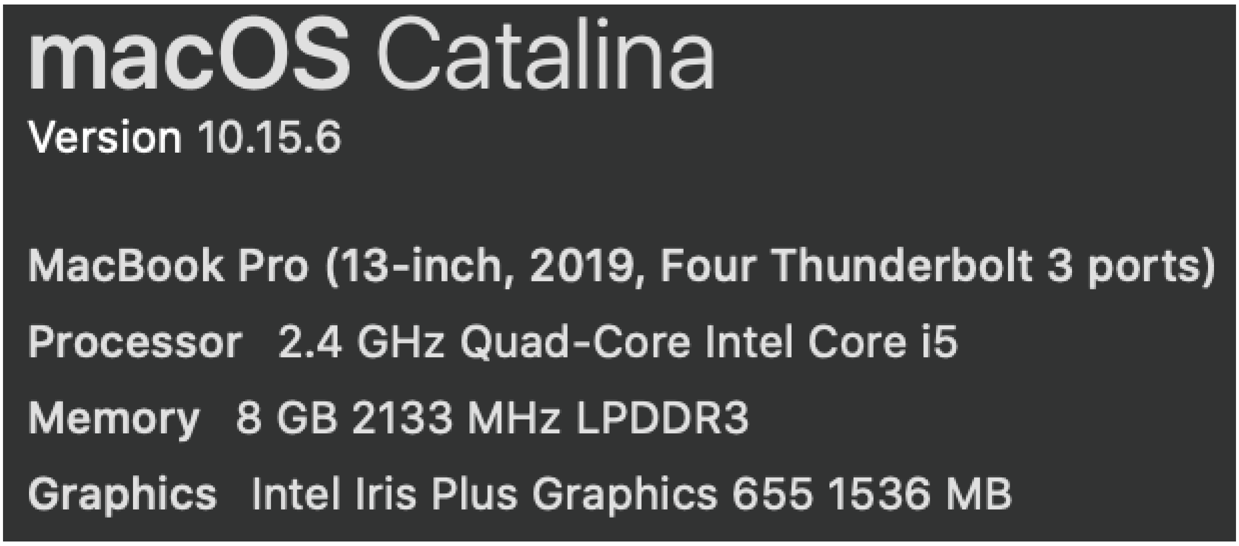
COMP5318 - Machine Learning and Data Mining: Assignment 1

- Aryan Bhatia : 490352056
- Mudit Malhotra : 490583269

HARDWARE AND SOFTWARE SPECIFICATIONS



Usual run time of this file on the above mentioned specifications is between **5:30 to 6 minutes**.

IMPORTING LIBRARIES AND INPUT DATA

In [64]:

```
import pandas as pd
import os
print(os.listdir("./Input/train"))
pd.set_option('display.max_columns', 10)

import pandas as pd
from sklearn.decomposition import PCA
from sklearn.preprocessing import MinMaxScaler
from sklearn.model_selection import train_test_split
from sklearn.neighbors import KNeighborsClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.naive_bayes import GaussianNB
from sklearn.metrics import accuracy_score
from sklearn.svm import SVC
from sklearn.model_selection import GridSearchCV
from sklearn.metrics import precision_recall_fscore_support
from sklearn.ensemble import BaggingClassifier

%matplotlib inline
```

```
['train.csv']
```

In [65]:

```
# train.csv including feature and label using for training model.
data_train_df = pd.read_csv('./Input/train/train.csv')
```

In [66]:

```
# Selecting input feature
data_train_feature = data_train_df.loc[:, "v1":"v784"].to_numpy()

# Selecting output lable
data_train_label = data_train_df.label.to_numpy()
```

In [67]:

```
# Train Test Split
X_train, X_test, y_train, y_test = train_test_split(
    data_train_feature, data_train_label, random_state=0, stratify=data_train_label)
```

In [68]:

```
# Performance Metrics Calculator Helper
def performance(y_true, y_pred, type):
    precision_test = precision_recall_fscore_support(y_true, y_pred, average='macro'
    print("Accuracy on " + type + " set: {:.3f}".format(accuracy_score(y_true, y_pre
    print("Precision on " + type + " set: {:.3f}".format(precision_test[0]))
    print("Recall on " + type + " set: {:.3f}".format(precision_test[1]))
    print("F-Score on " + type + " set: {:.3f}".format(precision_test[2]))
```

