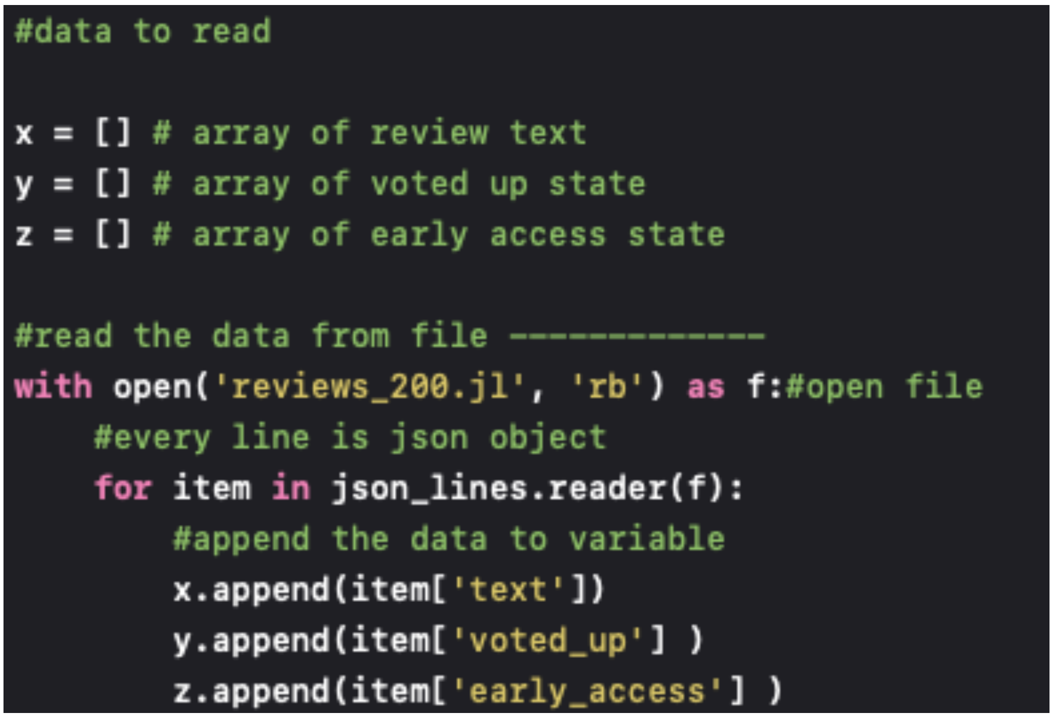
FINAL ASSINGMENT 2020

CS7CS4 Machine Learning

Hasan Erbilen

25th January 2021

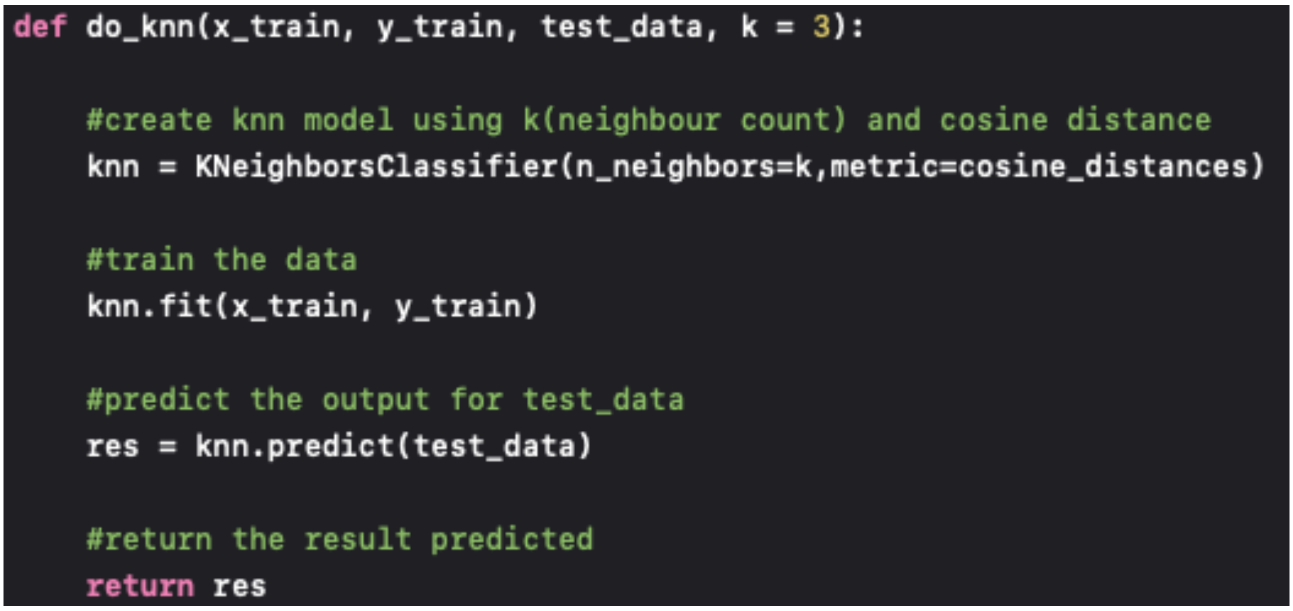
1)The goal is to evaluate whether the review text can be used to predict whether (i) the game was "voted up" by the reviewer; or (ii) the review is for an "early access" version of the game.We begin by reading the review file using the json_lines module. It takes a review txt array as x, voted up array as y and early access array as z. The length of all arrays are the same.



```python
#data to read

x = [] # array of review text
y = [] # array of voted up state
z = [] # array of early access state

#read the data from file -------------
with open('reviews_200.jl', 'rb') as f:#open file
    #every line is json object
    for item in json_lines.reader(f):
        #append the data to variable
        x.append(item['text'])
        y.append(item['voted_up'] )
        z.append(item['early_access'] )
```

