Scenario:-

A pharma company released a new drug that can cure cancer. However, this drug is fatal for patients with a particular set of molecules in their DNA. Let us assume that a single DNA strand is enough to identify if a drug is fatal for a person. Let us also assume that each molecule in the DNA can be one among four types. Problem is, each DNA strand is two billion molecules long, and it is hard to identify problematic molecules manually.

The company is interested in training a classifier to automatically decide if this treatment is safe for the patient. They have collected DNA strands from ten thousand people. The drug is fatal for hundred among these ten thousand people. They have provided both the DNA strand data, and its corresponding label (safe or fatal). For clarity, let's assume the label "safe" is a positive label and label "fatal" is a negative label.

Background :-

Cancer classification based on molecular level investigation has gained the interest of researches as it provides a systematic, accurate and objective diagnosis for different cancer types. Several recent researches have been studying the problem of cancer classification using data mining methods, machine learning algorithms and statistical methods to reach an efficient analysis for gene expression profiles.

Studying the characteristics of thousands of genes simultaneously offered a deep insight into cancer classification problem. It introduced an abundant amount of data ready to be explored. It has also been applied in a wide range of applications such as drug discovery, cancer prediction and diagnosis which is a very important issue for cancer treatment. Besides, it helps in understanding the function of genes and the interaction between genes in normal and abnormal conditions. That is done by monitoring the behavior of genes -gene expression data- under different conditions.

In this report , an effective linear classification model(**Logistic Regression**) is being proposed coupled with feature selection/elimination approach(**L1 regularization**).

Machine Learning Objectives and Constraints :-

Objective:- Classify each data-point into positive ("safe") or negative ("unsafe") classes .

Constraints:-

- **Interpretability : -** Model must give the result which can be interpreted .
- **Class probabilities are needed : -** For each data point , model must give the probabilities of its belongingness to each class .
- **Penalize the errors in class probabilities : -** In this problem errors can be very costly .
- **No Latency constraints while predicting .**

Workflow :-

1. Data Collection and Reading :-

Dataset is being given which contains data of DNA strands collected from 10000 people along with their class labels . In order to read the csv dataset we need to import **pandas** library .

2. Data Preprocessing :-

- Converting each class label into "1"(for positive) and "0"(for negative) label before classification .
- Given each molecule in DNA can be among 4 types , features of our dataset are categorical variable . Before classification we need to convert our features into numerical vectors . We have two approaches to do it :- 1. **OneHot Encoding** , 2. **Response Coding with laplace smoothing .** We will proceed with OneHot Encoding . Though it increases the dimensionality of our dataset but it works pretty well with linear classifiers . To do it we need to import **sklearn.preprocessing.OneHotEncoder** .

3. Splitting dataset into train and test (64:16:20) : -

Note:- Separate Cross validation set is not taken because we will perform stratified K-fold cross validation while hyperparameter tuning .

We split our dataset into following ratio train_x : cross_x: test_x = 64 : 16 : 20 .Similarly , our target class is also splitted in ratio train_y : cross_y : test_y = 64 : 16 : 20 . Splitting is random and ratio of different classes is maintained in each set after splitting . To do it we need to import **sklearn.model_selection.train_test_split .**

4. Checking the stability of features and labels across train , cross - validate and test datasets :-

We need to check the stability(distribution type and overlapping) of features and labels across train cross-validate and test datasets .

Problem :- What if datasets are not stable i.e train , cross-validate and test datasets do not follow same distribution type or they sparsely overlap with each other .

Solution :- Resample the whole dataset and again check for stability .

5. Model Selection :-

This is the most important choice we have to make . All of our predictions , accuracy etc depends on what machine learning model/algorithm we should choose . Here we will proceed with linear algorithm(**Logistic Regression**) . Lets see why we don't proceed with other models :-

- **Baseline models like Linear SVM :-** Linear SVM is a classical algorithm which works very well with classification task . But it performs poor in high dimension dataset and our dataset is very dimensional . Also this model is not so interpretable .
- **Ensemble Models :-** Don't give much edge over linear models . Very low interpretability . Very expensive during computation .
- **Model Stacking or Maximum Voting classifier :-** Performance depends much on dataset . Some of the models inside it won't perform well on high dimensional datasets .Very expensive during computation .
- **Neural Nets :-** Not enough data . Overfitting problem .

