Project Progress: Week 3

Women's E-Commerce Clothing Review Analysis

Course: ALY 6020 Predictive Analytics

Instructor: Dr. Marco Montes De Oca

Submitted By,

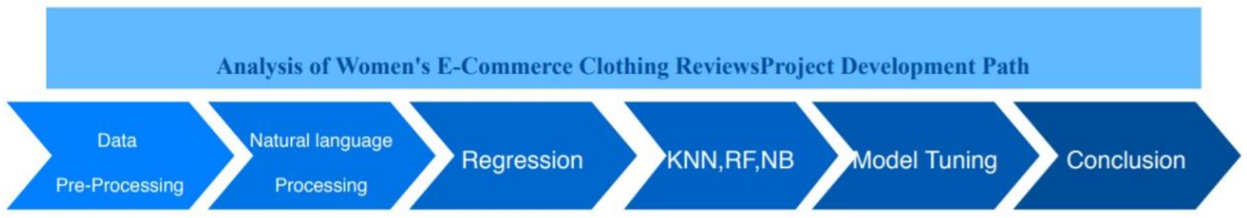
Ashlesha Kshirsagar

Shruti Avinash Pawar

11 June 2020

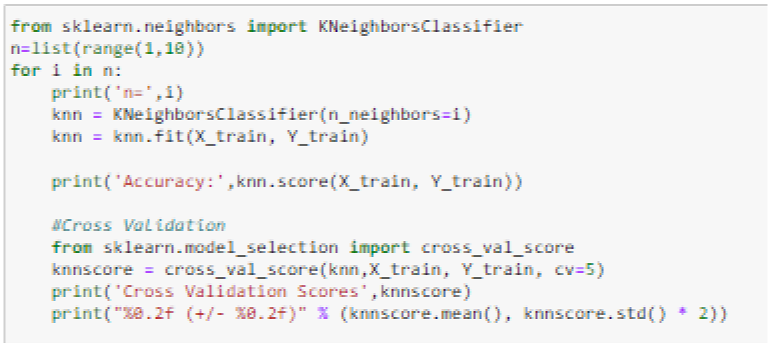
Introduction:

As per the project flow below, in progress report of week 2, we completed the steps from data preprocessing till logistic regression. This week, we have covered KNN and Random forest on the data extracted using NLP.


Analysis of Women's E-Commerce Clothing ReviewsProject Development Path
Data Pre-Processing
Natural language Processing
Regression
KNN,RF,NB
Model Tuning
Conclusion

k-nearest neighbors classifier:

The k-nearest neighbors classifier (k-NN) is a classical supervised method based on statistical data, which maps any feature vector X to the pattern class that appears most frequently among the k-nearest neighbors. The performance of a classifier depends on the interrelationship between sample size, number of features, and classifier complexity. Estimating the accuracy of a classifier is important not only to predict its future prediction accuracy, but also for choosing a good classifier from a given set. (Max Kuhn and Kjell)In training phases of those supervised classifiers, accuracy estimation is helpful for adjusting partial parameters until achieving expected accuracy rate. The accuracy of a classifier is the probability of correctly classifying a random or selected instance. Usually, the absolute accuracy is unknown, cannot be calculated, and must be estimated from given datasets.
In this week we have used KNN classification. we define our classifer, in this case KNN, fit it to our training data and evaluate its accuracy. We'll be using an arbitrary K but we will see later on how cross validation can be used to find its optimal value.



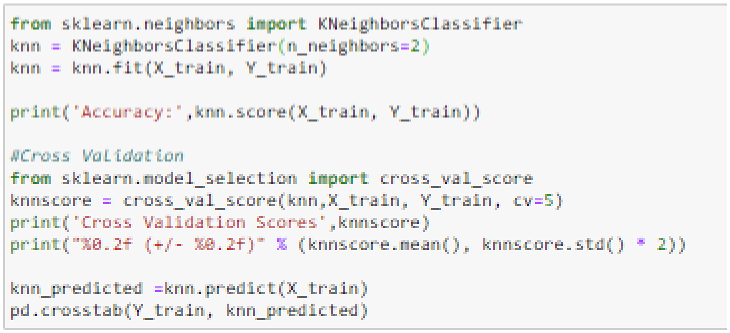
```
from sklearn.neighbors import KNeighborsClassifier
n=list(range(1,10))
for i in n:
    print('n=',i)
    knn = KNeighborsClassifier(n_neighbors=i)
    knn = knn.fit(X_train, Y_train)

    print('Accuracy:',knn.score(X_train, Y_train))

    #Cross Validation
    from sklearn.model_selection import cross_val_score
    knnscore = cross_val_score(knn,X_train, Y_train, cv=5)
    print('Cross Validation Scores',knnscore)
    print("%0.2f (+/- %0.2f)" % (knnscore.mean(), knnscore.std() * 2))
```

Cross-Validation is used for evaluating predictive models by partitioning the original sample into a training set to train the model, and a test set to evaluate it.

In k-fold cross-validation, the original sample is randomly partitioned into k equal size subsamples.scikit-learn comes in handy with its Cross_val_score method. We specify that we are performing 5 folds with the CV =5 parameter and that our scoring metric should be accuracy since we are in a classification setting. From output we got the maximum accuracy at K = 2 which is 87.47%.and accuracy got decreased afterwards due to under-fitting.

