Comparison & Analysis Of Various Machine Learning Classifiers And Naïve Approach To Select Best Model

Harshil Shah , Naman Shah & Bhavik Mistry
(All Are On-Campus Developers)

December 12th, 2015

Contents

1 Abstract **1**

2 Introduction **1**

3 Various Classifiers **2**

4 Results And Analysis **3**

5 Naïve Approach To Select Best Classifier Model **4**

6 Conclusion & Future Work **8**

7 Team Member's Responsibility **8**

References **9**

Appendices **10**

A Parsing Data **10**

B Different Classifiers **12**

C Random Partitioning Dataset **13**

D Naive Approach - Selecting Best Model **14**

1 Abstract

In this report, we have tried to present an empirical analysis of multiple supervised learning technologies : SVM, Linear Discriminant Analysis(LDA), Quadratic Discriminant Analysis(QDA), k-Nearest Neighbors, Naive Bayes. We have examined algorithms' effect on different types of dataset on basis of different criteria. By evaluating different learning algorithms using many performance criteria, we have also tried to come up with naive way to recommend best learning methodology according to naive client's requirements of classification and/or dataset's behaviour.

2 Introduction

There are so many very successful and "latest" machine learning classifiers available to use regarding dataset classification problem. [5] As far as the report goes, some of them are examined and explained in upcoming sections. For experimental purpose, different datasets were used. Datasets behaviour are different from each others - binary vs multi-class classifications, number of instances, number of features. Datasets are taken from UCI Machine Learning Repository & information is shown in below table. [10]

Index	**Dataset**	**Instances**	**Features**	**Classification**
1	Letter Recognition	20000	16	Binary
2	UCI HARS	10299	56	Multi-Class
3	Adult	30164	104	Binary
4	Internet Ads	2359	1558	Binary
5	CoverType	581012	54	Multi-Class
6	Farm-Ads	4143	54877	Binary

Table 1: Dataset Used

Considering different classifiers, How do one can decide what machine learning algorithm to choose for given problem or dataset? [8]Selection of any machine learning algorithm to $classify$ "data" depends on various factors like - behaviour of data, needed accuracy (higher/approximation), execution time of algorithm, RMSE restriction etcetera. [1] If it is possible to automate selection of algorithm, then we can speed up analysis and classification process for any type of data. In this project, we are trying to come up with model which can select best algorithm considering above mentioned criteria. [4]

3 Various Classifiers

We have measure different performance metrics by applying following algorithm on above mentioned datasets.

- **Linear Discriminant Analysis (LDA)** :- This method is used in dimensionality reduction as well as classification. It tries to find linear combination of features that separates two or more classes of objects better than other combination. This combination is further used as linear classifier. [14]

  This algorithm assumes that conditional probability of a sample being in one of the various class is normally distributed with specific mean and covariance. Under this assumption, Bayes optimal solution is to predict points as being from the second class if log of likelihood ratios is below some threshold.
  Advantages:-

  – This algorithm gives best outputs when possible outcome classes have the common covariance and all independent variables follows the multivariate normal distribution. [2]

- **Quadratic Discriminant Analysis (QDA)** :- QDA is generalize form of LDA. QDA assumes that measurements from each class are normally distributed. Key difference between QDA and LDA is that, in QDA no assumption is made about the covariance matrices, leading to quadratic decision boundary. [15]
  Advantages:-

  – This algorithm gives best outputs when possible outcome classes does not have the common covariance and all independent variables follows the multivariate normal distribution. [2]

- **k-Nearest Neighbours (k-NN)** :- This is the simplest method for classification. This method classifies a sample based on majority votes from its neighbors, with the object being assigned to the class most common among its k nearest neighbors. Commonly used distance metric is Euclidean distance. [13]

  Key things to consider :

  - For multi-class classification, k value should not be multiple of number of distinct classes.
  - Finding distance between samples is complex and time consuming process. Hence, k value should be small enough as classification can be completed fast.

- **Naive Bayes** :- Naive Bayes classifiers are supervised machine learning algorithms based on applying Bayes' theorem with assumption that any two feature pairs are totally independent from each other. [12] [3]

  These are probability models, which are combined with decision rule, works as a classifier. One common rule is to choose the maximum value. Naive Bayes classifier computes conditional probability for sample being in every possible

class and assigns label having maximum value for conditional probability.
Advantages:-

- These algorithms work best in case where data holds the conditional independence. They converge quicker than other algorithms. These are best fit in case where someone wants quick output and easy solutions. [2]

