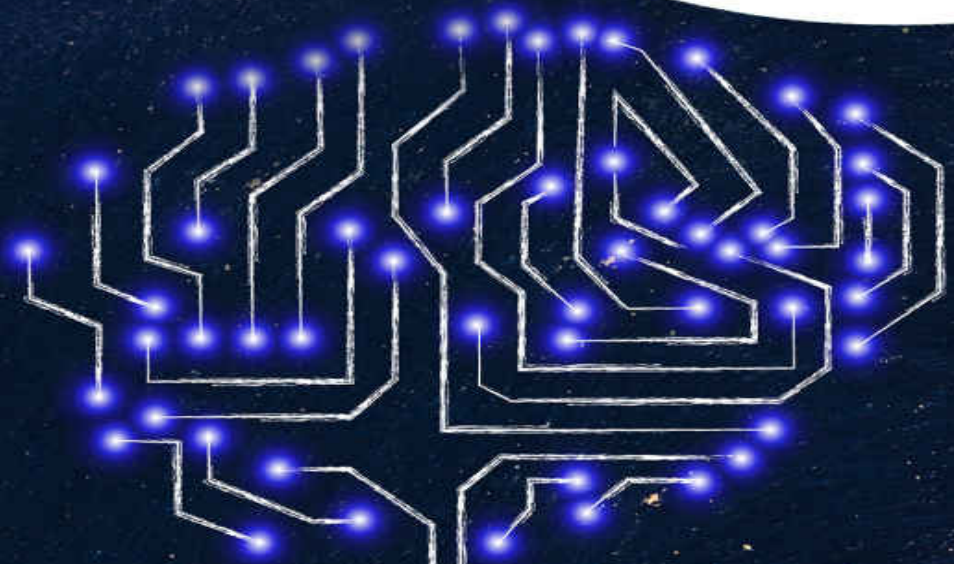
MACHINE LEARNING

GUÍA DETALLADA PARA IMPLEMENTAR ALGORITMOS DE MACHINE LEARNING CON PYTHON



RUDOLPH RUSSELL

Machine Learning

Guía Paso a Paso Para Implementar Algoritmos De Machine Learning Con Python

Rudolph Russell

Tabla de Contenido

CAPITULO 1

INDTODUCCIÓN AL MACHINE LEARNING

Teoría

Si pregunto acerca del “Machine Learning” probablemente imaginarás un robot o algo como el Exterminador. En realidad, el Machine Learning no solo está involucrado en la robótica, pero además en muchas otras aplicaciones. También puedes imaginar algo como filtros de spam al ser una de las primeras aplicaciones en Machine Learning el cual ayuda a mejorar la vida de millones de personas. En este capítulo, te presentaré lo que es el Machine Learning, y como funciona.

¿Qué es Machine Learning?

El Machine Learning es la práctica de programación de computadoras para aprender de los datos. En el ejemplo de arriba, el programa fácilmente podrá determinar si lo dado es importante o si es “spam” (correo electrónico no deseado). En el Machine Learning, los datos son conocidos como conjuntos de capacitación o ejemplos.

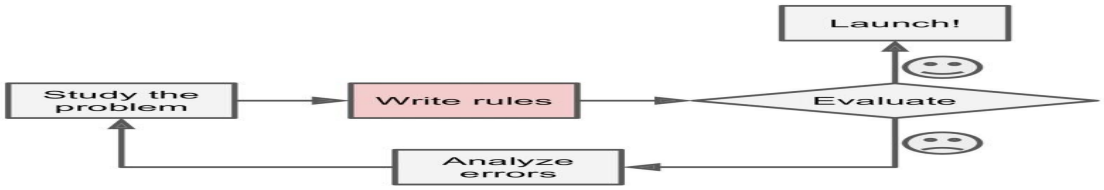
¿Por qué Machine Learning?

