1.
(1)Read into python using json_lines.reader the file can be opened in text or binary mode .
(2) Data Preprocess.
a)Clean the data with expressions.
This makes reviews being able to use regular expressions. So I define a function to clean the data and apply it to the reviews. (By the way, when I finish the codes and come back to compare with the program without clean I got the same results, so this can be skipped.)

```
def preprocess_reviews(reviews):
    reviews = [REPLACE_NO_SPACE.sub("", line.lower()) for line in reviews]
    reviews = [REPLACE_WITH_SPACE.sub(" ", line) for line in reviews]
    return reviews
```

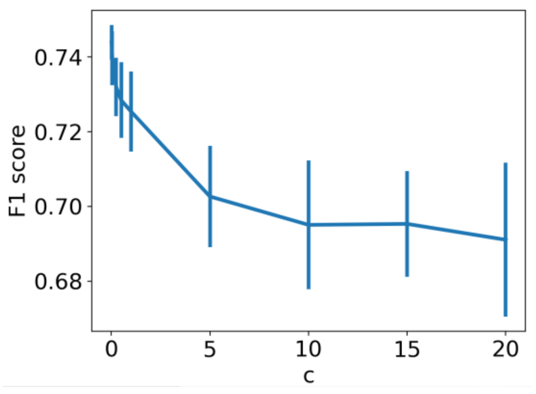
b)tokenization and stop words
Tokenization is the process of turning a meaningful piece of data and tokens serve as reference to the original data, but cannot be used to guess those values. Words such as the, a, is are very common and don’t help much so I would remove the stop words.

```
    def tokenize(text):
    toks = word_tokenize(text)
    s_toks = []
    for t in toks:
        s_toks.append(PorterStemmer().stem(t))
    return s_toks
sel_df_max = 0.1
vectorizer = TfidfVectorizer(
    stop_words=nltk.corpus.stopwords.words('english'),
    max_df=sel_df_max,
    tokenizer=tokenize)
X = vectorizer.fit_transform(x)
```

(3) Vectorization.
In order for this data to make sense to our machine learning algorithm I’ll need to convert each review to a numeric representation. tfidf(t,d) is large for a token that occurs a lot in document d but only rarely in overall collection of documents it’s a reasonable heuristic for identifying informative tokens.

```
from sklearn.feature_extraction.text import TfidfVectorizer
vectorizer = TfidfVectorizer(norm=None)
X = vectorizer.fit_transform(x_clean)
```

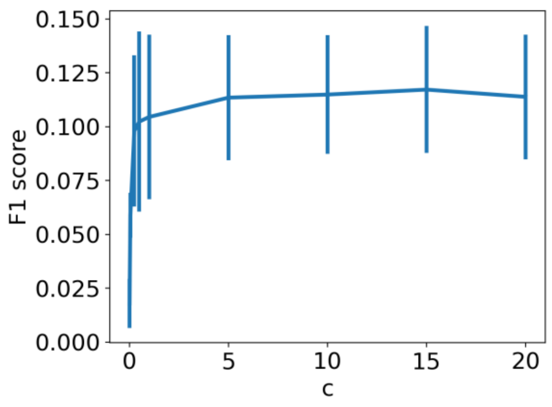
(4)Train the machine learning model
i) LogisticRegression : Logistic regression models the probabilities for classification problems with two possible outcomes. And to make prediction match features of the review text against previous examples from training data.
a)For (x,y) dataset:
First I use F1 score to choose the best hyperparameter of LR.



As we can see from the plot above, the F1 score gets lower as the C increases, so I would pick a small C value to better the LR model that is C=1.

b)For (x,z) dataset:

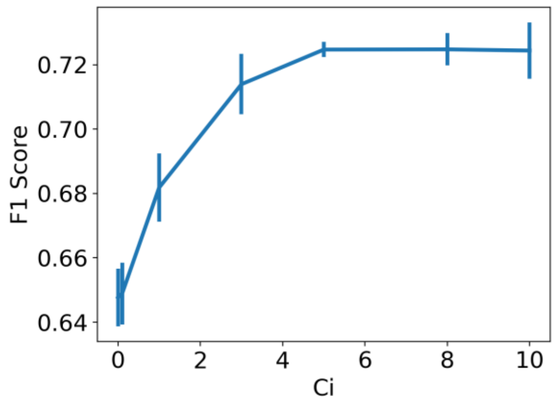
I use F1 score to choose the best hyperparameter of LR.