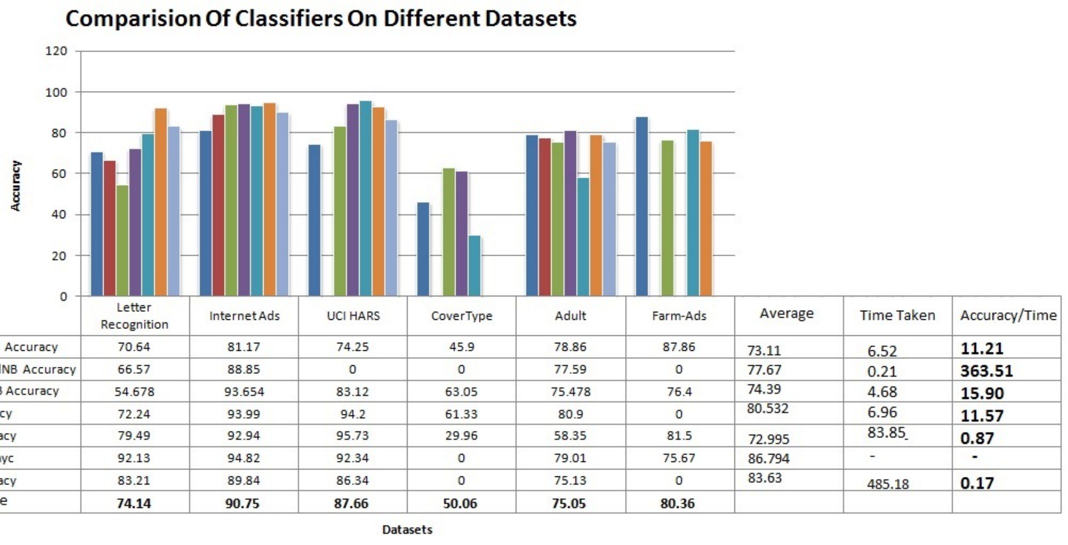
- **Gaussian Naive Bayes** :- This algorithm is used when data is following Gaussian distribution. Generally, continuous data associated with each class are assumed to be distributed according to Gaussian distribution.

- **Multinomial Naive Bayes** :- This algorithm is applied to multinomially distributed data. For n different trials, if there are k possible outcomes, then those data are distributed multinomially. This method is helpful in classifying categorical data.

- **Bernoulli Naive Bayes** :- This algorithm is helpful for the data that is distributed according to multivariate Bernoulli distribution, i.e., there may be multiple features but each one is assumed to be binary-valued. This class requires samples to be represented as binary-valued feature vectors.

- **Support Vector Machines (SVM)** :- This algorithm tries to identify a separating hyperplane that has largest distance between nearest data-points of any class.
  Advantages:-
  - These algorithms guarantee about the over-fitting. Moreover, with appropriate kernel, they perform well even if data is not linearly separable.
  - These algorithms tend to have higher accuracies. [9]

4 Results And Analysis



	Letter Recognition	Internet Ads	UCI HARS	CoverType	Adult	Farm-Ads	Average	Time Taken	Accuracy/Time
🟦 GuassianNB Accuracy	70.64	81.17	74.25	45.9	78.86	87.86	73.11	6.52	**11.21**
🟥 MultinomialNB Accuracy	66.57	88.85	0	0	77.59	0	77.67	0.21	**363.51**
🟩 BernoulliNB Accuracy	54.678	93.654	83.12	63.05	75.478	76.4	74.39	4.68	**15.90**
🟪 LDA Accuracy	72.24	93.99	94.2	61.33	80.9	0	80.532	6.96	**11.57**
🟦 QDA Accuracy	79.49	92.94	95.73	29.96	58.35	81.5	72.995	83.85	**0.87**
🟧 kNN Accurayc	92.13	94.82	92.34	0	79.01	75.67	86.794	-	**-**
🟦 SVM Accuracy	83.21	89.84	86.34	0	75.13	0	83.63	485.18	**0.17**
Average	**74.14**	**90.75**	**87.66**	**50.06**	**75.05**	**80.36**			

Figure 1: Accuracy

All the algorithms discussed in section 3 were applied on varity of datasets and results were generated and plotted on graph as shown in figure 1. Average observation was made as to understand behaviour of classifiers only.

All datasets seem to be converged perfectly except CoverType dataset. CoverType datasets is not easy to get converged by above classifiers. As shown, all datasets are working very differently with different classifiers. All Naive Bayes are simple probabilistic models, which implies that accuracy using this classifiers on any dataset will not be very much higher and not lower either. Another observation was made regarding computation time of each classifiers and *Accuracy/Computation Time* matrix was taken to have good reason about each classifiers. Having higher ratio of such matrix depicts good performance of classifier. As shown in graph, Multinomial Naive Bayes gives reasonably best performance amongst all the classifiers. On contrary, Discriminant Analysis tries to find best linear combination of features for classification. This behaviour of Discriminant Analysis, implies good accuracy compare to Naive Bayes but higher computation time too. Using Support Vectors, classification seems to be done very efficiently than any other classifiers, but execution time is also much higher.

In the nutshell, all classifiers are working very well on every datasets, but there were trade-offs between higher accuracy and lower converging time. In the upcoming section, we are trying to come up with generalize idea regarding which classifier algorithm needs to be chosen to have good performance.

5 Naïve Approach To Select Best Classifier Model

Now, choosing what algorithm / linear classifier will work best for given dataset, is *always* a question. Scikit Learn has provided one mind view(consider figure 2) regarding selection of classifier model. [7] But this model gives very abstract idea, further more, in real time application knowing data behaviour in all dataset is not possible, and model is not possible to use as data's behaviour is not always known beforehand. [1][4][2]

