

# 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

Esta 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 C
- 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

Una vez entrenados y probados los diferentes modelos utilizados en esta práctica fue muy interesante observar como la mayoría tienen un excelente desempeño para clasificar instancias.

Comenzaremos por mostrar los resultados obtenidos utilizando las máquinas de soporte vectorial, se utilizaron cuatro kernels distintos y se logran apreciar las diferencias entre cada uno. Después

se mostrarán los resultados obtenidos con métodos de clasificación utilizados en prácticas anteriores y por último se comprarán todos estos modelos para mostrar sus diferencias.

#### 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

**4.2.1 SVM con kernel lineal.** Al utilizar el kernel lineal se puede ver como la mayoría de las instancias se clasifican correctamente, algunas quedan fuera, como el número 1 con dos clasificaciones falsas, pero en general el desempeño de la SVM con kernel lineal nos permite clasificar las instancias satisfactoriamente, en este caso se clasificó para múltiples clases (10) y aún así logra hacerlo bien. La precisión obtenida del modelo fue de 0.9777

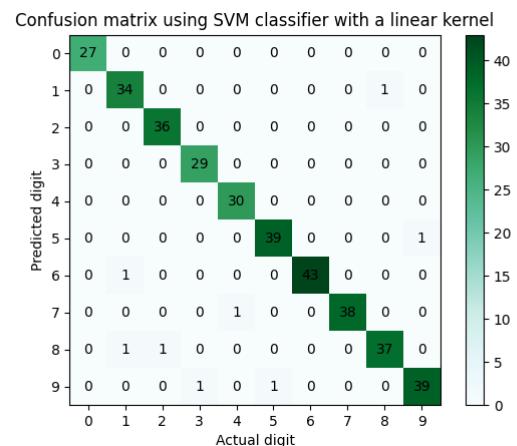
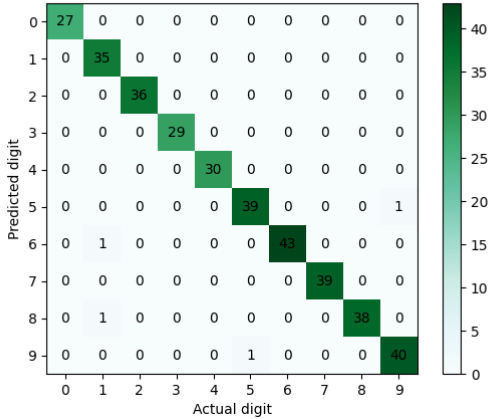


Figure 1: Matriz de confusión del modelo de máquina de soporte vectorial usando un kernel lineal

**4.2.2 SVM con kernel polinomial.** Al utilizar la maquina de soporte vectorial con kernel polinomial se observa una mejoría en el desempeño del modelo para clasificar, aunque el lineal ya era bueno este tiene aún mejores resultados. La matriz de confusión obtenida fue muy similar a la anterior, se observa que casi todos quedan clasificados correctamente y que las instancias clasificadas erróneamente son muy pocas (casi nulas). La precisión obtenida del modelo fue de 0.9888

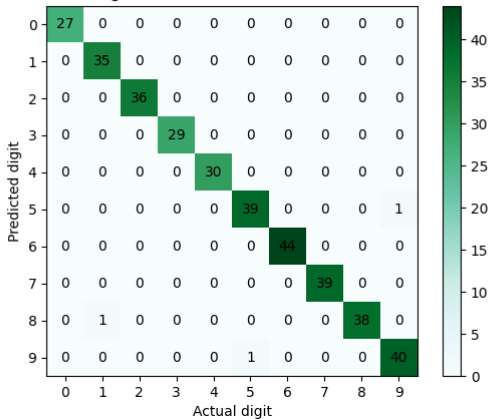
Confusion matrix using SVM classifier with a polynomial kernel



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

**4.2.3 SVM con kernel RBF.** Cuando se utilizó como kernel la función de base radial (RBF) se notó una mejoría en la precisión del modelo para clasificar, aunque dichas mejorías no sean tan significativas (diferencias de 1La precisión obtenida del modelo fue de 0.9916

