Subspace-Classification Approach for Simulated Tuple Class Assignment

Victor Robila*
victor.robila@gmail.com
Hunter College High School
New York, New York, USA

Robert Haralick
robert@haralick.org
City University of New York (CUNY) Graduate School
New York, New York, USA

Abstract

We developed a subspace classifier for measurement classification to provide an alternative to current deep learning approaches. Many modern neural networks cannot provide an explanation for their classification, and by providing a numeric path behind every classification, a subspace classifier solves this issue. We first use a bayesian method in which all the class conditional probabilities can be stored in memory. Then we made experiments with simulated class conditional distributions and defined a subspace classifier that does not store all the data in the class conditional probability arrays. This can use much larger distributions than the previous model as it uses much less memory so we expanded to cases where the measurement space is generated sequentially and everything does not have to be in the memory at the same time. For cases with distributions that fit in the memory we also compared a bayesian approaches with subspaces. We also compare the subspace classifier with a Python machine learning approach and find that our model outperforms it.

Keywords: neural networks, machine learning, n-tuple

ACM Reference Format:
Victor Robila and Robert Haralick. 2022. Subspace-Classification Approach for Simulated Tuple Class Assignment. In *Proceedings of Make sure to enter the correct conference title from your rights confirmation emai (Conference acronym 'XX).* ACM, New York, NY, USA, 11 pages. https://doi.org/XXXXXXX.XXXXXXX

1 Introduction

Machine learning has been growing in popularity in recent years and there have been many different approaches focusing on measurement classification [25][24][17]. One of the most common and widely discussed methods is deep learning, and it has found many applications in fields like law, economics, and medicine [5][18][28]. However, a common problem with deep learning is that networks can get so complicated that it is impossible to explain the reasoning behind an algorithm's decision, turning them into black boxes where the input and output is the only thing that humans can understand [12].

This problem is especially important when considering fields where bias may be important and where decisions can have important consequences. For example, there have been many discussions on how deep learning approaches in law can lead to unfair sentencing due to the implicit bias contained in datasets [19]. If these deep learning approaches are not used with supervision and their output is taken at face value, deep societal problems can emerge [30].

The need for explainable approaches in sensitive fields is why subspace classification, another type of machine learning approach, can be beneficial and useful [23]. Subspace classification is inherently based on probabilistic mathematics and therefore the reasoning for the classification for a measurement can be found directly by looking at the mathematics behind said classification.

Our approach

We developed a subspace classifier in the C programming language using Bayesian probabilistic statistics with the end goal of classifying simulated tuple values. The basic subspace classifier is discussed in section 4. We developed this tool with researchers in mind and allow a variety of customizations to be made to the data, such as changing the class distributions, measurements, subspaces, classification methods and more.

Our approach used synthetic generations of class conditional probabilities and the prior probabilities of the classes that were much larger than what a normal computer could run by using subspaces as defined in section 2 and described in section 4.4. [Expand here]

Following the development of the two versions of the algorithm, we also began optimizing the data generation process to result in better results. Data in the real world, unlike our simulated data are not completely random and usually have some underlying structures or patterns that benefit machine learning approaches, and we took slabs (or subsections) of a data hypercube to simulate such conditions. We tested using a variety of experimental protocols as discussed in section 5

Conference acronym 'XX, June 03–05, 2022, Woodstock, NY

ACM ISBN 978-1-4503-XXXX-X/18/06…$15.00
https://doi.org/XXXXXXX.XXXXXXX

and propose future work in section 7. In section 3 we define various terms and concepts that are needed to understand the subspace classifier's development.

In order to get a better understanding of the proposed subspace classifier's performance we also created another kind of machine learning model, a Support Vector Machine to see if our model outperforms a widely used standardized one.

2 Previous Work

Subspace Classification and N-Tuple Classification

There has been a lack of research on subspace classification in recent years, mostly due to the rapid rise and seemingly infinite applications of deep learning approaches. Subspace classification was originally described in a 1959 paper by Bledsoe and Browning where they introduced n-tuples for typewritten character recognition, which was the original application of subspace classification [4]. They represented the dots of black and white that made up typewritten characters as tuples and classified the type of character based on that. Alexander et al commercialized it, and it is one of the first pattern classification methods that is considered as a neural network [3].

In more recent years there have been several advances in n-tuple classification networks in almost all of their aspects. N-Tuple analysis for Optical Character Recognition has been revisited by Jung et al. who demonstrated that collections of n-tuples could be generated using a practical search algorithm and created a generator that can be used by researchers [10]. Memory usage in neural networks using n-tuples has also been optimized by Mitchell et al. which is similar to our subspace approach [15]. Jørgensen and Linneberg have shown that n-tuple classifiers do not always assign the highest output score to the class that an input belongs to and that n-tuple classifiers can be improved with biased training priors [9]. By relating an output probability with the probability of a certain class, it is possible to get n-tuple classifiers to be closer to Bayesian classifiers. The experimental protocols involved cross-validations.

Rohwer and Cressy, Tarling and Rohwer, and Morciniec and Rohwer have shown that n-tuple classification is similar in performance if not faster than most conventional methods and is usually much simpler as well [22][26][16]. Rohwer developed two probabilistic interpretations of the n-tuple recognition method which allowed it to be analyzed with Bayesian methods [21].

Subspace classifiers have also been used for speech recognition by Gunal and Edizkan who used linear subspace classifiers such as the Class Featuring Information Compression (CLAFIC), Multiple Similarity Method (MSM), and Common Vector Approach (CVA) [7]. They found that they were able to use the CLAFIC and CVA classifiers along with a feature extraction method described in their paper to reach high recognition rates.