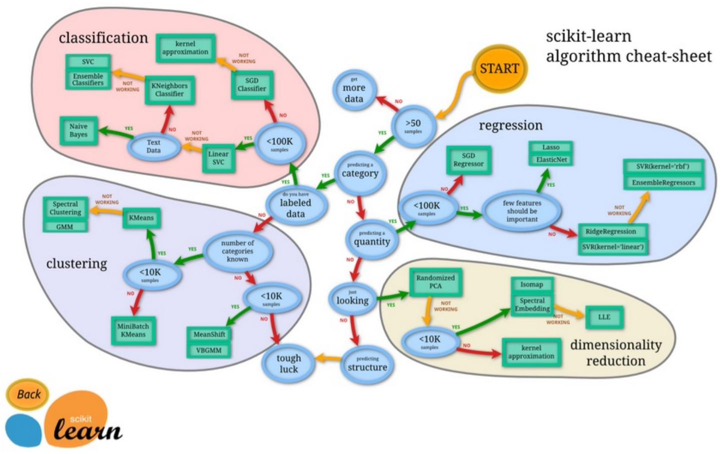


Figure 2: Choosing The Right Estimator - Mind View (Scikit Learn Model)

One another possible way to predict best model can be find out similarity between dataset : if some algorithm is working best for *one dataset* then that algorithm will also work better for *similar dataset* to previous one. As far as this project goes, finding similarity between dataset will require plenty of behavioural features and it seems out of scope in this project. Naive, yet credible model is proposed regarding selection. Classifier's performance can be evaluated on various performance indices: Threshold Metrics, Ordering Matrix, Probability Matrix, Timing Matrix. As any dataset can be multi-class classification problem, it is absolutely not needed to find out ordering matrix (ex. ROC Area). In this report, as threshold matrix *accuracy* of classifier, as Probability matrix *RMSE Error* of classification & as Timing matrix *time required to converge* is taken as performance variables. [2] To understand behavior of different performance parameters according to the different datasets, some observations were made. (consider figure 3 & 4)

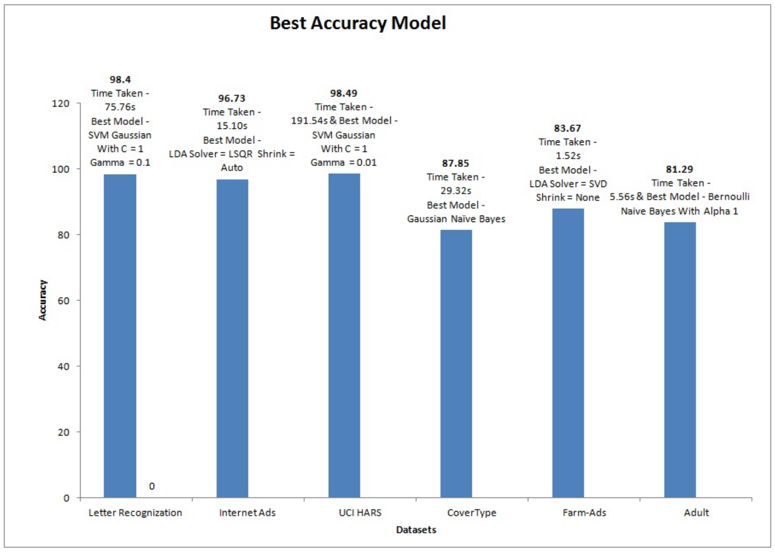


Figure 3: Best Model According To Accuracy For All Dataset

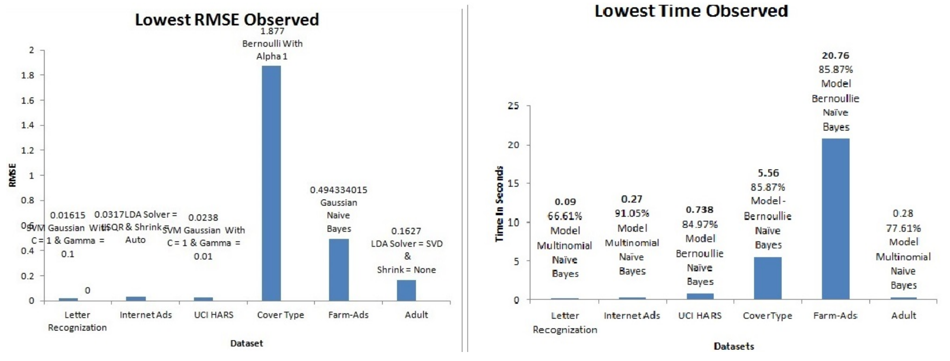


Figure 4: Best Model According To Lowest Time For Converging & Lowest Possible RMSE For All Dataset

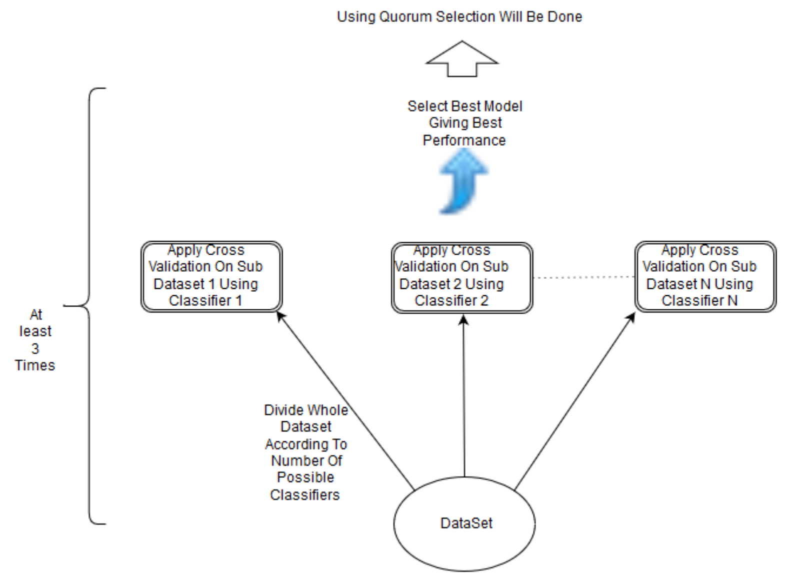
User's requirements can be very different - some wants accuracy as higher being time as thread off, some does want reasonable accuracy quickly, while some wants RMSE to be very lower.