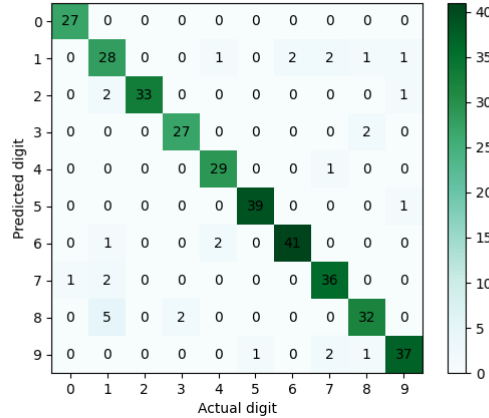
Confusion matrix using SVM classifier with a Radial Basis Function kernel



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

**4.2.4 SVM con kernel Sigmoide.** Para este punto los resultados de las maquinas de soporte vectorial eran muy prometedores, se pensaba que cada que se cambiaba el kernel habría mejoría, pero no fue el caso cuando se utilizó como kernel la sigmoide, la matriz de confusión nos muestra más errores al momento de clasificar las instancias aunque si clasifica la mayoría bien. Lo anterior también se ve reflejado en la precisión del modelo la cual fue más baja que los tres anteriores, seguramente el dataset y número de clases utilizadas no favoreció el desempeño de la SVM con este kernel. La precisión obtenida del modelo fue de 0.9138

Confusion matrix using SVM classifier with a sigmoid kernel



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

**4.2.5 Regresión logística.** Después de utilizar SVMs probando cuatro kernels distintos llega el momento de probar los métodos de clasificación ya utilizados anteriormente, cabe aclarar que no se habían utilizado para predecir tantas clases, así que se comenzó por probar el modelo de regresión logística, los resultados obtenidos fueron muy buenos pero no tan buenos como algunas SVMs, la matriz de confusión muestra algunos errores y esto a su vez se refleja en la precisión obtenida por el modelo, aunque sigue siendo una muy buena precisión y que se probó que la regresión logística funciona para clasificar múltiples clases no quita el hecho de que las SVMs tienen un mejor desempeño (para este caso utilizando este dataset). La precisión obtenida del modelo fue de 0.9472

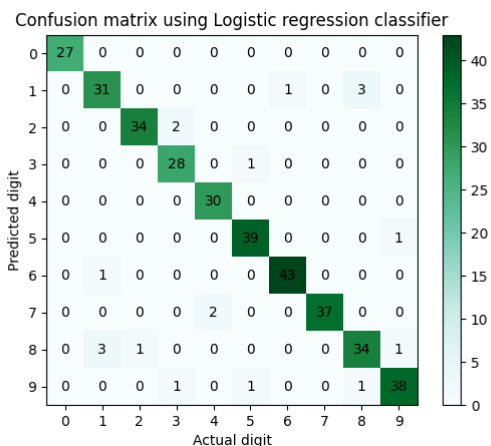


Figure 5: Matriz de confusión del modelo de regresión logística

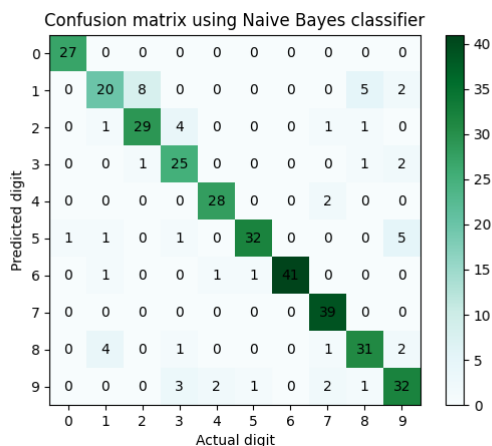


Figure 7: Matriz de confusión del modelo bayesiano ingenuo

4.2.6 *Knn*. Al utilizar como modelo de clasificación Knn el resultado fue que es uno de los mejores para clasificación múltiple, su desempeño fue muy similar a la mejor SVM (polinomial) y la matriz de confusión lo muestra, se ven muy pocos errores y que casi todos se encuentran clasificados correctamente. La precisión obtenida del modelo fue de 0.9888