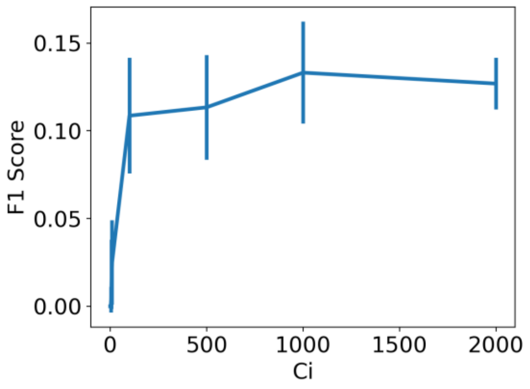
As we can see from the plot above, the F1 score gets higher as the C increases, but when it reaches 15 and over that would decrease the F1score so I would pick a suitable C value to better the LR model that is C=15.

ii)SVM : SVM tries to finds the “best” margin (distance between the line and the support vectors) that separates the classes and this reduces the risk of error on the data, while logistic regression does not, instead it can have different decision boundaries with different weights that are near the optimal point. SVMs don’t penalize examples for which the correct decision is made with sufficient confidence. This may be good for generalization. And in the case of review texts I will apply RBF kernel SVM, which deals with non-linearity and higher dimensions.

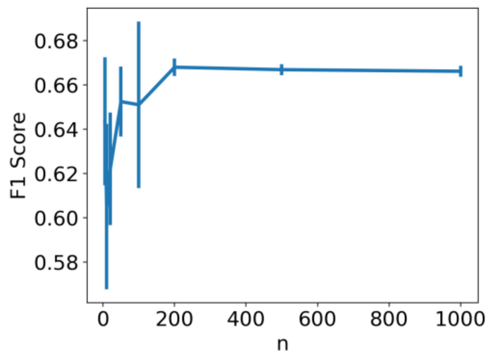
a)for (x,y) dataset. First is to choose the best hyperparameter, so I use a wide range of C to train the model and plot the f1-score. As the plot shows when Ci increases the score gets higher but when Ci>=5 the score is getting slightly smaller. So I would choose Ci=5 to get the best performance.



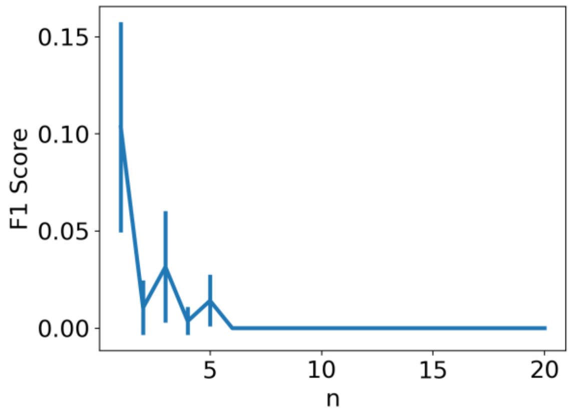
b)for (x,z) dataset. Repeat the same step as in (x,y), and this time I choose a wider range of Ci because it seems for (x,z) dataset it’s not that sensitive to hyperparameter Ci like (x,y). As we can see from the plot below, the score gets larger when Ci gets bigger, however when Ci reaches 1000 the upward trend ends so I will choose Ci=1000.



iii) KNN model: The K in the name of this classifier represents the k nearest neighbors, where k is an integer value specified by the user. Hence as the name suggests, this classifier implements learning based on the k nearest neighbors. The choice of the value of k is dependent on data. So I will select the best k for each dataset.
a)for (x,y) dataset as we can see from the plot when n=200 the F1score is the biggest with correspondingly low standard deviation, so I would choose n=200 for the KNN model.



