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

La clasificación es una de las tareas más importantes para una diversidad de aplicaciones, como el reconocimiento óptico de caracteres, la clasificación de imágenes, e incluso problemas biológicos y químicos como las expresiones de genes, o predicciones de la estructura de proteínas[3]. Esta práctica tiene como objetivo comparar un método de aprendizaje automático relativamente nuevo: las máquinas de soporte vectorial, utilizando un dataset de imágenes de dígitos.

1 INTRODUCCIÓN

Las máquinas de soporte vectorial son un conjunto de modelos de aprendizaje automático usados principalmente con el objetivo de clasificar un conjunto de datos, aunque también pueden ser usados para regresión o para detectar valores atípicos en el dataset.

La principal ventaja de estos modelos es que son muy efectivos en datasets con muchas dimensiones, incluso cuando se tiene un mayor número de dimensiones que de instancias[4].

Este modelo de aprendizaje automático fue introducido a principios de los años 90 por Vapnik[1] y ganó popularidad rápidamente debido a su desempeño prometedor y a que puede trabajar con muchas dimensiones y pocas muestras.

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

Este dataset está compuesto por 1797 imágenes de 8x8 pixeles, cada uno con un color monocromático en el rango entre el 0 y el 16. Cada una de las imágenes representa un dígito escrito a mano, pasado por un preprocesamiento de reconocimiento de caracteres para intentar eliminar la información considerada como ruido.

Este dataset fue originalmente publicado por el departamento de ingeniería informática de la Universidad Bogazici en Turquía[2], pero puede ser encontrado en el repositorio de Machine Learning de la Universidad de California en Irvine.

3.2 Máquinas de soporte vectorial

Para la implementación de la comparación de modelos de clasificación utilizamos la librería *scikit-learn*, que además de implementar las máquinas de soporte vectorial con la opción de cambiar la función de kernel, incluye también implementaciones para los demás modelos utilizados durante esta práctica.

Específicamente, para el desarrollo de esta práctica, utilizamos la clase *sklearn.svm.SVC*, que acepta un parámetro *kernel*, el cual cambiamos entre los valores "linear", "poly", "rbf" y "sigmoid" para así contrastar el desempeño de cada una de estas opciones.

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 cómo 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 compararán 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.

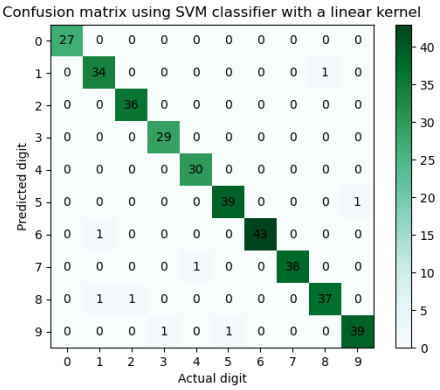

Figure 1: Matriz de confusión del modelo de máquina de soporte vectorial usando un kernel lineal

4.2.2 SVM con kernel polinomial. Al utilizar la máquina 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.

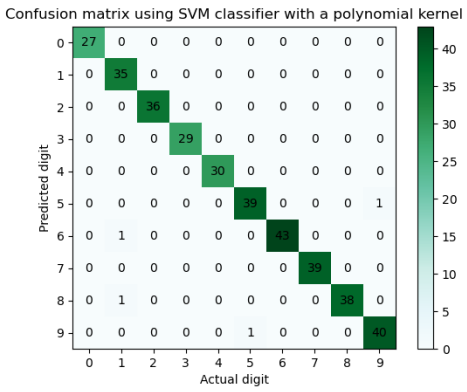

Figure 2: Matriz de confusión del modelo de máquina de soporte vectorial usando un kernel polinomial

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 1%-2%) nos permite observar cómo utilizando las mismas máquinas de soporte vectorial con diferente kernel nos permite hacer un modelo mejor entrenado.

La precisión obtenida del modelo fue de 0.9916.

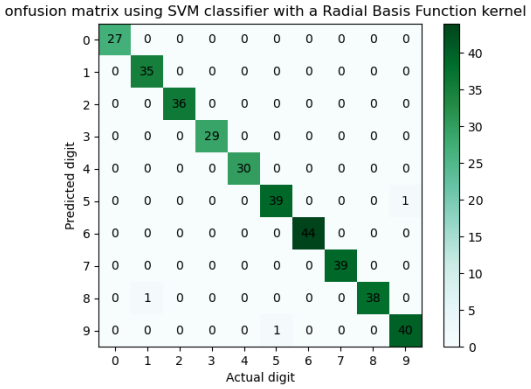

Figure 3: Matriz de confusión del modelo de máquina de soporte vectorial usando un kernel RBF

4.2.4 SVM con kernel Sigmoide. Para este punto los resultados de las máquinas 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.