ACCURACIES BEFORE PRE-PROCESSING

We ran the following code before pre processing to test the accuracies of various classifiers before pre processing. This is not a requirement in the assignment spec but we ran it in order to draw comparisons in the report.

```python
%%time
# Default KNN before pre-processing
knn = KNeighborsClassifier()
knn.fit(X_train, y_train)
y_pred = knn.predict(X_test)
performance(y_test, y_pred, 'test')
```

Python

```
Accuracy on test set: 0.841
Precision on test set: 0.844
Recall on test set: 0.841
F-Score on test set: 0.841
CPU times: user 10.2 s, sys: 1.96 s, total: 12.1 s
Wall time: 4.79 s
```

```python
%%time
# Default LogReg before pre-processing
logreg = LogisticRegression(max_iter = 5000)
logreg.fit(X_train, y_train)
y_pred = logreg.predict(X_test)
performance(y_test, y_pred, 'test')
```

Python

```
Accuracy on test set: 0.796
Precision on test set: 0.793
Recall on test set: 0.796
F-Score on test set: 0.794
CPU times: user 21min 4s, sys: 1min 16s, total: 22min 20s
Wall time: 3min 47s
```

```python
%%time
# Default NB before pre-processing
nb = GaussianNB()
nb.fit(X_train, y_train)

y_pred = nb.predict(X_test)
y_pred_train = nb.predict(X_train)
performance(y_test, y_pred, 'test')
performance(y_train, y_pred_train, 'train')
```

✓ 2.9s MagicPython

```
Accuracy on test set: 0.601
Precision on test set: 0.652
Recall on test set: 0.600
F-Score on test set: 0.574
Accuracy on train set: 0.605
Precision on train set: 0.661
Recall on train set: 0.605
F-Score on train set: 0.576
CPU times: user 1.83 s, sys: 1.07 s, total: 2.91 s
Wall time: 2.88 s
```

```
%%time
# Default SVM before pre-processing
svm = SVC()
svm.fit(X_train, y_train)

y_pred = svm.predict(X_test)
y_pred_train = svm.predict(X_train)
performance(y_test, y_pred, 'test')
performance(y_train, y_pred_train, 'train')
```

✓ 4m 15.3s MagicPython

```
Accuracy on test set: 0.874
Precision on test set: 0.873
Recall on test set: 0.874
F-Score on test set: 0.873
Accuracy on train set: 0.904
Precision on train set: 0.904
Recall on train set: 0.904
F-Score on train set: 0.904
CPU times: user 4min 13s, sys: 750 ms, total: 4min 14s
Wall time: 4min 15s
```

DATA PRE-PROCESSING FOR TRAINING DATA

In [69]:

```
# Normalisation
scaler = MinMaxScaler()
scaler.fit(X_train)

X_train_norm = scaler.transform(X_train)
X_test_norm  = scaler.transform(X_test)

pd.DataFrame(X_train_norm)
```

Out[69]:

	0	1	2	3	4	...	779	780	781	782	783
0	0.0	0.0	0.0	0.0	0.0	...	0.000000	0.00	0.000000	0.0	0.0
1	0.0	0.0	0.0	0.0	0.0	...	0.462745	0.38	0.082353	0.0	0.0
2	0.0	0.0	0.0	0.0	0.0	...	0.000000	0.00	0.000000	0.0	0.0
3	0.0	0.0	0.0	0.0	0.0	...	0.000000	0.00	0.000000	0.0	0.0
4	0.0	0.0	0.0	0.0	0.0	...	0.000000	0.00	0.000000	0.0	0.0
...	...	...	...	...	...	...	...	...	...	...	...
22495	0.0	0.0	0.0	0.0	0.0	...	0.000000	0.00	0.000000	0.0	0.0
22496	0.0	0.0	0.0	0.0	0.0	...	0.000000	0.00	0.000000	0.0	0.0
22497	0.0	0.0	0.0	0.0	0.0	...	0.000000	0.00	0.000000	0.0	0.0
22498	0.0	0.0	0.0	0.0	0.0	...	0.000000	0.00	0.000000	0.0	0.0
22499	0.0	0.0	0.0	0.0	0.0	...	0.203922	0.00	0.000000	0.0	0.0

22500 rows × 784 columns

In [70]:

```
# Dimension Reduction
pca = PCA(n_components=0.9).fit(X_train_norm)

X_train_pca = pca.transform(X_train_norm)
X_test_pca = pca.transform(X_test_norm)

pd.DataFrame(X_train_pca)
```

