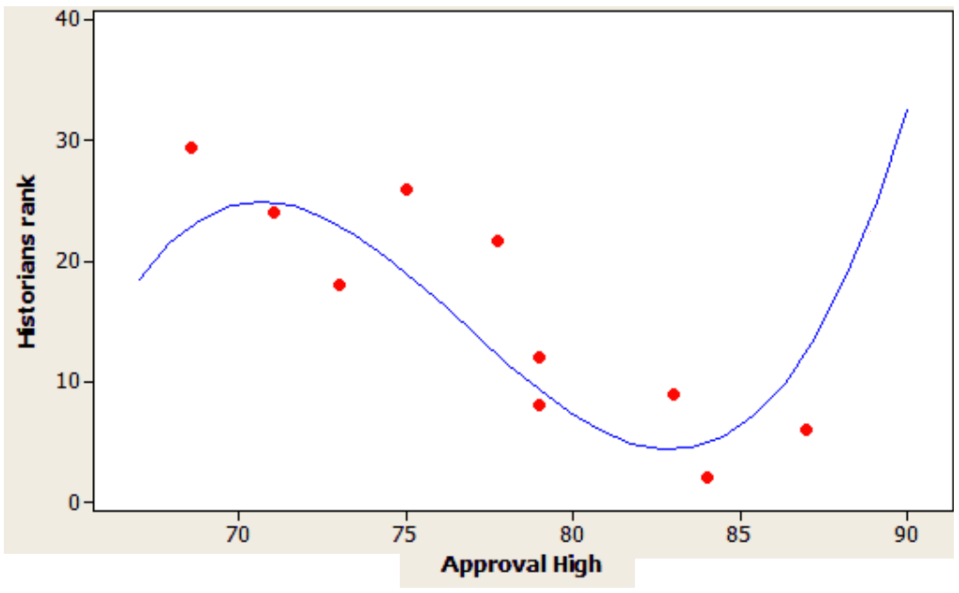

Figure II. Example of overfitting – a 3rd order polynomial model fit to data with a generally linear trend

Q II. K-fold cross-validation pseudocode

K-fold cross-validation is a resampling procedure used to evaluate machine learning models on a limited data sample. The procedure has a single parameter called k that refers to the number of groups that a given data sample is to be split into. For example k = 5 would result in 5-fold cross validation which is a common choice. Pseudocode is as below;

Pseudocode k-fold cross validation

```
For i in range(k):
        Randomly partition the dataset into:
        train_set(i)
        test_set(i)
        Train model on train_set(i)
        Predict on test_set(i)
        Calculate train and test scores:
        train_score(i)
        test_score(i)
```

Once the procedure has been repeated for each i up to k the overall mean train and test scores are calculated;

Mean_train_score = $\frac{1}{k}\sum_{i=1}^{k} train\_score(i)$

Mean_test_score = $\frac{1}{k}\sum_{i=1}^{k} test\_score(i)$

The mean scores give an estimate of the model's overall predictive performance.

Q III. k fold cross validation for hyperparameter selection to balance under- and over-fitting

K-fold cross validation involves partitioning a sample of data into complementary subsets, fitting the model to the training subset, and validating the model on the test set. Thus it can be used to train and test a model corresponding to a certain value of a model hyperparameter and the corresponding predictive performance can then be inspected. K-fold cross validation can then be repeated for a range of values of the hyperparameter and the results can then be compared to determine what is the optimal value of the hyperparameter. The method can be used to strike a balance between over- and under-fitting by visually inspecting the results and determining which value of the hyperparameter achieves the best trade-off between the training and test performance. Overfitting would occur if the model achieves a high training score and a significantly lower test score. This would occur if the model is overfitting to the specific training set and so it fails to generalise well on the unseen test set. Underfitting would occur if both the training and test predictive performance is poor as the model fails to capture the underlying trend in the data. The optimal model hyperparameter choice would have both a modestly good training and test score. In this case the model has not overfit to the training set and so it is able to generalise well to the unseen test data and so this value of the hyperparmeter would present as a good choice.