Explainable Machine Learning

In terms of the need for explainable machine learning, there have been several studies that discuss the ethical implications of bias in machine learning. Recently, the Federal Trade Commission has found that using what appear to be neutral machine learning models can lead to discrimination in jobs, housing, and banking [29]. There have also been studies such as Chen et al. that find that bias can arise in predictive health care [6]. The main problem is that models cannot explain their decisions and therefore humans cannot see if there is an implicit bias in play. As a result of this, there have been many attempts to introduce explainable machine learning into practice. For example, Lundberg et al. describe a system that predicts the risk of hypoxaemia and possible risk factors [13]. This wide range of applications means that subspace classification is very important.

3 Definitions and Notation

Bayes Rule

The Bayes rule is a way to find the conditional probability of an event if another event happens (P(A|B)). The Bayes rule states that this probability (P(A|B)) is equal to the probability of the other event happening given that the first event happened (P(B|A)) times the probability of the first event (P(A)) divided by the probability of the second event (P(B)). This formula can be mathematically written as:

$$P(A|B) = \frac{P(B|A)P(A)}{P(B)} \tag{1}$$

Measurement space and Subspaces

M is the notation used for the measurement space, or that defines all of the tuples in our dataset. Feature variables can have unique indexes. As an example, if there are N feature variables, the index set I can be defined as $\{I\} = \{1, ..., N\}$ where an index set is an ordered set of natural numbers such as $J = \{1, 2, 4, 5\}$.

The feature whose index is j takes its values from the range L_j. Measurement space or M can also be defined as [Not sure how to get this X].

Robert Haralick's definition of a subspace in Haralick and Yuksel can be used for this paper [8]:

4 Methods

4.1 Software Components

All our code was written in the C programming language. C is a general-purpose programming language and was chosen mainly for its dynamic memory allocation and the fact that it was easiest to apply to the project [27]. While all the code was written in C, we also used a .txt file for our input data file. We used a gcc compiler to compile our program [**?**].

We developed the subspace classifier using the Visual Studio Code Integrated Development Environment (IDE) and choose it for its simple, easy-to-use interface. We used VSCode version 1.72.2 [14].

For our comparison with a machine learning package we used Python since it is widely used in the programming community. Python is described as a object-oriented, high-level programming language with dynamic semantics [2]. We also used the Sci-Kit learn machine learning package due to its ease of use [20].

4.2 Background for Normal Subspace Classifier

4.2.1 Measurement Space, Tuple Generation, and True Class Generation. Each measurement in the subspace classifier is in the form of a binary tuple, with a length specified by the user. The total set of possible tuples with this given length is defined as the measurement space. The size of this measurement space is mathematically represented below, where length denotes the number of elements in the tuple.

$$Size\,of\,Measurement\,Space = 2^{length} \tag{2}$$

This experiment uses simulated data for all the tuples in the measurement space, which is generated by creating every possible combination of 0s and 1s with the given tuple length. These tuples are then randomly shuffled to simulate real-world data.

This experiment also uses simulated data for the true class of each element. The true class of a tuple is the tuple's actual class and is the value that will be compared to the assigned values when confusion matrices (described in section 1.6) are generated. An assigned class is the predicted class of a tuple given by the model. A true class is randomly assigned to each tuple in the measurement space. These true classes are in a ratio specified by the user. For example, in an experiment with 4 classes, the user can choose to use prior class probabilities such as 30, 40, 15, and 15.

4.2.2 Probability Data Generation. Marginal Probability Generation

Each tuple needs to be given a probability of existing. The array of all these probabilities is called the probability array. Given that every tuple in the measurement space is represented by a probability in this array, its total sum is 1. The values in this array are also randomly generated.

Class Conditional Probability Generation

The next subroutine in the subspace classifier creates class conditional probabilities for every class and tuple where a class conditional probability is defined as P(d | c) where d0 is a measurement tuple, c is a class, d is an element of measurement space D ($d \in D$), and c is an element of the set of all classes ($c \in C$). The specifics on how this is done are further expanded on the programming section (section 2.).

Subspace Probability Generation

Given that we have the class conditional probabilities, we can now find the probability for each subspace. We create subspaces by dividing each inputted tuple into smaller tuples of uniform length, such that these smaller tuples are disjoint. For example, if we have tuples of length 6, and want to have 2 subspaces, we will divide this tuple into two subspaces that each have length three.

We need to find the probability of each tuple in each of these subspaces to find the total probability of the tuple later on when assigning classes. We can use the following formula where M is the measurement space defined in section 3 with range $L_i, i \in I$:

$$P_J((J,y)|c) = \sum_{\{(I,x)\in M|\pi_J(I,x)=(J,y))\}} P_I((I,x)|c) \tag{3}$$

This formula shows the projection from the full space to a subspace indexed by a set J. We find the probability of the subspace given a class c C where C is the set of all classes ($P_J((J,y)|c)$ by projecting the original tuple into a subspace and adding up the class conditional probabilities of that tuple. To display this numerically, we can take an example where we have binary tuples of length 6, 2 subspaces for each tuple, and 4 classes. Since we have 2 subspaces for each tuple, and each tuple is of length 6, each subspace has length 3. The number of possible values for each of these subspaces is 8 since each tuple in each subspace is binary and $2^3 = 8$. We will have 2 subspace probability arrays since we have 2 subspaces, each of which will be in the shape 4 by 8 since we have 4 classes and 8 possible tuples.

Now, we need to fill all of these subspaces with class conditional distributions. Just like in the formula, we can take the class conditional probabilities for each of the tuples that contain the first subspace and add them to the 4 empty spaces in the subspace probability array. As an example, if we consider 000, the tuple for the first subspace, we will find all the tuples in the measurement space that have 000 as their first components and add the class conditional probabilities for those tuples into the place with 000 in the subspace probability array. Since in this case we have 2^8 or 64 possible tuples, we will add 8 probabilities for each class.