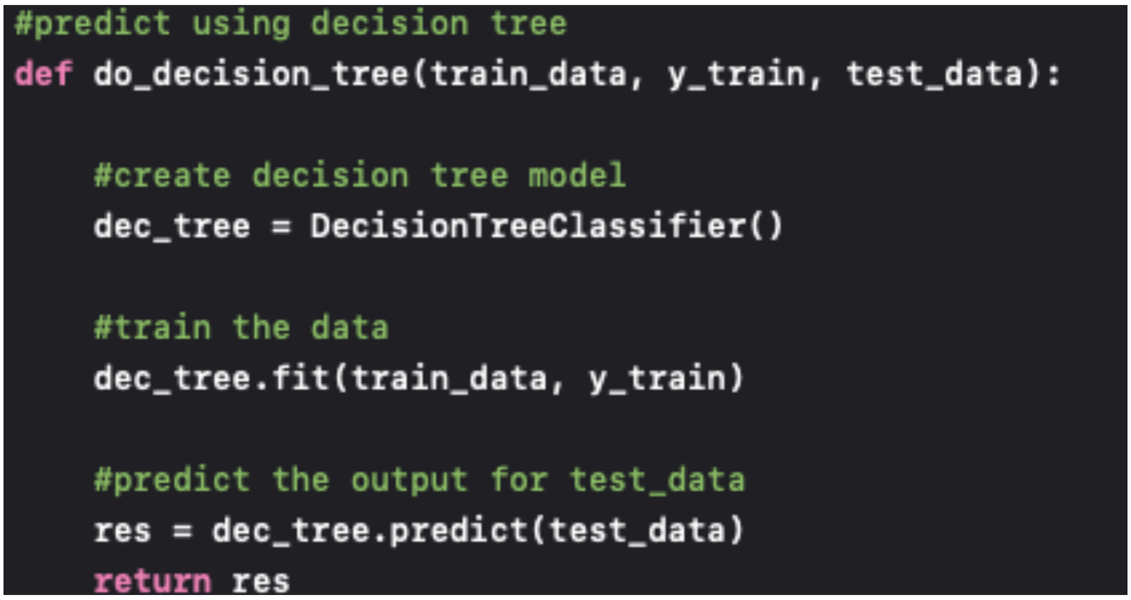
And then we create a prediction model for KNN, Decision Tree, SVM and Logistic   Regression.

a)KNN Classifier Model:First, we create the kNN model and train the dataFor given test data, we predict the y value ("voted up" or"early access").



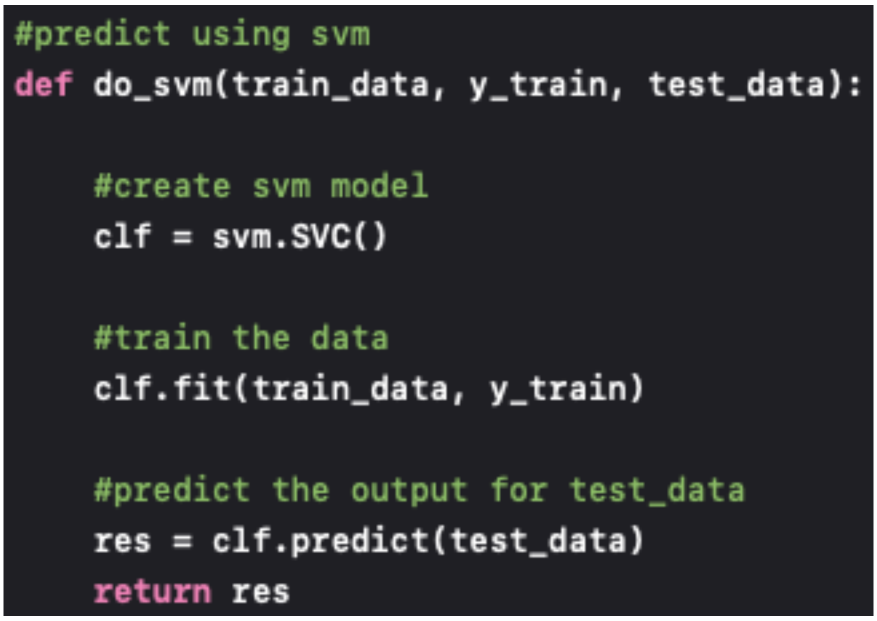
```python
def do_knn(x_train, y_train, test_data, k = 3):

    #create knn model using k(neighbour count) and cosine distance
    knn = KNeighborsClassifier(n_neighbors=k,metric=cosine_distances)

    #train the data
    knn.fit(x_train, y_train)

    #predict the output for test_data
    res = knn.predict(test_data)

    #return the result predicted
    return res
```

b) **Decision Tree Model:**First, we create the Decision Tree and train the data.For given test data, we predict the y value ("voted up" or"early access").