Asumamos que quisieras escribir el programa filtro sin usar métodos de Machine Learning. En este caso, tendrás que seguir los siguientes pasos:

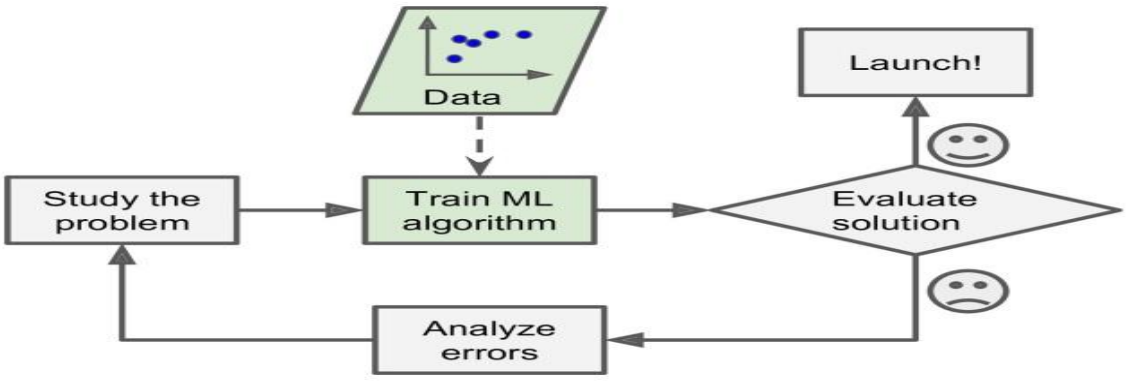
Al principio, echarías un vistazo a lo cómo se ven los correos electrónicos no deseados. Podrías seleccionarlos por las palabras o frases que usan, como “tarjeta de debido”, “gratis”, y muchas más, y además de patrones que son usados en los nombres de los remitentes o en el cuerpo del correo.

.Segundo, escribirías un algoritmo para detectar los patrones que hayas visto, y luego el software indicaría los correo electrónicos como “no deseado” si un numero de esos patrones es encontrado.

Finalmente, probarías el programa, y reharías los primeros dos pasos hasta que los resultados sean suficientemente buenos.



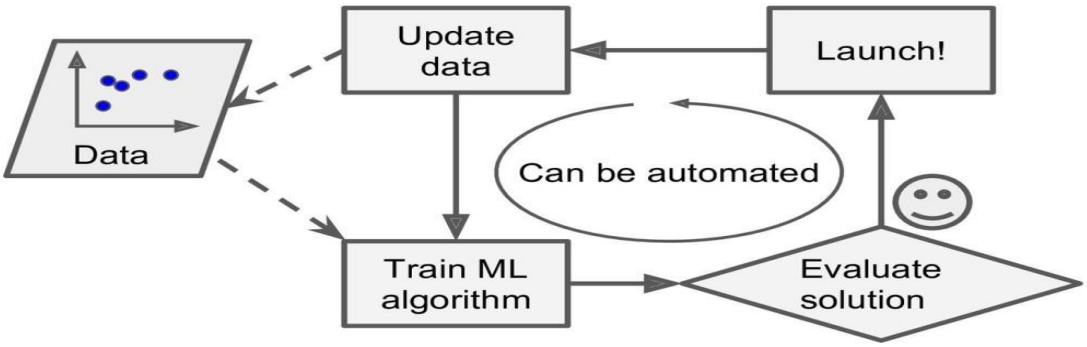
Porque el programa no es software, contienen una lista muy larga de reglas que son difíciles de mantener. Pero si desarrollas el mismo software usando AA (Machine Learning), podrás mantenerlo adecuadamente.



Adicionalmente, los remitentes de los correos electrónicos pueden cambiar las plantillas de los correos para que una palabra como “4You” ahora sea “para ti”, ya que sus correos han sido determinados como “no deseados”.

Los programas que usan técnicas tradicionales necesitarían ser actualizados, lo que significa que, si hubiese otros cambios, necesitarías actualizar tus códigos otra vez, otra vez, y otra vez.