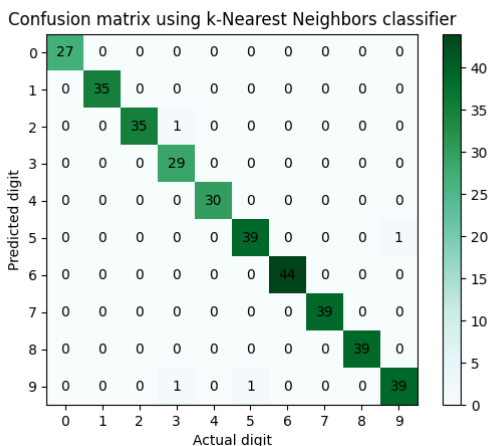


Figure 6: Matriz de confusión del modelo de k vecinos más cercanos

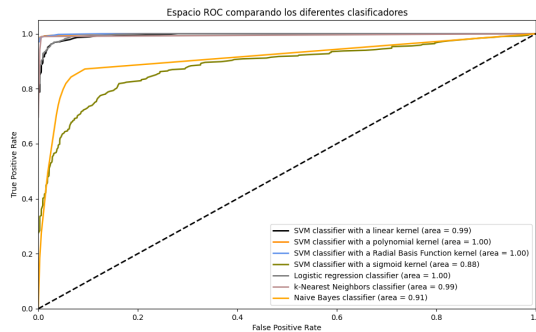
4.2.7 *Bayesiano ingenuo*. Por último se compara el modelo bayesiano ingenuo el cual no mostró una mejoría respecto a los modelos mencionados anteriormente, de hecho ha sido el modelo con peor desempeño, puede ser por que fueron muchas clases a clasificar. La precisión obtenida del modelo fue de 0.8444

### 4.3 Espacio ROC de los modelos

Para todos los clasificadores implementados en esta practica se observaron matrices de confusión muy similares, todos clasificaron la mayoría de las instancias correctamente lo cual es bueno, pero debido a que eran muchas clases la matriz de confusión se vuelve difícil de leer para poder comparar los clasificadores, en este caso una buena métrica podría ser compararlos en base a su precisión, pero si queremos tomar una vista más general del modelo y tomar en cuenta más que su precisión deberemos de utilizar el espacio ROC para compararlos.

El espacio ROC nos sirve para poder identificar las ventajas y desventajas de cada clasificador graficando el promedio de los falsos positivos y verdaderos positivos en un plano. Lo ideal para un modelo es que este su TPR esté lo más cercano al 1 y su FPR lo más cercano al 0, esa sería la precisión ideal de predicción. Se hará una pequeña modificación para poder emplearlos con múltiples clases y se tomará el macro para graficar el espacio ROC.

El resultado final fue que la mayoría de las SVMs tienen un excelente desempeño y son ideales para este dataset, a excepción de la SVM con kernel sigmoide la cual fue la más baja de todas, el segundo peor modelo fue bayes y después todas las SVMs junto con Knn y regresión logística presentan los mejores resultados (entre 0.99 y 1), recordando que el mayor desempeño es de 1.



**Figure 8: Gráfica de comparación de los modelos en el espacio ROC**

## 5 CONCLUSIONES Y REFLEXIONES

Esta practica fue todo un reto ya que no se habían utilizado clasificadores para múltiples clases, por lo cual no se sabía como iban a ser los resultados arrojados por los modelos, al implementarlos y observar las matrices de confusión obtenidas se aprecia que la mayoría de los clasificadores son muy buenos y tienen pocos errores (unos mas que otros), también se observa que los métodos de clasificación utilizados en practicas anteriores funcionaron óptimamente para múltiples clases.

Las maquinas de soporte vectorial muestran desempeño excelente para clasificar, pero al iterar sobre los diferentes kernels nos dimos cuenta que unos tienen mejor desempeño que otros, aunque depende del dataset y del caso de uso, se debe de tener en cuenta que la selección del kernel impactará el desempeño del modelo por lo que se recomienda hacer algo muy parecido con Knn para encontrar la K ideal, iterar sobre los kernels para ver cual tiene mejor precisión.