b)for (x,z) dataset I set the Ci range to `[1,2,3,4,5,6,7,8,9,10,12,15,20]`

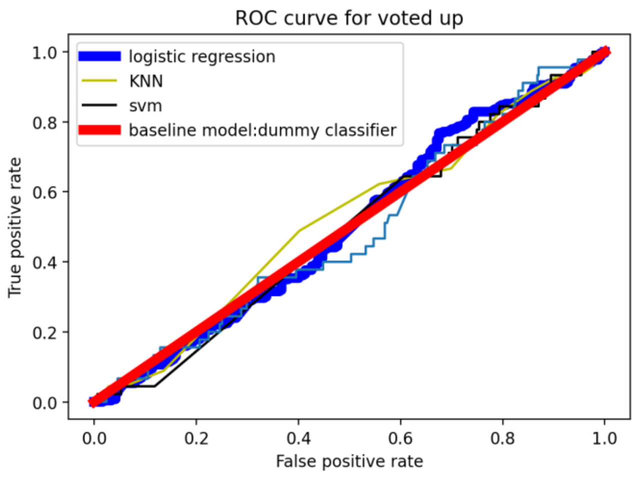


As we can see from the plot above, the performance of the model best at K=1 because it means choosing the closest training sample to the test sample. Since the test sample is in the training dataset, it'll choose itself as the closest and never make mistake. For this reason, the training error will be zero when K = 1, irrespective of the dataset.
(5)Baseline model: DummyClassifier, it doesn't generate any insight about the data and classifies the given data using only simple rules. The classifier's behavior is completely independent of the training data as the trends in the training data are completely ignored and instead uses one of the strategies to predict the class label, so I would choose it as the baseline model. "most_frequent" means the classifier always predicts the most frequent class label in the training data.

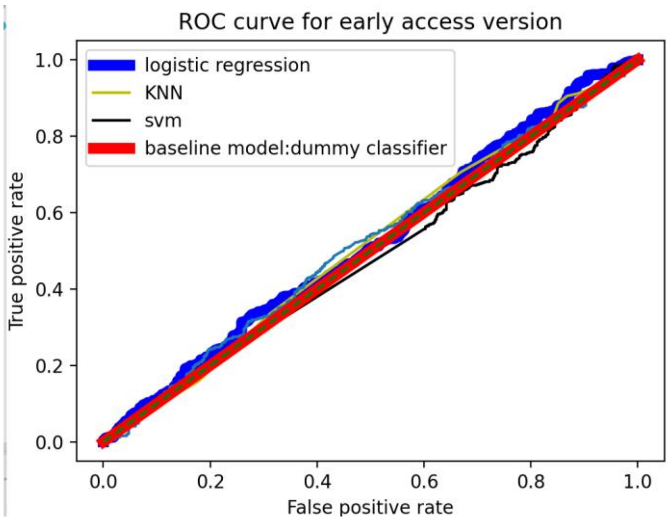
```
from sklearn.dummy import DummyClassifier
dummy_y = DummyClassifier(strategy="most_frequent").fit(X, y)
dummy_z = DummyClassifier(strategy="most_frequent").fit(X, z)
```

(6)ROC curve to compare.

a)for voted up: I had tried many times tuning the hyperparameters of all models, and no matter what I did the accuracy remained about 0.5 of all models, they are all unable to perform better than baseline model. So I would say given the text reviews it's not suitable for predictions of polarity of the content for voted up.

