RAPPORT DE KEVIN NGUETCHE POUR LE MINI-PROJET

A. Résumé du travail réalisé

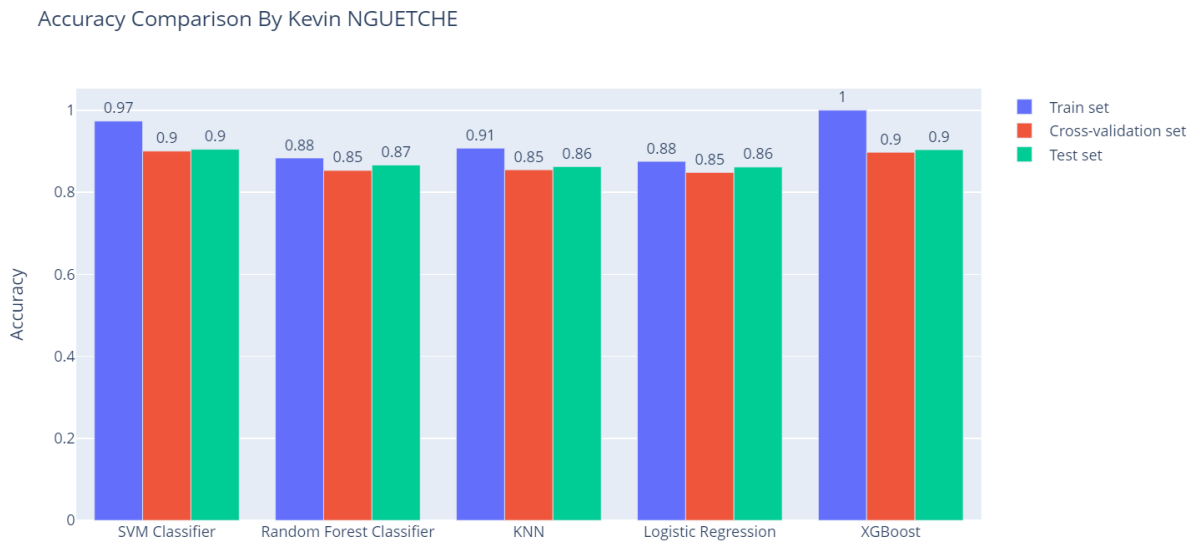
A l'aide de GridSearchCV, nous avons sélectionné les meilleurs paramètres de 5 différents modèles de Machine Learning sur le jeu de donnée Fashion MNIST.

Trois évaluations différentes ont été effectuées pour chaque modèle à savoir : une **évaluation sur les données d'entrainement**, une **cross validation à 5 blocs** et enfin une **évaluation sur les données de tests**.

Le modèle ayant fourni de meilleures performances sur les données de test est **SVM** avec **0.9 046** d'accuracy, suivi respectivement de **XGBOOST** avec **0.9037**, **Random Forest** avec **0.8899** , **KNN** avec **0.8625** et enfin **Regression logistic** avec **0.8614**.

Deux opérations de prétraitement ont été effectuées sur Fashion MNIST à savoir : **le mélange des données** (afin que toutes les classes soient représentées dans chaque portion du jeu de données) et **la normalisation Z-score** (afin de représenter les pixels sur un intervalle plus réduit que [0-255]).

B. Présentation des résultats



C. Résultats détaillés

Modèles	Accuracy Train set	Accuracy Cross-val	Accuracy Test set
SVM	0.9734	0.9003	0.9046
Random Forest	1.0000	0.8799	0.8899
KNN	0.9074	0.8547	0.8625
Logistic Regression	0.8752	0.8482	0.8614
XGBoost	1.0000	0.8974	0.9037

Dans les pages suivantes, nous présenterons plus en détail les étapes suivis.

I. JEU DE DONNEE FASHION MNIST

L'ensemble de données Fashion MNIST se compose de 70 000 images et chaque image a 784 caractéristiques (c'est-à-dire 28 × 28 pixels). Chaque pixel est une valeur de 0 à 255, décrivant l'intensité du pixel. 0 pour le blanc et 255 pour le noir.

Notre jeu de donnée a 10 classes d'étiquettes, numérotés de 0 à 9. Nous sommes donc en face d'un problème de classification multi-classes.

Suivant la consigne du projet, nous avons divisé notre jeu de données en 6/7 pour l'apprentissage soit 60 000 images, et 1/7 pour les tests soit 10 000 images.