Approach 1 : Shooting In The Blind (Only For Time Matrix)

As shown in above graphs, no particular generalized model is observed regarding performance criteria - Accuracy & RMSE. Although, for low timing criteria, Naive Bayes' variance seems to be working well for any dataset. As far as timing criteria is concerned, prediction regarding model selection will be to select Naive Bayes. This naive method is totally ***shooting in the blind***. Model selection cannot rely on this kind of method.

Approach 2 : Divide And Observe

Another approach should be defined in such a way that "understanding" of data is not required. But in that case, For selection of any algorithm, possible approach will be to compare every algorithm with each other and to evaluate performance. Then based on performance, best model will be selected. But comparison of each algorithm with consideration of all internal possible parameter on whole dataset is time intensive task (possible complexity : $O(N^2)$). To coup with time, dataset can be divided in N number of unbiased random sub-dataset and on each individual dataset, individual classifier can be evaluated by cross-validation. Based on each classifiers' performance, best model can be stated. This selection should be performed at least 3 times and based on quorum decision best model should be selected.(possible complexity : $k * O(N)$, $k >= 3$ & $< N$) (consider fig 5)



Using Quorum Selection Will Be Done
Select Best Model Giving Best Performance
Apply Cross Validation On Sub Dataset 1 Using Classifier 1
Apply Cross Validation On Sub Dataset 2 Using Classifier 2
Apply Cross Validation On Sub Dataset N Using Classifier N
At least 3 Times
Divide Whole Dataset According To Number Of Possible Classifiers
DataSet

Figure 5: Approach 2 To Select Best Model

In this approach, partition of data depends on number of candidate participated classifiers. Usually, that number is very high so each sub-samples will be very small. Giving model selection based on few example instance can be, sometimes, misguiding. This approach is also not feasible as partitioning depends on number of candidate participated classifiers.

Approach 3 : Naive, Yet Credible Approach

As mentioned in above approach, instead of doing partitioning based on number of candidate participated classifiers, dataset will be randomly partition in sub-dataset. All classifiers will be evaluated on subset only and best classifier will be selected. This process will need to be done at least 3-5 times, and then frequently selected model will be predicated as best model regarding appropriate performance criteria.

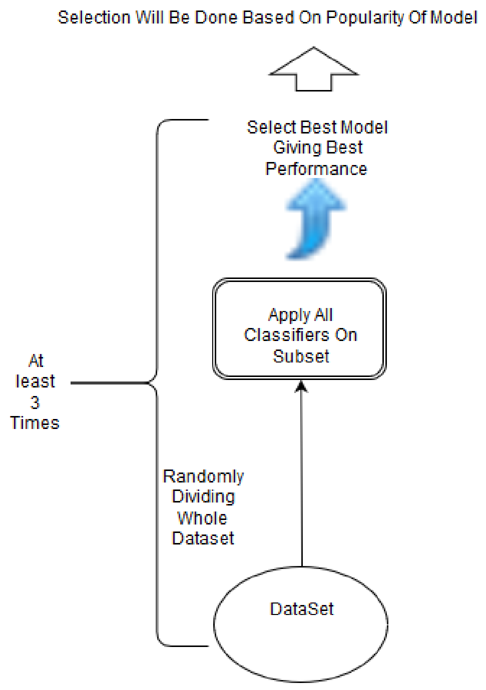

Selection Will Be Done Based On Popularity Of Model
Select Best Model Giving Best Performance
Apply All Classifiers On Subset
At least 3 Times
Randomly Dividing Whole Dataset
DataSet

Figure 6: Naive Approach

Now as in this approach as partitioning in independent and also system does not really have do ”understand” data behaviour, this approach, although being naive & simple, seems convincing. (predicted time complexity will be same approach 2 only, k * O(N), k >= 3 & < N)

To evaluate this approach, performance criteria will be *Accuracy*, *RMSE* & *Time*. In upcoming tables, predicted model and actual model's comparison is shown. Not matching model's associated performance deviation is written to find out tuning of model. Random partitioning of dataset was 30%. As results in graph 2 depicts that, in almost all the cases, model is predicting perfect classifier considering accuracy as performance criteria. Same thing also held true for other performance criteria. Even though, if predicted algorithm differs from actual algorithm, deviation in performance is not vast. So as far as performance criteria are concerned, model is working pretty well.

Now regarding selection of partition criteria, consider figure 3. Various partition paradigm were tested on 4 datasets only. As it is observed that, output regarding each partitions does not change. Model is also expected to be independent from random partition percentage. Which is also considered as good advantage of naive model but for better tuning good partition parameter will be 30% random partition of dataset.

