End Term Exam B

April 21, 2022

EE5180: Introduction to Machine Learning
Submitted by: Akash Sharma (EE21S056)

1 Problem Statement

You are a drug manufacturer and you want to study whether your drug, namely drug A, leads to a higher rate of recovery than drug B. What are the input samples/feature vectors you will use, the loss function, hypothesis classes and the output labels. Justify your answers.

How many samples of feature vectors will you require to obtain a PAC solution? Propose a learning algorithm and evaluate its performance using some toy data set.

Now, let us bring a small twist to the problem. Instead of wanting to study whether your drug is better, you want to show using your algorithm that your drug is indeed better. What is the difference between the above and the current scenario? How will you use the data set you had for the previous problem and obtain a PAC solution which favors drug A?

2 Solution

Before deciding the particulars of the learning algorithm, it is necessary to fix the design of the clinical trial for comparing the two drugs A and B, which are *assumed* to be used as treatment for a fictitious disease. Here, a fictitious disease is chosen since the feature selection for the input samples is randomized and depends on some of my prior knowledge (experience), for example *it is quite likely that a person's response to a drug depends on their age*; it also depends on the design of the trial which takes into account the outcomes that the experimenter is interested in. For this particular problem, we are interested in the **rate of recovery** of the patient population as function of the drug used out of the two given choices.

In case an existing disease was chosen, it would require disease specific knowledge about the factors affecting the treatment of such a disease and the feature vector would have to be selected appropriately. Another reason a fictitious disease has been chosen is to circumvent the problem of acquiring disease specific knowledge, though in the real world applications this is not an option.

2.1 Clinical Trial Setup

The general outcome of a drug trial is the measures of safety and efficacy for that particular drug in a highly controlled environment [1]. It is quite common to find the effects of a single drug on the selected patient population which is divided into two groups – treatment group which is subjected to the drug and a control group which is subjected to a placebo drug. However, comparing the effects of two different drugs to find which one is better needs a different method, it is suggested that a head-to-head trial [2] is done in such cases.

In a head-to-head trial or a randomized controlled trial [3] , from the total selected patients, two groups are chosen which are similar to each other in distribution of types of patients based on various factors that clinicians deem fit to be the indicators of similarity. Each group is then matched with either of the two drugs that are being compared and the results are noted, these results then would be indicative of the efficacy of each drug on a certain patient population because they were divided in such a way that no bias would be induced from the differences in the two groups since there are no differences (ideally).

One **problem** I faced while trying to design the trial was lack of public datasets which followed the head-to-head trial approach. Instead, I tried to relax the conditions required for the methods mentioned above and went with a naive direct approach for comparison of the two drugs [4], in this method the similarities between the two groups is not considered and a direct comparison of the effects of each drug is made, this leads to comparison errors caused by difference in distribution of types of patients in each group, however here simplicity of design takes precedence over consequential errors. A few assumptions have to made before we move forward -

- **Drug A** is still in trial phases and has not been tested on many patients.
- **Drug B** has been in the market for a long time and has both trial data and real world data available reflecting its effect on a variety of patients. Therefore, recovery rate of patients is known and has high accuracy with high confidence because of the sheer number of test and observational data samples.
- The disease can be treated (if the drug works) in a relatively smaller time frame, for example 14 days at maximum. This means after administering a drug to a patient, the effects can be noted within 14 days. Particularly for the trial study here, the effect of interest is if the patient recovered from the disease or not i.e., if all the symptoms have been alleviated and patient has returned to state of normal body function.

2.1.1 Need of learning algorithms

Typically, there is *no need of machine learning algorithms* for finding out the efficacy and safety measures of a drug since the goal of the drug trial is the same and would anyway be known once the study is concluded. However, it is well known that drug trial studies could take multiple years [5] due to factors such as induction of potential candidates for the study, the time required for multiple experiments/tests etc. In such cases, how is it possible to find out comparative performance of a drug on a larger patient population given a much smaller subset of such a population? Here, learning algorithms could be used for such predictions i.e., using a smaller subset of patient outcomes, we can predict the outcome for a larger subset with some arbitrary degree of confidence and accuracy.