Por otro lado, un programa que usa técnicas de AA automáticamente detectarán estos cambios hechos por los usuarios, y comenzará a indicarlos como “no deseados” sin la necesidad de que lo hagas manualmente.



Además, podemos usar AA para resolver problemas que son muy complejos los softwares que no usan AA. Por ejemplo, reconocimiento de discurso, cuando tú dices “uno” o “dos”, el programa debería poder distinguir la diferencia. Entonces, para esta tarea, deberás desarrollar un algoritmo que mide el sonudo.

Al final, el Machine Learning nos ayudará a aprender, y los algoritmos del Machine Learning nos puede ayudar a ver lo que hemos aprendidos.

¿Cuándo debemos usar Machine Learning?

- Cuando tengas un problema que requiera de largas listas de reglas para encontrar la solución. En este caso, las técnicas de Machine Learning pueden simplificar tu código y mejorar el desempeño.
- Problemas muy complejos para los que no haya solución con un enfoque tradicional.
- Ambientes no estables: softwares con Machine Learning se pueden adaptar a nuevos datos.

Tipos de Sistemas de Machine Learning

Hay diferentes tipos de sistemas de Machine Learning. Podemos dividirlos en categorías dependiendo si:

- Han sido capacitados con humanos o no
    - Supervisado
    - Sin supervisión
    - Semi-supervisado
    - Machine Learning de Refuerzo
- Si pueden aprender de forma incrementada
- Si pueden trabajar simplemente combinando nuevos puntos de datos, o si pueden detectar nuevos patrones en los datos, y luego construirán un modelo

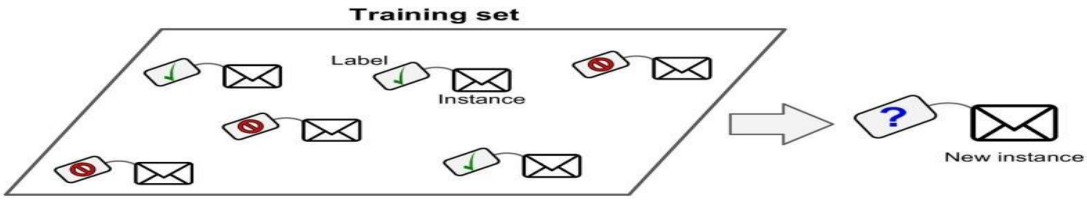
Machine Learning Supervisado y Sin Supervisión

Podemos clasificar los sistemas de Machine Learning de acuerdo al tipo y a la cantidad de supervisión humana durante la capacitación. Podrás encontrar 4 categorías principales, como hemos explicado antes.

- Machine Learning Supervisado
- Machine Learning sin Supervisión
- Machine Learning Semi-Supervisado
- Machine Learning de Refuerzo

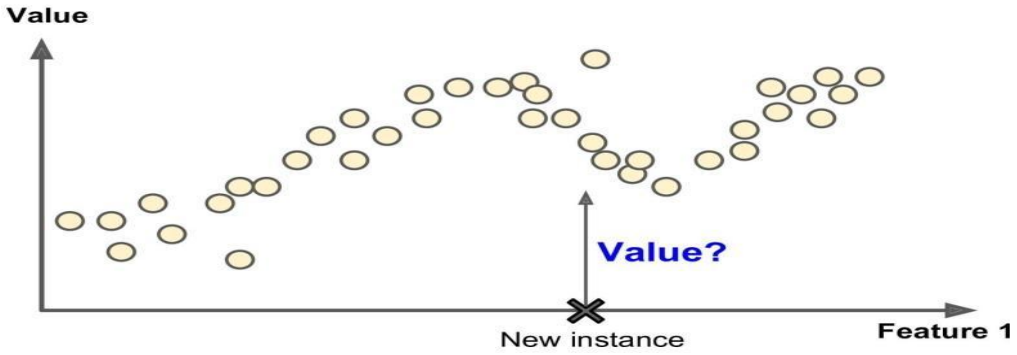
Machine Learning Supervisado

En este tipo de sistema de Machine Learning los datos con que tu alimentas el algoritmo, con la solución deseada, son referidos como “labels” (etiquetas).