4.2.3 Tuple Class Assignment. Basic Assignment

The first classification that we can do is simply examine which probability is greater for both the class conditional subspace probabilities. Given that we have multiple subspaces, we will assume that for each class the subspaces are probabilistically independent. Therefore, for a given tuple, we project the tuple into each subspace and multiply the class conditional probabilities.

Economic Gain Matrix and Bayes Rule Assignment

To assign classes to all binary tuples using either class conditional probabilities or subspace probabilities we can also use an economic gain matrix and the Bayes Decision

Table 1. Economic Gain Matrix

	Assigned	
True	c_1	c_2
c_1	0.4	0.2
c_2	0.1	6

Table 2. Probability of Measurement d_1 for each class

	Measurement
True Class	d_1
c_1	0.13
c_2	0.4

Table 3. Realistic Economic Gain Matrix

	Assigned			
True	d_1	d_2	d_3	...
c_1	1	0	0	...
c_2	0	1	0	...
c_3	0	0	1	...
...	..	...	...	...

Rule. An economic gain matrix is a matrix that shows how much benefit is given by each new assigned measurement depending on the class. As seen in Table 1, in a two-class environment, there are 4 gains, depending on the measurement's true class and assigned class.

In this example, the gain produced by assigning a measurement with true class c_1 to a measurement with true class c_1 is 0.4, the gain produced by assigning a measurement with true class c_1 to a measurement with true class c_2 is 0.1, etc. Since assigning a class c_2 to a class with true class c_2 is so much higher than everything else, the model will be biased towards assigning every measurement to c_2 since in most cases it will produce the most gain. The Bayes Decision Rule can then be applied using this matrix. Let us look at one measurement, d_1 that has probability 0.13 for true class c_1 and probability 0.6 for true class c_2 as seen in Table 2.

Using the economic gain matrix described in Table 2, the model will calculate the economic gain for each possibility. For example, the economic gain of assigning measurement d_1 to class c_1 is 0.13*0.4+0.1*0.6. Each measurement for d_1 is multiplied by the gain for its respective true class. The economic gain of assigning it to class c_2 is 0.13*0.2+6*0.6. The latter is clearly larger so this measurement will be assigned to class c_2.

For this experiment, the economic gain matrix will be an array filled with 0s except for the diagonal since there is no benefit to assigning a class that is incorrect as seen from table 3.

Table 4. Confusion Matrix Illustration

True class:	d_1	d_2	d_3
c_1	.12	.18	.3
c_2	.2	.16	.04
Assigned:	c_2	c_1	c_1

Table 5. Confusion Matrix

True class:	d_1	d_2	d_3
c_1	.12	.18	.3
c_2	.2	.16	.04
Assigned:	c_2	c_1	c_1

We used two forms of probability generation, subspace probabilities, and class conditional probabilities so we applied Bayes rule twice. For the class conditional probabilities, it was exactly as above. For the subspace probabilities, we have multiple probabilities for each tuple (there are multiple subspaces). Thus, we have to add these probabilities, and compare probabilities using these sums.

4.2.4 The Confusion Matrix. In addition to assigning classes to each of the measurement tuples in measurement space, we can also find the total probability of correct identification. The confusion matrix takes the assigned classes and looks at the probability of each measurement being correctly identified by taking P(c | d). Tables 3 and 4 use a Bayes Decision Rule applying the economic gain matrix with only 1s and 0s as described above and shows the calculations needed to find the probability of correct classification.

The probability of correct classification for these simulated data is 0.48+0.2=0.68 and is the sum of the probabilities of a measurement being correctly classified in the confusion matrix. Mathematically each of the elements in the confusion matrix can be expressed as following where D is the total size of the measurements pace, P_T is probability of true class, P_{TA} is the probability of the true class and assigned class being whatever is in the parenthesis and where f_d is the assigned class.

4.3 Programming for Normal Subspace Classifier

4.3.1 Program Inputs and Outputs and Other Information. The only inputs to the program are in the form of a text file provided by the user that define the parameters for which the classifier will run. These are in order, the number of runs, the size of measurement space, the length of each tuple in the measurement space, the length of each subspace, the number of classes, a random seed, and the probabilities for each of the classes. The probabilities for each of the classes are entered on different lines where the first line corresponds to the probability of the first class, second line for the probability of the second class etc. and where

Table 6. Mathematical Confusion Matrix

	Assigned:	
True	c_1	c_2
c_1	$\sum_{d\in D} f_d(c_1)P_T(c_1,d)$	$= \sum_{d\in D} f_d(c_2)P_T(c_1,d)$
c_2	$\sum_{d\in D} f_d(c_1)P_T(c_2,d)$	$= \sum_{d\in D} f_d(c_2)P_T(c_2,d)$

$$P_{TA}(c_1,c_1) = \sum_{d\in D} f_d(c_1)P_T(c_1,d) \tag{4}$$

$$P_{TA}(c_1,c_2) = \sum_{d\in D} f_d(c_2)P_T(c_1,d) \tag{5}$$

$$P_{TA}(c_2,c_1) = \sum_{d\in D} f_d(c_1)P_T(c_2,d) \tag{6}$$

$$P_{TA}(c_2,c_2) = \sum_{d\in D} f_d(c_2)P_T(c_2,d) \tag{7}$$

the number of lines is equal to the number of classes. If the inputs are not provided in this order and with the correct types (float/int) the program will stop running and alert the user that there is a mistake. Consequently, if the probabilities do not add to 100, the program will also stop running.

The data file is read, several variables are filled, and a class probability array is generated with the probabilities of each of the classes. Then, all arrays used in the program are defined and memory is allocated to them depending on their sizes using malloc.

The output of the program depends on whether the debug feature is on (section 4.3.9) but always contains the probability of correct identification for all four of the assignment methods and the average across trials and perturbations.