Model is observing that dataset's behaviour can also be represented as subset of dataset(un-biased). Disadvantage of model is it cannot work well in streaming data processing. In coming data's behaviour is unpredictable. Considering pros and cons, this naive model seems more reasonable and convincing to select best model to classify the data.

Dataset	Actual Model	Predicted Model Randomize 1	Predicted Model Randomize 2	Predicted Model Randomize 3	Predicted Model (Using Quorum)	Evaluation	Prediction Time	Actual Time
Letter Recognition	SVM With C = 1 & Gamma = 0.1	SVM With C = 1 & Gamma = 0.1	SVM With C = 1 & Gamma = 0.1	SVM With C = 1 & Gamma = 0.1	SVM With C = 1 & Gamma = 0.1	**True**	112.8	471.9
Internet Ads	LDA With Solver = LSQR & Shrink = Auto	LDA With Solver = LSQR & Shrink = Auto	LDA With Solver = LSQR & Shrink = Auto	LDA With Solver = LSQR & Shrink = Auto	LDA With Solver = LSQR & Shrink = Auto	**True**	102.9	242.80
UCI HARS	SVM With C = 1 & Gamma = 0.01	LDA With Solver = SVD & Shrink = None	SVM With C = 1 & Gamma = 0.01	LDA With Solver = SVD & Shrink = None	LDA With Solver = SVD & Shrink = None	0.97411 — 0.9849 0.01 Deviation	426.66	1294.90
Cover Type	Bernoulli Naive Bayes With Alpha = 1	Bernoulli Naive Bayes With Alpha = 1	Bernoulli Naive Bayes With Alpha = 1	Bernoulli Naive Bayes With Alpha = 1	Bernoulli Naive Bayes With Alpha = 1	**True**	868.53	9908.98
Farm Ads	Gaussian Naïve Bayes	Gaussian Naïve Bayes	Gaussian Naïve Bayes	Gaussian Naïve Bayes	Gaussian Naïve Bayes	**True**	-	-
Adult	LDA With Solver = SVD & Shrink = None	LDA With Solver = SVD & Shrink = None	LDA With Solver = SVD & Shrink = None	LDA With Solver = SVD & Shrink = None	LDA With Solver = SVD & Shrink = None	**True**	-	-

Table 2: Model Evaluation On Different Datasets

Dataset	Using 10% Partitioning	Using 20% Partitioning	Using 40% Partitioning
Letter Recognition	SVM , C = 1 & Gamma = 0.1	SVM , C = 1 & Gamma = 0.1	SVM , C = 1 & Gamma = 0.1
Internet Ads	LDA, Solver = LSQR & Shrink = Auto	LDA, Solver = LSQR & Shrink = Auto	LDA, Solver = LSQR & Shrink = Auto
UCI HARS	LDA, Solver = LSQR & Shrink = Auto	LDA, Solver = LSQR & Shrink = Auto	LDA, Solver = LSQR & Shrink = Auto
Cover Type	Bernoulli Naive Bayes, Alpha = 1	Bernoulli Naive Bayes, Alpha = 1	Bernoulli Naive Bayes, Alpha = 1

Table 3: Evaluating Sensitivity Of Model Based On Partitioning

6 Conclusion & Future Work

Every classifiers' behaviour were different for different datasets. Trade-off was observed between accuracy and time to converge algorithm. Furthermore, for relative dataset which classifier will perform best, was analyzed. Out of all possible approaches, naive approach seemed to be working accurately. According to having good performance of naive approach, it is obvious that behaviour of whole large dataset can be represented as behaviour of small subset(un-biased) of that dataset. In future, we want to test this model with variety of datasets and also want to implement dataset-dataset similarity matrix & enhance model selection technique.

7 Team Member's Responsibility

	Task	Team Members
Code Implementation	Implementation Of QDA & kNN	**Naman Shah**
	Implementation Of LDA & SVM	**Bhavik Mistry**
	Implementation Of Variance Of Naïve Bayes	**Harshil Shah**
	Implementation Of Naïve Approach To Select Best Model	**Harshil Shah**
	Data Collection	**Naman Shah** **Bhavik Mistry**
Poster	Poster Creation	**Naman Shah** **Bhavik Mistry** **Harshil Shah**
Report Writing	Report Of Introduction	**Harshil Shah** **Bhavik Mistry**
	Report Of Classifiers Description	**Naman Shah** **Bhavik Mistry**
	Report Of Result Analysis	**Harshil Shah** **Naman Shah**
	Report Of Naïve Apporach Description	**Harshil Shah**

Table 4: Team Members' Responsibility

References

[1] Caruana, Rich, and Alexandru Niculescu-Mizil. "An empirical comparison of supervised learning algorithms." Proceedings of the 23rd international conference on Machine learning. ACM, 2006.

[2] Choosing A Machine Learning Classifier By Edwin Chen
http://blog.echen.me/2011/04/27/choosing-a-machine-learning-classifier/

[3] McCallum, Andrew, and Kamal Nigam. "A comparison of event models for naive bayes text classification." AAAI-98 workshop on learning for text categorization. Vol. 752. 1998.

