Design and Implementation of Naïve Bayes

Aditi Helekar

March 6, 2017

Contents:

Abstract

A brief introduction to Naïve Bayes is provided and is implemented in Python. The implementation is demonstrated to identify the appropriate prediction on Testing data by training the models with Digit Recognition dataset and Amazon Baby products reviews dataset. There is a comparison mentioned about the different classifier available in scikit-learn library.

1. Problem Statement:

- Implement or train a Naive Bayes classifier using the Digit Recognition and Amazon Reviews datasets.
- Submit your implementation and analysis results showing the various settings/parameters you experimented with.
- Include a section in your report wrapping up the supervised learning assignments. At this point you have experimented with 6 different SL approaches to classification in two domains. Compare the results you've obtained. Which approach performed best? Consider various aspects of performance (accuracy, complexity, precision, recall, confusion matrix,...)

2. Approach:

I have implemented Naïve Bayes in Python using scikit-learn library. Scikit-learn is library designed for implementing Machine learning techniques in Python. Scikit-learn provides libraries SVM.

Naïve Bayes:

Naive Bayes methods are a set of supervised learning algorithms based on applying Bayes' theorem with the "naive" assumption of independence between every pair of features. In spite of their apparently over-simplified assumptions, naive Bayes classifiers have worked quite well in many real-world situations, famously document classification and spam filtering. They require a small amount of training data to estimate the necessary parameters. Naive Bayes learners and classifiers can be extremely fast compared to more sophisticated methods. The decoupling of the class conditional feature distributions means that each distribution can be independently estimated as a one dimensional distribution. This in turn helps to alleviate problems stemming from the curse of dimensionality.

GaussianNB implements the Gaussian Naive Bayes algorithm for classification. The likelihood of the features is assumed to be Gaussian.

class sklearn.naive_bayes.GaussianNB(*priors=None*)

MultinomialNB implements the naive Bayes algorithm for multinomially distributed data, and is one of the two classic naive Bayes variants used in text classification (where the data are typically represented as word vector counts, although tf-idf vectors are also known to work well in practice).

class sklearn.naive_bayes.MultinomialNB(*alpha=1.0*, *fit_prior=True*, *class_prior=None*)

alpha: Additive (Laplace/Lidstone) smoothing parameter (0 for no smoothing).

fit_prior: Whether to learn class prior probabilities or not. If false, a uniform prior will be used.

class_prior : Prior probabilities of the classes. If specified the priors are not adjusted according to the data.

BernoulliNB implements the naive Bayes training and classification algorithms for data that is distributed according to multivariate Bernoulli distributions; i.e., there may be multiple features but each one is assumed to be a binary-valued (Bernoulli, boolean) variable. Therefore, this class requires samples to be represented as binary-valued feature vectors; if handed any other kind of data, a BernoulliNB instance may binarize its input (depending on the binarize parameter).

class sklearn.naive_bayes.BernoulliNB(*alpha=1.0*, *binarize=0.0*, *fit_prior=True*, *class_prior=None*)

alpha: Additive (Laplace/Lidstone) smoothing parameter (0 for no smoothing).

binarize : Threshold for binarizing (mapping to booleans) of sample features. If None, input is presumed to already consist of binary vectors.

fit_prior: Whether to learn class prior probabilities or not. If false, a uniform prior will be used.

class_prior : Prior probabilities of the classes. If specified the priors are not adjusted according to the data.

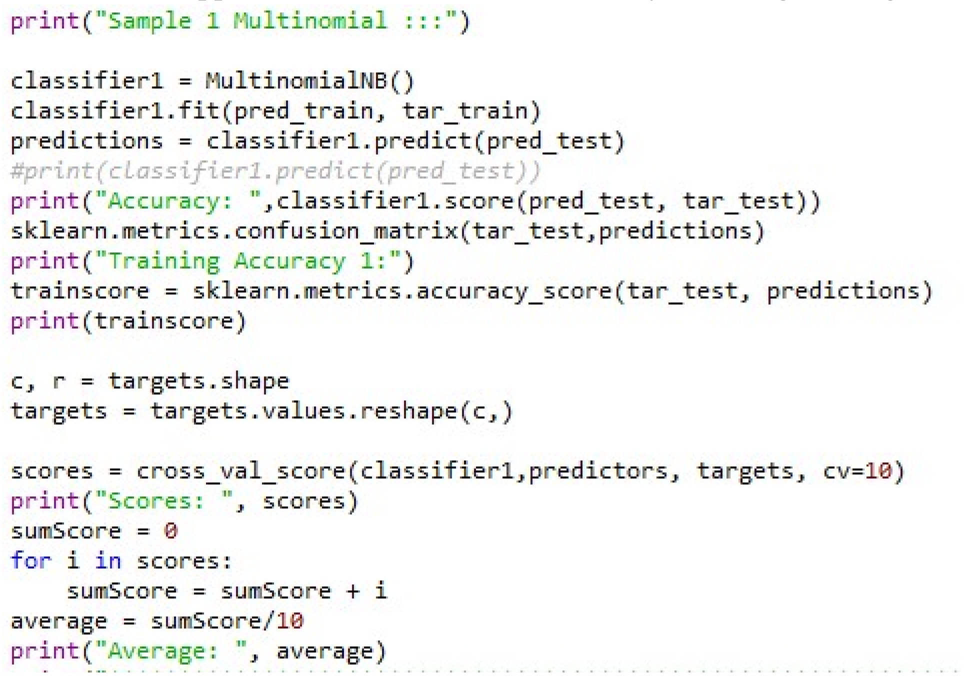
3. Implementation:

The basic algorithm used for both implementations is described below:
1. Read the csv file using pandas.io library and store it as a Dataframe.
2. Classify the attributes as samples and features to apply it to MultinomialNB classifier. Features are also referred as predictors and samples as responses.
3. Create an object of Classifier imported from scikit learn library.
4. Using fit function, fit the dataframe to tree model. Fit function finds patterns in the given dataset.
5. For cross-validation, divide the training dataset in train and test data using train_test_split function.
6. Predict the behavior of model using predict function with test data.
7. After cross validation use the test dataset and predict again.
8. Calculate accuracy using score function.

For Amazon Dataset, I first removed all the stopwords using the nltk toolkit. Later on, I converted the words to vectors and found the occurrences of each meaningful words using CountVectorizer. Multinomial Naïve Bayes is suitable to discrete features, so applying it to both model gave better results than other models.

<u>Naïve Bayes :</u>

I have verified the results by modifying the model from Bernoulli, Gaussian and Multinomial. Below code snippet shows the outline of Naïve Bayes for Digit Recognition Dataset:



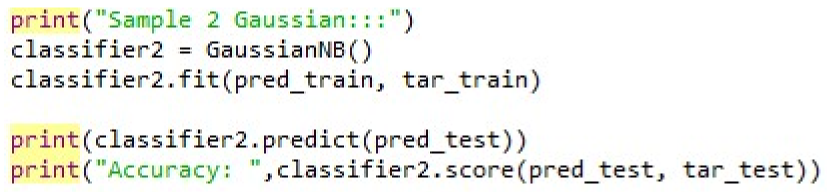
```
print("Sample 1 Multinomial :::")

classifier1 = MultinomialNB()
classifier1.fit(pred_train, tar_train)
predictions = classifier1.predict(pred_test)
#print(classifier1.predict(pred_test))
print("Accuracy: ",classifier1.score(pred_test, tar_test))
sklearn.metrics.confusion_matrix(tar_test,predictions)
print("Training Accuracy 1:")
trainscore = sklearn.metrics.accuracy_score(tar_test, predictions)
print(trainscore)

c, r = targets.shape
targets = targets.values.reshape(c,)

scores = cross_val_score(classifier1,predictors, targets, cv=10)
print("Scores: ", scores)
sumScore = 0
for i in scores:
    sumScore = sumScore + i
average = sumScore/10
print("Average: ", average)
```

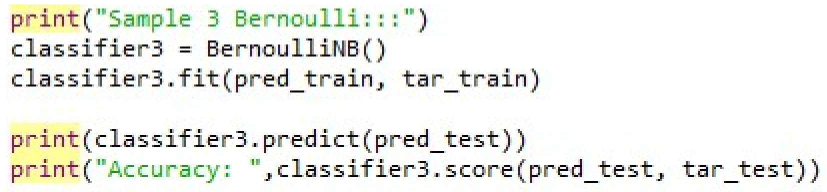
The above code shows the implementation of MultinomialNB classifier.



```
print("Sample 2 Gaussian:::")
classifier2 = GaussianNB()
classifier2.fit(pred_train, tar_train)

print(classifier2.predict(pred_test))
print("Accuracy: ",classifier2.score(pred_test, tar_test))
```

The above code shows GaussianNB implementation.



```
print("Sample 3 Bernoulli:::")
classifier3 = BernoulliNB()
classifier3.fit(pred_train, tar_train)

print(classifier3.predict(pred_test))
print("Accuracy: ",classifier3.score(pred_test, tar_test))
```

The above code shows BernoulliNB implementation.