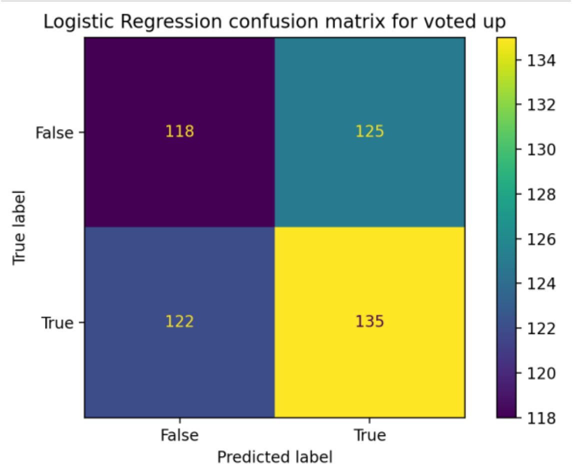


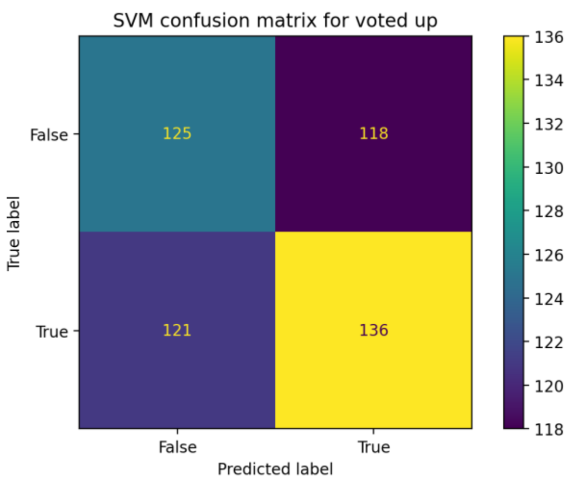
b)for early access version: As we can see from the plot below, all the models mostly overlap with the baseline model, which means the performances of these models are not good, that is, the text reviews are not suitable for predictions of early access version.

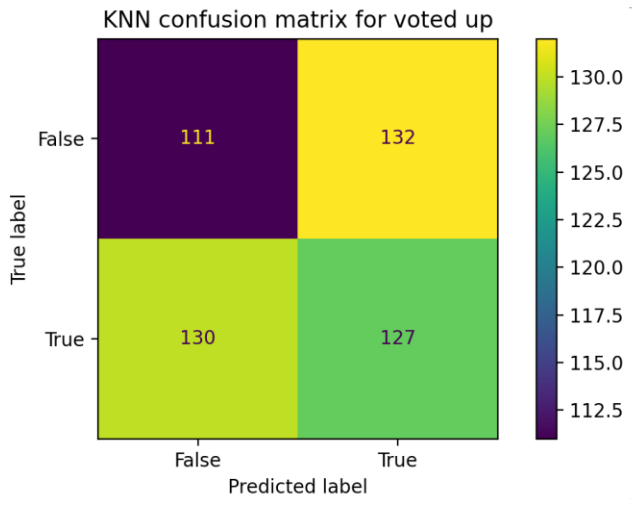


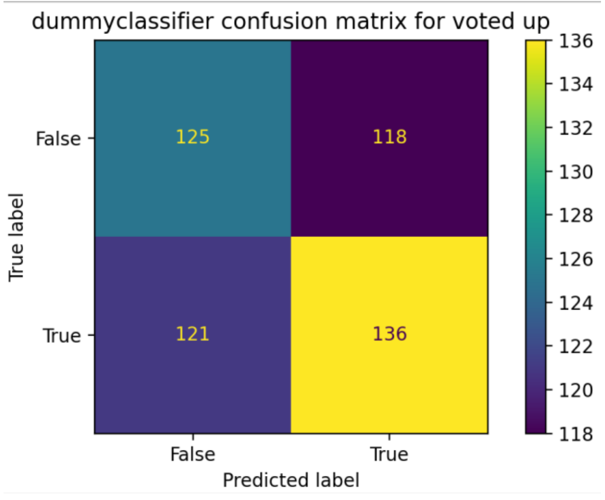
(7)confusion matrix

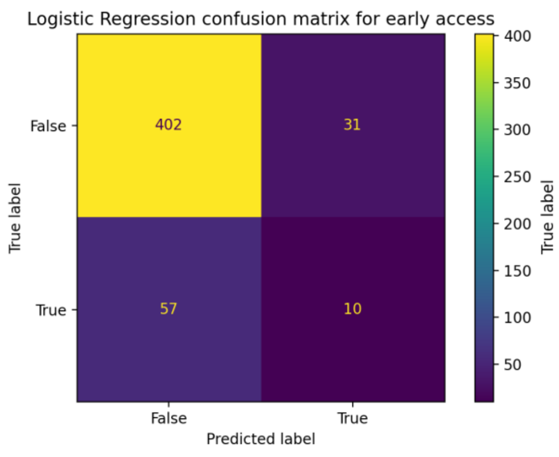
As we can see from the plots below, for voted up the models perform similar and models are not suitable for predictions of voted up. And this is the same as for early access, maybe the data is too unpredictable because we can't just tell by external words, language is magic and we say the words that have deep meaning within it.

