COMP3308: Introduction to Artificial Intelligence

Classification Assignment Report

Aneta Swianiewicz

May 2020

Contents

1 Introduction **2**

2 Aim **3**
2.1 Project Objectives . . . 3

3 Data **3**
3.1 The dataset . . . 3
3.2 Models . . . 4
3.2.1 k-Nearest Neighbours . . . 4
3.2.2 Naive Bayes . . . 5
3.3 Feature Selection . . . 5
3.3.1 Correlation-based Feature Selection . . . 6

4 Implementation **7**
4.1 MyClassifier . . . 7
4.1.1 Naive Bayes . . . 7
4.1.2 k-Nearest Neighbours . . . 8
4.2 Evaluator . . . 9

5 Results and Discussion **9**
5.1 Implemented Classifiers . . . 9
5.2 Weka Classifiers . . . 10

6 Conclusion **11**

7 Reflection **12**

Bibliography **13**

1 Introduction

According to the British National Health Service, "diabetes is a lifelong condition that causes a person's blood sugar level to become too high"[1]. Diabetes causes a range of serious symptoms, such as organ failure, which can eventually lead to death. Since 1980, the global prevalence of diabetes has virtually doubled[2]. As diabetes poses a growing threat to public health, tacking it effectively is essential. Early diagnosis and treatment are invaluable when it comes to successfully treating or managing any disease, especially one of such magnitude. Early diagnosis and treatment maximise patients' chances of slowing the rate of diabetes' progression, which, in turn, allows them to maintain their quality of life. Therefore, finding efficient and reliable ways of identifying individuals at risk is the best to ensure that they can be diagnosed and treated as soon as possible.

Machine learning techniques can be used as an aid in identifying at-risk individuals. A model can be very helpful by flagging individuals that match set-out criteria or even identify previously unknown patterns. Such a model would save time and resources by relieving healthcare workers and allowing them to direct focus into tasks that cannot be so easily automated. This has the potential to immensely improve the standard of care and save countless lives.

As previously mentioned, the method of identifying patients at risk of having diabetes should be efficient and reliable. The criterion of reliability is rather straightforward - the determination should be accurate. However, in case of potential misclassification, a false positive is preferred to a false negative, as its outcome has less of an adverse impact on an individual compared to a disease going untreated. Explainability is also an element that should be taken into account when considering reliability. When individuals have an understanding of how a model is making a decision, they can trust the outcome more. This is especially relevant when the determination made by a model turns out to be counter-intuitive or otherwise unexpected.

When it comes to efficiency, it should be considered in terms of both doctor and patient. Ideally, the data points needed for model input should be efficient to collect. Then, using the model - inputting the data points and computation needed to produce an output - should also be efficient. However, controlling the efficiency of a model can only be controlled to an extent. While we can control how simple or complex of a model to implement and how many features to include, it is assumed that other factors influencing efficiency - the type of software and hardware, or method of sourcing data points - are beyond our control.

Naturally, these concerns motivate the type of model which should be used to identify at-risk patients. Both reliability and efficiency seem to suggest opting for a model that is not overly complex. This report outlines the results of the implementation of two classification algorithms, Naive Bayes and k-Nearest Neighbours, which these requisites, in order to create a predictive classification model. The goal of this model is to accurately determine, based on a handful of personal attributes, whether an individual is exhibiting symptoms of diabetes.

2 Aim

This project aims to build a classifier that will accurately predict whether a given individual shows signs of diabetes. It will achieve this by meeting three project objectives stated below.

2.1 Project Objectives

1. Implement Naive Bayes Classifier
2. Implement k-NN Classifier
3. Evaluate accuracy of classifiers through stratified 10-fold cross-validation
4. Further evaluate the classifiers through external comparison

The objectives 1 and 2 constitute the main part of the project - building a predictive model using Python (version 3.7). Objective 3 aims to verify the reliability of classifiers and provides a basis for comparison between them. Then, Objective 4 is meant to expand on Objective 3 by establishing a better frame of reference. To achieve this, the performance of implemented models will be compared with performance of corresponding models in Weka, an established data analysis tool developed at the University of Waikato.

3 Data

3.1 The dataset

The classifiers will be based on Pima Indian Diabetes dataset . It was originally sourced from University of California, Irvine Machine Learning Repository. However, it has since been removed due to "permission restrictions"[3].