4. Observations:

Support Vector Machines :

1. For Digit Recognition dataset:

```
Sample 1 Multinomial :::
Accuracy:  0.922171353826
Training Accuracy 1:
0.922171353826
Scores:  [ 0.9038961   0.91168831  0.92708333  0.88802083  0.91927083  0.92167102
  0.91884817  0.91029024  0.90501319  0.91777188]
Average:  0.912355391477
************************************************************
Sample 2 Gaussian:::
[6 8 3 ..., 8 8 9]
Accuracy:  0.748855461086
Sample 3 Bernoulli:::
[6 7 3 ..., 8 8 9]
Accuracy:  0.864617396991
Testing::::
Sample 1 Multinomial :::
[1 1 3 ..., 8 9 8]
Accuracy:  0.889198218263
Sample 2 Gaussian:::
[1 8 3 ..., 8 8 8]
Accuracy:  0.786191536748
Sample 3 Bernoulli:::
[6 7 3 ..., 8 8 9]
Accuracy:  0.864617396991
```

2. For Amazon dataset:

   The output with MultinomialNB and default features.

```
[5 5 5 ..., 5 5 5]
Accuracy:
0.549753559693
```

5. Conclusion:

The MultinomialNB classifier for classification with discrete features (e.g., word counts for text classification). The multinomial distribution normally requires integer feature counts. However, in practice, fractional counts such as tf-idf may also work. Like MultinomialNB, this classifier is suitable for discrete data. The difference is that while MultinomialNB works with occurrence counts, BernoulliNB is designed for binary/boolean features. Gaussian Naive Bayes (GaussianNB)Can perform online updates to model parameters via partial_fit method.

For Digit recognition dataset:

1. It was found that Multinomial Naïve Bayes gave better accuracy than the Bernoulli Naive Bayes. The least accuracy was seen in Gaussian Naïve Bayes.
2. When we try to change the fit_prior parameter to false the accuracy drops as it will use uniform prior rather than learning class probabilities prior.

For Amazon dataset:

1. It was observed that Multinomial Naïve Bayes gave better accuracy.
2. When we try to change the fit_prior parameter to false the accuracy drops.
3. If we try to change alpha value the additive smoothing parameter the accuracy decreases.

6. Report on Supervised Learning techniques:

1. Digit Recognition dataset:

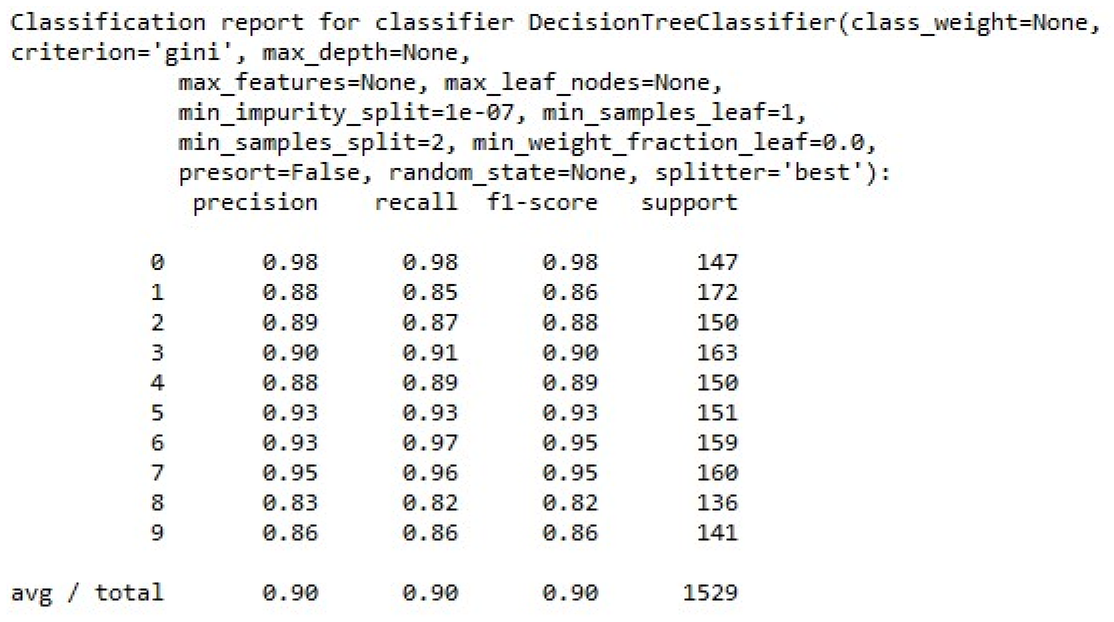
a. Accuracy:

The below table shows the training and testing accuracy for Digit recognition dataset for all the supervised learning techniques.

