Práctica 7
Máquinas de soporte vectorial

Mario Emilio Jiménez Vizcaíno
A01173359@itesm.mx
Tecnológico de Monterrey
Ingeniería en Tecnologías Computacionales
Monterrey, N.L., México

Jesus Abraham Haros Madrid
A01252642@itesm.mx
Tecnológico de Monterrey
Ingeniería en Tecnologías Computacionales
Monterrey, N.L., México

ABSTRACT

TODO

1 INTRODUCCIÓN

TODO

2 CONCEPTOS PREVIOS

- Programación básica en Python
- Conocimiento de las librerías *scikit-learn*, *matplotlib* y *numpy*
- Conocimientos básicos de estadística

3 METODOLOGÍA

Ésta práctica, al igual que la práctica pasada, implementar los modelos fue una tarea fácil, ya que la metodología para llevarla a cabo fue muy clara en la descripción de la actividad. Es por eso que nos guiamos de los pasos descritos en el documento de la práctica 7 para poder implementar los scripts de Python.

Para demostrar la eficacia y los resultados de las máquinas de soporte vectorial las comparamos contra tres modelos más:

- Un modelo de regresión logística
- Un modelo de k-vecinos más cercanos: durante nuestras pruebas encontramos el valor óptimo para k como 1, con una máxima precisión de 0.9889. El código de nuestra prueba puede ser encontrado en el apéndice B
- Un modelo de Bayes ingenuo

Para poder hacer el script fácil de usar con los diferentes modelos de clasificación se optó por leer como argumento de ejecución de programa el modelo con el que se quiere trabajar.

3.1 Dataset Digits

TODO

3.2 Máquinas de soporte vectorial

TODO

El código que ejecutamos para realizar el análisis del dataset se encuentra en el apéndice A.

4 RESULTADOS

TODO

4.1 Precisión de los modelos

Modelo	Precisión
MSV con kernel lineal	0.9777
MSV con kernel polinomial	0.9889
MSV con kernel RBF	0.9917
MSV con kernel sigmoide	0.9139
Regresión logística	0.9472
k-Vecinos más cercanos	0.9889
Bayes ingenuo	0.8444

4.2 Matriz de confusión de los modelos

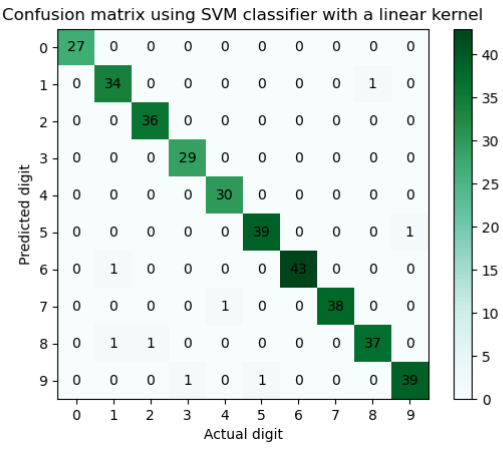

Figure 1: Matriz de confusión del modelo de máquina de soporte vectorial usando un kernel lineal

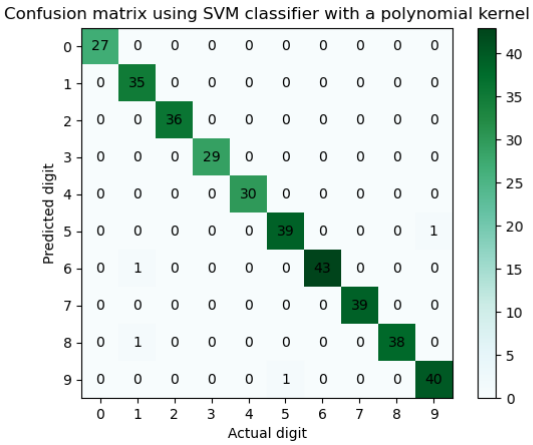

Figure 2: Matriz de confusión del modelo de máquina de soporte vectorial usando un kernel polinomial