- El Machine Learning supervisado agrupa tareas de clasificación. El programa de arriba es un buen ejemplo de esto porque ha sido capacitado con muchos correos electrónicos al mismo tiempo que sus clases

Otro ejemplo es predecir un valor numérico como el precio de un apartamento, dados un numero de características (ubicación, número de habitaciones, instalaciones), llamados predictores; este tipo de tarea es llamado regresión.



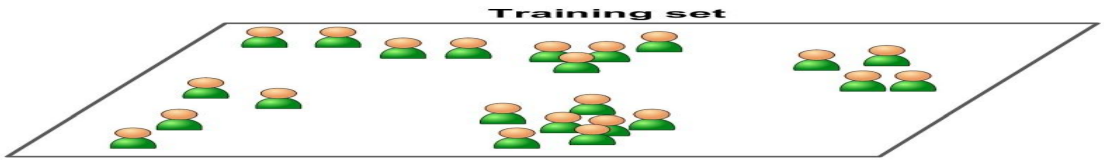
Debes tener en cuenta que algunos algoritmos de regresión puden ser usados para clasificación también, y viceversa.

Los Algoritmos Supervisados Mas Importantes

- K-nearest neighbors (KNN, vecinos más cercanos K)
- Red Neural
- Máquinas de soporte de vectores
- Regresión logística
- Arboles de decisiones y bosques aleatorios

Machine Learning No Supervisado

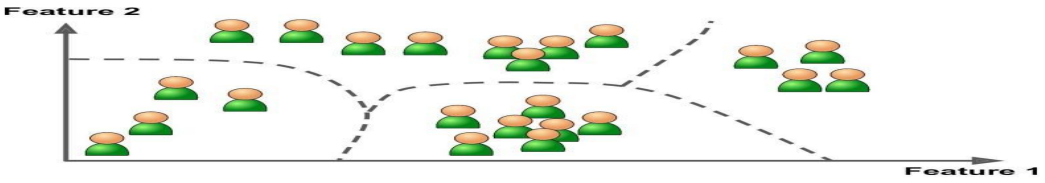
En este tipo de sistemas de Machine Learning, puedes suponer que los datos están sin etiqueta.



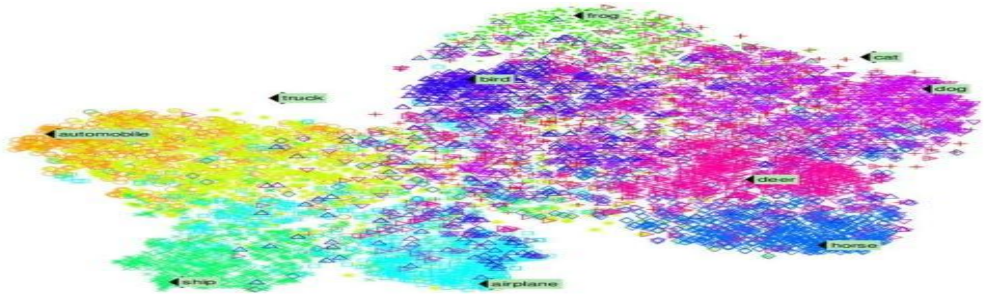
Los Más Importantes Algoritmos del Machine Learning No Supervisado

- **Agrupamiento (Clustering):** Medios k, análisis de agrupamiento jerárquico
- Machine Learning de Asociación de Regla: Eclat y A priori
- Visualización y Reducción de Dimensionalidad: Núcleo PCA, distribuido de t PCA.