II. PRETRAITEMENT

Les opérations de prétraitement effectuées sont :

- Le Mélange des données : afin que toutes les classes soient représentées dans chaque portion du jeu de données ; cette opération améliore les performances de la validation.

```
# Prétraitement
```

```
# Permutation du jeu de donnée
import numpy as np
np.random.seed(42)   # if you want reproducible results set the random seed value.
shuffle_index = np.random.permutation(60000)
x_train, y_train = x_train[shuffle_index], y_train[shuffle_index]
```

- La normalisation Z-score : afin de représenter les pixels sur un intervalle plus réduit que [0-255] (la plage [0-255] est très grande). Cette normalisation est donnée par la formule : ***x_normalisé = (x – moyenne) / écart-type***.

```
# Normalisation Z-score du Jeu de donnnée
from sklearn.preprocessing import StandardScaler

scaler = StandardScaler()
x_train = scaler.fit_transform(x_train.astype(np.float64))
x_test = scaler.fit_transform(x_test.astype(np.float64))
```

III. SELECTION DES MEILLEURS PARAMETRES DE CHAQUE MODELE

Afin d'effectuer une comparaison équitable de nos différents modèles, nous les avons comparés en utilisant leurs meilleurs paramètres associés. GridSearchCV a été utilisé.

- Meilleurs paramètres de SVM sur Fashion MNIST

```
# Trouvons les meilleurs hyper-paramètres pour nos différents models sur nos données
```

```
# SVM

classifier_svm = svm.SVC()
print("Listes des paramètres de l'SVM",classifier_svm.get_params(),"\n")

parameters = [{'C': [1, 10, 100, 1000], 'kernel': ['linear', 'rbf'], 'gamma': [0.001, 0.0001], 'max_iter':[1000, 10000],
               'decision_function_shape' : ['ovo', 'ovr']}]

grid_search = GridSearchCV(estimator=classifier_svm, param_grid=parameters, scoring='accuracy', cv=5)

grid_search.fit(x_train, y_train)

best_parameters = grid_search.best_params_
print("Meilleurs paramètres trouvés sont : %s" % best_parameters)
```

```
Meilleurs paramètres trouvés sont : {'C': 10, 'decision_function_shape': 'ovo', 'gamma': 0.001, 'kernel': 'rbf', 'max_iter':
10000}
```

- Meilleurs paramètres de Random Forest sur Fashion MNIST

```
# Random Forest

classifier_rf = RandomForestClassifier()

parameters = [{'random_state': [0, 10, 30, 42, 100], 'max_depth': [10, 50, 100], 'n_estimators': [10, 100, 1000]}]

grid_search = GridSearchCV(estimator=classifier_rf, param_grid=parameters, scoring='accuracy', cv=5)

grid_search.fit(x_train, y_train)

best_parameters = grid_search.best_params_s
print("Meilleurs paramètres de Random forest trouvés sont : %s" % best_parameters)
```

```
Meilleurs paramètres de Random forest trouvés sont : {'max_depth': 50, 'n_estimators': 100, 'random_state': 0}
```

- Meilleurs paramètres de KNN sur Fashion MNIST

```
# KNN

classifier_knn = KNeighborsClassifier()

parameters = [{'n_neighbors': [3, 4, 5, 8, 12]}]

grid_search = GridSearchCV(estimator=classifier_knn, param_grid=parameters, scoring='accuracy', cv=5)

grid_search.fit(x_train, y_train)

best_parameters = grid_search.best_params_
print("Meilleurs paramètres du KNN trouvés sont : %s" % best_parameters)
```

```
Meilleurs paramètres du KNN trouvés sont : {'n_neighbors': 4}
```

- Meilleurs paramètres de Logistic Regression sur Fashion MNIST

```
# Logistic Regression

classifier_lr = LogisticRegression()

parameters = [{'random_state': [0,42], 'C': [1, 10, 100, 1000], 'multi_class' : ['multinomial'],
               'solver' : ['lbfgs', 'liblinear'] }]

grid_search = GridSearchCV(estimator=classifier_lr, param_grid=parameters, scoring='accuracy', cv=5)

grid_search.fit(x_train, y_train)

best_parameters = grid_search.best_params_
print("Meilleurs paramètres de la regression logistique trouvés sont : %s" % best_parameters)

```

```
Meilleurs paramètres de la regression logistique trouvés sont : {'C': 100, 'multi_class': 'multinomial', 'random_state': 0,
'solver': 'lbfgs'}
```

- Meilleurs paramètres de XGBOOST sur Fashion MNIST

```
# XGBOOST
classifier_xgb = XGBClassifier()

parameters = [{'random_state': [0, 42], 'max_depth': [10, 50, 100], 'n_estimators': [10, 100, 1000]}]

grid_search = GridSearchCV(estimator=classifier_xgb, param_grid=parameters, scoring='accuracy', cv=5)

grid_search.fit(x_train, y_train)

best_parameters = grid_search.best_params_
print("Meilleurs paramètres de XGBOOST trouvés sont : %s" % best_parameters)

```

IV. Evaluation des 5 modèles

Maintenant que les meilleurs paramètres de chaque modèle ont été sélectionnés, évaluons ces derniers.