Codified name	Source description
NPreg	Number of times pregnant
Glucose	Plasma glucose concentration a 2 hours in an oral glucose tolerance test
BloodPr	Diastolic blood pressure (mm Hg)
Skin	Triceps skin fold thickness (mm)
Insulin	2-Hour serum insulin (mu U/ml)
BMI	Body mass index (weight in kg/(height in m)^2)
PGFunc	Diabetes pedigree function
Age	Age (years)
Class	Class variable ("yes" or "no")

Table 1: Attribute descriptions

The dataset contains 768 instances, with class split ratio of approximately 2:1 - there are 500 individuals classified as "no" and 268 classified as "yes". All attributes, aside from the class, take a numeric value. All attributes are enumerated in Table 1. The table includes a description of each attribute according to the source *names* file and the shortened attribute name.

The attributes that the classification model is based range from very rudimentary - age - to more complex - such as diabetes pedigree function. Given the diversity of the attributes, the range of numeric values taken they is quite varied.

3.2 Models

The classifiers used in this project employ k-Nearest Neighbours algorithm and Bayes theorem. Both k-NN and Naive Bayes classifiers are examples of lazy learning. This means that there is no training phase for the model before input can be passed through, making them very straight-forward and easy to implement. The training data are simply stored rather than abstracted into set of parameters. This approach is very useful for highly dynamic data, where generalizations made before can quickly become outdated. Given that storing a uncondensed dataset is space inefficient, it is most suitable for cases where only a small number of attributes is considered. This requirement is consistent with Pima dataset, as it takes only nine dimensions - eight numeric attributes and a class.

3.2.1 k-Nearest Neighbours

K-NN is a non-parametric model - it does not have a fixed parameter structure describing data distribution. K-NN stores known instances and then, given a query, finds k closest instances based on some chosen distance measure. The query is then assigned the class of the majority of neighbours in case of classification or mean of neighbour values in case of regression. However, in case of a large disparity between size of classes, the *majority vote* approach can lead to misclassification.

As with many models, high dimensionality can complicate execution of k-NN. This is because the

dimentionality amplifies the distance between any points, which makes the nearest points further away, making them appear like outliers.

The primary disadvantage of k-NN is sensitivity to noise and local structures. This can be countered with an appropriate choice of k. Additionally, in case of large data set, runtime of k-NN, $O(N)$, might be relatively inefficient. Then, binary trees or hash tables can be leveraged to improve execution time[4].

3.2.2 Naive Bayes

Naive Bayes classifier is a probabilistic model which uses Bayes' theorem to determine the output. The theorem "relates the *direct* probability of a hypothesis conditional on a given body of data, to the *inverse* probability of the data conditional on the hypothesis"[5]. It can be expressed by the following equation:

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

In the context of classification, the theorem is applied to find out the probability of a class given the attribute values. This is illustrated by: $P(class|x_n) = \frac{P(x_n|class)P(class)}{P(x_n)}$. A query is assigned to the class with highest probability given the query. Naive Bayes classifier to be quite scalable as it considers attribute probability in their individual dimension, hence avoiding the exponentially growing complexity in higher dimensions.

The principal assumption behind Naive Bayes classifier is the independence of variables. This follows from how the probability is calculated - by taking a product of the individual attribute probabilities: $P(x_1 \cap ... \cap x_n) = P(x_1)...P(x_n) = \prod_{i=1}^{n} P(x_i)$. This implies that, for any x_i and x_j, $P(x_i) = P(x_i|x_j)$ and defines independence.

Generally, the probability distribution is decided based on domain knowledge around the problem and used data. Alternatively, non-parametric methods such as Kernel density estimation can be used to estimate a probability distribution. In the case of the Pima dataset, it is assumed that the attributes follow a Gaussian distribution.

3.3 Feature Selection

Feature (or attribute, variable) selection is a method for limiting the number of model predictors based on some pre-selected algorithm in order to enhance accuracy. According to Huan Liu, "for a dataset with N features and M dimensions (or features, attributes), feature selection aims to reduce

M to M and $M \leq M$."[6] Feature selection is one of dimensional reduction techniques; it helps to simplify a model for it to be more generalize and less susceptible to the noise in training data and, hence, less prone to over-fitting. Other advantage of feature selection include reducing training time, however this aspect is not very apparent in our case, given the already small size of the training dataset.

In the case of the Pima dataset, the main consideration for feature reduction is eliminating noise that may be introduced by potentially redundant attributes in the case of k-NN and eliminating variables that could violate the independence assumption in the case of Naive Bayes.