Out[70]:

	0	1	2	3	4	...	79	80	81
0	2.927449	-5.553593	4.019518	-1.395135	1.064630	...	-0.011555	-0.025837	-0.095283
1	1.992421	0.442999	-2.368003	0.811313	-1.177249	...	-0.479448	0.015362	-0.228318
2	2.916436	-4.533643	2.285697	-2.685905	0.292165	...	-0.053102	0.513047	-0.128389
3	-6.189816	1.348044	-0.645078	-2.423576	1.313570	...	-0.153498	-0.184887	0.326095
4	-2.974141	-4.625361	1.107169	0.583185	0.232016	...	0.265475	0.093921	0.410109
...	...	...	...	...	...	...	...	...	...
22495	1.257244	5.409011	5.380687	3.621838	-2.909050	...	-0.069430	0.024253	-0.329398
22496	1.754955	-4.031853	0.866494	-0.421730	0.518514	...	-0.398757	-0.208873	0.206416
22497	-6.374198	0.561054	-0.780198	-2.838871	1.389735	...	0.245362	0.187905	-0.085312
22498	-5.730576	2.660773	-0.073577	-3.724108	1.928655	...	0.021028	0.012773	0.234765
22499	7.419908	2.280146	-1.712024	-0.641199	-0.133754	...	-0.154405	-0.063361	-0.290396

22500 rows × 84 columns

KNN

After Pre-Processing

In [71]:

```
%%time
# Accuracy of default KNN classifier after pre processing
knn = KNeighborsClassifier()
knn.fit(X_train_pca, y_train)

y_pred = knn.predict(X_test_pca)
y_pred_train = knn.predict(X_train_pca)
performance(y_test, y_pred, 'test')
performance(y_train, y_pred_train, 'train')
```

```
Accuracy on test set: 0.850
Precision on test set: 0.851
Recall on test set: 0.850
F-Score on test set: 0.850
Accuracy on train set: 0.895
Precision on train set: 0.895
Recall on train set: 0.895
F-Score on train set: 0.894
CPU times: user 16.6 s, sys: 6.73 s, total: 23.4 s
Wall time: 15.4 s
```

Parameter Tuning

```
%%time
# Parameter Tuning
param_grid = {'n_neighbors': [1, 3, 5, 11, 15], 'p': [1, 2]}

grid_search = GridSearchCV(KNeighborsClassifier(), param_grid, cv=5, return_train_score=True, n_jobs=-1)
grid_search.fit(X_train_pca, y_train)

print("Test set score: {:.2f}".format(grid_search.score(X_test_pca, y_test)))
print("Best parameters: {}".format(grid_search.best_params_))
print("Best cross-validation score: {:.2f}".format(grid_search.best_score_))
print("Best estimator:\n{}".format(grid_search.best_estimator_))
```

MagicPython

```
Test set score: 0.85
Best parameters: {'n_neighbors': 5, 'p': 1}
Best cross-validation score: 0.85
Best estimator:
KNeighborsClassifier(p=1)
CPU times: user 14.6 s, sys: 1.1 s, total: 15.7 s
Wall time: 6min 47s
```

In [72]:

```
%%time
# Create a KNN Classifier using best parameters
knn = KNeighborsClassifier(p=1)
knn.fit(X_train_pca, y_train)

y_pred = knn.predict(X_test_pca)
y_pred_train = knn.predict(X_train_pca)
performance(y_test, y_pred, 'test')
performance(y_train, y_pred_train, 'train')
```

```
Accuracy on test set: 0.851
Precision on test set: 0.852
Recall on test set: 0.851
F-Score on test set: 0.851
Accuracy on train set: 0.898
Precision on train set: 0.899
Recall on train set: 0.898
F-Score on train set: 0.898
CPU times: user 52.6 s, sys: 3.88 s, total: 56.5 s
Wall time: 56.9 s
```