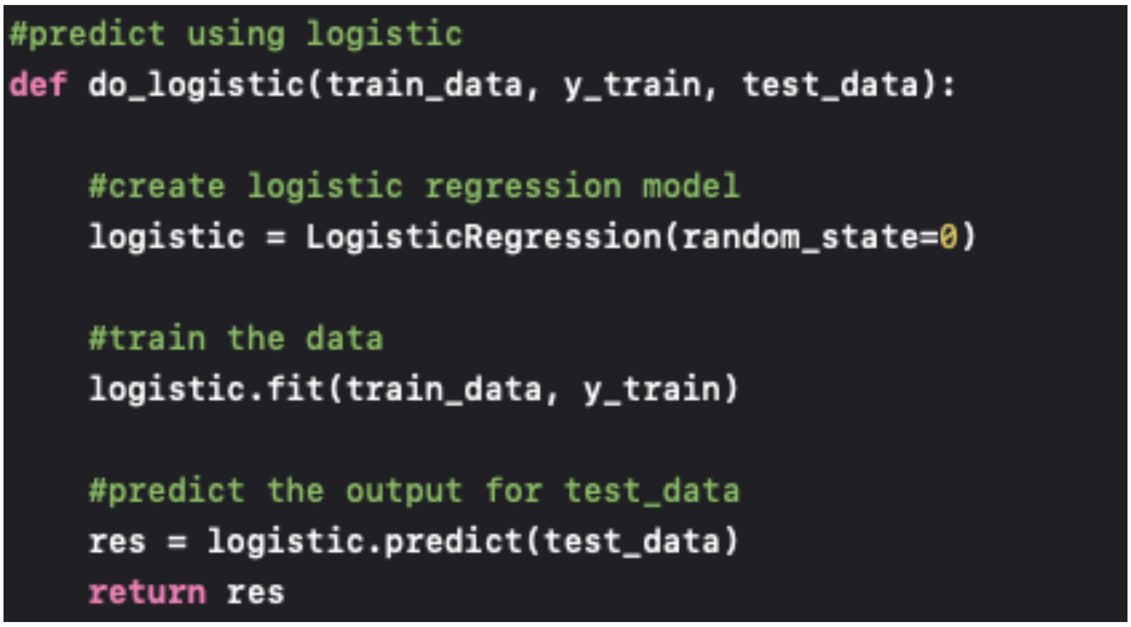
```python
#predict using decision tree
def do_decision_tree(train_data, y_train, test_data):

    #create decision tree model
    dec_tree = DecisionTreeClassifier()

    #train the data
    dec_tree.fit(train_data, y_train)

    #predict the output for test_data
    res = dec_tree.predict(test_data)
    return res
```

c)**SVM Classifier Model:**First, we create the SVM model and train the data.For given test data, we predict the y value ("voted up" or"early access").



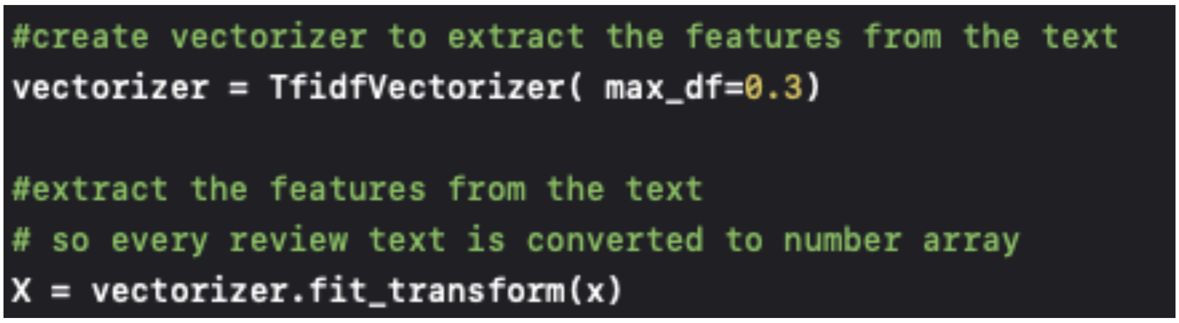
```
#predict using svm
def do_svm(train_data, y_train, test_data):

    #create svm model
    clf = svm.SVC()

    #train the data
    clf.fit(train_data, y_train)

    #predict the output for test_data
    res = clf.predict(test_data)
    return res
```

d) **Logistic Regression Model:**First, we create the Logistic Regression model and train the data.For given test data, we predict the y value (“voted up” or “early access”)



```
#predict using logistic
def do_logistic(train_data, y_train, test_data):

    #create logistic regression model
    logistic = LogisticRegression(random_state=0)

    #train the data
    logistic.fit(train_data, y_train)

    #predict the output for test_data
    res = logistic.predict(test_data)
    return res
```

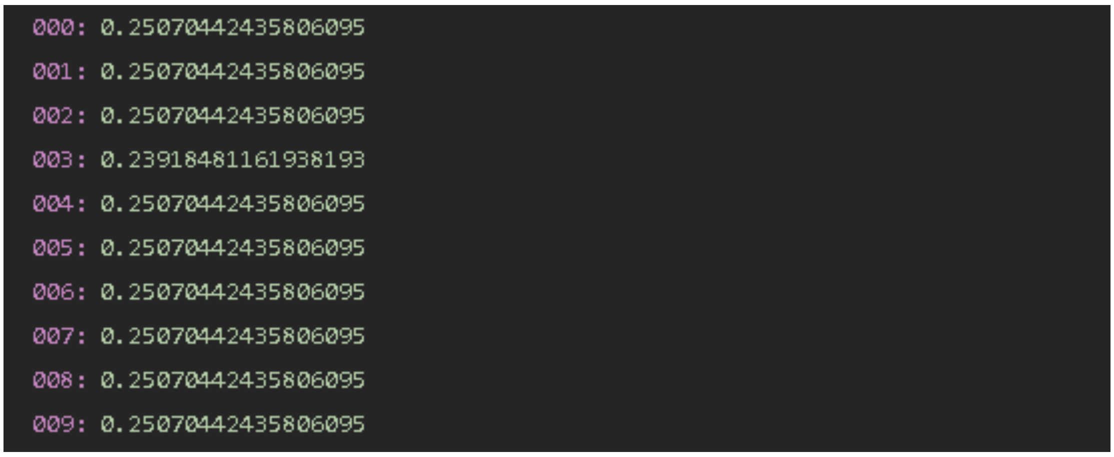
Next, we extract features from the review text



```
#create vectorizer to extract the features from the text
vectorizer = TfidfVectorizer( max_df=0.3)

#extract the features from the text
# so every review text is converted to number array
X = vectorizer.fit_transform(x)
```