4.3.2 Data Generation. The first subroutine used in the subspace classifier program generates the array of tuples that defines the measurement space. As mentioned previously each tuple is binary with a length defined by the user. To generate this array all the numbers up to the size of the measurement space are converted to 6-digit binary arrays. Following this, all arrays are added to the main measurement space,

4.3.3 True Class Generation and Shuffling. The true class assignment is the second subroutine that is used and takes in the percentages for each class, the number of classes, an empty true class array, and the number of measurements in the measurement space. The percentages entered by the user are checked to make sure that they add up to 100 (within a certain threshold) and the program ends if they do not. Then a double for loop is used to fill in the true class array with the correct number of each percentage. The threshold for the second for loop is:

$$\frac{percentage}{(100.0 * (float)(number\ of\ measurements)) + 0.5} \tag{8}$$

This is calculated for each class percentage. The 0.5 is important since the for loop takes the floor of the value of the fraction. Oftentimes, while the limit of the expression is a whole number like 33, the program ends up with 32.99999 and needs the extra 0.5 to make sure that the upper bound of the for loop is 33 and not 32. Following this, we fill the array and count the number of classes that were placed in the array. If there are too many true classes produced, the program ends, and if there are too few, a random class is added until the true class array. It was experimentally determined that this very rarely happens and does not have a significant effect on the data. Then, using the percentages for this part of the program, P(c) is calculated.

4.3.4 Shuffling. An important step that needs to be done following the creation of the true class array and the marginal probability array is shuffling all of their terms. This is done in another subroutine where each term is moved around to a random position. This ensures that the true class array is randomized and not just something like 1,1,1,1,1,2,2,2,3,3 as this may bias the data.

4.3.5 Marginal Probability Generation. We also need to generate the probability of each of the tuples in the measurement space for the classifier to function. The program does this by randomly generating values within a certain threshold and then normalizes them to add up to 1.

4.3.6 Generating Conditional Probabilities by Applying Bayes Theorem. Given that all the basic parts needed to find class conditional probabilities have been generated, it is now possible to find those probabilities. Class conditional probabilities P(d | c) where d is a measurement tuple, c is a class, d is an element of measurement space D (d D), and c is an element of the set of all classes (c C) are generated in this subroutine. The specifics on how this is done are further expanded in parts 2.7 and 2.8. Following the creation of the class conditional probabilities, we must now use the Bayes theorem to find the probability of a class given a measurement, P(c | d), c C, d D, which will be used to assign a class to each measurement. The formula is as follows where P(c) is the probability for each class, and P(d) is the probability for each measurement:

$$P(c|d) = \frac{(P(d|c)P(c))}{(P(d))} \tag{9}$$

A subroutine that applies Bayes theorem is used for this task and takes in the class conditional probabilities, the P(c) array, and the P(d) value to calculate P(c|d). 2.7 Perturbation A perturbation is defined as a set number added to a probability in order to mimic real-world data by allowing for some randomness in the data and is further described in this section. At this point we have generated a random true class for every measurement based on the percentage of each class which is given by the user, and generated P(c). The next step is to compute the class conditional probabilities (P(d |

$true\ class\ array: c_1, c_2$
$P(d_{1,2}) = 0.4, 0.6$
Table 7. $P(d_{1,2}c_{1,2})\ before\ perturbation:$

	d_1	d_2
c_1	0.4	0
c_2	0	0.6

Table 8. $P(d_{1,2}c_{1,2})\ after\ perturbation:$

	d_1	d_2
c_1	0.3	0.15
c_2	0.1	0.45

c)) and apply a perturbation (if any). The program originally starts assuming that there is no perturbation and assigns the value from P(d) to $P(d|c_{true_class_of_d})$ and assigns a 0 to every class that is not the true class of P(d). Then a perturbation is computed by taking out a percentage of the true class for P(d), dividing by the number of classes, and adding this quotient to all of the classes with P(d). A visualization of this is provided in Tables 7 and 8.

As shown, the true classes of d_1 and d_2 are c_1 and c_2 respectively and the probabilities of d_1 and d_2 are 10 and 20 respectively. Before the perturbation the values for class c_1 and measurement d_1 and the values for class c_2 and measurement d_2 were 10 and 20 respectively since c_1 is the true class of d_1 and 1 is the true class of d_2. We then take 50% of the value for the true classes (in this case 10, and 20 so 5 and 10), divide by the number of classes (2) and add this back to each value in the column, giving us 7.5, 2.5, 5, and 15.

After the class conditional probability array is filled, we normalize it such that the total sum of all the values in each row after perturbation is 1. This is since the total probability for each of the classes must be 1.

4.3.7 Subspace Class Conditional Probability Generation. The mathematics and explanation behind the subspace class conditional probability generation was already outlined in section 4.2.2. A three-dimensional array is generated and filled with 0s. The 0s are important as in rare cases the total sum of the 8 probabilities for the subspaces may be 0 and the program needs to reflect this. The next step in this subroutine first requires us to convert a tuple into a linear address. A linear address is a way to refer back to a given leaf in a tree or a tuple. In this case, we have the linear addresses (the index of the tuple) and we need to find the tuples in order to add their probabilities to the correct subspace. The lin_add_to_tuple function defined in Edsn Kensington's Documentation for Linear Address and its Inverse is used and the probability from the class conditional probability array is added to the corresponding subspace class conditional probability array [11]. Once all probabilities are added, the subroutine ends and the originally empty subspace probability array is updated with these new values.

$tuple = (0, 0, 1, 0, 1, 0)$
Table 9. Class Conditional Probability Array for Tuple

Class:	Probability:
c_1	0.01
c_2	0.2
c_3	0.003
c_4	0.1

$Assigned\ Class: c_2$

4.3.8 Class Assignments. Assignment Using Class Conditional Probabilities

The first task in this program is a basic assignment that relies on the class conditional probabilities. This assignment takes the class conditional probabilities for each tuple and assigns the class with the highest class conditional probability to the tuple. An example is provided in table 9 with 4 classes and a tuple of length 6.