Como ejemplo, supongamos que tienes muchos datos en el uso visitante de uno de nuestros algoritmos para detecar grupos con visitantes similares. Podría encontrar que el 65% de tus visitantes son masculinos a quienes les encanta ver películas en las noches, mientras que el 30% ve obras en las noches; en este caso, usando un algoritmo de agrupamiento, dividirá cada grupo en sub grupos.



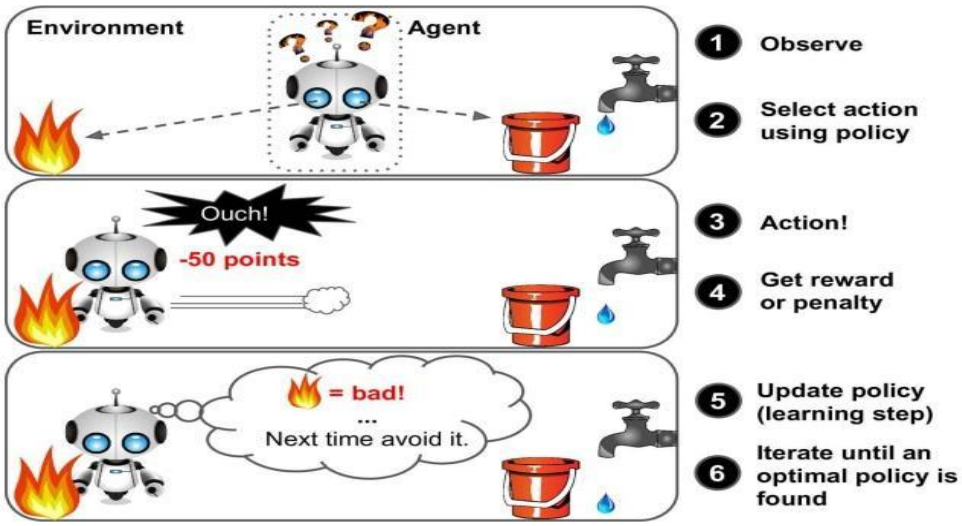
Hay algoritmos muy importantes, como los algoritmos de visualización; estos algoritmos son de Machine Learning sin supervisión. Necesitarás dales muchos datos y datos sin etiqueta como una salida, y entonces obtendrás visualización 2D o 3D como una salida.



La meta aquí es hacer la salida tan simple como sea posible sin perder nada de la información. Para manejar este problema, se combinaran muchas características relacionadas en una sola característica; por ejemplo, combinará la marca de un automóvil con su modelo. Esto es llamado extracción de característica.

Machine Learning de Refuerzo

Machine Learning de refuerzo es otro tipo de sistemas de Machine Learning. Un agente “Sistema AI” (Inteligencia Artificial) observará el ambiente, ejecutando diferentes acciones, y luego recibe recompensas t a cambio. Con este tipo, el agente debe aprender por sí mismo. Es llamado una póliza.



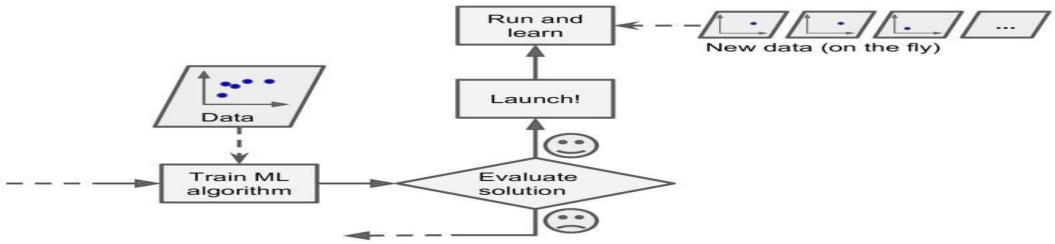
Puedes conseguir este tipo de enseñanza en muchas aplicaciones robóticas que aprenden como caminar