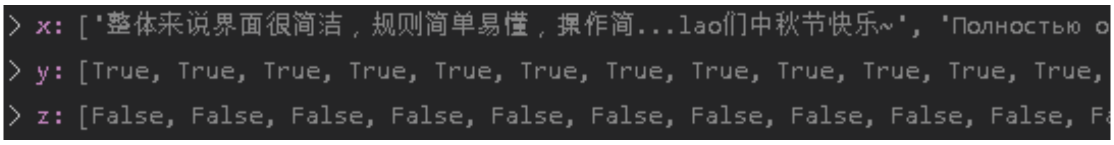
Every review is converted to a number array and the array length for every review will be the same.

The matrix size is 5000 * 49119



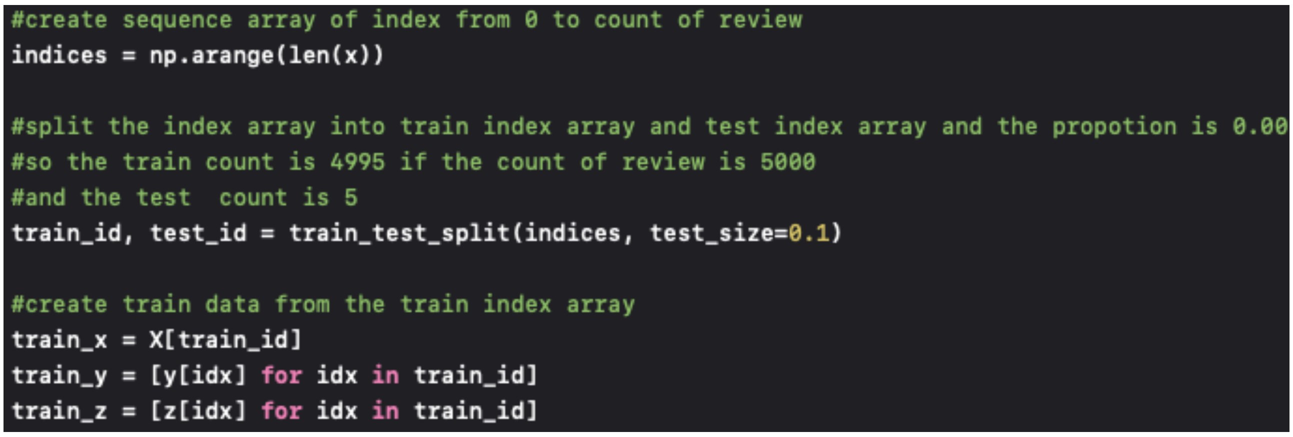
```
000: 0.25070442435806095
001: 0.25070442435806095
002: 0.25070442435806095
003: 0.23918481161938193
004: 0.25070442435806095
005: 0.25070442435806095
006: 0.25070442435806095
007: 0.25070442435806095
008: 0.25070442435806095
009: 0.25070442435806095
```

And the y and z are still remaining as Boolean array



```
> x: ['整体来说界面很简洁，规则简单易懂，操作简...lao们中秋节快乐~', 'Полностью о
> y: [True, True, True, True, True, True, True, True, True, True, True, True,
> z: [False, False, False, False, False, False, False, False, False, False, Fa
```