LOGISTIC REGRESSION

After pre-processing

In [73]:

```
%%time
# Accuracy of default LogReg Classifier after pre processing
logreg = LogisticRegression(max_iter = 5000)
logreg.fit(X_train_pca, y_train)

y_pred = logreg.predict(X_test_pca)
y_pred_train = logreg.predict(X_train_pca)
performance(y_test, y_pred, 'test')
performance(y_train, y_pred_train, 'train')
```

```
Accuracy on test set: 0.843
Precision on test set: 0.842
Recall on test set: 0.843
F-Score on test set: 0.842
Accuracy on train set: 0.854
Precision on train set: 0.853
Recall on train set: 0.854
F-Score on train set: 0.853
CPU times: user 34.3 s, sys: 3.37 s, total: 37.7 s
Wall time: 9.59 s
```

Parameter Tuning

```
%%time
# Parameter Tuning
param_grid = {'C': [100, 10, 1.0, 0.1, 0.01], 'solver':  ['liblinear', 'lbfgs']}

grid_search = GridSearchCV(LogisticRegression(max_iter = 5000), param_grid, cv=5, return_train_score=True, n_jobs=-1)
grid_search.fit(X_train_pca, y_train)

print("Test set score: {:.2f}".format(grid_search.score(X_test_pca, y_test)))
print("Best parameters: {}".format(grid_search.best_params_))
print("Best cross-validation score: {:.2f}".format(grid_search.best_score_))
print("Best estimator:\n{}".format(grid_search.best_estimator_))
```

✓ 2m 53.5s Python

```
Test set score: 0.84
Best parameters: {'C': 1.0, 'solver': 'lbfgs'}
Best cross-validation score: 0.84
Best estimator:
LogisticRegression(max_iter=5000)
CPU times: user 50.2 s, sys: 7.5 s, total: 57.7 s
Wall time: 2min 53s
```

No need for creating the best classifier after parameter tuning because, as seen above, the default parameters are already the best ones for LogisticRegression.

NAIVE BAYES

After pre-processing

In [74]:

```
%%time
# Accuracy of default NB Classifier after pre processing
nb = GaussianNB()
nb.fit(X_train_pca, y_train)

y_pred = nb.predict(X_test_pca)
y_pred_train = nb.predict(X_train_pca)
performance(y_test, y_pred, 'test')
performance(y_train, y_pred_train, 'train')
```

```
Accuracy on test set: 0.771
Precision on test set: 0.778
Recall on test set: 0.771
F-Score on test set: 0.772
Accuracy on train set: 0.773
Precision on train set: 0.778
Recall on train set: 0.773
F-Score on train set: 0.773
CPU times: user 184 ms, sys: 44.6 ms, total: 228 ms
Wall time: 178 ms
```

Parameter Tuning

```
%%time
# Parameter Tuning
param_grid_nb = { 'var_smoothing': np.logspace(0,-9, num=100) }

nbModel_grid = GridSearchCV(estimator=GaussianNB(), param_grid=param_grid_nb, verbose=1, cv=10, n_jobs=-1)
nbModel_grid.fit(X_train_pca, y_train)

print("Test set score: ", nbModel_grid.score(X_test_pca, y_test))
print("Best parameters: {}".format(nbModel_grid.best_params_))
print("Best cross-validation score: ", nbModel_grid.best_score_)
```

MagicPython

```
Fitting 10 folds for each of 100 candidates, totalling 1000 fits
Test set score:  0.772
Best parameters: {'var_smoothing': 0.0002848035868435802}
Best cross-validation score:  0.7708888888888887
CPU times: user 3.86 s, sys: 769 ms, total: 4.63 s
Wall time: 21.4 s
```

In [75]:

```
%%time
# Create a NB Classifier using best parameters and check accuracy
nb = GaussianNB(var_smoothing=0.0002848035868435802)
nb.fit(X_train_pca, y_train)

y_pred = nb.predict(X_test_pca)
y_pred_train = nb.predict(X_train_pca)
performance(y_test, y_pred, 'test')
performance(y_train, y_pred_train, 'train')
```

```
Accuracy on test set: 0.772
Precision on test set: 0.780
Recall on test set: 0.772
F-Score on test set: 0.774
Accuracy on train set: 0.774
Precision on train set: 0.781
Recall on train set: 0.774
F-Score on train set: 0.775
CPU times: user 139 ms, sys: 32.4 ms, total: 171 ms
Wall time: 170 ms
```