Machine Learning por Lote

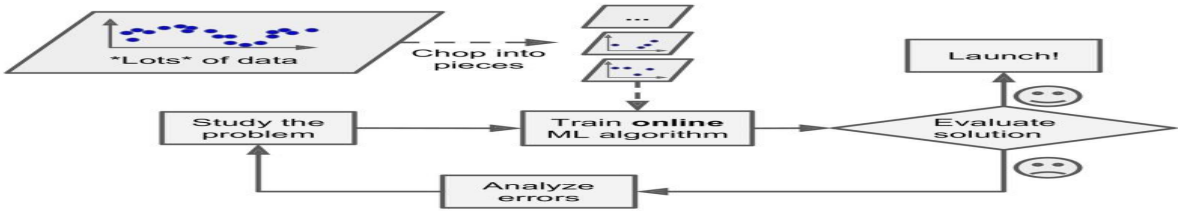
En este tipo de sistema de Machine Learning, el sistema no puede aprender de manera incrementada: El sistema debe obtener todos los datos necesarios. Esto quiere decir que necesitará muchos recursos y una gran cantidad de tiempo, así que siempre es hecho fuera de línea. Entonces, apara trabajar con este tipo de Machine Learning, lo primero que hay que hacer es capacitar el sistema, y luego lanzarlo sin ningún Machine Learning.

Machine Learning En-Línea

Este tipo de Machine Learning es lo opuesto al Machine Learning por lote. Lo que quiero decir es que aquí **el sistemas puede aprender de forma incrementada proveyendo al sistema con** todos los todos disponibles como ejemplos (grupos o individualmente), y luego el sistema puede aprender en el proceso.



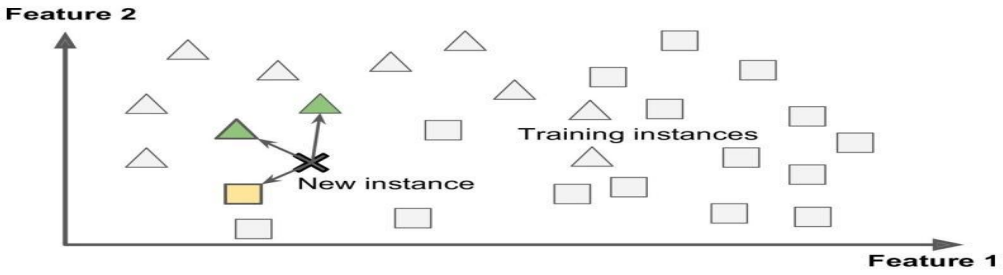
Puedes usar este tipo de sistema para problemas que requieran el flujo continuo de datos, los cuales además necesitan adaptarse rápidamente al cualquier cambio. Además puedes utilizar este tipo de sistemas para trabajar con grandes paquetes de datos.



Deberías saber que tan rápido tu sistema puede adaptarse a cualquier cambio en la "velocidad de Machine Learning". Si la velocidad es alta, significa que el sistema aprenderá rápidamente pero además olvidara datos viejos más rápidamente.

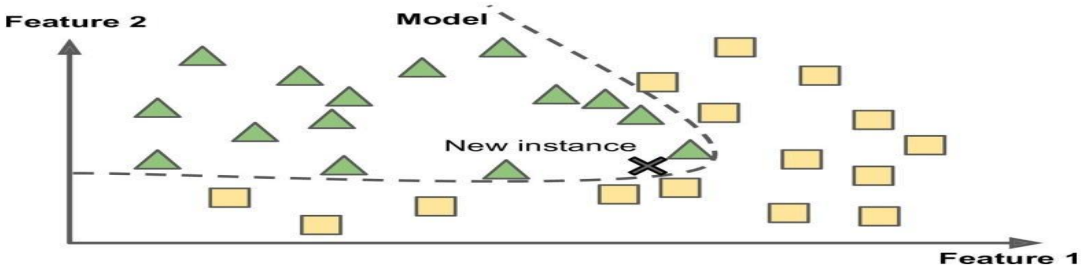
Machine Learning por Ejemplos

Este es el tipo de Machine Learning más simple que debes aprender de memoria. Al usar este tipo de Machine Learning en nuestro programa de correo electrónico, colocará una indicación (flag) en todos los correos que fueron indicados (flag) por los usuarios.