As shown, since class c_2 has the highest probability, it is assigned to the tuple. The program does this by looking at each tuple separately, setting a threshold value of 0, and updating the threshold for each subsequent class. If the next class has a higher probability than the previous, it is assigned to the tuple, and the threshold is updated to the probability for this class. This is then repeated until all the class probabilities are checked and the highest class is assigned

Assignment Using Subspace Class Conditional Probabilities

The same method defined in section 4.3.6 is applied for the Subspace Class Conditional Probabilities. However, there is one difference in that we have two subspaces for each tuple and each of these subspaces has their own probability. This means that they have to be combined in some way, and for the case of this experiment, they are multiplied. Further information on this is provided in section 6.

When a tuple is entered into this subroutine, the tuple is split into two parts and each of these subspaces are converted into linear addresses using the inverse linear address function defined in Edsn Kensington's Documentation for Linear Address and its Inverse [11]. Then the addresses are looked up in the subspace probability array and the two probabilities are multiplied. This is repeated for all classes and the class with the highest probability is assigned.

Assignment Using Class Conditional Probabilities and Bayes Rule

This section describes the assignment of the binary tuples based on their class conditional probabilities and Bayes Rule. The first step needed for this assignment is applying the Bayes rule to fill the array P(c | d). We already know P(d), P(c), and P(d | c) so we just apply it and build the array. Then, using the same method as the previous sections and the values produced for each measurement and class, we assign the class that has the highest probability to each measurement.

If we have more than one class where this is true, the program defaults to the first class with this equal probability. Then P(c | d) is normalized to add up to 1.

Assignment Using Subspace Class Conditional Probability Arrays and Bayes Rule

This assignment is a combination of the previous two sections. The same method is used to combine the two subspace probabilities, but these values are added instead of multiplied. Then, Bayes Rule is applied again.

The next step is very important for determining the probability of correct classification and involves building a confusion matrix. In a for loop, the confusion matrix is generated by summing up all the probabilities for which the assigned class was c1 and the true class was c1, the values for which the assigned class was c2 and the true class was c1, etc. until the second table is generated. Since the diagonal is the only one where the true class is the same as the assigned class, the probability of correct identification will be the sum of the values in the diagonal of the confusion matrix. Note that the total sum of the confusion matrix will always be 1.

Following the generation of this confusion matrix and determining the probability of correct identification and converting it to a percentage, the program is complete. The average accuracy and standard deviations are outputted in addition to the number of classes, measurements and runs.

For this project, 4 confusion matrices are generated with one for each assignment.

4.3.9 Additional Features. While mathematically the program is complete with just these basic steps, there are some programming additions that make the program easier to work with. The main one is a debugging feature which can be changed in the main program file. This is a global variable which is present in every subroutine and can turn on the ability to see every array and data value generated. However, there are several other subroutines that are needed to make this work. These are for printing a float or an int array, and for printing a float or an int vector. They are all quite similar and involve a for loop running through all of the values in the array and outputting them. There are also comments and descriptions of what each subroutine does at the beginning of every file to improve readability.

4.4 Subspace Classifier with Large Datasets

We also thought it would be interesting to develop a subspace classifier that is functionally the same as the one described in section 4.3 but fixes situations where the amount of data being processed by the algorithm is too much for the computers it is running on to handle by using subspaces. This means that if the same data file is provided to both programs, they should both output the same results.

Our Method

In order to allow the program to perform with a large amount of data we minimized the number of large arrays that the computer had to have stored in the memory through malloc. The largest arrays and thus the most problematic ones were the class conditional probability array, the array of all tuples, and the marginal probability arrays. In our previous classifier the subspace probability arrays were generated using the values inside the class conditional array and we lost a lot of performance. Instead, we generate the marginal probability and tuple for each measurement individually and send the probabilities directly into the subspace probability arrays. As soon as the measurement is generated and used, it is cleared from memory. Note that the marginal probability array was normalized in the previous classifier. Since we never generate this array and generate each probability one by one, the subspaces have inflated values. This was fixed by normalizing the subspace arrays.

Through this generation approach we were able to create another version of the subspace classifier that could perform well on very large datasets that overflow the memory of a given computing device.

4.5 Performance Optimization by Using Slabs

The subspace classifiers previously defined in sections 4.3 and 4.4 had several dimensions in their measurement space. Each of the tuples in this measurement space had a non-zero probability and therefore, a lot of randomness was induced. In order to fix this and simulate the more structured appearance of real-world data, a restriction in the form of a slab taken from the measurement space is used. A visualization of this is taking a section of a hyper-dimensional data cube. This can be defined through the following expression in an N-dimensional space:

$$\{x \in R^N | ax + b_1 \geq 0 \; and \; ax + b_2 \leq 0\} \tag{10}$$

Where b1 and b2 are chosen such that some tuple exists in this step and |b2-b1|>1 to not make the slab too small. The program interprets this equation by taking 2 values, a center value and another value that is added and subtracted from this center value to create b1 and b2:

$$center = 7$$
$$value = 2$$
$$\{x \in R^N | ax + 9 \geq 0 \; and \; ax + 5 \leq 0\} \tag{11}$$

The values for a are randomly generated and are between 1 and 1.5.

4.6 Python Machine Learning Model

We took the tuples and marginal probabilities and inputted them into a Support Vector Machine with the goal of classifying the measurements into true classes. In order to do this we took a dataset from the subspace classifier of 729 6-tuples and probability pairs and their associated true classes. We then split it into a 80%-20% training-testing ratio and ran the classification.

