1) Introduction

The Brief for this report outlined a binary classification task to be performed on a dataset containing 5,782 samples. All samples consist of 4608 features including both gist and CNN. 615 samples were assigned training and contained no null values .3,000 samples were assigned training and contained null values. Final 2,167 samples were assigned test data and contained null values. The samples represent images labelled as either 1 for sunny or 0 for dark. The outcome is to produce class labels for the test data provided.

1.1) Approach

Data Pre-Processing applied before training the classifiers. Pre-Processing includes imputing (1), normalization (2) and feature selection (3). Training data split into total, 100%confidence, 66%confidence sets. Classifiers chosen follow:

Perceptron **(4)** - linear machine learning algorithm used for binary classification tasks. Works by taking a row of input features and predicting a class label. Model fit using respective training data sets. Hyperparameters for Perceptron are learning rate and max epoch iterations. GridsearchCV is utilized to return optimal values for both (5). Accuracy scores calculated using RepeatedStratifieddKFold. Each training set given n_splits corresponding to its sample size and uniform n_repeats=5.

Logistic Regression **(6)** - similar linear decision boundary to classify data. Sigmoid function maps to binary values using the following formula: 1 / (1 + e^-value). Model fit using 80/20 train_test_split on respective data sets. Confusion matrix displays (true,false|positive,negative) values.

2) Methods

2.1) Data Pre-Processing

Imputing involves replacing null values with values of significance. Training2data and testdata contained null values. These were dealt with using 'mean' imputing. Each null value is replaced by the mean of all values for that feature. To optimize the accuracy of this method training2data was first separated into respective confidence and label values. Mean values calculated best represent the data. Testdata did not contain label or confidence values so did not need to be split. 'Mode' and 'median' imputing lowered accuracy for the task and were eliminated.

Normalization implemented using 'Standard Scaler' which creates uniform variance across features making them normally distributed.

Feature Selection involves reducing dimensionality from our model. Having more features than samples risks the curse of dimensionality (Bellman, 1957). PCA and chi2 included to reduce dimensionality. PCA is conducted after normalization as it is sensitive to non-uniform variance in features. In our model we specify that PCA must retain features such that the amount of variance explained is greater than 80%. Chi2 is conducted before normalization as it cannot take negative values. It takes as its parameters the number of features we wish to extract. 600 chosen as this value was similar to the no. extracted from PCA. The features returned have the highest Chi-square value which corresponds to their dependence on the function's correct prediction.

2.2) Classifiers

Perceptron - Activation is calculated for each sample by taking ∑(weight * feature value + bias) for each row. Bias set to 1, weights are initially random. Activation value compared with Perceptrons activation function to produce a prediction: >0: 1, <=0: 0, error is then calculated. Prediction then fed to the update rule: w(t+1)=w(t)+eta0*(e-p)*i where w=weight, eta0=learning rate, e=expectedlabel, p=predicted, i=input (Auer, Burgsteiner, Maass, 2008). Perceptrons update rule amends weights when predictions and actual labels do not match. Process is known as Stochastic gradient descent optimization;

weights converge to values that correspond to cost-function being minimized (Dasgupta, Kalai, Monteleoni, 2009). Hyperparameters epoch max_iter and learning rate were tuned using GridSearchCV.

3 Perceptron classifiers achieved optimal accuracy scores after 100 epochs. Implies 100 epochs suffices to achieve minimized cost function. All classifiers are set to 100 epochs. Learning rate corresponds to how much the weights are updated each epoch and is not uniform, results follow:
Case1=0.01, Case2=0.0001, Case3=0.01, Case4=0.001, Case5=0.001, Case6=0.0001, Case7=0.0001, Case8=0.05, where each case value maps to the index in the graph below. Accuracy scores calculated using RepeatedStratifiedKFold. Results tables for Perceptron and Logistic Regression can be found below. Detailing the algorithmic logic behind Logistic Regression is beyond the scope of this report. Confusion matrix scores help visualize LR performance on sunny or dark photos. CTD(87.5%+,82.9%-), CChi2(80.2%+,78.4% -), TTD(77.2%+,79%-), TChi2(70.1%+,73.2%-). Confident data had higher accuracy on sunny photos. Total data had higher accuracy on dark photos.