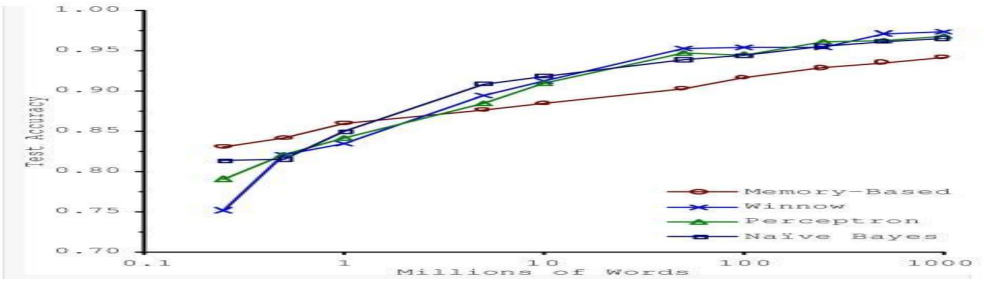
Machine Learning por Modelo

Este es otro tipo de Machine Learning en el cual el Machine Learning por ejemplos permite la construcción para hacer predicciones.



Insuficiente Cantidad o Malos Datos de Capacitación

Los sistemas de Machine Learning no son como los niños, quienes pueden distinguir entre manzanas y naranjas y todo tipo de colores y formas, pero ellos requieren una gran cantidad de datos para trabajar efectivamente, bien sea que trabajes con programas y problemas muy simples, o aplicaciones complejas como el procesamiento de imágenes y el reconocimiento de discurso. Aquí hay un ejemplo de la irrazonable efectividad de los datos, mostrando el proyecto de Machine Learning. El cual incluye datos simples y el problema complejo de NLP.



Datos de Calidad Pobre

Si estás trabajando con datos de capacitación que estas llenos de errores y valores atípicos, le será muy difícil al sistema poder detectar patrones, así que no trabajará apropiadamente. Entonces, si quieres que tu programa trabaje bien, debes pasar más tiempo limpiando tus datos de capacitación.

Características Irrelevantes

El sistemas solo podrá aprender si los datos de capacitación contienen suficientes características y datos que no sean muy irrelevantes. La parte más importante de cualquier proyecto de Machine Learning es desarrollar características buenas de “ingeniería de características”.

Ingeniería de Características

El proceso de ingeniería de características va así:

. **Selección de Características:** Seleccionar las características más útiles.

. **Extracción de Características:** Combinar características ya existentes para crear características más útiles.

. **Creación de Nuevas Características:** Creación de características nuevas, basadas en los datos.

Ensayo

Si quisieras asegúrate de que tu modelo esté trabajando bien y que el modelo pueda generalizarse con nuevos casos, puedes probar nuevos casos con éste al colocar el modelo en el ambiente y luego monitorear como se desempeñará. Este es un buen método, pero si tu modelo es inadecuado, el usuario se quejará.

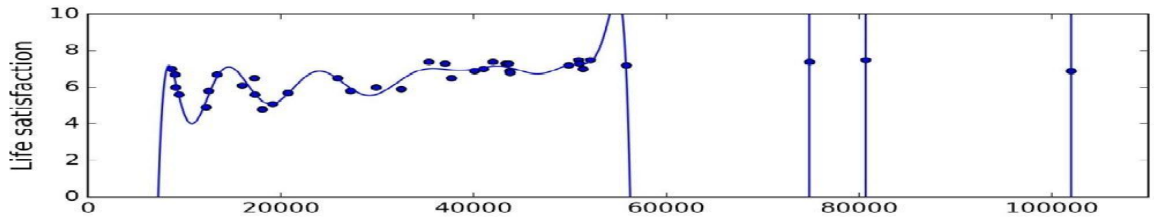
Deberías dividir tus datos en dos grupos, un grupo para capacitación y el otro grupo para pruebas, para que puedas capacitar tu modelo usando el primer grupo y probarlo usando el segundo. La generalización de error es la taza de error por evaluación de tu modelo en el grupo de prueba. El valor que obtengas te dirá si tu modelo es suficientemente bueno, y si trabajará apropiadamente.