3.3.1 Correlation-based Feature Selection

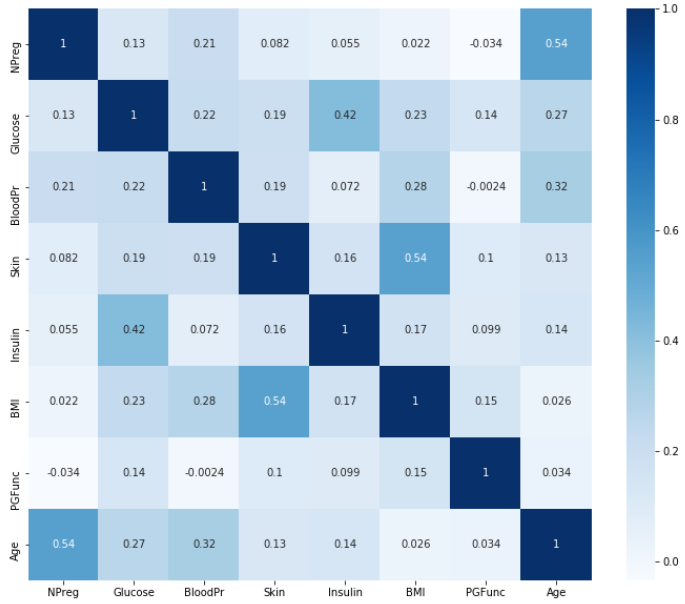

Figure 1: Pearson's correlation of variables

The method used in this project was Correlation-based Feature Selection. Figure 3 contains the attribute correlation matrix. It shows that the most correlated variable pairs - *NPreg, Age* and *BMI, Skin* - have a Pearson correlation coefficient of 0.54. This positive correlation is consistent with the general intuition: the older a person is, the more chances they had to be pregnant and a higher BMI would imply a thicker layer of subcutaneous fat, and greater skin fold thickness.

The feature selection was done through Weka's *Select Attributes* using the *CFS Subset Evaluator* function and *Best-First* search method. The attributes following attributes were selected: *Glucose, Insulin, BMI, PGFunc, Age*. Accordingly, *NPreg, Skin, BloodPr* have been left out.

4 Implementation

Both the classifiers and the evaluator programs were implemented in Python using only modules included in the Python Standard Library and Numpy.

4.1 MyClassifier

`MyClassifier.py` is the core element of the project. It is the program that takes in three elements in the following order: two files - training and testing - and user input determining the algorithm. Recognized user input for specifying classification model is either "NB" for Naive Bayes or "kNN" with k replaced with desired integer for k-Nearest Neighbours. `MyClassifier.py` then outputs a "yes" or "no" for each line of testing input.

The following sections, 4.1.1 and 4.1.2, contain code and descriptions of the functions invoking each classifier.

4.1.1 Naive Bayes

Naive Bayes classifier takes training and test sets. First, training test is split on class into *yes* and *no* instances. This is necessary for computing prior probabilities $P(class = yes)$ and $P(class = no)$. Then, for each class, the mean and standard deviation of each attribute is calculated. Following the assumption that attributes follow Gaussian probability distribution, probabilities for each attribute given each class are computed and multiplied withing their class to yield overall probability. Finally, probability that $P(class = yes)$ is computed. If $P(class = yes) \geq 0.5$, the output class is *yes*.

Listing 1: Function defining Naive Bayes classifier

```
def nb(train, test):
    # split on class, assume binary classification
    train_y = []
    train_n = []

    for i in range(len(train)):
        if train[i][-1] == 'yes':
            train_y.append(train[i][:-1])
        elif train[i][-1] == 'no':
            train_n.append(train[i][:-1])

    # convert class list to arrays
```

```
    train_y = np.array(train_y)
    train_n = np.array(train_n)

    # compute prior class probabilities
    p_y = len(train_y)/(len(train_y)+len(train_n))
    p_n = len(train_n)/(len(train_y)+len(train_n))

    # models (mean, std) for class 1 and 2
    model_y = model(train_y)
    model_n = model(train_n)

    #classify
    attribute_ps_y = []
    attribute_ps_n = []

    for i in range(len(test)):
        attribute_ps_y.append(normal_pdf(test[i], model_y[i][0], model_y[i][1]))
        attribute_ps_n.append(normal_pdf(test[i], model_n[i][0], model_n[i][1]))

    prob_if_y = np.prod(attribute_ps_y)
    prob_if_n = np.prod(attribute_ps_n)

    prob = (prob_if_y*p_y)/(prob_if_y*p_y+prob_if_n*p_n)

    #If there is ever a tie between the two classes, choose class yes
    if prob >= 0.5 and prob <= 1:
        return "yes"
    elif prob < 0.5 and prob >= 0:
        return "no"
```

4.1.2 k-Nearest Neighbours

Initially, the k is specified by the user. Given the training data and a query, the distance from query to each attribute is calculated. The chosen distance measure is Euclidean distance. Then, the list of obtained distances is sorted. The first k elements of this list are neighbours. Then, the neighbours' classes are tallied. The query is assigned a class of the majority of neighbours or, in case of a tie, class *yes*.