En general todos los modelos tuvieron muy buen desempeño, fue muy interesante ver como la precisión cambia entre cada uno pero oscilan en rangos muy similares.

Por último al observar el espacio ROC nos damos cuenta que el modelo ideal está entre algunas SVMs y regresión logística, pero fue interesante ver como Naive bayes tiene la peor precisión mientras que el modelo con el peor desempeño en el espacio ROC es la SVM con sigmoide, esto nos muestra que no solo debemos de seleccionar un modelo por su precisión si no que es valioso ver como se comporta en el espacio ROC para tomar decisiones.

### 5.1 Refrexi3n de Abraham

Esta fue una de las prácticas más interesantes hasta el momento ya que la complejidad del codigo no fue muy alta lo que nos permitió enfocarnos en el analisis de resultados y comparaci3n del desempeño de los clasificadores, me gust3 ver como se comportan al clasificar múltiples clases lo cual me será útil para mi proyecto, y también me pareci3 muy valioso saber que se tienen que probar diferentes kernels para ver cual es el más óptimo para una SVM.

También ver que el espacio ROC nos muestra una mejor vista de los modelos me será de gran utilidad para no guiarme solo por su precisión.

Este trabajo me ayud3 a comprender los conceptos de aprendizaje autom3tico vistos en clase y a desarrollar mi capacidad de analizar y comparar.

### 5.2 Reflexi3n de Mario

TODO

## REFERENCES

## A CÓDIGO PARA LA COMPARACIÓN DE MODELOS DE CLASIFICACIÓN

```
1 import sys
2 import matplotlib.pyplot as plt
3 import numpy as np
4 from sklearn.datasets import load_digits
5 from sklearn.metrics import accuracy_score, confusion_matrix
6 from sklearn.model_selection import train_test_split
7 from sklearn.naive_bayes import BernoulliNB
8 from sklearn.neighbors import KNeighborsClassifier
9 from sklearn.linear_model import LogisticRegression
10 from sklearn.svm import SVC
11
12 # Choose a classifier using the first parameter
13 possible_clfs = ["linear", "poly", "rbf", "sigmoid",
14                 "logistic", "knn", "bayes"]
15 if len(sys.argv) < 2 or sys.argv[1] not in possible_clfs:
16     print("Include an argument from the list:", possible_clfs)
17     exit(1)
18
19 classifier_type = sys.argv[1]
20 if classifier_type in ["linear", "poly", "rbf", "sigmoid"]:
21     clf = SVC(kernel=classifier_type)
22     classifier_name = "SVM classifier with a " + {
23         "linear": "linear kernel",
24         "poly": "polynomial kernel",
25         "rbf": "Radial Basis Function kernel",
26         "sigmoid": "sigmoid kernel",
27     }[classifier_type]
28 elif classifier_type == "logistic":
29     clf = LogisticRegression(multi_class="ovr", max_iter=1000)
30     classifier_name = "Logistic regression classifier"
31 elif classifier_type == "knn":
32     clf = KNeighborsClassifier(n_neighbors=1)
33     classifier_name = "k-Nearest Neighbors classifier"
34 elif classifier_type == "bayes":
35     clf = BernoulliNB()
36     classifier_name = "Naive Bayes classifier"
37
38 # Load the dataset and split it
39 digitsX, digitstest = load_digits(return_X_y=True)
40 trainX, testX, trainy, testy = train_test_split(
41     digitsX, digitstest, test_size=0.2, random_state=0)
42
43 # Fit the model and predict the labels
44 clf.fit(trainX, trainy)
45 predicty = clf.predict(testX)
46
47 # Print results
48 print(classifier_name)
49 print("Accuracy:", accuracy_score(testy, predicty))
50 cm = confusion_matrix(testy, predicty)
51 print("Confusion matrix:")
52 print(cm)
```

```

54 # Plot the confusion matrix
55 plt.figure()
56 plt.imshow(cm, interpolation="nearest", cmap="BuGn")
57 plt.title("Confusion matrix using " + classifier_name)
58 plt.colorbar()
59 plt.xticks(np.arange(10), np.arange(10).astype(str), size=10)
60 plt.yticks(np.arange(10), np.arange(10).astype(str), size=10)
61 plt.xlabel("Actual digit")
62 plt.ylabel("Predicted digit")
63 for x in range(10):
64     for y in range(10):
65         plt.annotate(cm[x][y], xy=(y, x),
66                     horizontalalignment="center",
67                     verticalalignment="center")
68 plt.savefig(classifier_type + "_cm.png")