[4] Standford - Choosing What Kind Of Classifier To Use
http://nlp.stanford.edu/IR-book/html/htmledition/choosing-what-kind-of-classifier-to-use-1.html

[5] Machine Learning Algorithms For Classification By Rob Schapire (Princeton University)
http://www.cs.princeton.edu/ schapire/talks/picasso-minicourse.pdf

[6] Model Selection - Wikipedia
https://en.wikipedia.org/wiki/Model_selection

[7] Choosing The Right Estimator - Scikit Learn
http://scikit-learn.org/stable/tutorial/machine_learning_map/

[8] http://stats.stackexchange.com/questions/7610/top-five-classifiers-to-try-first

[9] https://www.researchgate.net/post/Can_we_say_that_SVM_is_the_best_classifier_to_date14

[10] Dataset - UCI Repository
http://archive.ics.uci.edu/ml/datasets/

[11] Classifiers' Python Library :
http://scikit-learn.org/stable/supervised_learning.html#supervised-learning

[12] Naive Bayes - Wikipedia
https://en.wikipedia.org/wiki/Naive_Bayes_classifier

[13] kNN - Wikipedia
https://en.wikipedia.org/wiki/K-nearest_neighbors_algorithm

[14] LDA - Wikipedia
https://en.wikipedia.org/wiki/Linear_discriminant_analysis

[15] QDA - Wikipedia
https://en.wikipedia.org/wiki/Quadratic_classifier#Quadratic_discriminant_analysis

Appendices

Python libraries were used for classifiers' implementation. [11]

A Parsing Data

```
# Reading Data In Format
def generate_data(dataSet):

   if dataSet == "Random":
      X = np.array([[1, 1], [2, 1], [3, 2], [9, 102], [1, 80], [2, 70]])
      y = np.array([2, 2, 2, 1, 2, 9])

   elif dataSet == "Letter Recognition":
      f = open("Data/letter-recognition.data")

      data = np.loadtxt(f, delimiter=',')

      X = data[:, 1:]
      y = data[:, 0]

   elif dataSet == "InternetAdsDataset":
      f = open("Data/internet-ads.data")

      data = np.loadtxt(f, delimiter=',')

      y = data[:,-1]
      X = data[:,:-1]

   elif dataSet == "UCI_HARS":
      forX = open("Data/UCI_HARS/X_Whole.txt")
      forY = open("Data/UCI_HARS/y_Whole.txt")

      dataForX = np.loadtxt(forX)
      dataForY = np.loadtxt(forY)

      X = dataForX
      y =dataForY

   elif dataSet == "cover-type":
      data = np.loadtxt('Data/covertype.data',delimiter=',')
      y = data[:,-1]
      X = data[:,:-1]

   elif dataSet == "farm-ads":
      X, y = load_svmlight_file('Data/farm-ads')

      X = X.todense()

   elif dataSet == "amazon":
      data=np.genfromtxt('Data/amazon.txt', delimiter = ',')
      y = data[:,-1]
      X = data[:,:-1]

# X = X.todense()
```

```
   elif dataSet == "adult":
      data=np.loadtxt(fname = 'Data/adult.txt', dtype='str', delimiter = ',')
      enc = OneHotEncoder(categorical_features=[1,3,5,6,7,8,9,13], sparse = False)

      newData = enc.fit_transform(data)
      y = newData[:,-1]
      X = newData[:,:-1]

   elif dataSet == "DBWorld":
      print "Reading DBWorld Dataset"
      data = loadarff('Data/dbworld_bodies.arff')

      act_data = []
      for i in range(len(data[0])):
         act_data.append(list(data[0][i]))
      act_data_array = np.asarray(act_data)

      partition = act_data_array.shape[1] - 1
      X = act_data_array[:,:partition]
      y = act_data_array[:,partition:].reshape(act_data_array.shape[0])

   return X, y

if __name__=='__main__' :

   print "Working On letterRecognitionDataset"


#  Argument Will Be Changed For Diff Datasets
      X, y = generate_data("Letter Recognition")

      print "X Shape", X.shape
      print "y Shape", y.shape
```

B Different Classifiers

```
# Gaussian Naive Bayes
def gaussianNaiveBayes(X, y):
   classifier = GaussianNB()
# classifier.fit(X, y)

   return classifier

# MultiNomial Naive Bayes With Parameters
def multinomialNaiveBayes(X, y, alphaValue, fitPrior, classPrior):
   classifier = MultinomialNB()
   classifier.set_params(alpha = alphaValue, fit_prior = fitPrior, class_prior = classPrior)
# classifier.fit(X, y)

   return classifier

# Bernoulli Naive Bayes With Parameters
def bernoulliNaiveBayes(X, y, alphaValue, binarizeValue, fitPrior, classPrior):
   classifier = BernoulliNB()
   classifier.set_params(alpha = alphaValue, binarize = binarizeValue, fit_prior = fitPrior,
                  class_prior = classPrior)
# classifier.fit(X, y)

   return classifier

# Quadratic Discriminant Analysis
def quadraticDiscriminantAnalysis(X, y, reg_parameter):
   classifier = QuadraticDiscriminantAnalysis()

   classifier.set_params(reg_param=reg_parameter)
# classifier.fit(X, y)

   return classifier

# Linear Discriminant Analysis
def linearDiscriminantAnalysis(X, y, solverValue, shrinkageValue):
   classifier = LinearDiscriminantAnalysis(solver = solverValue, shrinkage = shrinkageValue)
# classifier.set_params(solver = solverValue, shrinkage = shrinkageValue)
# classifier.fit(X, y)

   return classifier

def nearestNeighbor(X, y, numberOfNeighbor, weight):
   classifier = KNeighborsClassifier()
   classifier.set_params(n_neighbors = numberOfNeighbor, weights = weight)
# classifier.fit(X, y)

   return classifier

def SVM_Gaussian(X, y, CValue, gammaValue):

   classifier = svm.SVC(C=CValue, kernel='rbf', gamma=gammaValue)

# classifier.fit(X, y)

   return classifier
```