Listing 2: Function defining k-NN classifier

```
def knn(k,train,test):
     # calculate distances from all training samples
    distances = [] #list of (index, distance, class)
    for i in range(len(train)):
        distances.append((i, heuristic(test, train[i][:-1]), train[i][-1]))

    distances.sort(key=lambda x: x[1])
    neighbours = distances[:k]
    votes_yes = len([i for i in neighbours if i[2] == 'yes'])
    votes_no = len([i for i in neighbours if i[2] == 'no'])

    if votes_no > votes_yes:
        return "no"
    elif votes_yes >= votes_no: # If there is a tie between the classes, choose yes
        return "yes"
```

4.2 Evaluator

In order to evaluate the performance of `MyClassifier.py`, a separate `Evaluator.py` program has been added as an extension to the core program. Evaluator takes multiple arguments - each constituting a fold. After taking in n pre-generated fold, Evaluator leaves one out and combines the remaining $n-1$ folds into a training set. The training set is used as the train argument in a classifier function. Then, the classifier predicts class for instances in the testing fold. The accuracy prediction is recorded. This process is repeated for every fold. Finally, the program outputs the average accuracy for Naive Bayes, 1-NN, 3-NN, and 5-NN classifiers. For further validation, a confusion matrix is also generated at every step, yielding n matrices. Then, a matrix containg the average of values across these martices is returned for each classifier. The results obtained by passing the 10 stratified folds comprising the Pima dataset can be found in the following section, in Tables 2 and 3.

5 Results and Discussion

In this section, I will outline the results achieved by the implemented classifiers as well as results given by ten different Weka classifiers.

5.1 Implemented Classifiers

	My1NN	My5NN	MyNB
No feature selection	68.35%	75.39%	75.26%
CFS - 5 features	68.23%	75.13%	76.17%
CFS - 4 features	69.65%	74.87%	76.04%

Table 2: Implemented classifiers accuracy summary

$MyNB$

		Prediction	
		yes	no
Actual	yes	16.3	8.5
	no	10.5	41.5

$My1NN$

		Prediction	
		yes	no
Actual	yes	14.9	12.4
	no	11.9	37.6

$My5NN$

		Prediction	
		yes	no
Actual	yes	16.7	8.8
	no	10.1	41.2

Table 3: Average confusion matrices (number of examples)

The accuracy results for implemented classifiers can be found in Table 2. The accuracy values for each given classifier are the average of accuracies of each fold in a 10-fold stratified cross-validation. The stratification and splitting was done with Scikit-learn library using `StratifiedKFold()` object and `split()` function. The 10-fold cross-validation allows us to maximise the amount of insight that can be derived from the provided training set by using it all of its elements for both training

and testing. Stratification is done to ensure that each fold is representative of the set, which aims to prevent skewing results.

Unsurprisingly, 1NN has the lowest performance out of all three classifiers. Relying on only one closest neighbour makes the prediction very unstable and easily misled by random noise or class outliers. Thanks to additional context and reference points, 5NN is able to consistently out-preform 1NN.

NB has an accuracy rate that is very similar to 5NN, differing by 1.18% at most. The proportion of precision and recall is also quite similar between the two classifiers - the recall rate was higher than the precision for both, which is in line with the intention of prioritising false positives over false negatives. Interestingly, when applying feature selection NB accuracy increased slightly, while 5NN decreased. A likely explanation is that reducing an already small number of attributes rendered the dataset unable to fully engender the nuance of true distribution, hence hindering k-NN performance.