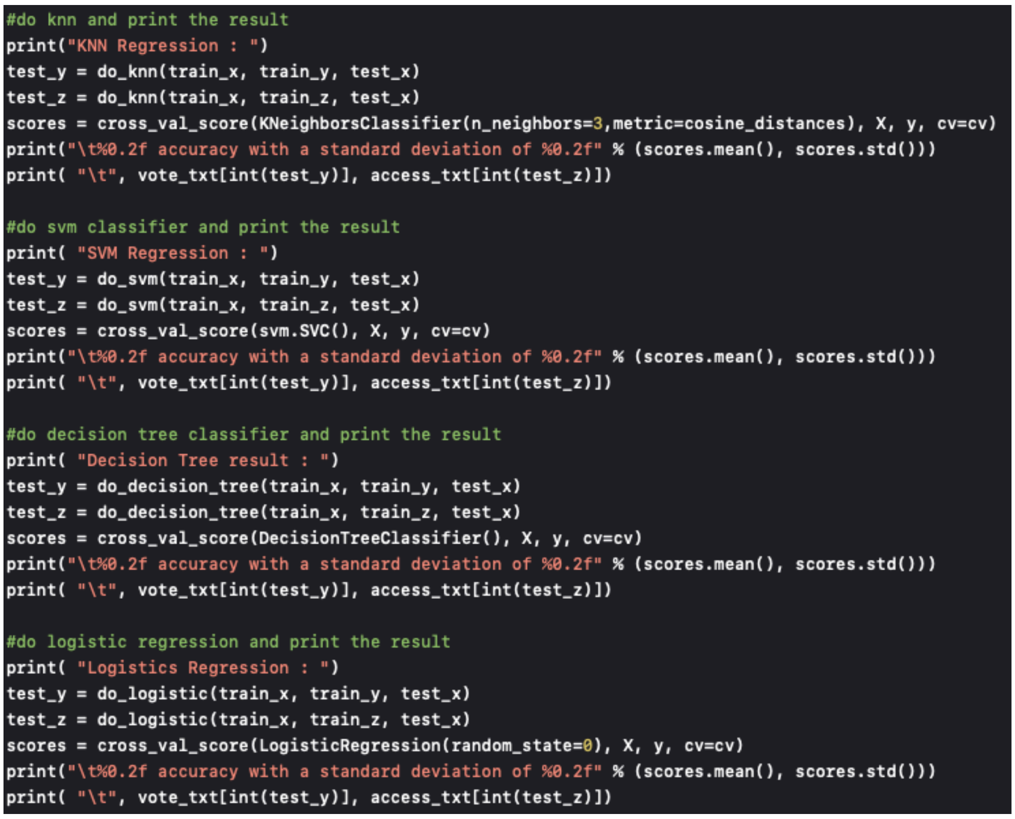
Then we split the data array into train data and test data using ratio 0.001.



```
#create sequence array of index from 0 to count of review
indices = np.arange(len(x))

#split the index array into train index array and test index array and the propotion is 0.00
#so the train count is 4995 if the count of review is 5000
#and the test  count is 5
train_id, test_id = train_test_split(indices, test_size=0.1)

#create train data from the train index array
train_x = X[train_id]
train_y = [y[idx] for idx in train_id]
train_z = [z[idx] for idx in train_id]
```

The test review count will be 5 if the total count is 5000

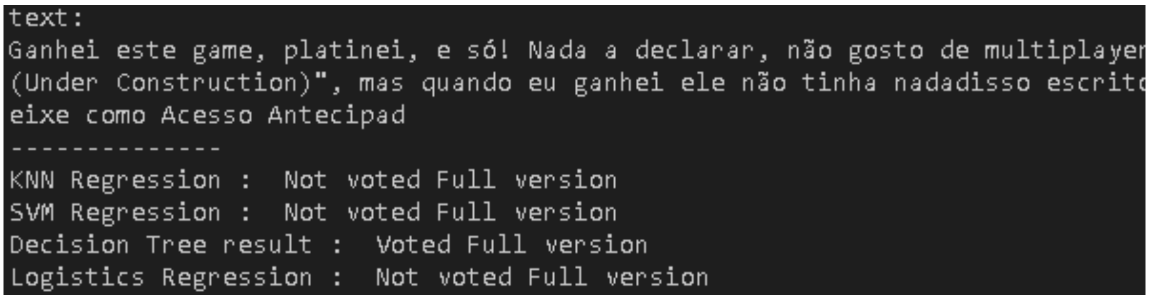
Then we do predict the “voted up” state and “early access” state using KNN, SVM, Decision Tree, Logistic ModeUsing the same model, we predict “voted up” and “early access”.



```
#do knn and print the result
print("KNN Regression : ")
test_y = do_knn(train_x, train_y, test_x)
test_z = do_knn(train_x, train_z, test_x)
scores = cross_val_score(KNeighborsClassifier(n_neighbors=3,metric=cosine_distances), X, y, cv=cv)
print("\t%0.2f accuracy with a standard deviation of %0.2f" % (scores.mean(), scores.std()))
print( "\t", vote_txt[int(test_y)], access_txt[int(test_z)])

#do svm classifier and print the result
print( "SVM Regression : ")
test_y = do_svm(train_x, train_y, test_x)
test_z = do_svm(train_x, train_z, test_x)
scores = cross_val_score(svm.SVC(), X, y, cv=cv)
print("\t%0.2f accuracy with a standard deviation of %0.2f" % (scores.mean(), scores.std()))
print( "\t", vote_txt[int(test_y)], access_txt[int(test_z)])

#do decision tree classifier and print the result
print( "Decision Tree result : ")
test_y = do_decision_tree(train_x, train_y, test_x)
test_z = do_decision_tree(train_x, train_z, test_x)
scores = cross_val_score(DecisionTreeClassifier(), X, y, cv=cv)
print("\t%0.2f accuracy with a standard deviation of %0.2f" % (scores.mean(), scores.std()))
print( "\t", vote_txt[int(test_y)], access_txt[int(test_z)])

#do logistic regression and print the result
print( "Logistics Regression : ")
test_y = do_logistic(train_x, train_y, test_x)
test_z = do_logistic(train_x, train_z, test_x)
scores = cross_val_score(LogisticRegression(random_state=0), X, y, cv=cv)
print("\t%0.2f accuracy with a standard deviation of %0.2f" % (scores.mean(), scores.std()))
print( "\t", vote_txt[int(test_y)], access_txt[int(test_z)])
```

If the review is good, the review will be “voted up” and if it is “early access”, the review will be badKNN

The output of the program will be as follows:



```
text:
Ganhei este game, platinei, e só! Nada a declarar, não gosto de multiplayer
(Under Construction)", mas quando eu ganhei ele não tinha nadadisso escrito
eixe como Acesso Antecipad
--------------
KNN Regression :  Not voted Full version
SVM Regression :  Not voted Full version
Decision Tree result :  Voted Full version
Logistics Regression :  Not voted Full version
```