2.1.2 Trial Design

The goal of this trial is to acquire enough data to find out *if drug A **leads** to higher recovery rate than drug B*. For each patient, the following measurements are recorded after a few tests -

1. Age (`Age`) - Age of each patient is recorded. It ranges from `15` to `74`.
2. Sex (`Sex`) - Recorded as Male (`M`) or Female (`F`).
3. Blood Pressure (`BP`) - Recorded as `HIGH`, `LOW` and `NORMAL`.

4. Cholesterol (`Cholesterol`) - Recorded as `HIGH` and `NORMAL`.
5. Dominant biomarker (`dom_marker`) - Another biomarker is chosen which dominates the condition of the patient and their reaction to the drug.

All of the above are *assumed* to be features of the patient's profile which affect the state of recovery of the patient once the drug is administered. These features will be used as elements of the *feature vectors* or *input samples* for the learning algorithm.

After taking the above measurements, the patient is administered with drug A and the effect is recorded in the dataset under the column `recovery` with choices as `YES` and `NO` where the former means that the patient has recovered from the disease and is healthy now, the latter means that patient's body has resisted treatment and is not healthy. One question still remains, how many patients are needed so that eventually the learning algorithm we choose can successfully predict what we want it to predict? For now, we will just *assume* that we need an arbitrary number of patients which are generally characteristic of such a study, say about 200. It should be noted that this number is *arbitrary* and could have been something else as well. The reason for selecting an arbitrary number will be explained in the upcoming sections where we discuss hypothesis classes.

2.1.3 Dataset Synthesis

Since there is no provision for performing the above study and no other matching study was found, one way to get a dataset is to synthesize it. Here I use some seed data that I found on Kaggle [6]. After performing some manipulation so that it fits our application, it looks like as shown below. The programming language used in this report is `Python 3.8`.

[1]:
```
import pandas as pd
import numpy as np
import warnings #comment this out in case warnings need to be seen
warnings.filterwarnings('ignore')

patient = pd.read_csv("patient.csv")
print(patient.head()) #show first five entries in the dataset
print("") #empty space for visibility
print("The dimensions of the dataset are",patient.shape) #print dimensions
```

```
   Age Sex      BP Cholesterol  dom_marker
0   23   F    HIGH        HIGH      25.355
1   47   M     LOW        HIGH      13.093
2   47   M     LOW        HIGH      10.114
3   28   F  NORMAL        HIGH       7.798
4   61   F     LOW        HIGH      18.043

The dimensions of the dataset are (200, 5)
```

It can be seen that there are 200 patients for whom the relevant data has been recorded. However, the `recovery` column is not shown. I will add that column later on after synthesizing more patients using a distribution representative of the original dataset which is shown above. The reason we need more patients is because we do not have a dataset for drug B. The strategy from here onwards is to synthesize more patient information from the seed dataset `patient` and then **randomly** label each patient with the `recovery` labels, `YES` or `NO` for drug B, this will give us a larger dataset with information about the recovery rate of drug B (which is in our control). After doing that, we will again randomly label the seed dataset `patient` with the `recovery` labels for drug A. Finally, we will have two datasets, each for drug A and B.

For patient data synthesis, a `Python` package named Synthetic Data Vault or `SDV` is used [7]. Use `pip install sdv` to install the package on the system.