3) Results

Accuracy & std scores: case Perceptron

	Accuracy	Standard dev.
1.Total Training Data	75.176%	11.044%
2.Confident T D	83.014%	10.189%
3.Unconfident T D	75.758%	11.635%
4.Total PCA T D	74.097%	11.218%
5.Confident PCA T D	80.283%	10.623%
6.Unconfident PCA T D	73.059%	11.495%
7.Total chi2 T D	70.595%	11.490%
8.Confident chi2 T D	79.617%	10.058%

Accuracy & confusion matrix scores: case LR

	Accuracy	True pos	True neg	False pos	False neg
TTD	78.008%	250	320	74	85
CTD	84.974%	77	87	11	18
TChi2	72.199%	220	300	91	110
CChi2	79.275%	73	80	18	22

3.1) Outcomes and implications

High learning rates produce inaccurate gradient descent optimization. Optimal learning rate for Total and 66% confidence datasets reduced after feature selection, whereas 100% confidence data rose. Feature selection acquires most relevant data which should require less learning. Much of the total training data was 66% confidence and considered less accurate. Feature selection appears inauspicious. Accuracy decreased in every case. Curse of dimensionality suggests pre-feature-selection training data sets may be inaccurate; their models are overfitting. Confident training data had the highest Perceptron and LR accuracy but highest dimensionality issues. Although accuracy values differ between Perceptron and LR, predictions for class labels were identical on pre-feature-selection training data. Two best performing Perceptrons: 2, 5, had 2.731% accuracy difference on training. Test predictions for these two classifiers had 85.833% uniformity.

4) Conclusion

Unfamiliarity with python resulted in verbose code. Failed to implement KNN imputing and alternative feature selections such as ‘lasso’. Confident PCA Training Data Perceptron is chosen for test predictions as the classifier with highest accuracy and lowest std after dealing with dimensionality issues. Cross-analysis into which features were most pertinent following feature selection methods was not conducted but could provide insight.

Bibliography:

Auer, P., Burgsteiner, H. and Maass, W., 2008. A learning rule for very simple universal approximators consisting of a single layer of perceptrons. *Neural Networks*, [online] 21(5), pp.786-795. Available at: <https://www.sciencedirect.com/science/article/pii/S0893608007002730> [Accessed 17 May 2021].

Bellman, R., 1957. *Dynamic Programming*. Princeton, N.J.: Princeton Univ. Pr. [Accessed 15 May 2021].

Dasgupta, S., Tauman Kalai, A. and Monteleoni, C., 2009. Analysis of Perceptron-Based Active Learning. *Journal of Machine Learning Research*, [online] pp.281-299. Available at: <https://www.jmlr.org/papers/volume10/dasgupta09a/dasgupta09a.pdf> [Accessed 17 May 2021].

Reference List - Code:

https://machinelearningmastery.com/perceptron-algorithm-for-classification-in-python/

https://machinelearningmastery.com/logistic-regression-for-machine-learning/

https://datatofish.com/logistic-regression-python/

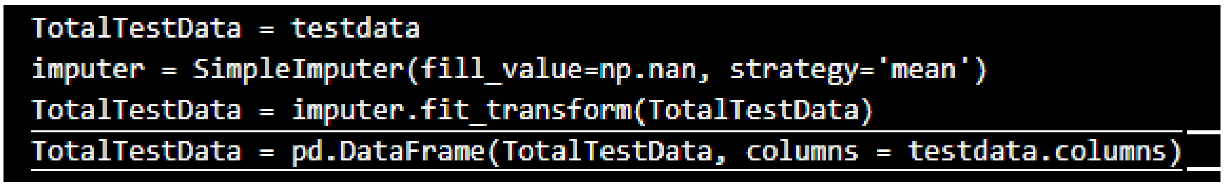
https://pandas.pydata.org/docs/reference/frame.html

https://scikit-learn.org/stable/index.html

Appendix:

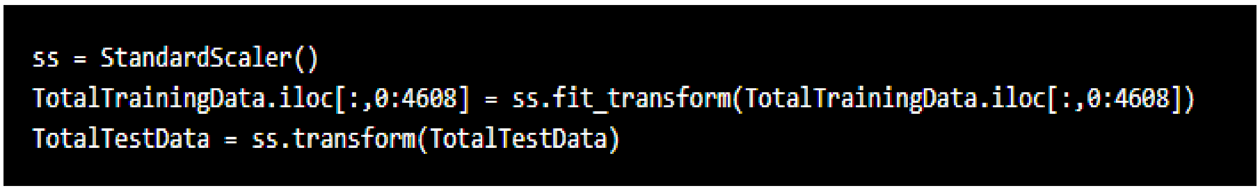
Code snippets that follow are based on their index mentioned in the above report.

1.'Mean' imputing example: test data



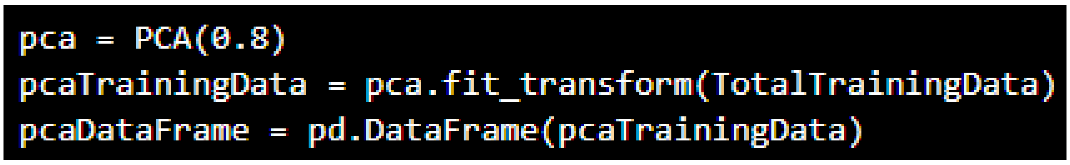
```
TotalTestData = testdata
imputer = SimpleImputer(fill_value=np.nan, strategy='mean')
TotalTestData = imputer.fit_transform(TotalTestData)
TotalTestData = pd.DataFrame(TotalTestData, columns = testdata.columns)
```

2.Normalization example: standard scaler



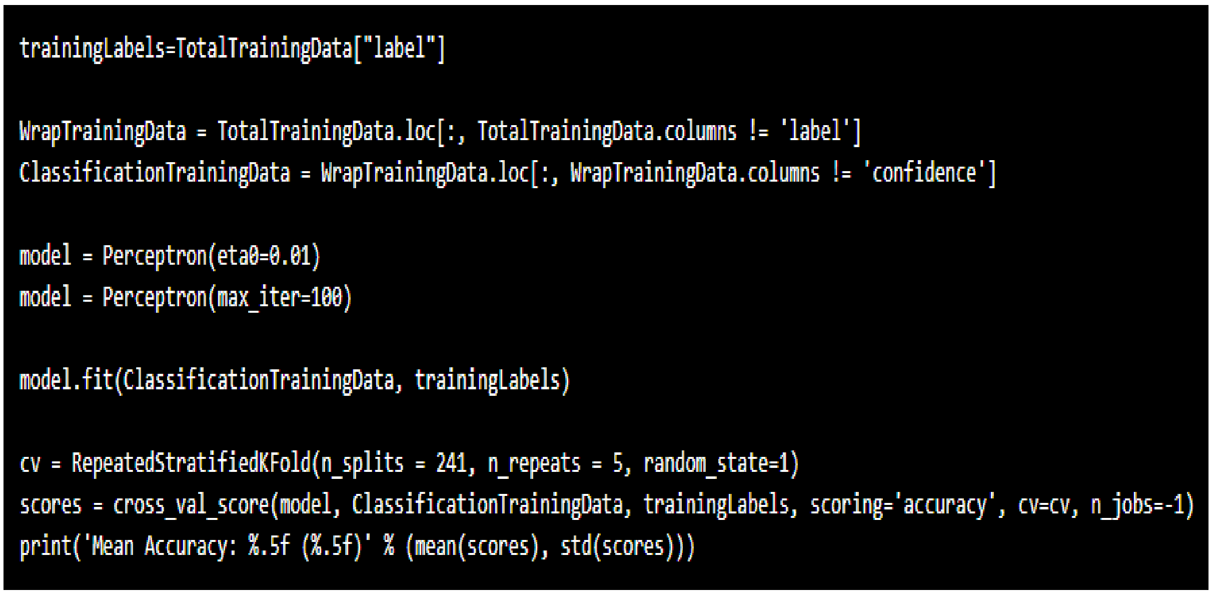
```
ss = StandardScaler()
TotalTrainingData.iloc[:,0:4608] = ss.fit_transform(TotalTrainingData.iloc[:,0:4608])
TotalTestData = ss.transform(TotalTestData)
```