Lets see why we choose linear model(**Logistic Regression**) over other models :-

- Works extremely well with high dimensional dataset .
- Model Interpretability is very good .
- Simple Model with simple assumptions .
- Computationally cheap .

To do this we import **sklearn.linear_model.SGDClassifier** with **loss** argument as “**log**” .

Problem :- Our dataset is imbalanced and will affect our predictions .

Solution :- We will balance our dataset . For that we can opt any of two approaches :-

1. Use **SMOTE** for upsampling
2. While using SGDClassifier , make **class_weight** argument as “**balanced**” .

Problem :- How to avoid the problem of **overfitting** and **underfitting** .

Solution :- We will introduce the **regularization** term which keeps a check over these problems .

6. Hyperparameter Tuning(alpha):-

Note:- We are not considering **elastic net** as regulariser .

Question :- Why hyperparameter tuning ?

Answer :- Hyperparameter(alpha) tuning provides our optimal weight vector for classification .

Code snippet of hyperparameter tuning:-

```
alpha = [10 ** x for x in range(-6, 3)]

cv_log_error_array = []

for i in alpha:

    print("for alpha =", i)
    clf = SGDClassifier(class_weight='balanced', alpha=i, penalty='l1', loss='log')
    clf.fit(train_x, train_y)
    sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
    sig_clf.fit(train_x, train_y)
    sig_clf_probs = sig_clf.predict_proba(cross_x)
    cv_log_error_array.append(log_loss(cross_y, sig_clf_probs, labels=clf.classes_, eps=1e-15))
best_alpha = np.argmin(cv_log_error_array).
```

To do this we need to import the libraries :- **sklearn.calibration.CalibratedClassifierCV** , **sklearn.metrics.classification.log_loss and numpy.**

Problem :- How to get important features or eliminate less important features ?

Solution :- To select features in the logistic regression model,we used the implementation of $L1$-regularised logistic regression , which follows **Zhu** and **Hastie's** symmetric formulation of binary logistic regression. The L 1 regularisation term, which is tuned by a user-defined parameter λ, shrinks the coefficients of less relevant features down to zero, thus discarding them.

7. Testing the model with best hyper parameter : -

Problem :- Which metric to choose and why ?

Solution :- The overall accuracy of a classifier is not very informative by itself because it does not tell us how well each class is classified . Also in this problem we need to look at each aspect of classification because even small mistake can end up killing a person . Therefore, we consider precision and recall along with confusion matrix. For binary classification, precision and recall are defined as:-

Precision = **True Positive / (True Positive + False Positive) .**

Precision provides a deep insight about how correctly/precisely we classify on test data.

Recall = True Positive / Total Positives .

Recall provides an insight about how many points we are classifying correctly .

Code snippet of testing :-

from sklearn.metrics import precision_recall_fscore as score
from sklearn.metrics import confusion_matrix

clf = SGDClassifier(class_weight='balanced', alpha=alpha[best_alpha], penalty='l1', loss='log')

y_pred = clf.predict(text_x)

print(confusion_matrix(test_y,y_pred))

Precision , **Recall** , **fscore** = score(test_y , y_pred).

print(**Precision**)

print(**Recall**)

print(**fscore**)

Question :- Is it possible to identify the exact molecules in the DNA that make the drug fatal, given just the classifier that you trained to identify if the drug is fatal?

Answer :- Yes , it is possible to identify the exact molecules in the DNA that make the drug fatal . This will proceed as follows :-

- prob_sorted = sorted(zip(clf.coef_[0], features_index))   # here features_index shows the index of encoded features
- print(prob_sorted[:10])

From above we get the feature indexes .Now to get the index of molecules , we will divide the feature indexes by 4(as each molecule is being OneHotEncoded into 4 numerical features) . If remainder is 0 , quotient is the index of the molecule otherwise quotient+1 is the index of molecule . Following this for all the feature indexes , we will get all the indexes of molecules .

Flow Chart

START

Importing Libraries

Data Collection

Data Reading

Data Preprocessing

Splitting Data into various Sets

Checking Dataset Stability

Model Selection

Hyperparameter Tuning

Testing with tuned hyperparameter

Calculating confusion matrix , F1 score , precision and recall

END