Q IV. Pros & Cons of a logistic regression classifier vs a kNN classifier.

Logistic regression is a statistical model that uses a logistic function to model a binary dependent variable. Advantages of a logistic regression classifier over a knn classifier include that;

i. Logistic Regression (lr) is an easy, fast and straight-forward classification method. Hyperparameter tuning is not critical to train a logistic regression model and so the default model can be used. This is in contrast to a kNN classifier in which choosing the number of neighbours 'k' is critical. To choose the optimal value of k, hyperparameter tuning would be required using cross validation for example. The distance metric used in the knn model must also be selected (e.g euclidean distance, manhattan distance etc). The value of the metric can change the predicted outcome and so there can be variability across expected results. Proper scaling of the features is necessary to ensure fair treatment across features. In addition lr has a significantly faster computation time than knn which can have a large computation cost especially if the sample size is large. This is in part due to the fact that it is a distance-based algorithm and so the cost of calculating distance between a new point and each existing point is high.
ii. Logistic regression is a parametric model whereas knn is a non-parametric model. In the former the model parameters θ explain the direction of association and intensity of the effect of the independent variables on the dependent variable which cannot be achieved with a knn. The lr model coefficients can be interpreted as indicators of feature importance. This can help with model explainability for the end user of the logistic regression model.
iii. Logistic Regression outputs probabilities along with classification results which allows a confidence level about the prediction to be derived. This is an advantage over the knn classifier which only gives the final classification label. For example, if a training case has a 90% probability for a class, and another has a 55% probability for the same class, it can be inferred which training examples are more accurate for the formulated problem.

Some disadvantages of a logistic regression classifier over a knn classifier include that;

i. Logistic regression cannot be applied to non-linear classification problems, it can only learn a linear decision boundary. Linearly separable data is rarely found in real world situations. So the transformation of non linear features is required which can be done by increasing the number of features such that the data becomes linearly separable in higher dimensions. This is in contrast to a knn classifier which can learn non-linear boundaries and so it is advantageous in that regard.

ii. There are several assumptions that the underlying data must meet in order for the logistic regression model to be valid. These include independence of errors, absence of multicollinearity, and a lack of strongly influential outliers. Colinearity amongst predictors can tamper the accuracy of the lr model. Outliers can also skew the results. Significantly less assumptions need to be met for the knn classifier to be used and so there is less of a barrier for use across a range of datasets which is advantageous.

iii. Logistic regression requires training a model on a training set while no training is required for a knn model which can save time.

Q V. Two examples of situations when a kNN classifier would give inaccurate predictions

A knn classifer would give inaccurate predictions in certain situations such as the two following cases;

i. If the data is randomly spread out and no obvious groupings of observations emerge. In this case there is no underlying trend between the features in metric space and the dependent variable and no useful information can be obtained from it. When a specific case is queried, the knn classifier will try to find the k nearest neighbours but since the data points are arbitrarily distributed in the metric space, the k nearest neighbours may not be an accurate representation of the true label of the query resulting in inaccurate predictions.

ii. A knn classifer could also give inaccurate predictions if the chosen value of k, the number of nearest neighbours is too large. If there is a large number of nearest neighbours, trends within smaller sub groups of the data would be missed. When a specific case is queried, the knn classifier would classify it as coming from the class of all k nearest neighbours, however it may belong to the class of a much smaller subset of it’s nearest neighbours that would only be detected if k was much smaller. Thus too large a value of k can lead to inaccurate predictions.