Algorithm	Training Accuracy(%)	Testing Accuracy(%)
Decision Trees	90.45	85.41
Neural Networks	97.18	94.59
K- Neareat Neighbors	98.29	97.88
Ada Boost	98.62	96.93
Support Vector Machines	98.23	96.38
Naïve Bayes	91.89	88.91

b. Classification report:

1. Decision trees:

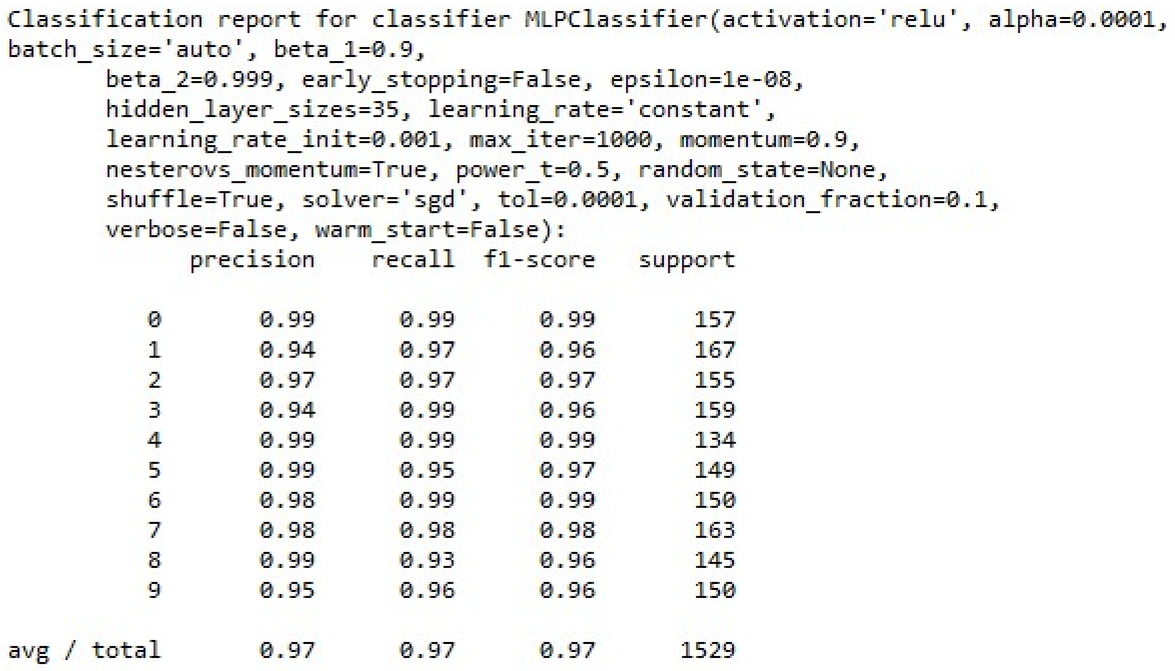


```
Classification report for classifier DecisionTreeClassifier(class_weight=None,
criterion='gini', max_depth=None,
            max_features=None, max_leaf_nodes=None,
            min_impurity_split=1e-07, min_samples_leaf=1,
            min_samples_split=2, min_weight_fraction_leaf=0.0,
            presort=False, random_state=None, splitter='best'):
             precision    recall  f1-score   support

          0       0.98      0.98      0.98       147
          1       0.88      0.85      0.86       172
          2       0.89      0.87      0.88       150
          3       0.90      0.91      0.90       163
          4       0.88      0.89      0.89       150
          5       0.93      0.93      0.93       151
          6       0.93      0.97      0.95       159
          7       0.95      0.96      0.95       160
          8       0.83      0.82      0.82       136
          9       0.86      0.86      0.86       141

avg / total       0.90      0.90      0.90      1529
```