- Evaluation de SVM

```
# SVM, Evaluation
classifier_svm = svm.SVC(C = 10.0, decision_function_shape = 'ovo', gamma = 0.001, kernel = 'rbf', max_iter = 10000)
classifier_svm.fit(x_train, y_train)
y_train_prd = classifier_svm.predict(x_train)

scores = cross_val_score(classifier_svm, x_train, y_train, cv=5)

y_test_prd = classifier_svm.predict(x_test)
acc_train_svm = accuracy_score(y_train,y_train_prd)
acc_val_svm = scores.mean()
acc_test_svm = accuracy_score(y_test,y_test_prd)

print("accuracy on train set:{:.4f}\naccuracy on cross validation set:{:.4f}\naccuracy on test set :{:.4f}".format(
acc_train_svm, acc_val_svm, acc_test_svm))
```

```
accuracy on train set:0.9734
accuracy on cross validation set:0.9003
accuracy on test set :0.9046
```

– Evaluation de Random Forest

```
# Random Forest, Evaluation
classifier_rf = RandomForestClassifier(random_state=0, n_estimators=100, max_depth=50)
classifier_rf.fit(x_train, y_train)
y_train_prd = classifier_rf.predict(x_train)

scores = cross_val_score(classifier_rf, x_train, y_train, cv=5)

y_test_prd = classifier_rf.predict(x_test)
acc_train_rf = accuracy_score(y_train,y_train_prd)
acc_val_rf = scores.mean()
acc_test_rf = accuracy_score(y_test,y_test_prd)

print("accuracy on train set:{:.4f}\naccuracy on cross validation set:{:.4f}\naccuracy on test set :{:.4f}".format(
acc_train_rf, acc_val_rf, acc_test_rf))
```

```
accuracy on train set:1.0000
accuracy on cross validation set:0.8799
accuracy on test set :0.8899
```

– Evaluation de KNN

```
# KNN, Evaluation
classifier_knn = KNeighborsClassifier(n_neighbors=4)
classifier_knn.fit(x_train, y_train)
y_train_prd = classifier_knn.predict(x_train)

scores = cross_val_score(classifier_knn, x_train, y_train, cv=5)

y_test_prd = classifier_knn.predict(x_test)
acc_train_knn = accuracy_score(y_train,y_train_prd)
acc_val_knn = scores.mean()
acc_test_knn = accuracy_score(y_test,y_test_prd)

print("accuracy on train set:{:.4f}\naccuracy on cross validation set:{:.4f}\naccuracy on test set :{:.4f}".format(
acc_train_knn, acc_val_knn, acc_test_knn))
```

```
accuracy on train set:0.9074
accuracy on cross validation set:0.8547
accuracy on test set :0.8625
```

– Evaluation de Logistic Regression

```
# Logistic Regression, Evaluation
classifier_lr = LogisticRegression(multi_class="multinomial", solver="lbfgs", C=100, random_state=0)
classifier_lr.fit(x_train, y_train)
y_train_prd = classifier_lr.predict(x_train)

scores = cross_val_score(classifier_lr, x_train, y_train, cv=5)

y_test_prd = classifier_lr.predict(x_test)
acc_train_lr = accuracy_score(y_train,y_train_prd)
acc_val_lr = scores.mean()
acc_test_lr = accuracy_score(y_test,y_test_prd)

print("accuracy on train set:{:.4f}\naccuracy on cross validation set:{:.4f}\naccuracy on test set :{:.4f}".format(
acc_train_lr, acc_val_lr, acc_test_lr))
```

```
accuracy on train set:0.8752
accuracy on cross validation set:0.8482
accuracy on test set :0.8614
```