Appendix – Code

```
# -*- coding: utf-8 -*-

"""

Created on Tue Dec 22 22:12:19 2020


@author: Hannah Craddock

"""


#******************************


#Data

import numpy as np

import matplotlib.pyplot as plt

import pandas as pd

import json_lines

import nltk

from google_trans_new import google_translator #from googletrans import
Translator

nltk.download('stopwords')

from sklearn.pipeline import Pipeline

from sklearn.feature_extraction.text import CountVectorizer,
TfidfTransformer

from sklearn.svm import LinearSVC

from sklearn.linear_model import LogisticRegression

from sklearn.model_selection import KFold, train_test_split

from sklearn.metrics import confusion_matrix, classification_report,
f1_score, roc_curve, auc

from sklearn.dummy import DummyClassifier

from sklearn.neighbors import KNeighborsClassifier

from sklearn.model_selection import GridSearchCV

from google_trans_new import google_translator
```

```
#Plot
%matplotlib qt
SMALL_SIZE = 8
MEDIUM_SIZE = 28
BIGGER_SIZE = 30

plt.rc('font', size=MEDIUM_SIZE);
plt.rc('axes', titlesize=MEDIUM_SIZE)     # fontsize of the axes title
plt.rc('axes', labelsize=MEDIUM_SIZE)    # fontsize of the x and y labels
plt.rc('xtick', labelsize=MEDIUM_SIZE)    # fontsize of the tick labels
plt.rc('ytick', labelsize=MEDIUM_SIZE)    # fontsize of the tick labels
plt.rc('legend', fontsize=MEDIUM_SIZE)    # legend fontsize
plt.rc('figure', titlesize=BIGGER_SIZE)  # fontsize of the figure title


#*************************
#Data
X_orig =[] ; y=[] ; z=[]
x_label = 'text'
y_label ='voted_up'
z_label = 'early_access'

with open ( 'reviews_228.jl' , 'rb') as f :
    for item in json_lines.reader(f):
        X_orig.append(item[x_label])
        y.append(item[y_label])
        z.append(item[z_label])

#Data preprocessing - Translate data to english
```

```
translator = google_translator()

X_df = pd.DataFrame()

X_df['text'] = X_orig

X_df['translated'] = X_orig

#Translate column

X_df['translated'] = X_df['text'].apply(translator.translate,
lang_tgt='en') #.apply(getattr, args=('text',))

#values

X = list(df['translated'].values)


#***************************************************

#Model 1 - SVM


#Use k fold cross validation to use optimal value of penalty term c

def SVM_choose_c(X, y, c_range, num_splits, y_label, plot_color):

    '''Support Vector Classification - Implement 5 fold cross validation to
determinine optimal C'''


    #Param setup

    kf = KFold(n_splits = num_splits, shuffle = True)

    mean_f1 =[]; std_f1 =[];

    X = np.array(X)

    y = np.array(y)


    #Loop through each k fold

    for c_param in c_range:

        print('c = {}'.format(c_param))

        f1_temp = []


        model = Pipeline([('vect', CountVectorizer(stop_words =
nltk.corpus.stopwords.words('english'))),

                          ('tfidf', TfidfTransformer()),
                          ('clf', LinearSVC(C = c_param))])

        for train_index, test_index in kf.split(X):

            model.fit(X[train_index], y[train_index])
            ypred = model.predict(X[test_index])
            f1_temp.append(f1_score(y[test_index],ypred))

        #Get mean & variance
        mean_f1.append(np.array(f1_temp).mean())
        std_f1.append(np.array(f1_temp).std())

    #Plot
    #plt.figure()
    plt.errorbar(c_range, mean_f1, yerr=std_f1, color = plot_color)
    plt.xlabel('c penalty term')
    plt.ylabel('f1 score')
    plt.title('SVC - CV for penalty term c, X = review text, y = 
{}'.format(y_label))
    plt.savefig('./svm_{}'.format(y_label))
    plt.show()