2. **Neural Networks:**

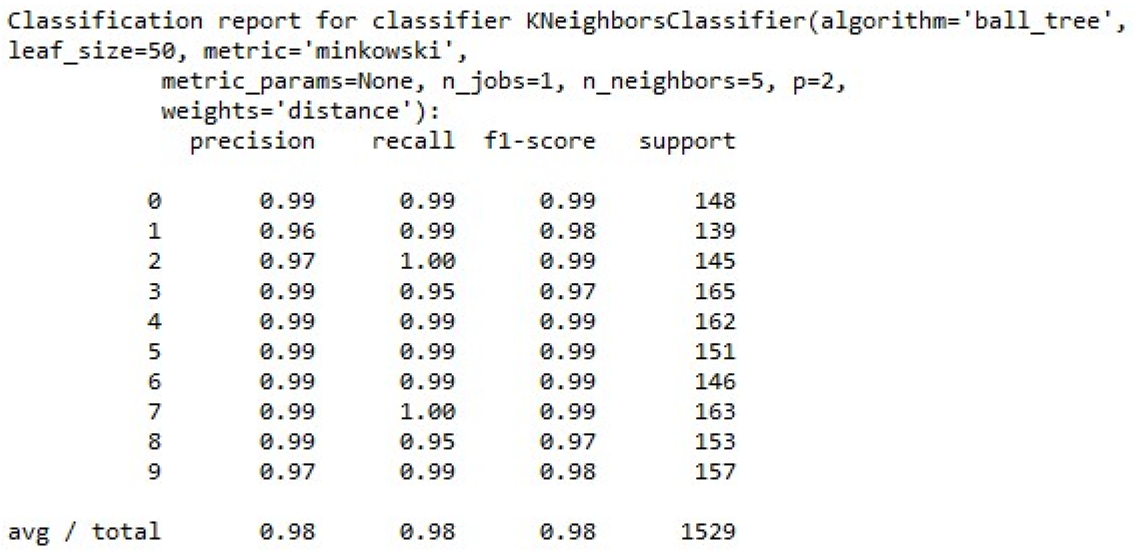


```
Classification report for classifier MLPClassifier(activation='relu', alpha=0.0001,
batch_size='auto', beta_1=0.9,
       beta_2=0.999, early_stopping=False, epsilon=1e-08,
       hidden_layer_sizes=35, learning_rate='constant',
       learning_rate_init=0.001, max_iter=1000, momentum=0.9,
       nesterovs_momentum=True, power_t=0.5, random_state=None,
       shuffle=True, solver='sgd', tol=0.0001, validation_fraction=0.1,
       verbose=False, warm_start=False):
             precision    recall  f1-score   support

          0       0.99      0.99      0.99       157
          1       0.94      0.97      0.96       167
          2       0.97      0.97      0.97       155
          3       0.94      0.99      0.96       159
          4       0.99      0.99      0.99       134
          5       0.99      0.95      0.97       149
          6       0.98      0.99      0.99       150
          7       0.98      0.98      0.98       163
          8       0.99      0.93      0.96       145
          9       0.95      0.96      0.96       150

avg / total       0.97      0.97      0.97      1529
```

3. **K Nearest Neighbors:**

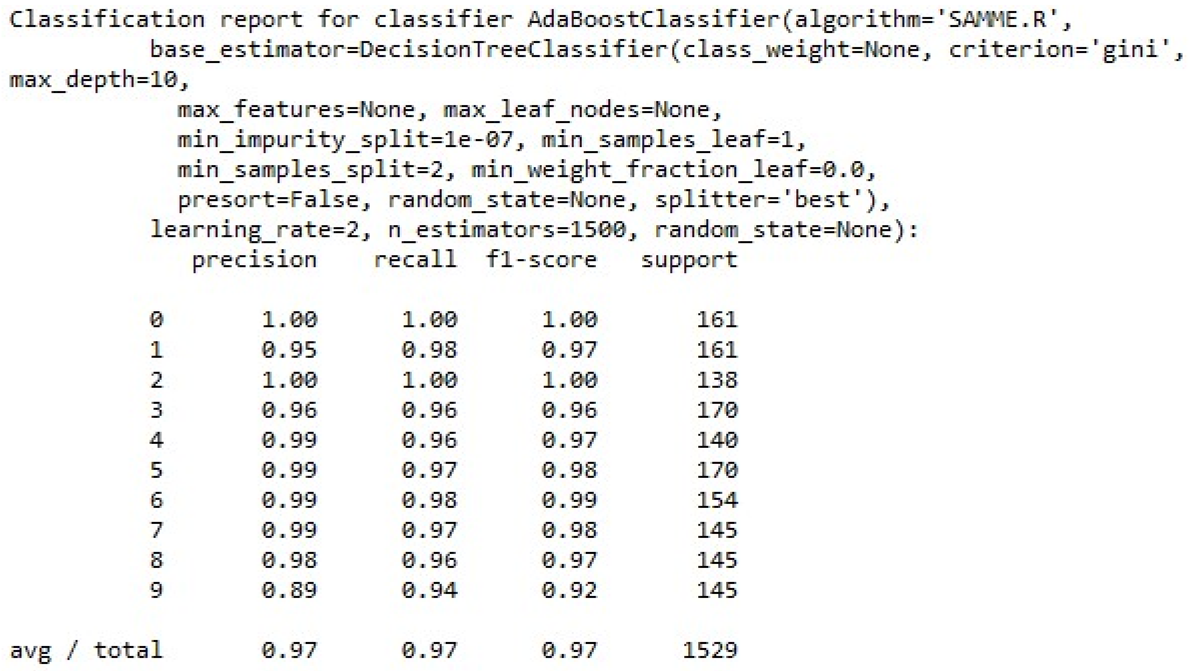


```
Classification report for classifier KNeighborsClassifier(algorithm='ball_tree',
leaf_size=50, metric='minkowski',
           metric_params=None, n_jobs=1, n_neighbors=5, p=2,
           weights='distance'):
             precision    recall  f1-score   support

          0       0.99      0.99      0.99       148
          1       0.96      0.99      0.98       139
          2       0.97      1.00      0.99       145
          3       0.99      0.95      0.97       165
          4       0.99      0.99      0.99       162
          5       0.99      0.99      0.99       151
          6       0.99      0.99      0.99       146
          7       0.99      1.00      0.99       163
          8       0.99      0.95      0.97       153
          9       0.97      0.99      0.98       157

avg / total       0.98      0.98      0.98      1529
```