[2]:
```
# Adding serial_num column for creating unique identifier for a patient
patient.insert(loc=0, column='serial_num', value=np.arange(len(patient)))
patient.head()
```

[2]:
```
   serial_num  Age Sex      BP Cholesterol  dom_marker
0           0   23   F    HIGH        HIGH      25.355
1           1   47   M     LOW        HIGH      13.093
2           2   47   M     LOW        HIGH      10.114
3           3   28   F  NORMAL        HIGH       7.798
4           4   61   F     LOW        HIGH      18.043
```

[3]:
```
from sdv.tabular import GaussianCopula
'''
GaussianCopula is one of the models used for fitting the seed dataset, other
options available are CTGan, CopulaGAN etc. GaussianCopula model will look
for joint marginal distribution of multiple features/columns in the dataset
and synthesize a new dataset with statistically similar distribution.
'''
# use serial number as unique identifier
synth_model = GaussianCopula(primary_key='serial_num')
synth_model.fit(patient) #fit the synthesis model

extend_patient = synth_model.sample(2000) # create dataset with 2000 patients
print(extend_patient.head())
print("")
print("The dimensions of extend_patient are",extend_patient.shape)
```

```
   serial_num  Age Sex      BP Cholesterol  dom_marker
0           0   46   F  NORMAL        HIGH      17.986
1           1   35   M  NORMAL        HIGH      12.031
2           2   18   F    HIGH      NORMAL      20.377
3           3   39   F    HIGH        HIGH      12.298
4           4   68   M  NORMAL        HIGH       7.864

The dimensions of extend_patient are (2000, 6)
```

Now, we have a dataset with 2000 patients which is *statistically representative* of the seed dataset.

To evaluate how well the synthesized dataset represents the original dataset, we can use various metrics. Here, I have used *Chi-Squared Test* and *Kolmogorov-Smirnov Test*. The score ranges from `0` to `1` and higher the score, better the synthesis. As can be seen from `CSTest`, the synthesis is acceptable but for `KSTest` it is comparatively lower.

[4]:
```
from sdv.evaluation import evaluate
evaluate(extend_patient, patient, metrics=['CSTest', 'KSTest'], aggregate =␣
 ↪False)
```

[4]:
```
   metric                                    name  raw_score  \
0  CSTest                             Chi-Squared   0.992125
1  KSTest  Inverted Kolmogorov-Smirnov D statistic   0.669500

   normalized_score  min_value  max_value      goal error
0          0.992125        0.0        1.0  MAXIMIZE  None
1          0.669500        0.0        1.0  MAXIMIZE  None
```

Now, let us create `drugA` dataset with 200 patients and corresponding `recovery` labels `YES` and `NO` as described earlier. Also, we do the same for `drugB` with 2000 patients.

[5]:
```
'''
Creating drugB dataset.The probabilities for 'YES' defines the
likeliness of the drug to work on the patient. For drug B, this
likeliness is 0.7 irrespective of other columns for a specific patient.
'''

recovery_B = np.random.choice(['YES', 'NO'], size = 2000, p = [.7, .3])
extend_patient['recovery'] = recovery_B.tolist()
drugB = extend_patient
drugB.head()
```

[5]:
```
   serial_num  Age Sex      BP Cholesterol  dom_marker recovery
0           0   46   F  NORMAL        HIGH      17.986       NO
1           1   35   M  NORMAL        HIGH      12.031      YES
2           2   18   F    HIGH      NORMAL      20.377      YES
3           3   39   F    HIGH        HIGH      12.298      YES
4           4   68   M  NORMAL        HIGH       7.864      YES
```

[6]:
```
'''
Creating drugA dataset.The probabilities for 'YES' defines the
likeliness of the drug to work on the patient. For drug A, this
likeliness is 0.8 irrespective of other columns for a specific patient.
'''

recovery_A = np.random.choice(['YES', 'NO'], size = 200, p = [.8, .2])
patient['recovery'] = recovery_A.tolist()
drugA = patient
drugA.head()
```

[6]:
```
   serial_num  Age Sex      BP Cholesterol  dom_marker recovery
0           0   23   F    HIGH        HIGH      25.355      YES
1           1   47   M     LOW        HIGH      13.093      YES
```

```
2             2   47   M     LOW         HIGH      10.114      YES
3             3   28   F  NORMAL         HIGH       7.798       NO
4             4   61   F     LOW         HIGH      18.043       NO
```

Finally, we have our datasets –

- `drugA` dataset: This is the data acquired from clinical trial held by the drug manufacturer as described earlier in the trial design section. Number of patients is 200 and each patient is labeled for if they recovered from the disease when administered with drug A.
- `drugB` dataset: This is taken as the old data from clinical trials held for drug B. Number of patients is 2000 and each patient is labeled if they recovered from the disease when administered with drug B.