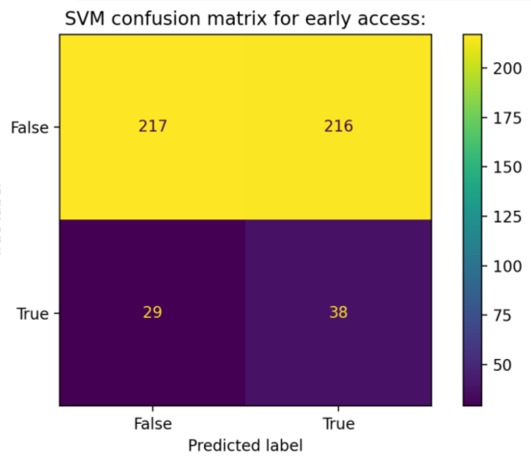


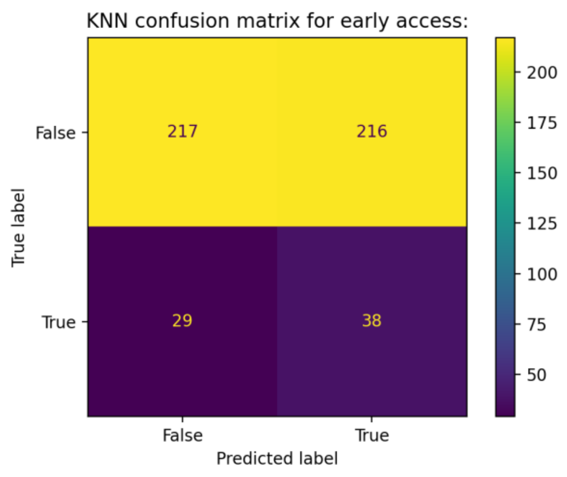


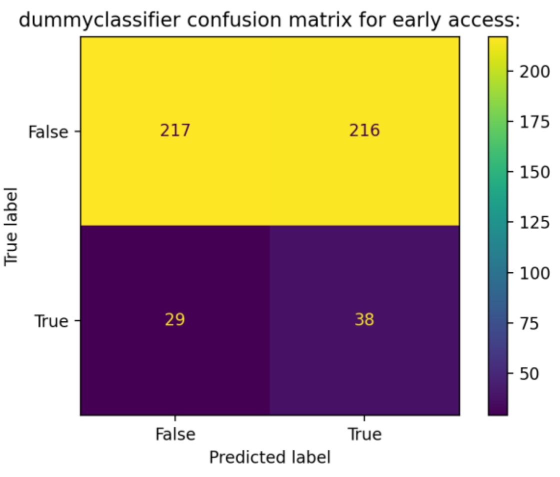












2.

(i) What are under-fitting and over-fitting, give an example of each. [5 marks]

Overfitting: overfitting means our model fits too closely to our data. The fitted line will go exactly through every point in the graph and this may fail to make predictions on future data reliably.
Underfitting: the counterpart of overfitting, happens when a machine learning model is not complex enough to accurately capture relationships between a dataset's features and a target variable. An underfitted model results in problematic or erroneous outcomes on new data, or data that it wasn't trained on, and often performs poorly even on training data.
Example:
For lasso regression: When C is small and close to 0, and as for the model it keeps the same trend and doesn't have much change because all coefficients are zero , the accuracy is 0 too , which means the model is underfitting, cannot capture the underlying trend of the data; And if we tune the alpha bigger and bigger , the accuracy of the training data would increase, imagine if the C is infinity , which means the model couldn't even tolerate a little error , then all the data could be considered , then the training is overfitting, since the data is shaped like a convex surface but the prediction is not convex near the edges or the prediction is much steeper than the training data near the boundary.

(ii) Give pseudo-code implementing k-fold cross-validation. [5 marks]

```
from sklearn.model_selection import KFold
kf = KFold(n_splits=N)
for train, test in kf.split(X):
    define the model
    model.fit(X[train], y[train])
    ypred = model.predict(X[test])
    evaluate performance on y[test],ypred
```

(iii) Why does k-fold cross-validation provide a way to select a model hyperparameter so as to strike a balance between over/under-fitting. [5 marks]