4. **Ada Boost:**

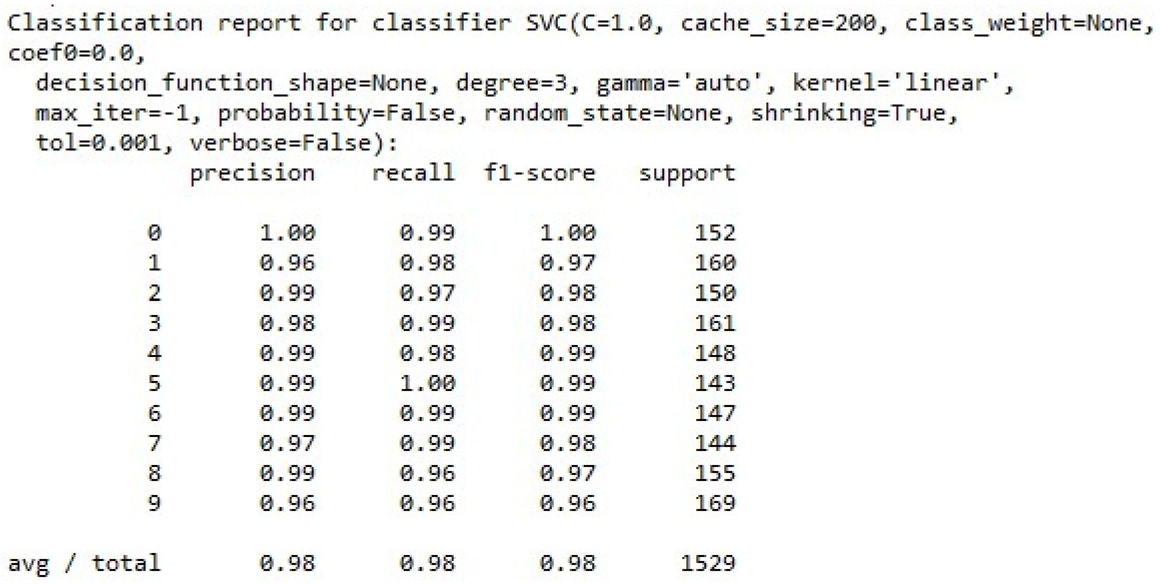


```
Classification report for classifier AdaBoostClassifier(algorithm='SAMME.R',
          base_estimator=DecisionTreeClassifier(class_weight=None, criterion='gini',
max_depth=10,
            max_features=None, max_leaf_nodes=None,
            min_impurity_split=1e-07, min_samples_leaf=1,
            min_samples_split=2, min_weight_fraction_leaf=0.0,
            presort=False, random_state=None, splitter='best'),
          learning_rate=2, n_estimators=1500, random_state=None):
             precision    recall  f1-score   support

          0       1.00      1.00      1.00       161
          1       0.95      0.98      0.97       161
          2       1.00      1.00      1.00       138
          3       0.96      0.96      0.96       170
          4       0.99      0.96      0.97       140
          5       0.99      0.97      0.98       170
          6       0.99      0.98      0.99       154
          7       0.99      0.97      0.98       145
          8       0.98      0.96      0.97       145
          9       0.89      0.94      0.92       145

avg / total       0.97      0.97      0.97      1529
```