2.1.4 Comments on Synthetic data

1. It is imperative that since the data is synthetic and the disease and drugs work exist in an imaginary situation, the inferences from this data might not make sense in the real world. For example, a patient with *relatively healthier* feature set can be inferred as non-recoverable from the disease which in real life would be highly unlikely given that there are no other factors affecting the patient's recovery. In such cases, the inferences made by the learning algorithm can turn out to be gibberish (in the real world but not in the imaginary world where the problem is set) not due to the model's incapability to learn but due to the dataset itself.
2. Another thing to keep in mind is that the dataset could turn out to be *non-informative* or it could also be *non-separable* i.e., when all feature vectors are plotted in an `n`-dimensional feature space where `n` is the number of features, the feature vectors could possibly show no patterns or clusters in which case, classification would not be possible for the given sample size.
3. There is lack of prior knowledge about the data since the setting is imaginary, in such a case, some exploratory data analysis would be needed to decide the candidate hypothesis classes.
4. An **significant** improvement on assignment of `recovery` labels would have been to use a clustering model to find clusters in the feature space and use those clusters for assignment. However, due to time constraint for this exam, I have avoided that.

Considering these warnings, it is possible that any model we choose may not be able to learn very well but in case, the conditions in the second comment do not come to pass, a model could learn well given the data it has been presented.

2.2 The Learning Model

We set this problem as a **classification** problem in a **supervised** setting. Given a patient, a model would have to predict if they will recover or not when given drug A. Firstly, using the `drugA` dataset, we will train our model and check its accuracy using train-test data split. Secondly, we will calculate the recovery rate using the model for prediction on a dataset which has been labeled for drug B as well (`drugB` dataset). Finally, we will compare the recovery rate for the drugs.

In the following subsections, we define our learning problem in detail and justify our choices for each of the components involved in the learning setup.

2.2.1 Feature Vectors

Before even defining the learning model in this section, I created datasets with particular feature sets or columns in mind. The feature vectors or the input samples are just the patient's data `(Age, Sex, BP, Cholesterol, dom_marker)` as shown in the datasets. Formally for $x \in \mathcal{X}$, x takes the form

$$x^k = \left(x_1^k, x_2^k, ..., x_m^k\right)$$

where $k = |\mathcal{X}|$ and m is number of features. Here, k will be equal to however many samples we choose to train our model with and $m = 5$ as can seen be from the dataset.

2.2.2 Loss Function

Since this problem is a *binary classification problem*, the simplest choice would be the **0-1 loss**[8] where the loss function outputs `0` if the predicted value is equal to the true value and `1`, otherwise.

2.2.3 Hypothesis Classes & PAC Learnability

For this particular problem of drug comparison, there is little prior knowledge that can be used for *inducing a bias*, therefore choosing hypothesis classes with the help of domain knowledge is not possible. It is possible to reduce the number of hypothesis candidates by visualizing the data as well, however it would not be possible for feature vectors with features greater than 3 since visualization would not be possible in those cases. Earlier in section 2.1.1, while discussing the number of samples taken in the trial study, I decided to take an arbitrary number because for learning a PAC solution we would require the **cardinality of the hypothesis class** which in fact has not been chosen due to the aforementioned reasons. Therefore, *no comment on the sample complexity could be made.* As we will see in the next section, when we choose our learner, we can easily change hypothesis classes as well.