#***********************************
#Model 2 - Knn

def knn_choose_k(X, y, k_range, num_splits, y_label, plot_color):
    '''knn - Implement 5 fold cross validation for determinine optimal k'''

    #Param setup
    kf = KFold(n_splits = num_splits, shuffle = True)
    mean_f1 =[]; std_f1 =[];
    X = np.array(X)
    y = np.array(y)

    #Loop through each k fold
    for k in k_range:
        print('k = {}'.format(k))
        f1_temp = []

        model = Pipeline([('vect', CountVectorizer(stop_words =
nltk.corpus.stopwords.words('english'))),
                          ('tfidf', TfidfTransformer()),
                          ('clf', KNeighborsClassifier(n_neighbors = k,
weights= 'uniform'))])

        for train_index, test_index in kf.split(X):

            model.fit(X[train_index], y[train_index])
            ypred = model.predict(X[test_index])
            f1_temp.append(f1_score(y[test_index],ypred))

        #Get mean & variance
        mean_f1.append(np.array(f1_temp).mean())
        std_f1.append(np.array(f1_temp).std())

    #Plot
    plt.figure(figsize = (12, 10))
    plt.errorbar(k_range, mean_f1, yerr=std_f1, color = plot_color)
    plt.xlabel('k')
    plt.ylabel('f1 score')
    plt.title('CV for k in knn, X = review text, y = {}'.format(y_label))
    plt.savefig('./knn_{}'.format(y_label))
    plt.show()


#**********************************

#Determine optimal model performance

def optimal_models_performance (X, y, optimal_k, optimal_C, y_label):

    ''' Grid search for optimal nlp classifier models (svm and knn). Plot
ROC curves, generate confusion matrices and classification report '''


    #1. Split the data

    testSizeX = 0.33 #67:33 split

    Xtrain, Xtest, ytrain, ytest = train_test_split(X, y, test_size=
testSizeX, random_state=42)


    #SVM

    svm_model = Pipeline([('vect', CountVectorizer(stop_words =
nltk.corpus.stopwords.words('english'))),

                      ('tfidf', TfidfTransformer()),

                      ('clf', LinearSVC())])


    #Knn

    knn_model = Pipeline([('vect', CountVectorizer(stop_words =
nltk.corpus.stopwords.words('english'))),

                      ('tfidf', TfidfTransformer()),

                      ('clf', KNeighborsClassifier(n_neighbors = optimal_k,
weights= 'uniform'))])


    #Dummy classifier

    dummy_model = DummyClassifier(strategy='most_frequent').fit(Xtrain,
ytrain)



    #Grid search

    parameters = {'vect__ngram_range': [(1, 1), (1, 2)],
                  'tfidf__use_idf': (True, False)}

    #************************************************

    #Svm: Train svm model

    svm_gs = GridSearchCV(svm_model, parameters, n_jobs=-1)


    #Performance - best performing

    print('*********************************************')

    print('====== \n Results for svm grid search model:')


    svm_gs = svm_gs.fit(Xtrain, ytrain)

    print(svm_gs.best_params_)

    predicted = svm_gs.predict(Xtest)


    print(confusion_matrix(ytest, predicted))

    print(classification_report(ytest, predicted))


    #************************************************

    #Train knn model

    knn_gs = GridSearchCV(knn_model, parameters, n_jobs=-1)


    #Performance - best performing

    print('*********************************************')

    print('====== \n Results for knn grid search model:')


    knn_gs = knn_gs.fit(Xtrain, ytrain)

    print(knn_gs.best_params_)

    predicted = knn_gs.predict(Xtest)


    print(confusion_matrix(ytest, predicted))
    print(classification_report(ytest, predicted))

    #********************************************

