UNIVERSITÉ BISHOP'S UNIVERSITY

UNIVERSITÉ BISHOP'S UNIVERSITY

A Project Report On

"Predict Disease Classes Using Genetic Microarray Data"

Submitted in Partial Fulfillment Of

CS 590 - Data Mining Project

To

Dr. Layachi Bentabet

By

Bhoopalsinh Musale	**002269332**
Syed Malik Muzaffar	**002269955**

UNIVERSITÉ BISHOP'S UNIVERSITY

Step 1: Load Data

- For loading gene data which has comma-separated values.
- Using Pandas library, we load the data and perform transpose.

Load Dataset Function

```
def load_data():
    '''
        Load data from CSV file
        returns X,Y and random seeds
    '''
    dataFrame = pd.read_csv('pp5i_train.gr.csv')
    dataFrame.set_index('SNO', inplace=True)
    dataFrame = dataFrame.transpose()
    dataFrame.reset_index(drop=True, inplace=True)

    y = pd.read_csv('pp5i_train_class.txt')
    dataFrame = pd.concat([dataFrame, y], axis=1)
    myRndSeeds = 72
    dataFrame = dataFrame.sample(frac=1, random_state=myRndSeeds).
reset_index(drop=True)
    print(dataFrame.shape)
    print(dataFrame.head())

    X = dataFrame.drop('Class', axis=1)

    y = dataFrame['Class']

    return X, y, myRndSeeds
```

Step 2: Data Cleaning

- Threshold both train and test data to a minimum value of 20, maximum of 16,000.

Dataset Pre-processing Function

```
def clean_data(X):
    '''
        Thresholding both train and test data
        to a minimum value of 20, maximum of 16,000.
    '''
    X.clip(upper=16000, lower=20, inplace=True)
    print(X.shape)
    X = X.loc[:, X.max() - X.min() > 2]
    print(X.shape)
    return X
```

Step 3: Feature Selection and T-test transformer

- Feature Selection selects features that provide significant contribution to our prediction output. It is an essential step in this project.
- So, for performing feature selection in this project, we have performed a t-test for each class from the sample of that class and in remaining classes. The genes with highest absolute t- values are selected from each gene class. We have calculated t-values as following:

$$t = \frac{\overline{X_1} + \overline{X_2}}{\sqrt{{s_1^2}/{n1} + {s_2^2}/{n2}}}$$

- When two population variances are not assumed to be equal, Welch's t-test can be used.
- Transformer is implemented which can be compatible with sklearn.pipelines. It selects top w features with highest t-scores for each class and selects features using t-score. Implementation of t-test from scypy library has been used here.

UNIVERSITÉ BISHOP'S UNIVERSITY

T-test Score Feature Selection Class

```
class TScoreSelection(BaseEstimator, TransformerMixin):
    def __init__(self, w=3):
        self.w = w

    def fit(self, X, y=None):
        X = check_array(X)
        self.input_shape_ = X.shape

        # Check that X and y have correct shape
        X, y = check_X_y(X, y)

        # Store the classes seen during fit
        self.labels_ = unique_labels(y)
        self.tValuesDF = pd.DataFrame(columns=self.labels_)
        self.sortedIndexes = pd.DataFrame(columns=self.labels_)

        for label in self.labels_:
            sample1 = X[y == label]
            sample2 = X[y != label]

# In canse of both samples have variance 0 I put some arbitrary values on sample 1
            zeroVarBothColsIdx = (
                np.var(sample1, axis=0) + np.var(sample2, axis=0)) == 0
            sample1[:, zeroVarBothColsIdx] = 10e6
            sample1[0, zeroVarBothColsIdx] = 1
            # perform the t test
            t = st.ttest_ind(sample1, sample2, equal_var=False)
            # I set t-values to 0 for the columns where both samples had 0 variance
            t[0][zeroVarBothColsIdx] = 0
            self.tValuesDF[label] = np.abs(t[0])
            self.sortedIndexes[label] = self.tValuesDF.sort_values(by=label,
                                                                   ascending=False).index


        # Return the transformer
        return self

    def transform(self, X):
        # Check is fit had been called
        check_is_fitted(self, ['input_shape_'])

        # Input validation
        X = check_array(X)

        # union of indexes from the top w columns for each label
        self.selCols = np.unique(
            self.sortedIndexes[:][0:self.w].values.flatten())

        return X[:, self.selCols]
```

Step 4: Classification and their Individual Result

I. K-NN (K=2,3,4)

- KNN classification is supervised learning algorithm. Here, we are given a training dataset which has training observations (x, y) and would like to capture relationship between x and y. Our motive here is to learn a function h:X→Y so that when given an unseen observation x, h(x) can predict the corresponding output y.
- Following output screenshot shows results of KNN classifier with K= [2,3,4] and K=3 gives highest mean test score. But we have tried different values of K also.