C Random Partitioning Dataset

```
# Shuffling Data And Randomly Dividing Whole Dataset
def randomPartition(X, y):
   partition_Ration = 0.4
   number_Rows = X.shape[0]
   number_Train = int(round(number_Rows*partition_Ration))
   number_Test = number_Rows - number_Train

   rows = np.arange(number_Rows)
   np.random.shuffle(rows)

   training = rows[:number_Train]
   testing = rows[number_Train:]

   y = y.reshape((-1,1))

   X_Training = X[training,:]
   y_Training = y[training,:]

   y_Training = y_Training.reshape(1, -1)[0]

   return X_Training, y_Training
```

D Naive Approach - Selecting Best Model

```
# Using Cross Validation, We are choosing best model in respective model criteria.
def compareAccuracy(X, y, randomFactor, runMulti, performance="accuracy"):
   randomization = randomFactor
   typeOfCompare = performance
   bestAccuracy = 0
   bestModel = ""
   print "-- -- -- --"

   print "Running Multinomial? ", runMulti

   for i in range(randomization):
      print "Iteration", i
      print "-- -- -- --"

      cross_validate=cross_validation.StratifiedKFold(y, 5, shuffle=True);
      gaussianStartTime = time.time()
      print "****Gaussian Naive Bayes****"
      classifierGaussian = gaussianNaiveBayes(X, y)

      valid_cross = cross_validation.cross_val_score(classifierGaussian, X, y, cv = cross_validate,
                  scoring=typeOfCompare)

      accuracyUsingGaussian = np.mean(valid_cross)
      print "Accuracy Using Gaussian", accuracyUsingGaussian

      if accuracyUsingGaussian == bestAccuracy:
         bestModel = bestModel + ", Gaussian Naive Bayes"
      if accuracyUsingGaussian > bestAccuracy:
         bestAccuracy = accuracyUsingGaussian
         bestModel = "Gaussian Naive Bayes"

      gaussianExecutionTime = time.time() - gaussianStartTime
      print "Execution Time In Gaussian Naive Bayes", str(gaussianExecutionTime)

      print "-- -- -- --"

      if runMulti:
         multiStartTime = time.time()

         print "****Multinomial Naive Bayes****"

         alphaRange = [0.001, 0.01, 0.1, 1]
         for a in alphaRange:
            print "Alpha ", a
            cross_validate=cross_validation.StratifiedKFold(y, 5, shuffle=True);

            alphaValue = a
            fitPrior=True
            classPrior=None
            classifierMultinomial = multinomialNaiveBayes(X, y, alphaValue, fitPrior, classPrior)

            print "classifier Multinomial", classifierMultinomial

            accuracyUsingMultinomial = np.mean(cross_validation.cross_val_score(classifierMultinomial,
                  X, y, cv = cross_validate, scoring=typeOfCompare))

            print "Accuracy Using Multinomial ", accuracyUsingMultinomial
            if accuracyUsingMultinomial == bestAccuracy:
                bestModel = bestModel + ", Multinomial Naive Bayes With Alpha " + str(alphaValue)
            if accuracyUsingMultinomial > bestAccuracy:
                bestAccuracy = accuracyUsingMultinomial
                bestModel = "Multinomial Naive Bayes With Alpha " + str(alphaValue)

        multiExecutionTime = time.time() - multiStartTime

        print "Multinomial Execution Time", str(multiExecutionTime)

    print "-- -- -- --"

    bernoulliStartTime = time.time()
    print "****Bernoulli Naive Bayes****"
    alphaRange = [0, 0.001, 0.01, 0.1, 1]
    for a in alphaRange:
        print "Alpha ", a
        cross_validate=cross_validation.StratifiedKFold(y, 5, shuffle=True);

        alphaValue = a
        binarize = 0
        fitPrior=True
        classPrior=None
        classifierBernoulli = bernoulliNaiveBayes(X, y, alphaValue, binarize, fitPrior, classPrior)

        print "classifier Bernoulli", classifierBernoulli

        accuracyUsingBernoulli = np.mean(cross_validation.cross_val_score(classifierBernoulli,
            X, y, cv = cross_validate, scoring=typeOfCompare))
        print "Accuracy Using Bernoulli ", accuracyUsingBernoulli

        if accuracyUsingBernoulli == bestAccuracy:
            bestModel = bestModel + ", Bernoulli Naive Bayes With Alpha " + str(alphaValue)
        if accuracyUsingBernoulli > bestAccuracy:
            bestAccuracy = accuracyUsingBernoulli
            bestModel = "Bernoulli Naive Bayes With Alpha " + str(alphaValue)

    bernoulliExecutionTime = time.time() - bernoulliStartTime

    print "Bernoulli Execution Time", bernoulliExecutionTime
    print "-- -- -- --"

    print "-- -- -- --"
    quadraticStartTime = time.time()
    print "****Quadratic Discriminant Analysis****"
    regParameterRange = [0, 0.0001, 0.001]
    for r in regParameterRange:
        print "Reg Parameter ", r