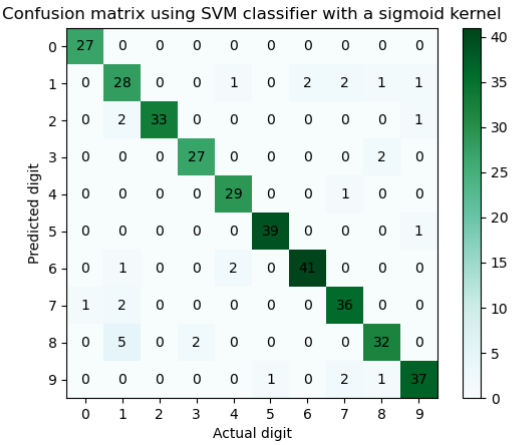

Figure 4: Matriz de confusión del modelo de máquina de soporte vectorial usando un kernel sigmoide

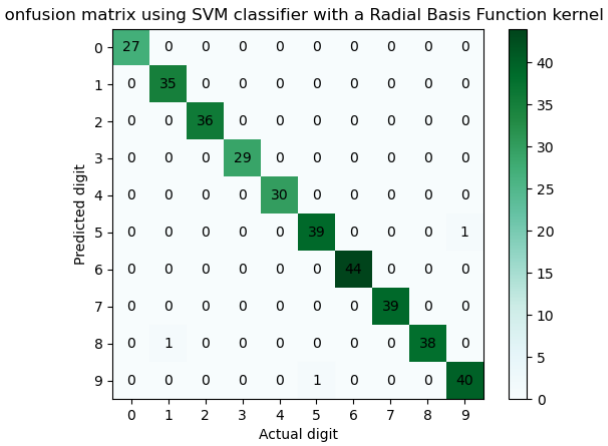

Figure 3: Matriz de confusión del modelo de máquina de soporte vectorial usando un kernel RBF