K-NN Score

	mean_fit_time	std_fit_time	mean_score_time	std_score_time	param_classify	param_classify__n_neighbors
12	0.063440	0.002741	0.007848	0.000129	KNeighborsClassifier(algorithm='auto', leaf_si...	3
21	0.053632	0.001682	0.006433	0.000275	KNeighborsClassifier(algorithm='auto', leaf_si...	4
13	0.058528	0.004633	0.006777	0.000504	KNeighborsClassifier(algorithm='auto', leaf_si...	3
18	0.054408	0.002052	0.006542	0.000258	KNeighborsClassifier(algorithm='auto', leaf_si...	3
11	0.061706	0.002437	0.007071	0.000090	KNeighborsClassifier(algorithm='auto', leaf_si...	3

II. Decision Tree

- Decision Tree algorithm belongs to supervised learning algorithms family, but it can be also used for solving regression and classification problems.
- By learning simple decision rules put from prior data i.e. training data, Decision Tree model can create a training model which predicts the class or value of target variable.
- For prediction of class label for a record, we start from a root of tree. Then we compare values of root attribute with record's attribute. With help of comparison we can follow the branch corresponding to value and jump to next node.

Decision Tree Score

	mean_fit_time	std_fit_time	mean_score_time	std_score_time	param_classify	param_classify__criterion
78	0.052294	0.001500	0.005202	0.000161	DecisionTreeClassifier(ccp_alpha=0.0, class_we...	gini
113	0.052228	0.000396	0.005318	0.000270	DecisionTreeClassifier(ccp_alpha=0.0, class_we...	entropy
138	0.053521	0.000588	0.005183	0.000098	DecisionTreeClassifier(ccp_alpha=0.0, class_we...	entropy
159	0.055541	0.001910	0.005583	0.000394	DecisionTreeClassifier(ccp_alpha=0.0, class_we...	entropy
77	0.052195	0.001628	0.005305	0.000337	DecisionTreeClassifier(ccp_alpha=0.0, class_we...	gini

III. Neural Network (MLP Classifier)

- MLP Multi-layer Perceptron classifier connects to a neural network. It depends on an underlying Neural Network to perform the task of classification unlike Support Vectors or Naive Bayes.
- Implementation of MLP Classifier takes no more effort than implementation of Support Vectors or Naive Bayes or any other classifiers from Scikit-Learn.

MLP Classifier Score

mean_fit_time	std_fit_time	mean_score_time	std_score_time	param_classify	param_classify__activation	param_classify__alpha
0.502029	0.207907	0.008172	0.001638	MLPClassifier(activation='logistic', alpha=0.0...	logistic	0.001
2.542058	0.303187	0.013660	0.000137	MLPClassifier(activation='logistic', alpha=0.0...	logistic	0.05
1.976578	0.221125	0.012142	0.002369	MLPClassifier(activation='logistic', alpha=0.0...	logistic	0.05
0.948243	0.052448	0.008451	0.002237	MLPClassifier(activation='logistic', alpha=0.0...	logistic	0.005
1.545984	0.049438	0.008211	0.001720	MLPClassifier(activation='logistic', alpha=0.0...	logistic	0.05

IV. Naïve Bayes

- Naive Bayes, a probabilistic machine learning model that's used for classification task. The classifier is based on the Bayes theorem.