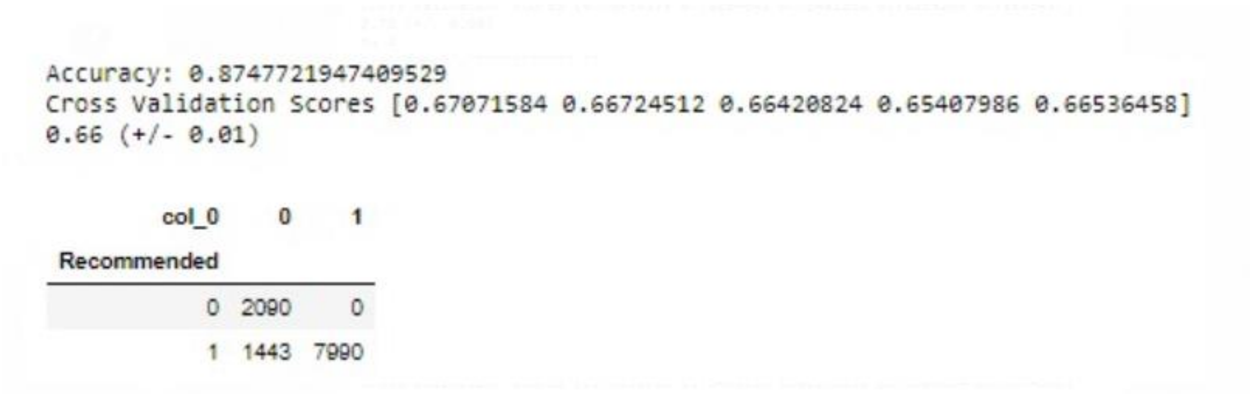


```
from sklearn.neighbors import KNeighborsClassifier
knn = KNeighborsClassifier(n_neighbors=2)
knn = knn.fit(X_train, Y_train)

print('Accuracy:',knn.score(X_train, Y_train))

#Cross Validation
from sklearn.model_selection import cross_val_score
knnscore = cross_val_score(knn,X_train, Y_train, cv=5)
print('Cross Validation Scores',knnscore)
print("%0.2f (+/- %0.2f)" % (knnscore.mean(), knnscore.std() * 2))

knn_predicted =knn.predict(X_train)
pd.crosstab(Y_train, knn_predicted)
```



```
Accuracy: 0.8747721947409529
Cross Validation Scores [0.67071584 0.66724512 0.66420824 0.65407986 0.66536458]
0.66 (+/- 0.01)

        col_0     0     1
  Recommended
            0  2090     0
            1  1443  7990
```

	Actual Yes	Actual No
Recommended No	True Positive	False Positive
	2090	0
Recommend Yes	False Negative	True Negative
	1443	7990

For K=2 we have got accuracy as 87.47%. to increase the accuracy of the model we can check with different K values along with different cross validation values (cv= 5,10 etc).
The cross validation and accuracy decreased across all the combinations for n in KNN due to the smaller data set size. The model still performs consistently with at a very high level.

Random Forest Model:
Random forest is a group of decision trees. These decision trees are fit on different training dataset and the process of picking the dataset is known as sampling with replacement. The output of the random forest model is an average of all the group decision tree output which reduces the bias. However, deciding the optimum number of trees is an important task as higher number of trees would result into higher sample repetitions which also have the potential of causing bias. The number of trees is a hyper parameter in Random forest model, and it can be tuned using cross validation to determine the number.(Brownlee, 2020)

```python
from sklearn import ensemble
rfc = ensemble.RandomForestClassifier(n_estimators=100)
rfc = rfc.fit(X_train, Y_train)
```

```python
test_prediction = rfc.predict(X_test)
print(confusion_matrix(Y_test,test_prediction))
```

We continued with the train and test dataset to fit a Random forest model at three different n_estimators (tree size) 100, 500 and 1000. Below was the output of the Random Forest Models:

Table1: Summary of Random Forest Model output

N_estimator	Accuracy	MAE	MSE	RMSE
100	83.16%	0.16880	0.16886	0.41093
500	83.31%	0.16684	0.16684	0.40845
1000	83.03%	0.16967	0.16967	0.41191

From table 1, we can see that the accuracy at 1000 n_estimators is the least, and highest at 500. The RMSE increased as the tree size increased from 500 to 1000. On the other hand, we can see a drop in RMSE from 100 to 500 and an increase in accuracy. Thus our optimum tree size is somewhere around 500 and we can determine that using tuning.

Further steps of action:

1. Fitting Naïve based model.
2. Tuning of logistic, Naïve based, KNN and Random forest model.
3. Changing the train and test dataset split and checking the impact on accuracy of these models.
4. Conclude by picking the model with highest accuracy.

Bibliography:

- Max Kuhn and Kjell Johnson (2013), Applied Predictive Modeling
- Ritchie Ng K-nearest Neighbors (KNN) Classification Model Retrieved from https://www.ritchieng.com/machine-learning-k-nearest-neighbors-knn/
- Brownlee, J. (2020). How to Develop a Random Forest Ensemble in Python. Retrieved 12 June 2020, from https://machinelearningmastery.com/random-forest-ensemble-in-python/