```

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

```

1 from sklearn.datasets import load_digits
2 from sklearn.model_selection import train_test_split
3 from sklearn.neighbors import KNeighborsClassifier
4
5 digitsX, digitSy = load_digits(return_X_y=True)
6 trainX, testX, trainy, testy = train_test_split(
7     digitsX, digitSy, test_size=0.2, random_state=0)
8
9 for neighbors in [1, 2, 3, 4, 5, 7, 10, 15, 20, 25, 30, 40, 50]:
10     clf = KNeighborsClassifier(n_neighbors=neighbors)
11     clf.fit(trainX, trainy)
12     accuracy = clf.score(testX, testy)
13     print("Accuracy using {:2} neighbors: {:.4f}".format(neighbors, accuracy))

```

## C CÓDIGO PARA LA COMPARACIÓN Y GRAFICACIÓN DEL ESPACIO ROC

```
1 import matplotlib.pyplot as plt
2 import numpy as np
3 from sklearn.datasets import load_digits
4 from sklearn.metrics import accuracy_score, confusion_matrix
5 from sklearn.model_selection import train_test_split
6 from sklearn.naive_bayes import BernoulliNB
7 from sklearn.neighbors import KNeighborsClassifier
8 from sklearn.linear_model import LogisticRegression
9 from sklearn.svm import SVC
10 from sklearn.multiclass import OneVsRestClassifier
11 from sklearn.preprocessing import label_binarize
12 from sklearn.metrics import roc_curve, auc
13 from scipy import interp
14 from itertools import cycle
15
16 def calculateROC(testy, predicty):
17     # Compute ROC curve and ROC area for each class
18     fpr = dict()
19     tpr = dict()
20     roc_auc = dict()
21     for i in range(10):
22         fpr[i], tpr[i], _ = roc_curve(testy[:, i], predicty[:, i])
23         roc_auc[i] = auc(fpr[i], tpr[i])
24
25     # First aggregate all false positive rates
26     all_fpr = np.unique(np.concatenate([fpr[i] for i in range(n_classes)]))
27
28     # Then interpolate all ROC curves at this points
29     mean_tpr = np.zeros_like(all_fpr)
30     for i in range(n_classes):
31         mean_tpr += interp(all_fpr, fpr[i], tpr[i])
32
33     # Finally average it and compute AUC
34     mean_tpr /= n_classes
35
36     fpr["macro"] = all_fpr
37     tpr["macro"] = mean_tpr
38     roc_auc["macro"] = auc(fpr["macro"], tpr["macro"])
39     return [classifier_name, fpr["macro"], tpr["macro"], roc_auc["macro"]]
40
41
42 # Plot all ROC curves
43 def plotROCs(ROCs):
44     lw = 2
45     plt.figure(figsize=(12, 7), dpi=100)
46
47     colors = cycle(['black', 'darkorange', 'cornflowerblue', 'olive', 'gray', 'rosybrown', 'orange', 'darkgray'])
48     for i, color in zip(range(len(ROCs)), colors):
49         plt.plot(ROCs[i][1], ROCs[i][2], color=color, lw=lw,
50                 label=ROCs[i][0] + ' (area = {0:0.2f})'
51                     ''.format(ROCs[i][3]))
52
53     plt.plot([0, 1], [0, 1], 'k--', lw=lw)
54     plt.xlim([0.0, 1.0])
55     plt.ylim([0.0, 1.05])
56     plt.xlabel('False Positive Rate')
57     plt.ylabel('True Positive Rate')
58     plt.title('Espacio ROC comparando los diferentes clasificadores')
59     plt.legend(loc="lower right")
60     plt.savefig("ROC_curve.png")
61     plt.show()
62
```