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

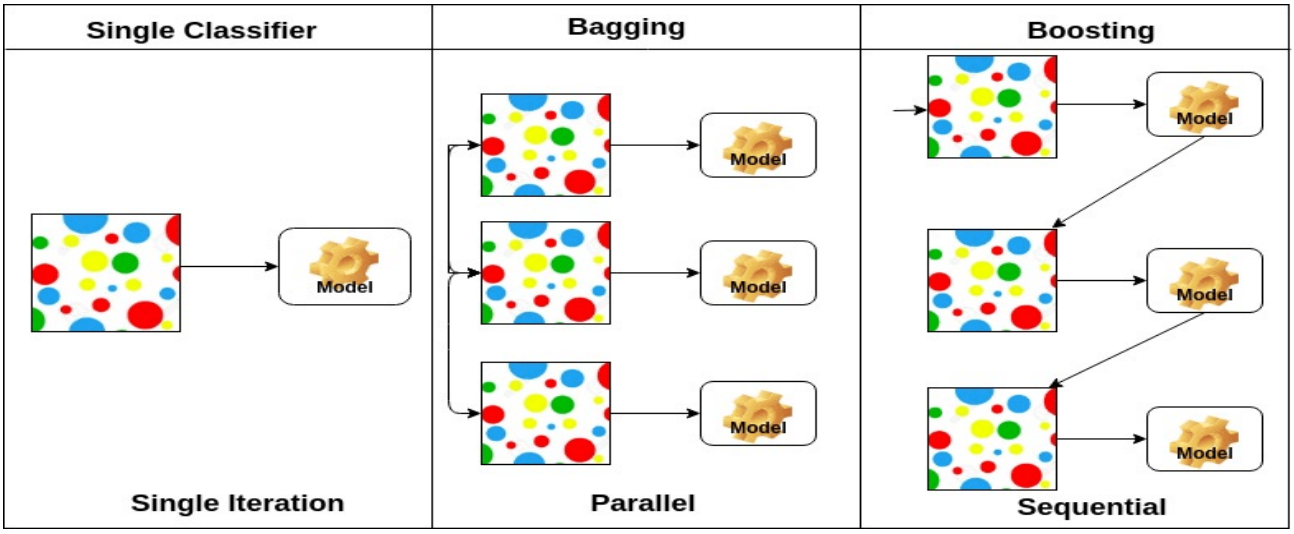
- Using Bayes theorem, we can find the probability of A happening, given that B has occurred. Here, A is the hypothesis and B is the evidence. The assumption here is that the features are independent. Presence of one feature will not affect the other. Hence called naive.

Naïve Bayes Classifier Score

mean_fit_time	std_fit_time	mean_score_time	std_score_time	param_classify	param_featureSelection__w	params	split0_test_score
0.393018	0.007773	0.006969	0.000085	GaussianNB(priors=None, var_smoothing=1e-09)	12	{'classify': GaussianNB(priors=None, var_smoot...	0.785714
0.395834	0.009526	0.007217	0.000440	GaussianNB(priors=None, var_smoothing=1e-09)	10	{'classify': GaussianNB(priors=None, var_smoot...	0.714286
0.393227	0.011151	0.006770	0.000279	GaussianNB(priors=None, var_smoothing=1e-09)	2	{'classify': GaussianNB(priors=None, var_smoot...	0.500000
0.386070	0.006099	0.007025	0.000363	GaussianNB(priors=None, var_smoothing=1e-09)	4	{'classify': GaussianNB(priors=None, var_smoot...	0.428571
0.393847	0.009774	0.006962	0.000126	GaussianNB(priors=None, var_smoothing=1e-09)	6	{'classify': GaussianNB(priors=None, var_smoot...	0.500000

V. AdaBoost Classifier

- Bagging means bootstrap aggregation. It is combination of multiple learners to reduce the variance of estimates. For example, random forest trains M Decision Tree, you can train M different trees on different random subsets of the data and perform voting for final prediction. Bagging consist of algorithms Random Forest and Extra Trees.
- Boosting algorithms, set of the low accurate classifier combined to create a highly accurate classifier. Low accuracy classifier (or weak classifier) offers the accuracy better than we can say the flipping of a coin. Highly accurate classifier (or strong classifier) offers error rate closer to 0. This algorithm is able to track the model who failed accurate prediction. The following three algorithms have gained massive popularity in data science competitions:
    - AdaBoost (Adaptive Boosting)
    - Gradient Tree Boosting
    - XGBoost
- Stacking (or stacked generalization) is an ensemble learning technique. It combines multiple base classification models' predictions to form a new data set. This new data is treated as the input data to another classifier. The classifier is employed to solve this problem. Stacking is also called as blending.



UNIVERSITÉ BISHOP'S UNIVERSITY

AdaBoost Classifier Score

mean_fit_time	std_fit_time	mean_score_time	std_score_time	param_classify	param_featureSelection__w	params
0.502445	0.005657	0.015822	0.000556	AdaBoostClassifier(algorithm='SAMME.R', base_e...	25	{'classify': AdaBoostClassifier(algorithm='SAM...
0.500107	0.007167	0.015988	0.000332	AdaBoostClassifier(algorithm='SAMME.R', base_e...	15	{'classify': AdaBoostClassifier(algorithm='SAM...
0.502075	0.013482	0.015607	0.001293	AdaBoostClassifier(algorithm='SAMME.R', base_e...	30	{'classify': AdaBoostClassifier(algorithm='SAM...
0.500440	0.007808	0.015482	0.000506	AdaBoostClassifier(algorithm='SAMME.R', base_e...	20	{'classify': AdaBoostClassifier(algorithm='SAM...
0.496071	0.016122	0.015527	0.000284	AdaBoostClassifier(algorithm='SAMME.R', base_e...	6	{'classify': AdaBoostClassifier(algorithm='SAM...