3.Feature selection example: PCA



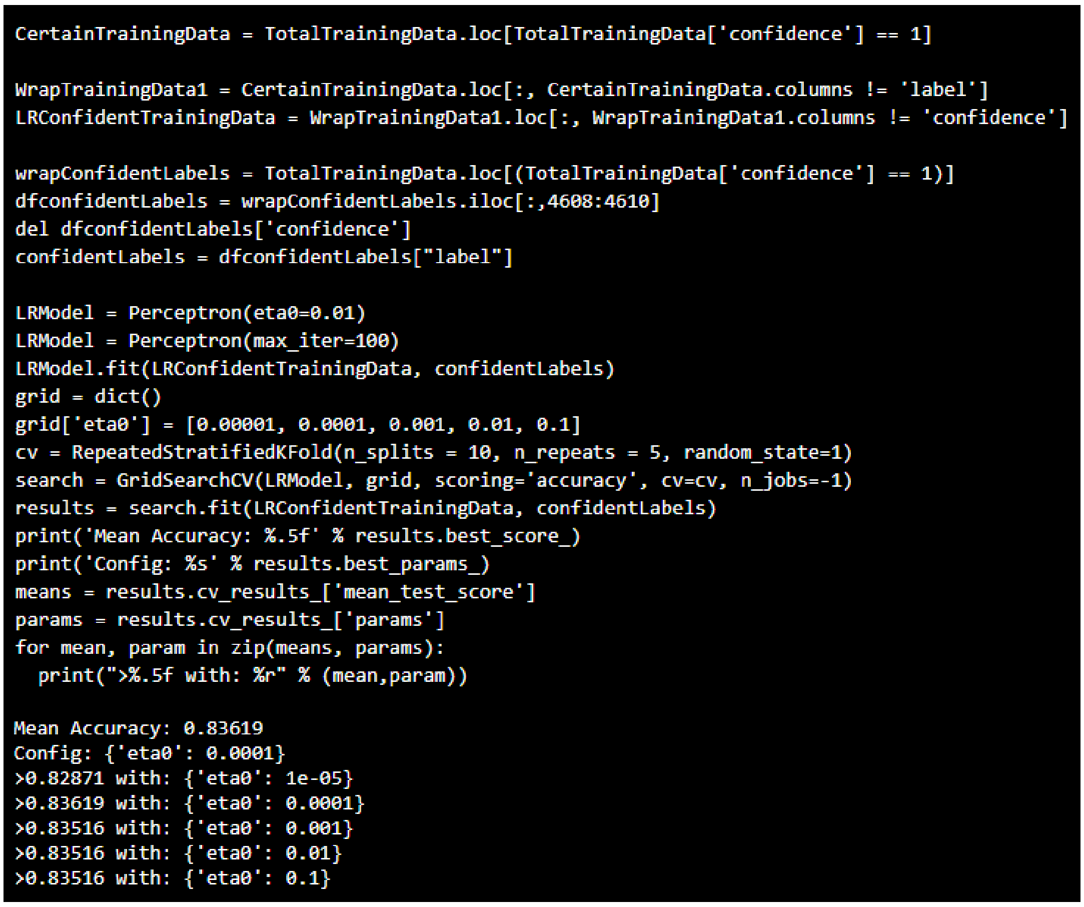
```
pca = PCA(0.8)
pcaTrainingData = pca.fit_transform(TotalTrainingData)
pcaDataFrame = pd.DataFrame(pcaTrainingData)
```