5. **Support Vector Machines:**

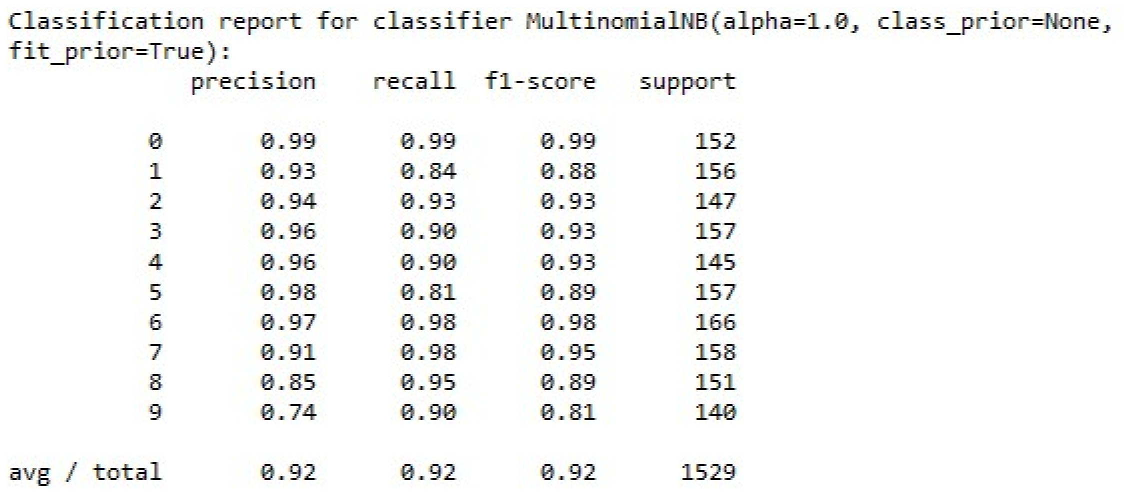


```
Classification report for classifier SVC(C=1.0, cache_size=200, class_weight=None,
coef0=0.0,
  decision_function_shape=None, degree=3, gamma='auto', kernel='linear',
  max_iter=-1, probability=False, random_state=None, shrinking=True,
  tol=0.001, verbose=False):
             precision    recall  f1-score   support

          0       1.00      0.99      1.00       152
          1       0.96      0.98      0.97       160
          2       0.99      0.97      0.98       150
          3       0.98      0.99      0.98       161
          4       0.99      0.98      0.99       148
          5       0.99      1.00      0.99       143
          6       0.99      0.99      0.99       147
          7       0.97      0.99      0.98       144
          8       0.99      0.96      0.97       155
          9       0.96      0.96      0.96       169

avg / total       0.98      0.98      0.98      1529
```

6. **Naïve Bayes:**



```
Classification report for classifier MultinomialNB(alpha=1.0, class_prior=None,
fit_prior=True):
             precision    recall  f1-score   support

          0       0.99      0.99      0.99       152
          1       0.93      0.84      0.88       156
          2       0.94      0.93      0.93       147
          3       0.96      0.90      0.93       157
          4       0.96      0.90      0.93       145
          5       0.98      0.81      0.89       157
          6       0.97      0.98      0.98       166
          7       0.91      0.98      0.95       158
          8       0.85      0.95      0.89       151
          9       0.74      0.90      0.81       140

avg / total       0.92      0.92      0.92      1529
```

c. Conclusion:

From the accuracy table, we can say that for Digit recognition dataset K- Nearest Neighbors gives the best accuracy whereas decision trees gave the least accuracy. The principle behind nearest neighbor methods is to find a predefined number of training samples closest in distance to the new point, and predict the label from these. The number of samples can be a user-defined constant (k-nearest neighbor learning), or vary based on the local density of points (radius-based neighbor learning). The distance can, in general, be any metric measure: standard Euclidean distance is the most common choice. Neighbors-based methods are known as non-generalizing machine learning methods, since they simply “remember” all of its training data (possibly transformed into a fast indexing structure such as a Ball Tree or KD Tree.).

1. In K nearest neighbor technique, we found that as we increase the number of neighbors the accuracy decreases which means that to get precise result , number of neighbors should be as low as possible.
2. Distance metric Euclidean distance gives better results than Manhattan distance.
3. Leaf size reduces the memory usage so smaller leaf_size is advisable.

We have used Ball tree approach which partitions data in a series of nesting hyper-spheres. This makes tree construction more costly than that of the KD tree, but results in a data structure which can be very efficient on highly-structured data, even in very high dimensions.

From the classification report we can see that it builds a text report showing the main classification metrics. So it gives the precision for each classification metric like KNN gives 0.98 precision for identifying the sample as 0. The recall gives the ratio of number of true positives to the number of false negatives. That’s the ability of classifier to find all positives.