Thus, instead of fixing the hypothesis class $\mathcal{H}$ ahead of time, we can go for different models and use validation to find the best model as discussed in Chapter 7 [8].

2.2.4 Output Labels

As mentioned earlier, this is a binary classification problem and the classification labels are `YES` when the patient recovers from the disease within 14 days upon taking drug A and `NO` when the patient does not recover.

2.3 Learning Algorithms & their Performance

For a classification task, many learning algorithms are present [9] such as logistic regression, naive Bayes, Support Vector Machines (SVMs), Decision Trees, Random Forest etc. For this particular task, I choose to use SVMs due to their *robustness* and *ability to deal with highly multidimensional feature spaces* [8]. One other reason is that if SVM is chosen to be the learner for our problem, it is easier to use different hypothesis classes by simply changing the **kernel** function which is quite easy to implement as well. Before, using these learning algorithms, it is important to make the data ready for input.

2.3.1 Data Preprocessing

Let us first remove features which have no significance for learning the classification task.

[7]:
```
# removing serial_num since it has no significance for classification
drugA = drugA.drop(['serial_num'], axis = 1)
drugA.head()
```

[7]:
```
   Age Sex      BP Cholesterol  dom_marker recovery
0   23   F    HIGH        HIGH      25.355      YES
1   47   M     LOW        HIGH      13.093      YES
2   47   M     LOW        HIGH      10.114      YES
3   28   F  NORMAL        HIGH       7.798       NO
4   61   F     LOW        HIGH      18.043       NO
```

Let us also check out the data type of each feature in the dataset. Below, it can be seen that the features `Sex`, `BP`, `Cholesterol` and `recovery` are categorical.

[8]:
```
print(drugA.dtypes)
```

```
Age              int64
Sex             object
BP              object
Cholesterol     object
dom_marker     float64
recovery        object
dtype: object
```

For dealing with categorical variables, we can use encoding [11] according to the type of categorical feature. `recovery` will be our target variable, therefore it needs to be encoded as well. For `NO`, we can encode it as `0` and for `YES`, encode it as `1`.

[9]:
```
drugA.loc[drugA["recovery"] == "YES", "recovery"] = 1
drugA.loc[drugA["recovery"] == "NO", "recovery"] = 0

#change recovery variable to numeric to avoid one-hot encoding
drugA['recovery'] = pd.to_numeric(drugA['recovery'])
```

[10]:
```
# using get_dummies method to one-hot encode all categorical variables
ohe_drugA = pd.get_dummies(drugA)
ohe_drugA.head()
```

[10]:
```
   Age  dom_marker  recovery  Sex_F  Sex_M  BP_HIGH  BP_LOW  BP_NORMAL  \
0   23      25.355         1      1      0        1       0          0
1   47      13.093         1      0      1        0       1          0
2   47      10.114         1      0      1        0       1          0
3   28       7.798         0      1      0        0       0          1
4   61      18.043         0      1      0        0       1          0
```

```
   Cholesterol_HIGH  Cholesterol_NORMAL
0                 1                   0
1                 1                   0
2                 1                   0
3                 1                   0
4                 1                   0
```

Let us also check the distribution of target labels to see if we have a balanced or an imbalanced dataset.

[11]:
```
ohe_drugA['recovery'].value_counts()/np.float(len(ohe_drugA))
```

[11]:
```
1    0.81
0    0.19
Name: recovery, dtype: float64
```

2.3.2 Training and testing algorithms

First, we create a input sample dataset `X` and `y` is the target data.

[12]:
```
# drop target variable
X = ohe_drugA.drop(['recovery'], axis = 1)
y = ohe_drugA['recovery'] #create target dataframe
```

Now, we do a train-test split using `scikit-learn` functions. From here on, the code has been majorly reused from [12].

[13]:
```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.2,␣
 ↪random_state = 42)
X_train.shape, X_test.shape
```

[13]:
```
((160, 9), (40, 9))
```

Feature scaling [13] to standardize independent features.