4.Perceptron example: Total Training Data



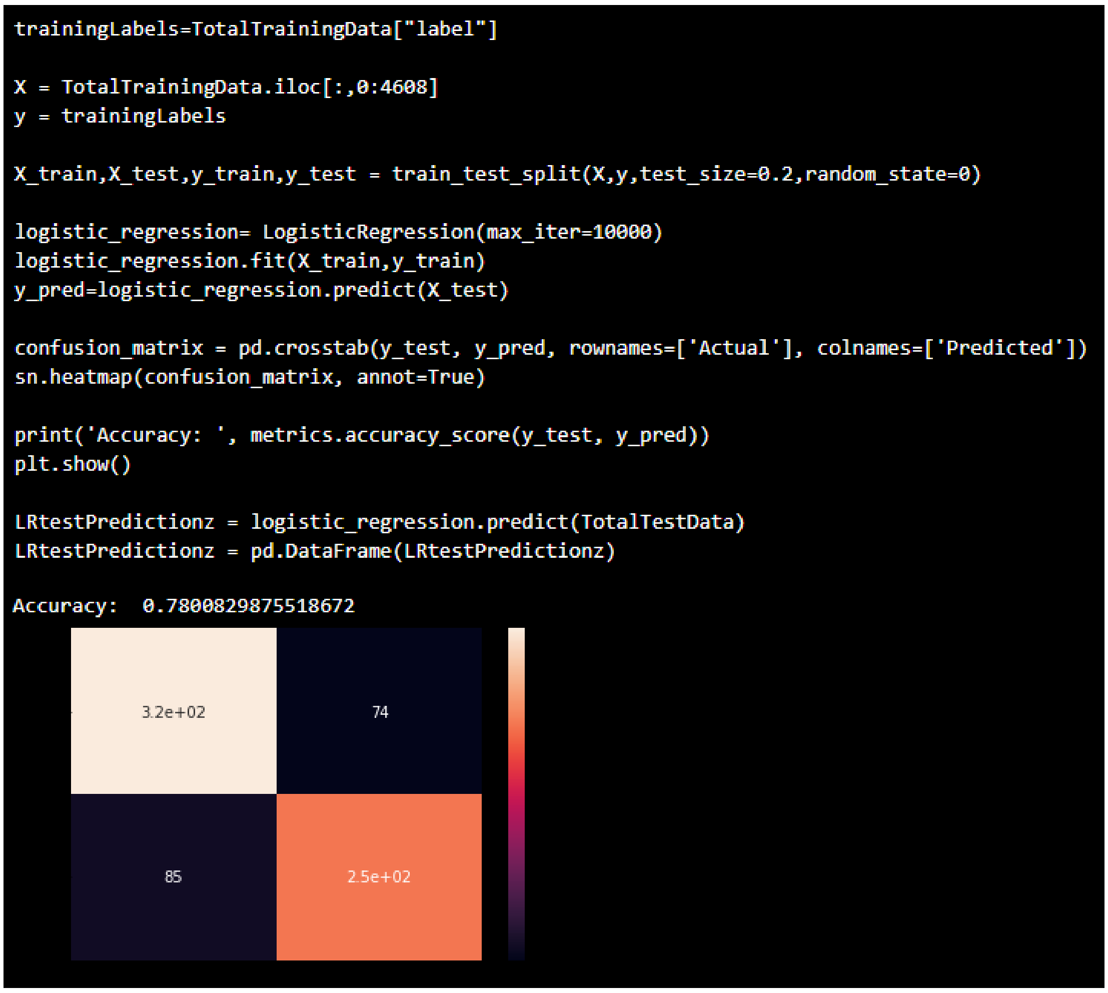
```
trainingLabels=TotalTrainingData["label"]

WrapTrainingData = TotalTrainingData.loc[:, TotalTrainingData.columns != 'label']
ClassificationTrainingData = WrapTrainingData.loc[:, WrapTrainingData.columns != 'confidence']

model = Perceptron(eta0=0.01)
model = Perceptron(max_iter=100)

model.fit(ClassificationTrainingData, trainingLabels)

cv = RepeatedStratifiedKFold(n_splits = 241, n_repeats = 5, random_state=1)
scores = cross_val_score(model, ClassificationTrainingData, trainingLabels, scoring='accuracy', cv=cv, n_jobs=-1)
print('Mean Accuracy: %.5f (%.5f)' % (mean(scores), std(scores)))
```