SVM

After pre-processing

In [76]:

```
%%time
# Accuracy of default SVC classifier after pre-processing
svm = SVC()
svm.fit(X_train_pca, y_train)

y_pred = svm.predict(X_test_pca)
y_pred_train = svm.predict(X_train_pca)
performance(y_test, y_pred, 'test')
performance(y_train, y_pred_train, 'train')
```

```
Accuracy on test set: 0.878
Precision on test set: 0.877
Recall on test set: 0.878
F-Score on test set: 0.877
Accuracy on train set: 0.898
Precision on train set: 0.898
Recall on train set: 0.898
F-Score on train set: 0.898
CPU times: user 1min 9s, sys: 389 ms, total: 1min 9s
Wall time: 1min 10s
```

Parameter Tuning

```
%%time
# Parameter Tuning
param_grid = {'C': [100, 10, 1.0, 0.1, 0.01], 'gamma': [1, 0.1, 0.01, 0.001, 0.0001],
              'kernel':  ['linear', 'poly', 'rbf', 'sigmoid']}

grid_search = GridSearchCV(SVC(), param_grid, cv=5, return_train_score=True, n_jobs=-1)
grid_search.fit(X_train_pca, y_train)

print("Test set score: {:.2f}".format(grid_search.score(X_test_pca, y_test)))
print("Best parameters: {}".format(grid_search.best_params_))
print("Best cross-validation score: {:.2f}".format(grid_search.best_score_))
print("Best estimator:\n{}".format(grid_search.best_estimator_))
```

MagicPython

```
Test set score: 0.89
Best parameters: {'C': 10, 'gamma': 0.01, 'kernel': 'rbf'}
Best cross-validation score: 0.88
Best estimator:
SVC(C=10, gamma=0.01)
CPU times: user 18.2 s, sys: 418 ms, total: 18.7 s
Wall time: 2h 38min 50s
```

In [77]:

```
%%time
# Create a SVM Classifier using best parameters
svm = SVC(C=10, gamma=0.01)
svm.fit(X_train_pca, y_train)

y_pred = svm.predict(X_test_pca)
y_pred_train = svm.predict(X_train_pca)
performance(y_test, y_pred, 'test')
performance(y_train, y_pred_train, 'train')
```

```
Accuracy on test set: 0.887
Precision on test set: 0.886
Recall on test set: 0.887
F-Score on test set: 0.886
Accuracy on train set: 0.934
Precision on train set: 0.934
Recall on train set: 0.934
F-Score on train set: 0.933
CPU times: user 1min, sys: 397 ms, total: 1min
Wall time: 1min 1s
```

BAGGING (SVM)

```
%%time
# Parameter Tuning for Bagging Classifier
param_grid = {'base_estimator': [SVC(), SVC(C=10, gamma=0.01)], 'n_estimators': [2, 5, 10],
              'bootstrap': [True, False], 'max_samples': [0.2, 0.4, 0.6, 0.8, 1.0]}

grid_search = GridSearchCV(BaggingClassifier(), param_grid, cv=5, return_train_score=True, n_jobs=-1)
grid_search.fit(X_train_pca, y_train)

print("Test set score: {:.3f}".format(grid_search.score(X_test_pca, y_test)))
print("Best parameters: {}".format(grid_search.best_params_))
print("Best cross-validation score: {:.3f}".format(grid_search.best_score_))
print("Best estimator:\n{}".format(grid_search.best_estimator_))
```

✓ 159m 56.1s MagicPython

```
Test set score: 0.887
Best parameters: {'base_estimator': SVC(C=10, gamma=0.01), 'bootstrap': False, 'max_samples': 1.0, 'n_estimators': 2}
Best cross-validation score: 0.883
Best estimator:
BaggingClassifier(base_estimator=SVC(C=10, gamma=0.01), bootstrap=False,
                  n_estimators=2)
CPU times: user 43.2 s, sys: 663 ms, total: 43.9 s
Wall time: 2h 39min 56s
```

In [78]:

```
%%time
# Using bagging Ensemble on a set of SVCs with best parameters
bclf = BaggingClassifier(base_estimator=SVC(C=10, gamma=0.01),
    bootstrap=False, n_estimators=2).fit(X_train_pca, y_train)

y_pred = bclf.predict(X_test_pca)
y_pred_train = bclf.predict(X_train_pca)
performance(y_test, y_pred, 'test')
performance(y_train, y_pred_train, 'train')
```

```
Accuracy on test set: 0.887
Precision on test set: 0.886
Recall on test set: 0.887
F-Score on test set: 0.886
Accuracy on train set: 0.934
Precision on train set: 0.934
Recall on train set: 0.934
F-Score on train set: 0.933
CPU times: user 2min, sys: 891 ms, total: 2min
Wall time: 2min 2s
```