[14]:
```
from sklearn.preprocessing import StandardScaler
cols = X_train.columns
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)
X_train = pd.DataFrame(X_train, columns=[cols])
X_test = pd.DataFrame(X_test, columns=[cols])
```

Hypothesis Class I : Radial Basis Functions Here, the kernel function used with SVM is radial basis function (rbf), changing the parameter `C` which handles outliers results in different hypothesis in this class. Here, we consider three hypothesis with `C` = 1, 100 and 1000. The code cell below implements `C` = 1.0 which is the default value.

[15]:
```
# import SVC classifier
from sklearn.svm import SVC
# import metrics to compute loss and accuracy
from sklearn.metrics import zero_one_loss
from sklearn.metrics import accuracy_score
# instantiate classifier with default hyperparameters
svc=SVC()
# fit classifier to training set
svc.fit(X_train,y_train)
# make predictions on test set
y_pred=svc.predict(X_test)
# compute and print accuracy score
print('Model accuracy score with default hyperparameters: {0:0.4f}'.␣
 ↪format(1-zero_one_loss(y_test, y_pred)))
```

```
Model accuracy score with default hyperparameters: 0.8250
```

[16]:
```
# instantiate classifier with rbf kernel and C=100
svc=SVC(C=100.0)
# fit classifier to training set
svc.fit(X_train,y_train)
# make predictions on test set
y_pred=svc.predict(X_test)
# compute and print accuracy score
print('Model accuracy score with rbf kernel and C=100.0 : {0:0.4f}'.␣
 ↪format(1-zero_one_loss(y_test, y_pred)))
```

```
Model accuracy score with rbf kernel and C=100.0 : 0.8000
```

[17]:
```
# instantiate classifier with rbf kernel and C=1000
svc=SVC(C=1000.0)
# fit classifier to training set
svc.fit(X_train,y_train)
# make predictions on test set
y_pred=svc.predict(X_test)
# compute and print accuracy score
print('Model accuracy score with rbf kernel and C=1000.0 : {0:0.4f}'.␣
 ↪format(1-zero_one_loss(y_test, y_pred)))
```

```
Model accuracy score with rbf kernel and C=1000.0 : 0.8000
```

Hypothesis Class II : Linear Functions Here, in this class, the kernel function is limited to linear functions. As before, the value of `C` is changed and three possible hypothesis are made.

[18]:
```
# instantiate classifier with linear kernel and C=1.0
linear_svc=SVC(kernel='linear', C=1.0)
# fit classifier to training set
linear_svc.fit(X_train,y_train)
```

```python
# make predictions on test set
y_pred_test=linear_svc.predict(X_test)
# compute and print accuracy score
print('Model accuracy score with linear kernel and C=1.0 : {0:0.4f}'.␣
 ↪format(1-zero_one_loss(y_test, y_pred_test)))
```

```
Model accuracy score with linear kernel and C=1.0 : 0.8250
```

[19]:
```python
# instantiate classifier with linear kernel and C=100.0
linear_svc100=SVC(kernel='linear', C=100.0)
# fit classifier to training set
linear_svc100.fit(X_train, y_train)
# make predictions on test set
y_pred=linear_svc100.predict(X_test)
# compute and print accuracy score
print('Model accuracy score with linear kernel and C=100.0 : {0:0.4f}'.␣
 ↪format(1-zero_one_loss(y_test, y_pred_test)))
```

```
Model accuracy score with linear kernel and C=100.0 : 0.8250
```

[20]:
```python
# instantiate classifier with linear kernel and C=1000.0
linear_svc1000=SVC(kernel='linear', C=1000.0)
# fit classifier to training set
linear_svc1000.fit(X_train, y_train)
# make predictions on test set
y_pred=linear_svc1000.predict(X_test)
# compute and print accuracy score
print('Model accuracy score with linear kernel and C=1000.0 : {0:0.4f}'.␣
 ↪format(1-zero_one_loss(y_test, y_pred_test)))
```

```
Model accuracy score with linear kernel and C=1000.0 : 0.8250
```

From the accuracy score, it can be seen that **Linear SVM** is a better hypothesis class since all the hypothesis have lesser loss score as compared to **Radial Basis Function SVM**. However, this accuracy can be improved if we had better prior knowledge to narrow down the hypotheses. Now, let us select linear SVM model for checking underfitting or overfitting.