    #Dummy model

    print('********************************************')

    print('====== \n Results for dummy model:')

    dummy_model_fitted = dummy_model.fit(Xtrain, ytrain)

    predicted = dummy_model_fitted.predict(Xtest)

    print(confusion_matrix(ytest, predicted))

    print(classification_report(ytest, predicted))

    #********************************************

    #ROC plots

    plt.figure()

    #svm model

    scores = svm_gs.decision_function(Xtest)

    fpr, tpr, _= roc_curve(ytest, scores)

    plt.plot(fpr,tpr, label = 'SVM')

    print('SVM AUC = {}'.format(auc(fpr, tpr)))

    #knn model

    scores = knn_gs.predict_proba(Xtest)[:,1]

    fpr, tpr, _= roc_curve(ytest, scores)

    plt.plot(fpr,tpr, color = 'r', label = 'knn')

    print('knn AUC = {}'.format(auc(fpr, tpr)))

    #Baseline Model
```

```
    scores_bl = dummy_model_fitted.predict_proba(Xtest)

    fpr, tpr, _= roc_curve(ytest, scores_bl[:, 1])

    plt.plot(fpr,tpr, color = 'orange', label = 'baseline model')

    print('AUC = {}'.format(auc(fpr, tpr)))


    #Random Choice

    plt.plot([0, 1], [0, 1],'g--')


    #Labels

    plt.xlabel('False positive rate')

    plt.ylabel('True positive rate')

    plt.title('ROC Curve. X = review text, y = {}'.format(y_label))


    plt.legend(['Svm', 'Knn', 'Baseline (most freq)','Random Classifier'])

    plt.savefig('./roc_{}'.format(y_label))

    plt.show()



#**********************************************************

#Apply Functions


#**************************

#Part 1 y = voted up


#Split x and y into train and test sets for subsequent model fitting

testSizeX = 0.33 #67:33 split

Xtrain, Xtest, ytrain, ytest = train_test_split(X, y, test_size= testSizeX,
random_state=42)


#i. SVM - choose C

c_range = np.geomspace(0.001, 1000, num = 7)
c_range = np.concatenate((c_range, c_range/2))

c_range = np.sort(c_range)

#c_range = np.linspace(start=0.001, stop=1000, num=15)


num_splits = 5

plot_color = 'blue'

SVM_choose_c(X, y, c_range, num_splits, y_label, plot_color)


#Repeat (smaller range)

c_range = np.geomspace(0.001, 10, num = 7)

c_range = np.concatenate((c_range, c_range/2))

c_range = np.sort(c_range)


SVM_choose_c(X, y, c_range, num_splits, y_label, plot_color)


#ii. knn - Choose k

k_range = [2,5,10,20, 50, 75, 100, 150, 200, 250, 300]

plot_color = 'green'

knn_choose_k(X, y, k_range, num_splits, y_label, plot_color)


#iii.Grid search - optimal model

optimal_k = 150

optimal_c = 1

optimal_models_performance (X, y, optimal_k, optimal_c, y_label)


#***************************#***************************#*********************

#Part 2. X = review test, z = early access

#Split x and y ('z') into train and test sets for subsequent model fitting

testSizeX = 0.33 #67:33 split

Xtrain, Xtest, ztrain, ztest = train_test_split(X, z, test_size= testSizeX,
random_state=42)
```

```
#i. SVM - choose C

c_range = np.geomspace(0.001, 1000, num = 10)

c_range = np.concatenate((c_range, c_range/2))

c_range = np.sort(c_range)


num_splits = 5

plot_color = 'orange'

SVM_choose_c(X, z, c_range, num_splits, z_label, plot_color)


#ii. knn - Choose k

k_range = [2,5,10,20, 50, 75, 100, 150, 200, 250, 300]

plot_color = 'green'

knn_choose_k(X, z, k_range, num_splits, z_label, plot_color)


#Repeat

k_range = [2,3,4,6,8,10,12,15]

knn_choose_k(X, z, k_range, num_splits, z_label, plot_color)


#iii.Grid search - optimal model

optimal_k = 150

optimal_c = 50

optimal_models_performance(Xtrain, ytrain, Xtest, ytest, optimal_k,
optimal_c, z_label)
```