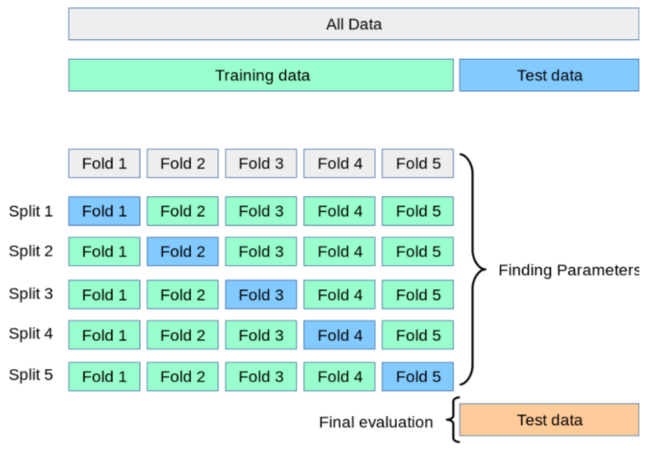


Figure 2.3 the step of nested k-fold cv to select model hyperparameter

The k-fold cross-validation procedure divides a limited dataset into k non-overlapping folds. Each of the k folds is given an opportunity to be used as a held back test set whilst all other folds collectively are used as a training dataset. And the k-fold cross-validation procedure for model hyperparameter optimization is nested inside the k-fold cross-validation procedure for model selection. Let's use five folds as an example. We perform a series of train and evaluate cycles where each time we train on 4 of the folds and test on the 5th, called the hold-out set. We repeat this cycle 5 times, each time using a different fold for evaluation. At the end, we average the scores for each of the folds to determine the overall performance of a given model. This allows us to optimize the model before deployment without having to use additional data. Use cross-validation to select the best model by creating models with a range of different hyperparameters,

and evaluate each one using k-fold cross-validation. Most importantly, MSE(the average distance between the prediction and the real value squared)could be calculated and we can tell from the plot : a model that is underfit will have high training and high testing error while an overfit model will have extremely low training error but a high testing error.
(iv) Discuss three pros and cons of a logistic regression classifier vs a kNN classifier.
[5 marks]

1) KNN is a non-parametric model, where logistic regression is a parametric model and KNN is comparatively slower than logistic regression.
2) KNN supports non-linear solutions where logistic regression supports only linear solutions.
3) Logistic regression can derive confidence level (about its prediction), whereas KNN can only output the labels.

v) Give two examples of situations when a kNN classifier would give inaccurate predictions. Explain your reasoning. [5 marks]
1.Dataset is very large. As calculating distances between each data instance would be very costly. It’s not great for large datasets, since the entire training data is processed for every prediction. Time complexity for each prediction is O(MNlog(k)) where M is the dimension of the data, N is the size or the number of instances in the training data.
2.Missing values of outliners. KNN is sensitive to outliers and missing values so a single mislabeled example can change the class boundaries. This could specially be a bigger problem for larger dimensions, if there is an outlier in one dimension, since the average separation tends to be higher for higher dimensions (curse of dimensionality), outliers can have a bigger impact.