[21]:
```python
# prediction on training dataset
y_pred_train = linear_svc.predict(X_train)
```

[22]:
```python
# checking for overfitting
print('Training-set accuracy score: {0:0.4f}'. format(1-zero_one_loss(y_train,␣
 ↪y_pred_train)))
```

```
Training-set accuracy score: 0.8063
```

[23]:
```python
# print the scores on training and test set
```

```python
print('Training set score: {:.4f}'.format(linear_svc.score(X_train, y_train)))
print('Test set score: {:.4f}'.format(linear_svc.score(X_test, y_test)))
```

```
Training set score: 0.8063
Test set score: 0.8250
```

From above, it is obvious that our model performs averagely because of the hypothesis classes that we selected. Thus, the guarantees that this model provides for its predictions is not strong. To get stronger guarantees i.e., lesser values of ϵ and δ, we need more data for training.

2.4 Difference between finding which is better versus being indeed better?

Now that we have selected our hypothesis (however inaccurate it may be), we can find whether drug A has better recovery rate than drug B. However, when we try to do that, we do not have particulary a high confidence with the predicted recovery rate for drug A. In that case, we cannot be sure if our drug is *indeed* better i.e., in terms of PAC solution, the probability of being accurate is not very high. To say one drug is *indeed* better than the other, we would need to provide the prediction accurately with probability 1 but for that sample complexity would have to very high (infinite) since sample complexity is inversely proportional to confidence parameter δ.

There is another way to comment on if one drug is better than the other or not and that is through finding the **effectiveness** of the drug [1] rather than the **efficacy** which means we need prove using a learning algorithm that the drug is more effective (it performs well in the real world not only in a controlled lab environment). To do that, a learning algorithm would need to find the relationship between efficacy and effectiveness. This can be done through a thorough study of relationship between surrogate-based outcomes and patient-oriented outcomes [1].

For a PAC solution to favor drug A, the dataset should also reflect superiority of drug A otherwise, our model will never favor A.

2.5 Conclusion

Drug comparison requires a lot of domain-specific study and trials have to be designed very carefully. A machine learning algorithm without an inductive bias will not be good learner since it will have to search in a very large hypothesis space for which large amount of data is required which is not alway available as in cases of new drugs being introduced in the market or in trial phases.

3 References

[1] Drug Efficacy and Safety

[2] Eduara Vieta, Nuria Cruz,Head to head comparisons as an alternative to placebo-controlled trials, European Neuropsychopharmacology, Volume 22, Issue 11, 2012, Pages 800-803, ISSN 0924-977X, DOI

[3] Randomised Trials

[4] Kim H, Gurrin L, Ademi Z, Liew D. Overview of methods for comparing the efficacies of drugs in the absence of head-to-head clinical trial data. Br J Clin Pharmacol. 2014;77(1):116-121,DOI

[5] Clinical Trial Results

[6] https://www.kaggle.com/datasets/pablomgomez21/drugs-a-b-c-x-y-for-decision-trees

[7] SDV Documentation

[8] Shai Shalev-Shwartz and Shai Ben-David. 2014. Understanding Machine Learning: From Theory to Algorithms. Cambridge University Press, USA.

[9] Machine Learning Algorithms for Classification

[10] Lecture notes, EE5180 (Spring 2022), IIT Madras

[11] One Hot Encoding & Label Encoding

[12] SVM Classifier Tutorial

[13] Feature Scaling