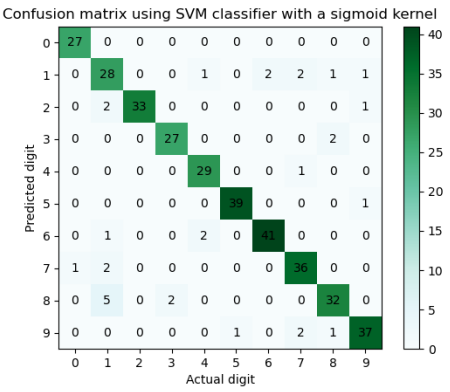

Figure 4: Matriz de confusión del modelo de máquina de soporte vectorial usando un kernel sigmoide

4.2.5 Regresión logística. Después de utilizar SMVs 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 sienda 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.

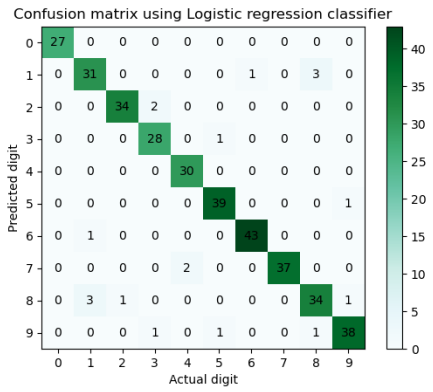

Figure 5: Matriz de confusión del modelo de regresión logística

4.2.6 k-NN. Al utilizar como modelo de clasificación k-NN 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.

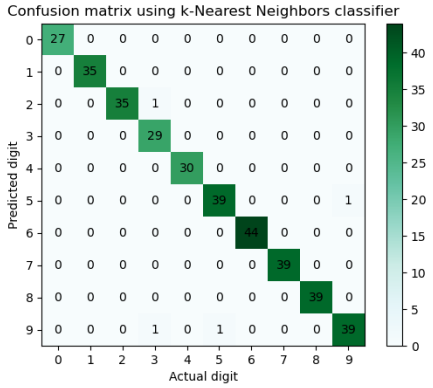

Figure 6: Matriz de confusión del modelo de k vecinos más cercanos

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 porque fueron muchas clases a clasificar.

La precisión obtenida del modelo fue de 0.8444.

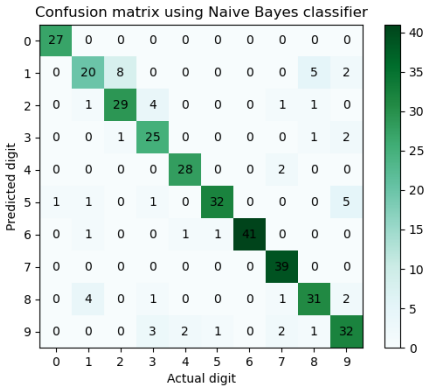

Figure 7: Matriz de confusión del modelo bayesiano ingenuo

4.3 Espacio ROC de los modelos

Para todos los clasificadores implementados en esta práctica 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 k-NN y regresión logística presentan los mejores resultados (entre 0.99 y 1), recordando que el mayor desempeño es de 1.

Para comparar los modelos en el espacio ROC utilizamos el código expuesto en el apéndice C.

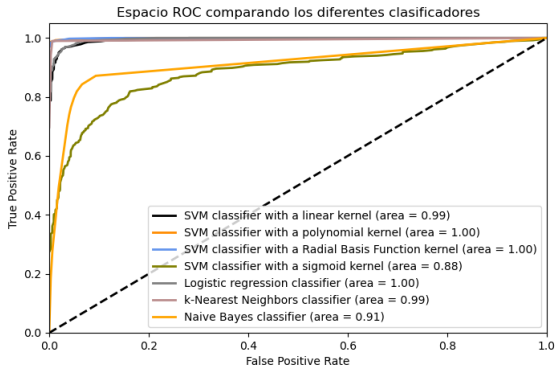

Figure 8: Gráfica de comparación de los modelos en el espacio ROC

5 CONCLUSIONES Y REFLEXIONES

Esta práctica 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 más que otros), también se observa que los métodos de clasificación utilizados en prácticas anteriores funcionaron óptimamente para múltiples clases.

Las máquinas 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 esto 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 k-NN para encontrar la k ideal, iterar sobre los kernels para ver cuál tiene mejor precisión.

En general todos los modelos tuvieron muy buen desempeño, fue muy interesante ver cómo 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 sólo 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 Refrexión de Abraham

Esta fue una de las prácticas más interesantes hasta el momento ya que la complejidad del código no fue muy alta lo que nos permitió enfocarnos en el analisis de resultados y comparación del desempeño de los clasificadores, me gustó ver cómo se comportan al clasificar múltiples clases lo cual me será útil para mi proyecto, y también me pareció muy valioso saber que se tienen que probar diferentes kernels para ver cuál 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 sólo por su precisión.

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

5.2 Reflexión de Mario

Considero que en esta práctica fue muy enriquecedor poder comparar directamente múltiples modelos de aprendizaje automático para visualizar cómo se comportan, qué tan fácil es usarlos y principalmente qué tan precisos son para realizar su objetivo común, que en este caso fue clasificar un dataset con muchas dimensiones (cada imagen se componía de 64 enteros).