Step 4: Best Model Selection

- We evaluate all above classifiers and with different hyperparameters to find out best estimator.
- For this, we made use of sklearn's **Pipeline** class. It sequentially applies a list of transforms and a final estimator. Intermediate steps of the pipeline must implement fit and transform methods. The final estimator only needs to implement fit. The transformers in the pipeline can be cached using memory argument.
- The motive of the pipeline is to assemble several steps that can be cross validated altogether while setting different parameters. For this, it enables setting parameters of the various steps using their names and the parameter name separated by a '__'. A step's estimator may be replaced by setting the parameter with its name to another estimator, or a transformer removed by setting it to 'passthrough' or None.
- Feature selection and the model training are done using cross-validation, in order to avoid data leakage.

Code Snippet: Hyperparameter Tuning and Best Model Estimation

```
# Top Gene Selection
    N_GENES = [2, 4, 6, 8, 10, 12, 15, 20, 25, 30]
    N_LAYERS = [(32,), (64,), (128,)]

    # Hyperparameter Optimization
    tuned_parameters = [
        # KNN Classifier(2,3,4)
        {'featureSelection__w': N_GENES,
         'classify': [KNeighborsClassifier()],
         'classify__n_neighbors': [2, 3, 4]
         },
        # Decision Tree Classifier(J48 algorithm)
        {'featureSelection__w': N_GENES,
         'classify': [tree.DecisionTreeClassifier()],
         'classify__criterion':['gini', 'entropy'],
         'classify__min_samples_leaf': [1, 3, 5],
         'classify__max_depth': [3, 6, 9],
         'classify__presort': [True]
         },
        # Neural Network Multi-label Classifier
        {'featureSelection__w': N_GENES,
         'classify': [MLPClassifier()],
         'classify__hidden_layer_sizes': N_LAYERS,
         'classify__activation': ['logistic'],
         'classify__alpha':[0.05, 0.01, 0.005, 0.001],
         'classify__max_iter':[1000],
         'classify__solver': ['lbfgs'],
         'classify__verbose': [True]
         },
        # Naïve Bayes Classifier
        {'featureSelection__w': N_GENES,
         'classify': [naive_bayes.GaussianNB()]
         },
        # AdaBoost Classifier
        {'featureSelection__w': N_GENES,
         'classify': [AdaBoostClassifier()]
         }
    ]

    # Model Selection using Pipeline and Cross validation
    kfolds = KFold(n_splits=5, shuffle=True, random_state=myRndSeeds)
    model = GridSearchCV(pipe, tuned_parameters, cv=kfolds,
                         return_train_score=True)
    model.fit(X, y)
    results = pd.DataFrame(model.cv_results_)

    print(results.sort_values(by='mean_test_score', ascending=False).head())

    # Best Model
    best_estimator_ = model.best_estimator_
    print(best_estimator_)
```

UNIVERSITÉ BISHOP'S UNIVERSITY

Best Model

```
best_estimator_ = model.best_estimator_
best_estimator_
```

```
Pipeline(memory='/tmp/tmp_ab1za2f',
         steps=[('featureSelection', TScoreSelection(w=15)),
                ('classify',
                 MLPClassifier(activation='logistic', alpha=0.05,
                               batch_size='auto', beta_1=0.9, beta_2=0.999,
                               early_stopping=False, epsilon=1e-08,
                               hidden_layer_sizes=(64,),
                               learning_rate='constant',
                               learning_rate_init=0.001, max_fun=15000,
                               max_iter=1000, momentum=0.9, n_iter_no_change=10,
                               nesterovs_momentum=True, power_t=0.5,
                               random_state=None, shuffle=True, solver='lbfgs',
                               tol=0.0001, validation_fraction=0.1,
                               verbose=True, warm_start=False))],
         verbose=False)
```

Predictions of Test Dataset with Best Model

```
# Running best model on Test dataset
testDataFrame = pd.read_csv('pp5i_test.gr.csv')
testDataFrame.set_index('SNO', inplace=True)
X_test = testDataFrame.transpose()
X_test.reset_index(drop=True, inplace=True)

# Generating output Y for given Test Dataset
Y = pd.DataFrame()
Y['predicted'] = model.predict(X_test)
finalResult = Y

# Final Output
print(finalResult)
```

UNIVERSITÉ BISHOP'S UNIVERSITY

Final Output (Predications)

	predicted
0	MGL
1	RHB
2	MGL
3	MED
4	RHB
5	RHB
6	MED
7	MED
8	MED
9	EPD
10	MED
11	EPD
12	MED
13	MED
14	RHB
15	MED
16	MED
17	MGL
18	MED
19	MED
20	MED
21	RHB
22	RHB