5 Results

5.1 Normal Subspace Classifier

5.1.1 Experimental Protocols. We tested the subspace classifier using 4 experimental protocols who's input data files are defined in Appendix A.

Increasing Number of Subspaces

The first interesting experiment that can be done with the classifier is looking at what happens as the number of subspaces increases while all other variables stay the same. This would also be the same as decreasing the size of each subspace. The control probabilities of correct identification can be found by running the input in Appendix A.2.1.

There is currently one subspace and a measurement space of size 729 with dimension 3. To see if increasing the number of subspaces has any effect on the accuracy, the same run file can be run using 2,3, and 6 instead of the 1 for the number of subspaces. This would run the program with 2, 3 and 6 subspaces. The same thing can be done more rigorously with a higher-dimensioned measurement space. The control run file for a measurement space of dimension 36 is in Appendix A.2.2

In this run file, the maximum integer value in tuples has been changed to 2 to allow the program to run in a reasonable amount of time. This run file can then be run with 2,3,4,6,9,12,18, and 36 subspaces to view the relationship between number of subspaces and probability of correct identification.

It was expected that increasing the number of subspaces will be beneficial and give a higher probability of correct identification since the measurements will be split up more and there will be more overlap. In addition, it was also expected that this trend will be very slightly exponential as the number of subspaces gets closer to the number of dimensions in the measurement space.

Increasing perturbation

In addition to looking at how the number of subspaces affects the probabilities of correct identification; it may also be interesting to look at what happens when perturbation is increased. This can be done automatically by turning on the perturbation feature, which will run the experiment for each perturbation value between 0 and 1 with an increment of 0.05. The run file for this experiment can be found in Appendix A.3.

This experiment has been tried already on a Bayes classifier and the trend showed a linear decrease as the perturbation decreased. It was expected that there would be a linear decrease in percent correct identification as the perturbation decreases will hold true for the subspace classifier. Further experiments combining the change in the number of subspaces and perturbation can also be done. By using the same run file as above and changing the number of subspaces to 2, 3 and 6, several graphs can be produced and overlayed. It was expected that the same trend will continue and the trend between perturbation and accuracy will be roughly linear but will get steeper as there are more subspaces.

Increasing the Number of Classes

Increasing the Number of Classes is another possible experiment. The run file for the control value is in Appendix A.4.

The same program can then be run with 2, 3, and 4 classes to see if there is any difference in the probability of correct identification. It was expected that as the number of classes increases, the probability of correct identification will decrease since there will be less values to choose from.

Change in Accuracy While Increasing Slab Size

It is also interesting to observe the effect of increasing the size of the slab in the hyperdimensional measurement space cube. This is equivalent to decreasing the number of 0s in the class conditional probability array. The maximum number of non-zero values in the measurement space for this experiment is 729, and the minimum is 0 if no measurements are inside the slab. It is expected that the probability of correct classification will increase for the subspace classifiers when the size of the slab is increased. The experiment can be tested by running the parameters in the program's run file (Appendix A.5). The value that is added and subtracted from the middle (in this case 0.25), can be increase to 0.5, 1 and 2 to develop a comparison.

5.1.2 Results for Experimental Protocols. Increasing Number of Subspaces

The subspace classifiers did show a change as the number of subspaces increased as shown from Figure 1. This experiment did not test perturbation, so the non-subspace classifiers remained at 100% probability of correct identification throughout the experiment. There is a general downward trend in accuracy as the number of subspaces increases, and it also seems that the variance also decreases as the number of subspaces approaches 6. The Bayes subspace classifier was always better than the non-Bayes subspace classifier. Across all the subspaces, the average accuracy and median for the bayes subspace classifier were 47.09 and 30.88 respectively. The average accuracy and median for the non-bayes subspace classifier were 47.06 and 30.84 respectively. These findings disprove the hypothesis that accuracy would improve as the number of subspaces increased.

Increasing Perturbation with Increasing Number of Subspaces for Bayes Subspace Classifier

It was found that the probability of correct identification increased at a greater rate for experiments with lower numbers of subspaces as shown from Figure 2. With one subspace, the trend seems to be closer to the like x=y, but the slope of the line decreases as subspaces were added. There is also an increase in variance as the perturbation decreases towards 0. All the probabilities of correct identification begin at the

same 25.0 value, which is just a random guess amongst the 4 classes and slowly increases from there.

Increasing the Number of Classes

Increasing the number of classes did have an effect on the accuracies of the subspace classifiers regardless of whether they used the Bayes rule or not as shown from Figure 3. Since perturbation was not tested for this experiment, the non-subspace classifiers scored 100% and stayed the same even when the number of classes was changed. It seemed that as the number of classes increased, the probability of correct classification decreased. This decrease seems to slow down as more classes are added. This finding validates the hypothesis that accuracy would decrease as the number of classes increases.

Change in Accuracy While Increasing Slab Size

As the size of the slab in the measurement space increased, the accuracy of the two subspace classifiers decreased as shown from Figure 4. The subspace classifier that used Bayes rule outperformed the subspace classifier that did not for all trials, but the largest difference was when the range had a size of 0.5. The two class conditional probability classifiers were unaffected. This experiment is equivalent to decreasing the number of 0s in the class conditional probabilities.