Appendix:
```
    import json_lines
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score, plot_confusion_matrix
from sklearn.model_selection import train_test_split
from sklearn.svm import LinearSVC
from sklearn.svm import SVC
import numpy as np
import matplotlib.pyplot as plt
import nltk
nltk.download('punkt')
nltk.download('stopwords')
from nltk.tokenize import word_tokenize
from nltk.stem import PorterStemmer

#read into python
x=[]
y=[]
z=[]
with open('reviews_80.jl','rb') as f:
    for item in json_lines.reader(f):
        x.append(item['text'])
        y.append(item['voted_up'])
        z.append(item['early_access'])


def tokenize(text):
    toks = word_tokenize(text)
    s_toks = []
    for t in toks:
        s_toks.append(PorterStemmer().stem(t))
    return s_toks
sel_df_max = 0.1
vectorizer = TfidfVectorizer(
    stop_words=nltk.corpus.stopwords.words('english'),
    max_df=sel_df_max,
    tokenizer=tokenize)
X = vectorizer.fit_transform(x)

#clean data
"""
import re
REPLACE_NO_SPACE = re.compile("[.;:!\'?,\"()\[\]]")
REPLACE_WITH_SPACE = re.compile("(<br\s*/><br\s*/>)|(\-)|(\/)")


def preprocess_reviews(reviews):
    reviews = [REPLACE_NO_SPACE.sub("", line.lower()) for line in reviews]
    reviews = [REPLACE_WITH_SPACE.sub(" ", line) for line in reviews]
    return reviews

x_clean = preprocess_reviews(x)
#vectorize
from sklearn.feature_extraction.text import TfidfVectorizer
vectorizer = TfidfVectorizer(norm=None,max_df =0.1)
X = vectorizer.fit_transform(x_clean)
"""


#split the data
Xtrain, Xtest, ytrain, ytest = train_test_split(X, y, test_size=0.1)
Xtrain, Xtest, ztrain, ztest = train_test_split(X, z, test_size=0.1)
#logistic regression
    #select best hyperparameter
        #for y
"""
mean_error_y=[]
std_error_y=[]
crange_y=[0.01, 0.05, 0.25, 0.5, 1, 5, 10, 15, 20]
for c in crange_y:
    lr = LogisticRegression(C=c)
    lr.fit(Xtrain, ytrain)
    from sklearn.model_selection import cross_val_score
    scores = cross_val_score(lr, X, y, cv=5, scoring='f1')
    mean_error_y.append(np.array(scores).mean())
    std_error_y.append(np.array(scores).std())
    print("Final Accuracy: %s"% accuracy_score(ytest, lr.predict(Xtest)))
import matplotlib.pyplot as plt
plt.rc('font', size = 18)
plt.errorbar(crange_y, mean_error_y, yerr=std_error_y, linewidth=3)
plt.xlabel('c'); plt.ylabel('F1 score')
plt.show()
"""

        #final lr
lr_y = LogisticRegression(C=1)
lr_y.fit(Xtrain, ytrain)
fig1=plt.figure()
dispy1=plot_confusion_matrix(lr_y, Xtest, ytest)
print("Logistic Regression confusion matrix for voted up:")
print(dispy1.confusion_matrix)
plt.title("Logistic Regression confusion matrix for voted up")
lr_z = LogisticRegression(C=15)
lr_z.fit(Xtrain, ztrain)
dispz1=plot_confusion_matrix(lr_z, Xtest, ztest)
print("Logistic Regression confusion matrix for early access:")
print(dispz1.confusion_matrix)
plt.title("Logistic Regression confusion matrix for early access")
"""
mean_error_y=[]
std_error_y=[]
crange_y=[0.01, 0.05, 0.25, 0.5, 1, 5, 10, 15, 20]
for c in crange_y:
    lr = LogisticRegression(C=c)
    lr.fit(Xtrain, ztrain)
    from sklearn.model_selection import cross_val_score
    scores = cross_val_score(lr, X, z, cv=5, scoring='f1')
    mean_error_y.append(np.array(scores).mean())
    std_error_y.append(np.array(scores).std())
    import matplotlib.pyplot as plt

plt.rc('font', size = 18)
plt.errorbar(crange_y, mean_error_y, yerr=std_error_y, linewidth=3)
plt.xlabel('c'); plt.ylabel('F1 score')
plt.show() """ """
#SVM
    #for y select best C value in SVC

mean_error=[]
```

```
std_error=[]
Ci_range = [0.001,0.1,1,3,5,8,10]
for Ci in Ci_range:
    model = SVC(C=Ci, kernel='rbf')
    from sklearn.model_selection import cross_val_score
    scores = cross_val_score(model, X, y, cv=5, scoring='f1')
    mean_error.append(np.array(scores).mean())
    std_error.append(np.array(scores).std())
import matplotlib.pyplot as plt