- kNN vs SVM
  - ✓ kNN attempts to approximate the underlying distribution of the data in a non-parametric fashion
  - ✓ SVM assumes there exists a hyper-plane separating the data points (quite a restrictive assumption).
  - ✓ SVM takes care of outliers better than KNN
  - ✓ If training data is much larger than number of features(m>>n), KNN is better than SVM. SVM outperforms KNN when there are bigger features and lesser training data.
- kNN vs Logistic Regression
  - ✓ KNN is a non-parametric model, where Logistic Regression is a parametric model.
  - ✓ KNN is comparatively slower than Logistic Regression.
  - ✓ KNN supports non-linear solutions where Logistic Regression supports only linear solutions.
  - ✓ Logistic Regression can derive confidence level (about its prediction), whereas KNN can only output the labels.
- Logistic Regression vs SVM
  - ✓ SVM can handle non-linear solutions whereas logistic regression can only handle linear solutions.
  - ✓ Linear SVM handles outliers better, as it derives maximum margin solution.
  - ✓ Hinge loss in SVM outperforms log loss in Logistic Regression

No	Model Name	Accuracy(%)
1	kNN	0.69
2	Decision Tree	0.67
3	Logistic Regression	0.73
4	SVM	0.48

Figure1:Comparison of Accuracy with using Cross Validation

- Our train data count is smaller than the feature count and the train data is not linear so kNN,Decision Tree Classifier and LR are better than SVM.

2)

i)

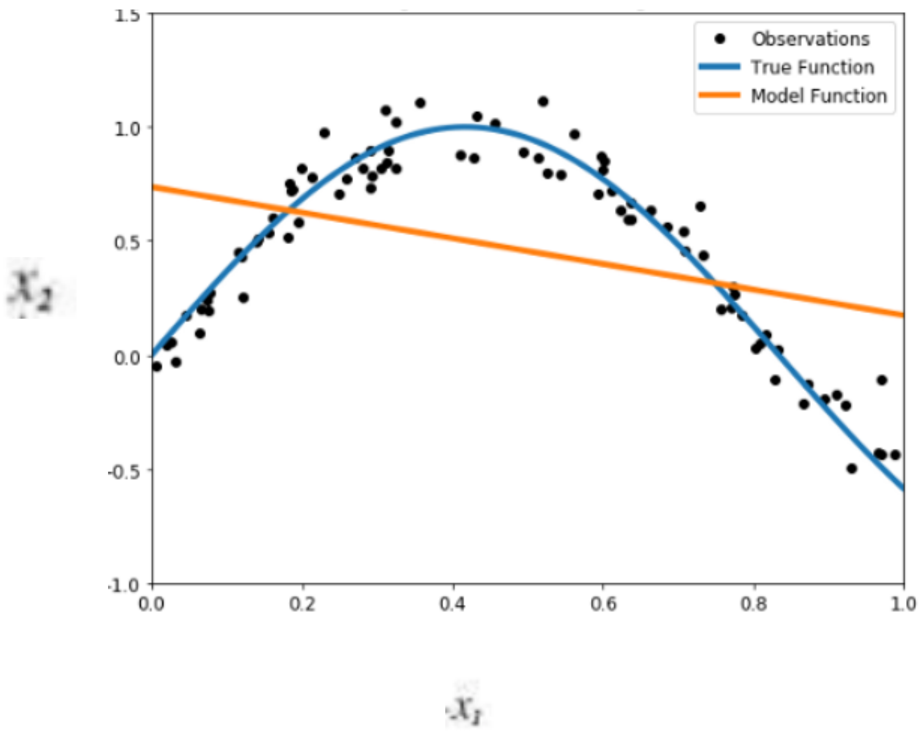


Figure 2:Example of Underfitting

-Underfitting occurs when a statistical model cannot adequately capture the underlying structure of the data.