The evaluation was initially performed on the full dataset, without feature selection. Then, it was repeated after removing three features - *Number of Pregnancies*, *Blood Pressure*, and *Skin thickness* - which were deemed most redundant by Weka Correlated Feature Selection algorithm. For reference, the accuracy test was repeated with one additional feature - *Diabetic Pedigree Function* - removed. *Number of Pregnancies* and *Skin thickness* attributes both were significantly correlated with *Age* and *BMI* respectively, so removing them allowed for greater independence between features. This is especially noticeable considering that the jump in performance accuracy is greatest in Naive Bayes classifier, which relies on such independence. Nevertheless, in the case of Pima dataset, the improvements made thanks to feature selection are very modest. This is because the number of features is already small, so the curse of dimensionality or similar complexity issues are not particularly significant here. If anything, the accuracy could potentially be improved if there were more independent and relevant attributes available.

5.2 Weka Classifiers

	ZeroR	1R	1NN	5NN	NB	DT	MLP	SVM	RF
No feature selection	65.10%	70.83%	67.84%	74.48%	75.13%	71.74%	75.39%	76.30%	74.87%
CFS	65.10%	70.83%	69.01%	74.48%	76.30%	73.31%	75.78%	76.69%	75.91%

Table 4: Weka classifiers accuracy summary

The accuracy results for Weka classifiers can be found in Table 4. They provide an invaluable background for evaluating the implemented classifiers, as they better present the limitations of

the dataset. Since the Weka performance accuracies are rather similar, both to each other and to its counterparts implemented in the project, there seems to be a a ceiling to how well does the information contained in the training data generalize. The achieved accuracies were *sufficiently good* as it produced better results than a random choice would. However, their predictions are not highly reliable.

The best result was achieved by Support Vector Machines, closely followed by Naive Bayes. As non-parametric models, SVM are quite flexible and were able to capture the distribution best.

The worst results were ZeroR and 1NN. This was to be expected, since these models are extremely crude, allowing for no nuance. ZeroR, given that the majority of classes happens to be *no*, returns this class for any given query. Hence, the class split ratio is clearly reflected in performance accuracy.

Across the board, CFS did not reduce performance for any classifier. Again, it provided meager improvement to the original performance. This is most likely because the dataset predictors were already condensed to the most useful ones.

6 Conclusion

The aim of the project has been to create a method of accurately predicting if an individual would be symptomatic of diabetes based on eight personal features. Two implemented classifiers have fulfilled that task with 5-Nearest Neighbours yielding the best accuracy of 75.39% for dataset with all features and Naive Bayes yielding the best accuracy of 76.17% for dataset with selected features. Weka classifiers achieved comparable accuracy on both full and CFS datasets.

The primary consideration arising from the results of the project is the quality and generalizablity of training data. Obtaining a set that contains more diversity amongst its members could help significantly improve generalizability. The Pima people are a small subset of general population and the general trends are most likely not reflected in sample based solely on them. Using a homogeneous set is not suitable as a baseline for wider predictions as biased samples are not reliable. Moreover, a set of 768 examples is fairly small. Procuring additional instances, using a larger set, or integrating more sets should produce much better results.

When it comes to kNN, other distance measures, such as Minkowski or Manhattan distance, could be applied and the performances achieved by them compared to the one generated by Euclidean distance.

Possible tweaks around Naive Bayes classifier is checking its performance with only two most

correlated variables - *Number of Pregnancies* and *Skin thickness* - removed, rather than all three removed by CFS. Possibly, *BloodPr* variable could have helped the accuracy of made predictions.

7 Reflection

The assignment was incredibly useful for refining both understanding and practical skills connected to k-NN and Naive Bayes algorithms. Implementing the classifiers from scratch constituted a valuable programming experience. However, successfully implementing them would have been impossible without first achieving a profound and intimate understanding of the topic. Hence, the theory and practice mutually reinforced themselves, helping me better grasp Machine Learning algorithms.

Bibliography

[1] N. Choices, "Diabetes," NHS, 2019. [Online]. Available: https://www.nhs.uk/conditions/diabetes/

[2] "Diabetes," Who.int, 10 2018. [Online]. Available: https://www.who.int/news-room/fact-sheets/detail/diabetes

[3] "Uci machine learning repository: Data set," archive.ics.uci.edu. [Online]. Available: https://archive.ics.uci.edu/ml/datasets/pima+indians+diabetes

[4] S. J. Russell and P. Norvig, *Artificial Intelligence : a Modern Approach.* Prentice-Hall, 2010.

[5] J. Joyce, "Bayes' theorem," *plato.stanford.edu*, 06 2003. [Online]. Available: https://plato.stanford.edu/archives/spr2019/entries/bayes-theorem/

[6] H. Liu, *Feature Selection.* Boston, MA: Springer US, 2010, pp. 402–406. [Online]. Available: https://doi.org/10.1007/978-0-387-30164-8_306