Por otra parte, pienso que el desarrollo de este reporte fue mucho más claro y directo gracias a la experiencia que hemos acumulado durante el semestre utilizando *sklearn*, que definitivamente es una herramienta con muchísimas más opciones, algoritmos y modelos que nunca se me hubieran ocurrido pero espero aprender durante lo que falta del semestre.

REFERENCES

[1] Corinna Cortes and Vladimir Vapnik. 1995. Support vector machine. *Machine learning* 20, 3 (1995), 273–297.

[2] Dheeru Dua and Casey Graff. 2017. UCI Machine Learning Repository. http://archive.ics.uci.edu/ml

[3] K SRIVASTAVA Durgesh and B Lekha. 2010. Data classification using support vector machine. *Journal of theoretical and applied information technology* 12, 1 (2010), 1–7.

[4] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. 2011. Scikit-learn: Machine Learning in Python. *Journal of Machine Learning Research* 12 (2011), 2825–2830.

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))
```

C CÓDIGO PARA LA COMPARACIÓN Y GRAFICACIÓN DEL ESPACIO ROC

```
import matplotlib.pyplot as plt
import numpy as np
from sklearn.datasets import load_digits
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
from sklearn.multiclass import OneVsRestClassifier
from sklearn.preprocessing import label_binarize
from sklearn.metrics import roc_curve, auc
from itertools import cycle


def calculateROC(classifier_name, testy, predicty):
    # Compute ROC curve and ROC area for each class
    fpr = dict()
    tpr = dict()
    roc_auc = dict()
    for i in range(10):
        fpr[i], tpr[i], _ = roc_curve(testy[:, i], predicty[:, i])
        roc_auc[i] = auc(fpr[i], tpr[i])

    # First aggregate all false positive rates
    all_fpr = np.unique(np.concatenate([fpr[i] for i in range(n_classes)]))

    # Then interpolate all ROC curves at this points
    mean_tpr = np.zeros_like(all_fpr)
    for i in range(n_classes):
        mean_tpr += np.interp(all_fpr, fpr[i], tpr[i])

    # Finally average it and compute AUC
    mean_tpr /= n_classes

    return [classifier_name, all_fpr, mean_tpr, auc(all_fpr, mean_tpr)]


def plotROCs(ROCs):
    """ Plot all ROC curves """
    plt.figure(figsize=(8, 5), dpi=100)

    colors = cycle(['black', 'darkorange', 'cornflowerblue', 'olive',
                    'gray', 'rosybrown', 'orange', 'darkviolet', 'crimson', 'slategray'])
    for i, color in zip(range(len(ROCs)), colors):
        plt.plot(ROCs[i][1], ROCs[i][2], color=color, lw=2,
                 label=ROCs[i][0] + ' (area = {0:0.2f})'
                 ''.format(ROCs[i][3]))

    plt.plot([0, 1], [0, 1], 'k--', lw=2)
    plt.xlim([0.0, 1.0])
    plt.ylim([0.0, 1.05])
    plt.xlabel('False Positive Rate')
    plt.ylabel('True Positive Rate')
    plt.title('Espacio ROC comparando los diferentes clasificadores')
    plt.legend(loc="lower right")
    plt.savefig("ROC_curve.png")
    plt.show()
```

```
ROCs = []
n_classes = 10
clfs = ["linear", "poly", "rbf", "sigmoid",
        "logistic", "knn", "bayes"]

# Load the dataset and split it
digitsX, digitsy = load_digits(return_X_y=True)
digitsy = label_binarize(digitsy, classes=[0, 1, 2, 3, 4, 5, 6, 7, 8, 9])
trainX, testX, trainy, testy = train_test_split(
    digitsX, digitsy, test_size=0.2, random_state=0)

for clf_name in clfs:
    if clf_name in ["linear", "poly", "rbf", "sigmoid"]:
        clf = SVC(kernel=clf_name)
        predicty = OneVsRestClassifier(clf).fit(
            trainX, trainy).decision_function(testX)
        classifier_name = "SVM classifier with a " + {
            "linear": "linear kernel",
            "poly": "polynomial kernel",
            "rbf": "Radial Basis Function kernel",
            "sigmoid": "sigmoid kernel",
        }[clf_name]
    elif clf_name == "logistic":
        clf = LogisticRegression(multi_class="ovr", max_iter=1000)
        predicty = OneVsRestClassifier(clf).fit(
            trainX, trainy).decision_function(testX)
        classifier_name = "Logistic regression classifier"
    elif clf_name == "knn":
        clf = KNeighborsClassifier(n_neighbors=1)
        predicty = OneVsRestClassifier(clf).fit(trainX, trainy).predict(testX)
        classifier_name = "k-Nearest Neighbors classifier"
    elif clf_name == "bayes":
        clf = BernoulliNB()
        predicty = OneVsRestClassifier(clf).fit(trainX, trainy).predict(testX)
        classifier_name = "Naive Bayes classifier"

    ROCs.append(calculateROC(classifier_name, testy, predicty))

plotROCs(ROCs)
```