DATA PRE-PROCESSING FOR BLIND TESTING DATA

In [ ]:

```
# test_input.csv includes 5000 samples used for label prediction. Test samples do nc
data_test_df = pd.read_csv('./Input/test/test_input.csv', index_col=0)
```

In [ ]:

```
# Data Normalisation
output_test_norm = scaler.transform(data_test_df.to_numpy())

pd.DataFrame(output_test_norm)
```

	0	1	2	3	4	...	779	780	781	782	783
0	0.0	0.0	0.000000	0.000000	0.0	...	0.000000	0.000	0.000000	0.0	0.0
1	0.0	0.0	0.000000	0.000000	0.0	...	0.000000	0.000	0.000000	0.0	0.0
2	0.0	0.0	0.000000	0.000000	0.0	...	0.000000	0.000	0.000000	0.0	0.0
3	0.0	0.0	0.000000	0.000000	0.0	...	0.000000	0.000	0.000000	0.0	0.0
4	0.0	0.0	0.000000	0.000000	0.0	...	0.000000	0.000	0.000000	0.0	0.0
...	...	...	...	...	...	...	...	...	...	...	...
4995	0.0	0.0	0.000000	0.000000	0.0	...	0.000000	0.000	0.000000	0.0	0.0
4996	0.0	0.0	0.000000	0.000000	0.0	...	0.000000	0.000	0.000000	0.0	0.0
4997	0.0	0.0	0.000000	0.000000	0.0	...	0.000000	0.000	0.000000	0.0	0.0
4998	0.0	0.0	0.000000	0.000000	0.0	...	0.000000	0.000	0.000000	0.0	0.0
4999	0.0	0.0	0.038462	0.006849	0.0	...	0.596078	0.536	0.172549	0.0	0.0

5000 rows × 784 columns

In [ ]:

```
# Dimension Reduction
output_test_pca = pca.transform(output_test_norm)

pd.DataFrame(output_test_pca)
```

	0	1	2	3	4	...	79	80	81
0	2.030060	-5.433903	1.980836	-1.898502	-1.143763	...	0.163122	0.084214	0.142833
1	1.971218	-5.661186	1.332592	-1.375247	-0.702272	...	-0.407891	0.289796	-0.219731
2	0.527241	-5.844876	1.605396	-1.962709	-2.800743	...	-0.056665	0.077233	0.170281
3	-0.334794	-3.352547	-0.780979	1.740940	0.259736	...	-0.145004	-0.078867	-0.272437
4	-1.495075	-2.585312	-0.883327	0.401559	0.713086	...	0.370981	0.381618	-0.013033
...	...	...	...	...	...	...	...	...	...
4995	2.789104	-3.193621	0.806616	0.593713	-0.233942	...	0.041395	-0.117392	-0.476785
4996	2.489568	5.947833	-0.281650	1.209622	3.791362	...	-0.535142	0.092127	0.591728
4997	-5.653882	-0.605367	-1.218067	3.388302	-0.931545	...	-0.497402	-0.183685	-0.146962
4998	7.083845	1.511506	0.441606	1.107137	0.528356	...	-0.400921	0.021544	-0.081451
4999	8.087741	4.290206	-0.369143	0.989579	-0.053555	...	0.064191	-0.306762	0.019435

5000 rows × 84 columns

EXPORTING OUTPUT

In [ ]:

```
# Helper function to export csv file storing predictions of a classifier on the blir
def export_predictions(filename, classifier):
    predictions = []
    filepath = './Output/' + filename + '.csv'

    for i in output_test_pca:
        prediction = classifier.predict([list(i)])
        predictions.append(prediction[0])

    output_df = pd.DataFrame(predictions, columns = ['label'])
    output_df.to_csv(filepath, sep=",", float_format='%d', index_label="id")
```

Example Usage: `export_predictions('test_output', knn)` . This will create a file "test_output.csv" in the Output folder which will store the predictions of the KNN classifier for the blind testing data.

Classifier Options for the `classifier` argument:

- knn
- logreg (LogisticRegression)
- nb (Naive Bayes)
- svm
- bclf (BaggingClassifier on SVM)

In [ ]:

```
export_predictions('test_output', knn)
```

PERFORMANCE COMPARISON OF CLASSIFIERS