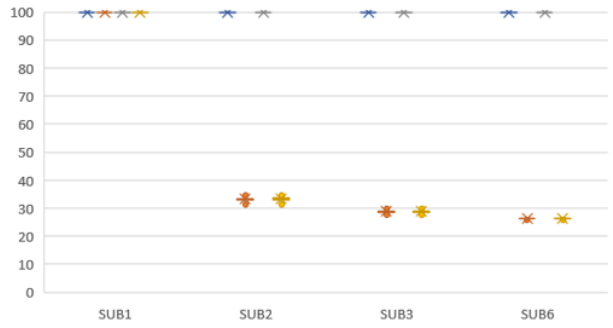

Figure 1. Accuracy vs Number of Subspaces
Blue=Non-Bayes Class Conditional Probability Classifier
Red=Non-Bayes Subspace Classifier
Gray=Bayes Class Conditional Probability Classifier
Yellow=Bayes Subspace Classifier

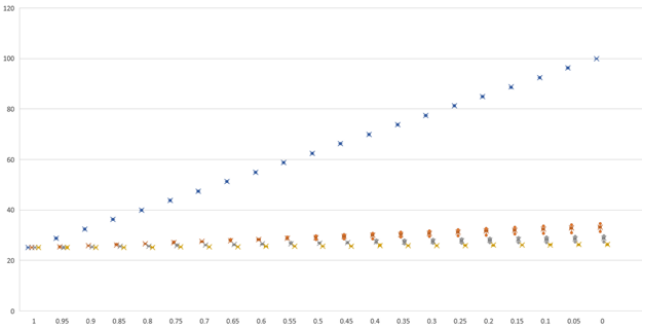

Figure 2. Increasing Perturbation with Increasing Number of Subspaces for Bayes Subspace Classifier
Blue= Bayes Subspace Classifier 1 Subspace
Red= Bayes Subspace Classifier 2 Subspace
Gray = Bayes Subspace Classifier 3 Subspace
Yellow= Bayes Subspace Classifier 6 Subspace

5.2 Subspace Classifier with large Datasets

The subspace classifier that was adapted to allow for large datasets was successful in having the same results as the normal subspace classifier and did work as expected.

5.3 Comparison Between Python Machine Learning Model and Subspace Classifier

As mentioned previously, to show the importance of this development, we compared our subspace classifier to a similar Python-based machine learning model. Our subspace

Figure 3. Accuracy vs Number of Classes
Blue=Non-Bayes Class Conditional Probability Classifier
Red=Non-Bayes Subspace Classifier
Gray=Bayes Class Conditional Probability Classifier
Yellow=Bayes Subspace Classifier

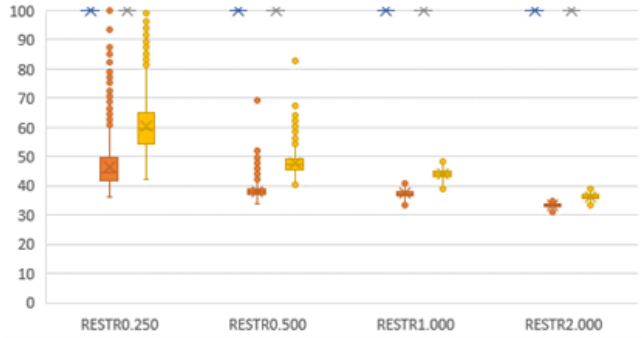

Figure 4. Change in Accuracy While Increasing Slab Size
Blue=Non-Bayes Class Conditional Probability Classifier
Red=Non-Bayes Subspace Classifier
Gray=Bayes Class Conditional Probability Classifier
Yellow=Bayes Subspace Classifier

classifier had a 25.172 accuracy on the testing data without any data optimisation using a slab and the python model had a 20.183 accuracy, demonstrating that our model has better accuracy than standard accepted models.

6 Conclusions and Future Work

We have defined the development of a subspace classifier for measurement classification and its expansion to large datasets that cannot be run on the memory of normal computers. We have also outlined a comparison between our subspace classifier and a Python machine learning model from a package and have shown that it equals or outperforms the Python model. We have also described several definitions and notations needed to understand this paper. In the future we plan to continue working on optimizing the performance of the classifier even on this extreme scale of data randomization and do more comparisons between the subspace classifiers and other machine learning models. For example, testing whether neural networks can see some hidden trend in the data might be interesting.

References

[1]]gcc [n. d.].

[2] 2022. What is python? executive summary. https://www.python.org/doc/essays/blurb/

[3] Igor Aleksander, WV Thomas, and PA Bowden. 1984. WISARD· a radical step forward in image recognition. *Sensor review* (1984).

[4] Woodrow Wilson Bledsoe and Iben Browning. 1959. Pattern recognition and reading by machine. In *Papers presented at the December 1-3, 1959, eastern joint IRE-AIEE-ACM computer conference*. 225–232.

[5] Ilias Chalkidis and Dimitrios Kampas. 2019. Deep learning in law: early adaptation and legal word embeddings trained on large corpora. *Artificial Intelligence and Law* 27, 2 (2019), 171–198.

[6] Irene Y Chen, Emma Pierson, Sherri Rose, Shalmali Joshi, Kadija Ferryman, and Marzyeh Ghassemi. 2021. Ethical machine learning in healthcare. *Annual review of biomedical data science* 4 (2021), 123–144.

[7] Serkan Gunal and Rifat Edizkan. 2007. Use of novel feature extraction technique with subspace classifiers for speech recognition. In *IEEE International Conference on Pervasive Services*. IEEE, 80–83.

[8] Robert M Haralick and Ahmet Cem Yuksel. 2020. The N-Tuple Subspace Classifier: Extensions and Survey. *IEEE Transactions on Systems, Man, and Cybernetics: Systems* 51, 1 (2020), 22–39.

[9] Thomas Martini Jorgensen and Christian Linneberg. 1999. Theoretical analysis and improved decision criteria for the n-tuple classifier. *IEEE Transactions on Pattern Analysis and Machine Intelligence* 21, 4 (1999), 336–347.

[10] D-M Jung, Mukkai S Krishnamoorthy, George Nagy, and Andrew Shapira. 1996. N-tuple features for OCR revisited. *IEEE Transactions on Pattern Analysis and Machine Intelligence* 18, 7 (1996), 734–745.