plt.rc('font', size=18);
plt.errorbar(Ci_range,mean_error,yerr=std_error,linewidth=3)
plt.xlabel('Ci'); plt.ylabel('F1 Score')
plt.show()
print("finish")"""
       #the final model
svm_y=SVC(C=5, kernel='rbf',probability=True).fit(Xtrain,ytrain)
dispy2=plot_confusion_matrix(svm_y, Xtest, ytest)
print("SVM confusion matrix for voted up:")
print(dispy2.confusion_matrix)
plt.title("SVM confusion matrix for voted up")
svm_z=SVC(C=1000,kernel='rbf',probability=True).fit(Xtrain,ztrain)
dispz2=plot_confusion_matrix(svm_y, Xtest, ztest)
print("SVM confusion matrix for early access:")
print(dispz2.confusion_matrix)
plt.title("SVM confusion matrix for early access:")
"""
    # for z select best C value in SVC
mean_error=[]
std_error=[]
Ci_range = [0.001,0.1,1,3,5,8,10,100,500,1000,2000]
for Ci in Ci_range:
    model = SVC(C=Ci, kernel='rbf')
    from sklearn.model_selection import cross_val_score
    scores = cross_val_score(model, X, z, cv=5, scoring='f1')
    mean_error.append(np.array(scores).mean())
    std_error.append(np.array(scores).std())
import matplotlib.pyplot as plt
plt.rc('font', size=18);
plt.errorbar(Ci_range,mean_error,yerr=std_error,linewidth=3)
plt.xlabel('Ci'); plt.ylabel('F1 Score')
plt.show()
"""
#KNN
from sklearn.neighbors import KNeighborsRegressor, KNeighborsClassifier
#final KNN
knn_y=KNeighborsClassifier(n_neighbors=200).fit(Xtrain, ytrain)
dispy3=plot_confusion_matrix(knn_y, Xtest, ytest)
print("KNN confusion matrix for voted up:")
print(dispy3.confusion_matrix)
```

```python
plt.title("KNN confusion matrix for voted up")

knn_z=KNeighborsClassifier(n_neighbors=1).fit(Xtrain, ztrain)
dispz3=plot_confusion_matrix(svm_y, Xtest, ztest)
print("KNN confusion matrix for early access:")
print(dispz3.confusion_matrix)
plt.title("KNN confusion matrix for early access:")
"""
mean_error=[]
std_error=[]
n_range = [1,2,3,4,5,6,7,8,9,10,12,15,20,40,50]
from sklearn import metrics

for n in n_range:
   model_knn = KNeighborsClassifier(n_neighbors=n).fit(Xtrain, ytrain)
   y_pred=model_knn.predict(Xtest)
   from sklearn.model_selection import cross_val_score
   scores = cross_val_score(model_knn, X, y, cv=5, scoring='f1')
   mean_error.append(np.array(scores).mean())
   std_error.append(np.array(scores).std())
   print(metrics.accuracy_score(ytest, y_pred))
import matplotlib.pyplot as plt
plt.rc('font', size=18);
plt.errorbar(n_range,mean_error,yerr=std_error,linewidth=3)
plt.xlabel('n'); plt.ylabel('F1 Score')
plt.show()
"""
#baseline model:
from sklearn.dummy import DummyClassifier
dummy_y = DummyClassifier(strategy="most_frequent").fit(Xtrain, ytrain)
dispy4=plot_confusion_matrix(svm_y, Xtest, ytest)
print("Dummy classifier matrix for voted up:")
print(dispy4.confusion_matrix)
plt.title("dummyclassifier confusion matrix for voted up")
dummy_z = DummyClassifier(strategy="most_frequent").fit(Xtrain, ztrain)
dispz4=plot_confusion_matrix(svm_y, Xtest, ztest)
print("dummyclassifier matrix for early access:")
print(dispz4.confusion_matrix)
plt.title("dummyclassifier confusion matrix for early access:")
plt.show()

#ROC curve
#for y
from sklearn.metrics import roc_curve,auc

fpr1, tp1, _ =roc_curve(ytest,lr_y.decision_function(Xtest))
plt.plot(fpr1,tp1,label='logistic regression',c='b',linewidth=6)

y_scores_knn = knn_y.predict_proba(Xtest)
```

```
fpr2, tp2, threshold1 = roc_curve(ytest, y_scores_knn[:, 1])
plt.plot(fpr2,tp2,label='KNN',c='y')

y_scores_svm = svm_y.predict_proba(Xtest)
fpr3, tp3, threshold2 = roc_curve(ytest, y_scores_svm[:, 1])
plt.plot(fpr3,tp3,label='svm',c='black')

plt.xlabel('False positive rate')
plt.ylabel('True positive rate')
plt.plot([0, 1], [0, 1], color='g',linestyle='--')

#baseline
y_scores_dummy = dummy_y.predict_proba(Xtest)
fpr4, tp4, threshold3 = roc_curve(ytest, y_scores_dummy[:, 1])
plt.plot(fpr4,tp4,label='baseline model:dummy classifier',c='r',linewidth=6)
plt.title("ROC curve for voted up")
plt.legend()
plt.show()
```