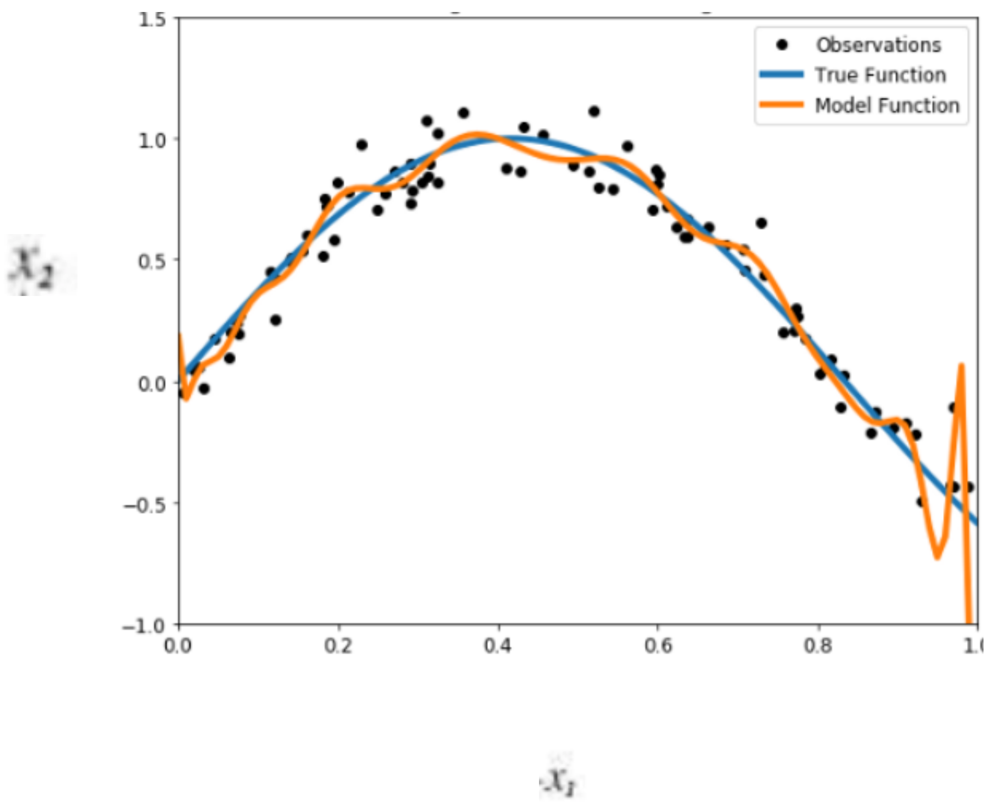
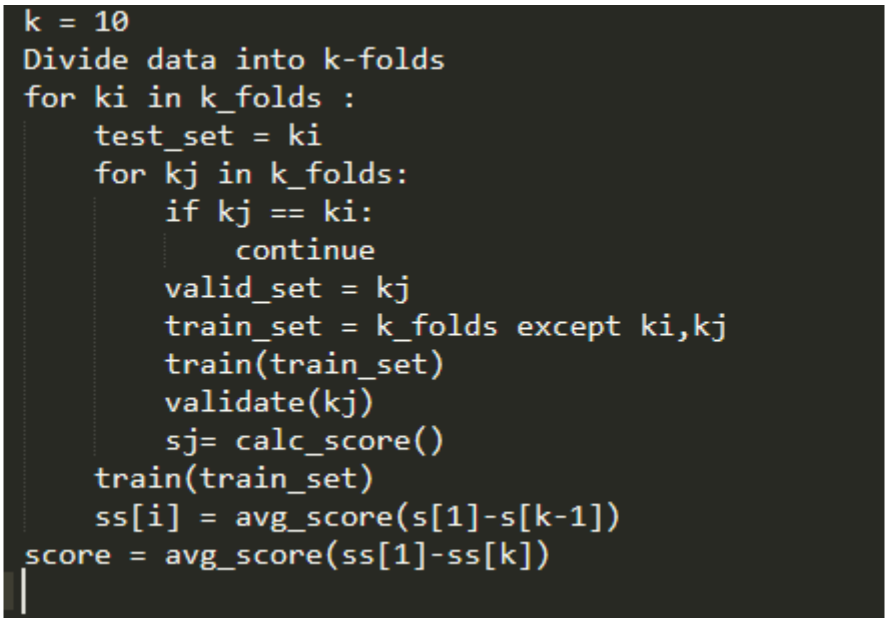


Figure3:Example of Overfitting

- Overfitting is "the production of an analysis that corresponds too closely or exactly to a particular set of data.
  It may therefore fail to fit additional data or predict future observations reliably.

ii)



```
k = 10
Divide data into k-folds
for ki in k_folds :
    test_set = ki
    for kj in k_folds:
        if kj == ki:
            continue
        valid_set = kj
        train_set = k_folds except ki,kj
        train(train_set)
        validate(kj)
        sj= calc_score()
    train(train_set)
    ss[i] = avg_score(s[1]-s[k-1])
score = avg_score(ss[1]-ss[k])
```

iii)

K-fold cross validation method allows using different training and testing data which will avoid overfitting and give better generalization ability.

- ✓ To use all the data for both training and testing purposes.

  Each section is used for training and testing so we are able to make predictions on all of our data

- ✓ To get more metrics

  Our accuracy is similar in all our folds, say 92.0, 91.5, 92.0, 92.5 and 91.8 then it means our data(algorithm) is consistent and we can be confident

  But we could end up in a slightly different scenario, say 92.0, 44.0, 91.5, 92.5 and 91.8 then we have to go back and make sure that our data is what we think it is.

- ✓ Use Models Stacking,We can predict by passing all of our model iv)

Category	Logistic Regression	KNN
Parametric	Yes	No
Speed	Fast	Slow
Linear solution	Yes	No

Figure 4: Pros and cons of a logistic regression classifier vs a kNN classifier

v)

No	Home Owner	Job Experience(0-5)	Defaulted(out)
1	No	1	Yes
2	Yes	5	Yes
3	No	5	No
4	Yes	0	No

Figure5: First Example inaccurate predictions

If k is 2, our data is (HO = Yes, JE = 1) =>the nearest neighbors is 1st and 4th but the output is Yes and No, so the prediction result will be wrong.

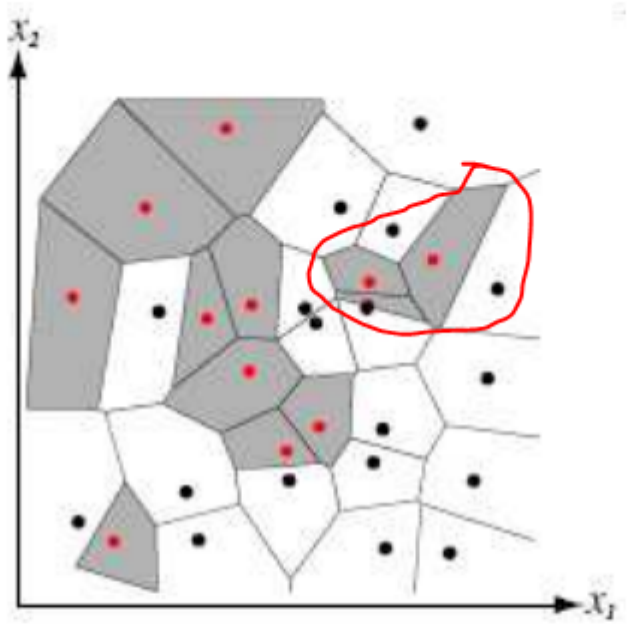


Figure 6:Second Example inaccurate predictions

It is the result of k = 1.The points in the red blob will be referred to noise.So if we try to predict the result in this red blob, the result will be red point.But indeed, it is black point.Deciding k is most important