        cross_validate=cross_validation.StratifiedKFold(y, 5, shuffle=True);

        regParameter = r
                classifierQDA = quadraticDiscriminantAnalysis(X, y, float(regParameter))

        print "classifier QDA", classifierQDA

        accuracyUsingQDA = np.mean(cross_validation.cross_val_score(classifierQDA, X, y,
            cv = cross_validate, scoring=typeOfCompare))
        print "Accuracy Using QDA ", accuracyUsingQDA

        if accuracyUsingQDA == bestAccuracy:
            bestModel = bestModel + ", QDA With Reg_Parameter " + str(regParameter)
        if accuracyUsingQDA > bestAccuracy:
            bestAccuracy = accuracyUsingQDA
            bestModel = "QDA With Reg_Parameter " + str(regParameter)

    quadraticExecutionTime = time.time() - quadraticStartTime
    print "Quadratic Execution Time", str(quadraticExecutionTime)
    print "-- -- -- --"

    lieanerDAStartTime = time.time()
    print "****Linear Discriminant Analysis****"
    solverRange = ["svd","lsqr", "eigen"]

    for solve in solverRange:
        shrinkageRange = "auto"
        print "Solver", solve

        if solve == "svd":
            shrinkageRange = None

        print "Shrink", shrinkageRange
        cross_validate=cross_validation.StratifiedKFold(y, 5, shuffle=True);

        classifierLDA = linearDiscriminantAnalysis(X, y, solve, shrinkageRange)
        print "classifier LDA", classifierLDA

        accuracyUsingLDA = np.mean(cross_validation.cross_val_score(classifierLDA, X, y,
            cv = cross_validate, scoring=typeOfCompare))
        print "Accuracy Using LDA ", accuracyUsingLDA

                if accuracyUsingLDA == bestAccuracy:
            bestModel = bestModel + ", LDA With Solver " + str(solve) + " Shrink " + str(shrinkageRange)
        if accuracyUsingLDA >= bestAccuracy:
            bestAccuracy = accuracyUsingLDA
            bestModel = "LDA With Solver " + str(solve) + " Shrink " + str(shrinkageRange)

    lieanerDAExecutionTime = time.time() - lieanerDAStartTime

    print "Linear DA Execution Time", str(lieanerDAExecutionTime)
    print "-- -- -- --"

    nearestStartTime = time.time()
    print "****Nearest Neighbor****"

    possibleN1 = int(len(y) * 0.01)
    if possibleN1 > 100:
        possibleN1 = 100
    if possibleN1 == 0:
        possibleN1 = 1
    neighbors = [possibleN1]

    for n in neighbors:
        print "Number Of Neighbors", n
        cross_validate=cross_validation.StratifiedKFold(y, 5, shuffle=True);
        classifierKNN = nearestNeighbor(X, y, n, "distance")
        print "classifier KNN", classifierKNN
        accuracyUsingKNN = np.mean(cross_validation.cross_val_score(classifierKNN, X, y,
            cv = cross_validate, scoring=typeOfCompare))
        print "Accuracy Using KNN ", accuracyUsingKNN
```

```
            if accuracyUsingKNN == bestAccuracy:
                bestModel = bestModel + ", KNN With Neighbor " + str(n) + " Weight " + str("Distance")
            if accuracyUsingKNN > bestAccuracy:
                bestAccuracy = accuracyUsingKNN
                bestModel = "KNN With Neighbor " + str(n) + " Weight " + str("Distance")

        nearestExecutionTime = time.time() - nearestStartTime
        print "Nearest Neighbor Execution Time", str(nearestExecutionTime)

        print "-- -- -- --"

        SVM_GaussianStartTime = time.time()
        print "****SVM Gaussian****"

        Cs = [0.01, 0.1, 1]
        gammas = [0.01, 0.1]

        for eachC in Cs:
            for eachGamma in gammas:
                cross_validate=cross_validation.StratifiedKFold(y, 5, shuffle=True);

                classifierSVM_Gaussian = SVM_Gaussian(X, y, eachC, eachGamma)
                accuracyUsingSVM_Gaussian = np.mean(cross_validation.cross_val_score(classifierSVM_Gaussian,
                    X, y, cv = cross_validate, scoring=typeOfCompare))

                print "Accuracy Using SVM_Gaussian "+ str(eachC) + " Gamma" + str(eachGamma) +
                        " Accuracy"+str(accuracyUsingSVM_Gaussian)

                if accuracyUsingSVM_Gaussian == bestAccuracy:
                    bestModel = bestModel + ", SVM With Gaussian C " + str(eachC) + " Gamma" + str(eachGamma)
                            if accuracyUsingSVM_Gaussian >= bestAccuracy:
                    bestAccuracy = accuracyUsingSVM_Gaussian
                    bestModel = "SVM With Gaussian C " + str(eachC) + " Gamma" + str(eachGamma)

        SVM_GaussianExecutionTime = time.time() - SVM_GaussianStartTime
        print "SVM Gaussian Execution Time", str(SVM_GaussianExecutionTime)
    print "Best Accuracy ", bestAccuracy
    print "Best Model ", bestModel
```