- Evaluation de XGBOOST

```
# XGBOOST, Evaluation
classifier_xgb = XGBClassifier(random_state=0, n_estimators=100, max_depth=50)
classifier_xgb.fit(x_train, y_train)
y_train_prd = classifier_xgb.predict(x_train)

scores = cross_val_score(classifier_xgb, x_train, y_train, cv=5)

y_test_prd = classifier_xgb.predict(x_test)
acc_train_xgb = accuracy_score(y_train,y_train_prd)
acc_val_xgb = scores.mean()
acc_test_xgb = accuracy_score(y_test,y_test_prd)

print("accuracy on train set:{:.4f}\naccuracy on cross validation set:{:.4f}\naccuracy on test set :{:.4f}".format(
acc_train_xgb, acc_val_xgb, acc_test_xgb))
```

```
accuracy on train set:1.0000
accuracy on cross validation set:0.8974
accuracy on test set :0.9037
```

V. COMPARAISON DES MODELES

Visualisons à présent nos modèles en fonction de leur évaluation sur les données d'entrainements, les données avec Cross validation à 5 blocs et les données de Test.

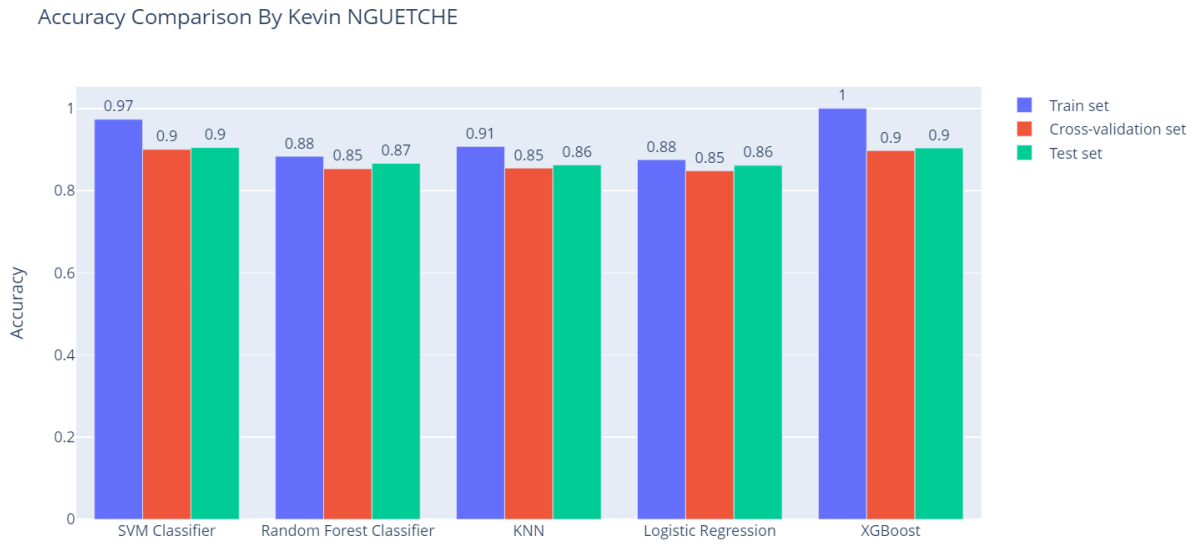
```
# Comparaison de nos modèles
acc_combine = {'Model':  ['SVM Classifier', 'Random Forest Classifier', 'KNN', 'Logistic Regression','XGBoost'],
        'Accuracy_Tra': [acc_train_svm, acc_train_rf, acc_train_knn, acc_train_lr,acc_train_xgb],
        'Accuracy_Val': [acc_val_svm, acc_val_rf, acc_val_knn, acc_val_lr,acc_val_xgb],
        'Accuracy_Test': [acc_test_svm, acc_test_rf, acc_test_knn, acc_test_lr,acc_test_xgb]
        }


```

```
# Schéma de comparaison

fig = go.Figure(data=[
    go.Bar(name='Train set', x=acc_combine['Model'], y=acc_combine['Accuracy_Tra'],text=np.round(acc_combine['Accuracy_Tra']
    go.Bar(name='Cross-validation set', x=acc_combine['Model'], y=acc_combine['Accuracy_Val'],text=np.round(acc_combine['Acc
    go.Bar(name='Test set', x=acc_combine['Model'], y=acc_combine['Accuracy_Test'],text=np.round(acc_combine['Accuracy_Test'
])

fig.update_layout(barmode='group',title_text='Accuracy Comparison By Kevin NGUETCHE',yaxis=dict(
        title='Accuracy'))
fig.show()
```



On peut constater à travers la figure que la performance maximale sur les données Fashion MNIST est de **90%**. Elle est atteinte en utilisant soit l'approche One vs One de **SVM** ; soit en fixant le nombre d'estimateurs de **XGBOOST** à 100 pour un random_state à 0.