[11] Edsn Kensington and Robert Haralick. 2022. Documentation for Linear Address and its Inverse.

[12] Octavio Loyola-Gonzalez. 2019. Black-box vs. white-box: Understanding their advantages and weaknesses from a practical point of view. *IEEE Access* 7 (2019), 154096–154113.

[13] Scott M Lundberg, Bala Nair, Monica S Vavilala, Mayumi Horibe, Michael J Eisses, Trevor Adams, David E Liston, Daniel King-Wai Low, Shu-Fang Newman, Jerry Kim, et al. 2018. Explainable machine-learning predictions for the prevention of hypoxaemia during surgery. *Nature biomedical engineering* 2, 10 (2018), 749–760.

[14] Microsoft. 2021. Visual studio code - code editing. redefined. https://code.visualstudio.com/

[15] Richard James Mitchell, JM Bishop, and Paul R Minchinton. 1996. Optimising memory usage in n-tuple neural networks. *Mathematics and computers in simulation* 40, 5-6 (1996), 549–563.

[16] Michal Morciniec and Richard Rohwer. 1995. The n-tuple classifier: Too good to ignore. (1995).

[17] Thuy TT Nguyen and Grenville Armitage. 2008. A survey of techniques for internet traffic classification using machine learning. *IEEE communications surveys & tutorials* 10, 4 (2008), 56–76.

[18] Saeed Nosratabadi, Amirhosein Mosavi, Puhong Duan, Pedram Ghamisi, Ferdinand Filip, Shahab S Band, Uwe Reuter, Joao Gama, and Amir H Gandomi. 2020. Data science in economics: comprehensive review of advanced machine learning and deep learning methods. *Mathematics* 8, 10 (2020), 1799.

[19] Ioannis Pastaltzidis, Nikolaos Dimitriou, Katherine Quezada-Tavarez, Stergios Aidinlis, Thomas Marquenie, Agata Gurzawska, and Dimitrios Tzovaras. 2022. Data augmentation for fairness-aware machine learning: Preventing algorithmic bias in law enforcement systems. In *2022 ACM Conference on Fairness, Accountability, and Transparency*. 2302–2314.

[20] Fabian Pedregosa, Gaël Varoquaux, Alexandre Gramfort, Vincent Michel, Bertrand Thirion, Olivier Grisel, Mathieu Blondel, Peter Prettenhofer, Ron Weiss, Vincent Dubourg, et al. 2011. Scikit-learn: Machine learning in Python. *the Journal of machine Learning research* 12 (2011), 2825–2830.

[21] RJ Rohwer. 1995. Two Bayesian Treatments of the n-tuple recognition method. (1995).

[22] Richard Rohwer and David Cressy. 1989. Phoneme classification by boolean networks.. In *EUROSPEECH*. 2557–2560.

[23] Ribana Roscher, Bastian Bohn, Marco F Duarte, and Jochen Garcke. 2020. Explainable machine learning for scientific insights and discoveries. *Ieee Access* 8 (2020), 42200–42216.

[24] FM Serra Bragança, S Broomé, Marie Rhodin, S Björnsdóttir, V Gunnarsson, JP Voskamp, E Persson-Sjodin, W Back, Gabriella Lindgren, M Novoa-Bravo, et al. 2020. Improving gait classification in horses by using inertial measurement unit (IMU) generated data and machine learning. *Scientific reports* 10, 1 (2020), 1–9.

[25] Markus Stocker, Paula Silvonen, Mauno Rönkkö, and Mikko Kolehmainen. 2016. Detection and classification of vehicles by measurement of road-pavement vibration and by means of supervised machine learning. *Journal of Intelligent Transportation Systems* 20, 2 (2016), 125–137.

[26] Roland Tarling and Richard Rohwer. 1993. Efficient use of training data in the n-tuple recognition method. *Electronics Letters* 24, 29 (1993), 2093–2094.

[27] Barbara Thompson. 2022. What is C programming language? basics, introduction, history. https://www.guru99.com/c-programming-language.html

[28] Fei Wang, Lawrence Peter Casalino, and Dhruv Khullar. 2019. Deep learning in medicine—promise, progress, and challenges. *JAMA internal medicine* 179, 3 (2019), 293–294.

[29] Ian Weiner. 2021. FTC declares racially biased algorithms in artificial intelligence unfair and deceptive, prohibited by law. https://www.lawyerscommittee.org/ftc-declares-racially-biased-algorithms-in-artificial-intelligence-unfair-and-deceptive-prohibited-by-law/

[30] Douglas Yeung, Inez Khan, Nidhi Kalra, and Osonde Osoba. 2021. *Identifying Systemic Bias in the Acquisition of Machine Learning Decision Aids for Law Enforcement Applications*. JSTOR.

A Experimental Protocols

A.1 Data Input File Format

Number of Trials: xxx
Size of Measurement Space: xxx
Number of Dimensions in Measurement Space: xxx
Number of Subspace: xxx
Number of Classes: xxx
Value to be Added to Center: xxx
Center: xxx
Maximum Integer Value in Tuple: xxx
Seed: xxx
Probability of Class 1: xxx
Probability of Class 2: xxx
...

A.2 Protocol 1

A.2.1 Protocol 1a.

10,000
729
6
1
4
2
7
3
200221412424
25.0
25.0
25.0
25.0

A.2.2 Protocol 1b.

10,000
68719476736
36
1
12
2
7
2
200221412424
25.0
25.0
25.0
25.0

A.3 Protocol 2

10,000
729
6
2
4
2
7
3
200221412424
25.0
25.0
25.0
25.0

A.4 Protocol 3

10,000
729
6
1
1
2
7
3
200221412424
25.0
25.0
25.0
25.0

A.5 Protocol 4

10,000
729
6
2
4
0.25
5
3
200221412424
25.0
25.0
25.0
25.0