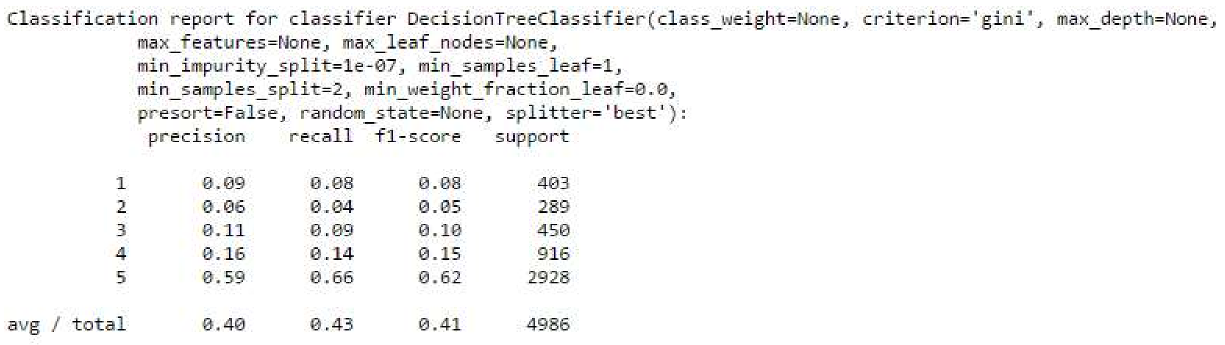
2. Amazon Dataset:

a. Accuracy:

Algorithm	Testing Accuracy(%)
Decision Trees	42.81
Neural Networks	56.23
K- Neareat Neighbors	59
Ada Boost	57.9
Support Vector Machines	60
Naïve Bayes	54.97

b. Classification report:

1. Decision Trees:

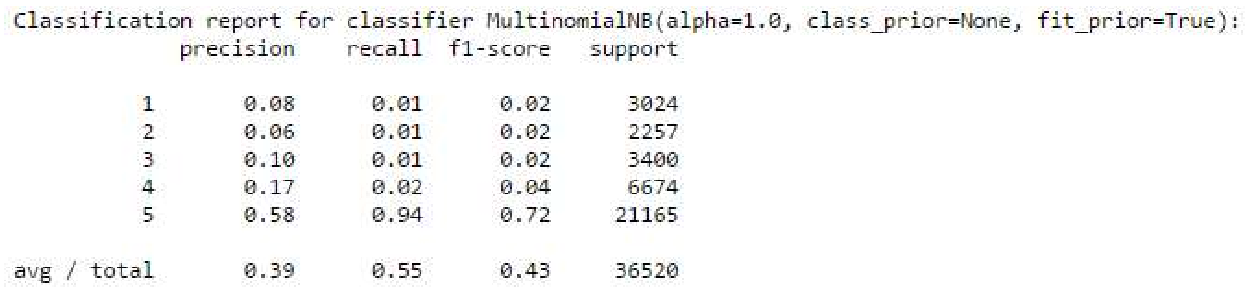


```
Classification report for classifier DecisionTreeClassifier(class_weight=None, criterion='gini', max_depth=None,
            max_features=None, max_leaf_nodes=None,
            min_impurity_split=1e-07, min_samples_leaf=1,
            min_samples_split=2, min_weight_fraction_leaf=0.0,
            presort=False, random_state=None, splitter='best'):
             precision    recall  f1-score   support

          1       0.09      0.08      0.08       403
          2       0.06      0.04      0.05       289
          3       0.11      0.09      0.10       450
          4       0.16      0.14      0.15       916
          5       0.59      0.66      0.62      2928

avg / total       0.40      0.43      0.41      4986
```

2. Naïve Bayes:



```
Classification report for classifier MultinomialNB(alpha=1.0, class_prior=None, fit_prior=True):
             precision    recall  f1-score   support

          1       0.08      0.01      0.02      3024
          2       0.06      0.01      0.02      2257
          3       0.10      0.01      0.02      3400
          4       0.17      0.02      0.04      6674
          5       0.58      0.94      0.72     21165

avg / total       0.39      0.55      0.43     36520
```

From the accuracy table, we can say that for Amazon dataset K- Nearest Neighbors gives the best accuracy whereas decision trees gave the least accuracy.

7. References:

1. Naïve Bayes, http://scikit-learn.org/stable/modules/naive_bayes.html
2. Bernoulli Classifier, http://scikit-learn.org/stable/modules/generated/sklearn.naive_bayes.BernoulliNB.html#sklearn.naive_bayes.BernoulliNB
3. Multinomial Classifier, http://scikit-learn.org/stable/modules/generated/sklearn.naive_bayes.MultinomialNB.html#sklearn.naive_bayes.MultinomialNB
4. Gaussian Classifier, http://scikit-learn.org/stable/modules/generated/sklearn.naive_bayes.GaussianNB.html#sklearn.naive_bayes.GaussianNB
5. Cross-validation, http://scikit-learn.org/stable/modules/cross_validation.html
6. Classification report, http://scikit-learn.org/stable/modules/generated/sklearn.metrics.classification_report.html