5.Tuning Hyperparameters case learning rate: GridCV Total Training Data



```
CertainTrainingData = TotalTrainingData.loc[TotalTrainingData['confidence'] == 1]

WrapTrainingData1 = CertainTrainingData.loc[:, CertainTrainingData.columns != 'label']
LRConfidentTrainingData = WrapTrainingData1.loc[:, WrapTrainingData1.columns != 'confidence']

wrapConfidentLabels = TotalTrainingData.loc[(TotalTrainingData['confidence'] == 1)]
dfconfidentLabels = wrapConfidentLabels.iloc[:,4608:4610]
del dfconfidentLabels['confidence']
confidentLabels = dfconfidentLabels["label"]

LRModel = Perceptron(eta0=0.01)
LRModel = Perceptron(max_iter=100)
LRModel.fit(LRConfidentTrainingData, confidentLabels)
grid = dict()
grid['eta0'] = [0.00001, 0.0001, 0.001, 0.01, 0.1]
cv = RepeatedStratifiedKFold(n_splits = 10, n_repeats = 5, random_state=1)
search = GridSearchCV(LRModel, grid, scoring='accuracy', cv=cv, n_jobs=-1)
results = search.fit(LRConfidentTrainingData, confidentLabels)
print('Mean Accuracy: %.5f' % results.best_score_)
print('Config: %s' % results.best_params_)
means = results.cv_results_['mean_test_score']
params = results.cv_results_['params']
for mean, param in zip(means, params):
  print(">%.5f with: %r" % (mean,param))

Mean Accuracy: 0.83619
Config: {'eta0': 0.0001}
>0.82871 with: {'eta0': 1e-05}
>0.83619 with: {'eta0': 0.0001}
>0.83516 with: {'eta0': 0.001}
>0.83516 with: {'eta0': 0.01}
>0.83516 with: {'eta0': 0.1}
```

6.Logistic Regression: Total Training Data



```
trainingLabels=TotalTrainingData["label"]

X = TotalTrainingData.iloc[:,0:4608]
y = trainingLabels

X_train,X_test,y_train,y_test = train_test_split(X,y,test_size=0.2,random_state=0)

logistic_regression= LogisticRegression(max_iter=10000)
logistic_regression.fit(X_train,y_train)
y_pred=logistic_regression.predict(X_test)

confusion_matrix = pd.crosstab(y_test, y_pred, rownames=['Actual'], colnames=['Predicted'])
sn.heatmap(confusion_matrix, annot=True)

print('Accuracy: ', metrics.accuracy_score(y_test, y_pred))
plt.show()

LRtestPredictionz = logistic_regression.predict(TotalTestData)
LRtestPredictionz = pd.DataFrame(LRtestPredictionz)

Accuracy:  0.7800829875518672
```