Appendix

```python
from types import ClassMethodDescriptorType
import json_lines
import numpy as np

from sklearn.feature_extraction.text import TfidfVectorizer

from sklearn.model_selection import train_test_split
from sklearn.model_selection import cross_val_score
from sklearn.model_selection import ShuffleSplit

from sklearn.metrics.pairwise import cosine_distances
from sklearn.metrics import accuracy_score

from sklearn.neighbors import KNeighborsClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.linear_model import LogisticRegression
from sklearn import svm

vote_txt = ["Not voted", "Voted" ]
access_txt = ["Full version", "Early Access" ]

#predict using knn
def do_knn(x_train, y_train, test_data, k = 3):

    #create knn model using k(neighbour count) and cosine distance
    knn = 
KNeighborsClassifier(n_neighbors=k,metric=cosine_distances)

    #train the data
    knn.fit(x_train, y_train)

    #predict the output for test_data
    res = knn.predict(test_data)

    #return the result predicted
    return res

#predict using decision tree
def do_decision_tree(train_data, y_train, test_data):

    #create decision tree model
    dec_tree = DecisionTreeClassifier()
    #train the data
    dec_tree.fit(train_data, y_train)

    #predict the output for test_data
    res = dec_tree.predict(test_data)
    return res

#predict using logistic
def do_logistic(train_data, y_train, test_data):

    #create logistic regression model
    logistic = LogisticRegression(random_state=0)

    #train the data
    logistic.fit(train_data, y_train)

    #predict the output for test_data
    res = logistic.predict(test_data)
    return res

#predict using svm
def do_svm(train_data, y_train, test_data):

    #create svm model
    clf = svm.SVC()

    #train the data
    clf.fit(train_data, y_train)

    #predict the output for test_data
    res = clf.predict(test_data)
    return res

#data to read 

x = [] # array of review text 
y = [] # array of voted up state
z = [] # array of early access state

#read the data from file -------------
with open('reviews_200.jl', 'rb') as f:#open file
    #every line is json object
    for item in json_lines.reader(f):
        #append the data to variable
        x.append(item['text'])
        y.append(item['voted_up'] )
        z.append(item['early_access'] )

#create vectorizer to extract the features from the text
vectorizer = TfidfVectorizer( max_df=0.3)
```

```
#extract the features from the text
# so every review text is converted to number array
X = vectorizer.fit_transform(x)

#create sequence array of index from 0 to count of review
indices = np.arange(len(x))

#split the index array into train index array and test index array
and the propotion is 0.001
#so the train count is 4995 if the count of review is 5000
#and the test  count is 5
train_id, test_id = train_test_split(indices, test_size=0.1)

#create train data from the train index array
train_x = X[train_id]
train_y = [y[idx] for idx in train_id]
train_z = [z[idx] for idx in train_id]

#select the first text for test
test_text = x[test_id[0]]

#extract the features from the test text
test_x = vectorizer.transform([test_text])
cv = ShuffleSplit(n_splits=5, test_size=0.1, random_state=0)
#output the test text
print('text:\n%.300s\n---------------'%test_text)

#do knn and print the result
print("KNN Regression : ")
test_y = do_knn(train_x, train_y, test_x)
test_z = do_knn(train_x, train_z, test_x)
scores = 
cross_val_score(KNeighborsClassifier(n_neighbors=3,metric=cosine_d
istances), X, y, cv=cv)
print("\t%0.2f accuracy with a standard deviation of %0.2f" % 
(scores.mean(), scores.std()))
print( "\t", vote_txt[int(test_y)], access_txt[int(test_z)])

#do svm classifier and print the result
print( "SVM Regression : ")
test_y = do_svm(train_x, train_y, test_x)
test_z = do_svm(train_x, train_z, test_x)
scores = cross_val_score(svm.SVC(), X, y, cv=cv)
print("\t%0.2f accuracy with a standard deviation of %0.2f" % 
(scores.mean(), scores.std()))
print( "\t", vote_txt[int(test_y)], access_txt[int(test_z)])

#do decision tree classifier and print the result
print( "Decision Tree result : ")
test_y = do_decision_tree(train_x, train_y, test_x)
test_z = do_decision_tree(train_x, train_z, test_x)
scores = cross_val_score(DecisionTreeClassifier(), X, y, cv=cv)
```

```
print("\t%0.2f accuracy with a standard deviation of %0.2f" %
(scores.mean(), scores.std()))
print( "\t", vote_txt[int(test_y)], access_txt[int(test_z)])

#do logistic regression and print the result
print( "Logistics Regression : ")
test_y = do_logistic(train_x, train_y, test_x)
test_z = do_logistic(train_x, train_z, test_x)
scores = cross_val_score(LogisticRegression(random_state=0), X, y,
cv=cv)
print("\t%0.2f accuracy with a standard deviation of %0.2f" %
(scores.mean(), scores.std()))
print( "\t", vote_txt[int(test_y)], access_txt[int(test_z)])
```