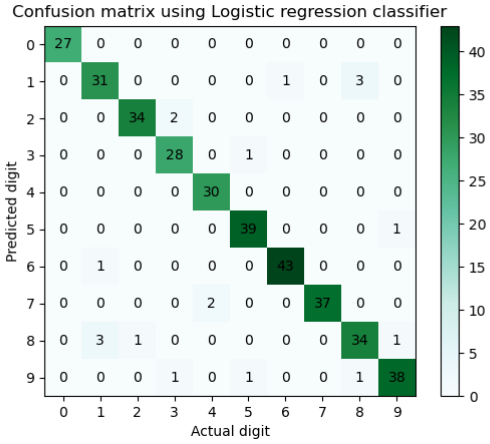

Figure 5: Matriz de confusión del modelo de regresión logística

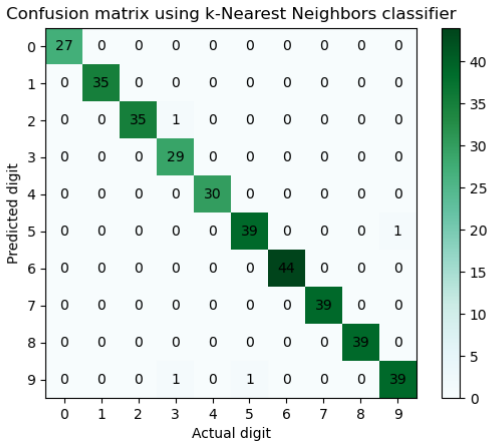

Figure 6: Matriz de confusión del modelo de k vecinos más cercanos

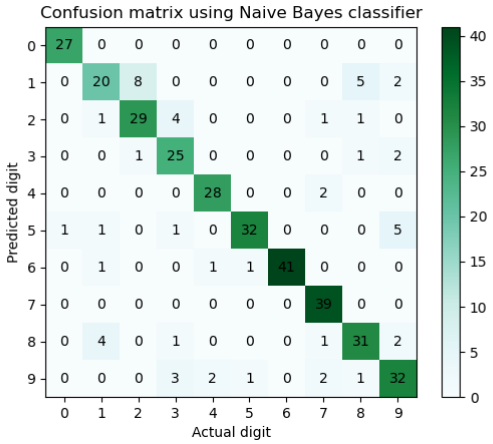

Figure 7: Matriz de confusión del modelo bayesiano ingenuo

4.3 Espacio ROC de los modelos

TODO

5 CONCLUSIONES Y REFLEXIONES

TODO

5.1 Refrexión de Abraham

TODO

5.2 Reflexión de Mario

TODO

REFERENCES

A CÓDIGO PARA LA COMPARACIÓN DE MODELOS DE CLASIFICACIÓN

```
import sys
import matplotlib.pyplot as plt
import numpy as np
from sklearn.datasets import load_digits
from sklearn.metrics import accuracy_score, confusion_matrix
from sklearn.model_selection import train_test_split
from sklearn.naive_bayes import BernoulliNB
from sklearn.neighbors import KNeighborsClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.svm import SVC

# Choose a classifier using the first parameter
possible_clfs = ["linear", "poly", "rbf", "sigmoid",
                 "logistic", "knn", "bayes"]
if len(sys.argv) < 2 or sys.argv[1] not in possible_clfs:
    print("Include an argument from the list:", possible_clfs)
    exit(1)

classifier_type = sys.argv[1]
if classifier_type in ["linear", "poly", "rbf", "sigmoid"]:
    clf = SVC(kernel=classifier_type)
    classifier_name = "SVM classifier with a " + {
        "linear": "linear kernel",
        "poly": "polynomial kernel",
        "rbf": "Radial Basis Function kernel",
        "sigmoid": "sigmoid kernel",
    }[classifier_type]
elif classifier_type == "logistic":
    clf = LogisticRegression(multi_class="ovr", max_iter=1000)
    classifier_name = "Logistic regression classifier"
elif classifier_type == "knn":
    clf = KNeighborsClassifier(n_neighbors=1)
    classifier_name = "k-Nearest Neighbors classifier"
elif classifier_type == "bayes":
    clf = BernoulliNB()
    classifier_name = "Naive Bayes classifier"

# Load the dataset and split it
digitsX, digitsy = load_digits(return_X_y=True)
trainX, testX, trainy, testy = train_test_split(
    digitsX, digitsy, test_size=0.2, random_state=0)

# Fit the model and predict the labels
clf.fit(trainX, trainy)
predicty = clf.predict(testX)

# Print results
print(classifier_name)
print("Accuracy:", accuracy_score(testy, predicty))
cm = confusion_matrix(testy, predicty)
print("Confusion matrix:")
print(cm)
```

```
# Plot the confusion matrix
plt.figure()
plt.imshow(cm, interpolation="nearest", cmap="BuGn")
plt.title("Confusion matrix using " + classifier_name)
plt.colorbar()
plt.xticks(np.arange(10), np.arange(10).astype(str), size=10)
plt.yticks(np.arange(10), np.arange(10).astype(str), size=10)
plt.xlabel("Actual digit")
plt.ylabel("Predicted digit")
for x in range(10):
    for y in range(10):
        plt.annotate(cm[x][y], xy=(y, x),
                     horizontalalignment="center",
                     verticalalignment="center")
plt.savefig(classifier_type + "_cm.png")
```

B CÓDIGO PARA LA COMPARACIÓN DE K PARA EL MODELO DE K-VECINOS MÁS CERCANOS

```
from sklearn.datasets import load_digits
from sklearn.model_selection import train_test_split
from sklearn.neighbors import KNeighborsClassifier

digitsX, digitsy = load_digits(return_X_y=True)
trainX, testX, trainy, testy = train_test_split(
    digitsX, digitsy, test_size=0.2, random_state=0)

for neighbors in [1, 2, 3, 4, 5, 7, 10, 15, 20, 25, 30, 40, 50]:
    clf = KNeighborsClassifier(n_neighbors=neighbors)
    clf.fit(trainX, trainy)
    accuracy = clf.score(testX, testy)
    print("Accuracy using {:2} neighbors: {:.4f}".format(neighbors, accuracy))
```