Si la taza de error es baja, el modelo es bueno y se desempeñará apropiadamente. En contraste, si tu taza es alta, esto significa que tu modelo se desempeñara mal y no trabajará apropiadamente. Mi consejo para ti es usar el 80% de los datos para capacitación y el 20% restante para propósitos de prueba, para que sea muy simple evaluar y probar un modelo.

Sobreajustando los Datos

Si estás en un país en el extranjero y alguien te roba algo, podrías decir que todos son ladrones. Esto es una sobregeneralización, en el Machine Learning es llamado, “sobreajustar”. Esto significa que las máquinas hacen lo mismo: pueden desempeñarse bien cuando están trabajando con los datos de capacitación, pero no pueden generalizarlos apropiadamente. Por

ejemplo, en la siguiente figura encontrarás un modelo de alto grado de satisfacción de vida que sobreajusta los datos, pero que trabaja bien con los datos de capacitación.



¿Cuándo ocurre esto?

El sobreajuste ocurre cuando el modelo es muy complejo para la cantidad de datos de capacitación dados.

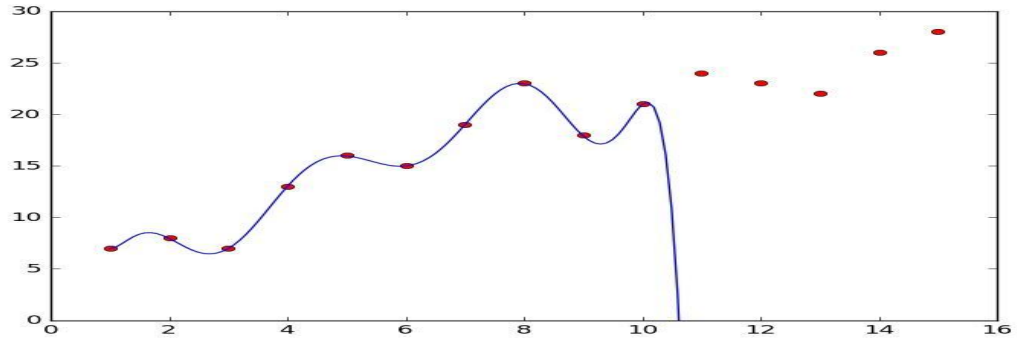
Solución

Para resolver el problema de “sobreajuste”, debes hacer lo siguiente:

- Reúne más datos para “datos de capacitación”
- Reduce los niveles de ruido
- Selecciona uno que menos parámetros

Subajustando los Datos

Pro el nombre, podemos decir que subajustar es el opuesto de sobreajustar, y encontrarás esto cuando el modelos es muy simple de aprender. Por ejemplo, usando el ejemplo de calidad de vida, la vida real es más compleja que tu modelo, así que las predicciones no arrojarán lo mismo, incluso en las muestras de capacitación.



Soluciones

Para solucionar este problema:

- Selecciona el modelo más potente, el cual tienen muchos parámetros
- Alimenta tu algoritmo con las mejores características. Aquí, me refiero a ingeniería de características
- Reduce las restricciones de tu modelo

EJERCISIOS

En este capítulo, habremos cubierto muchos conceptos del Machine Learning. Los siguientes capítulos serán muy prácticos, y escribirás códigos, pero deberías responder las siguientes preguntas, solo para estar seguros de que estas en la vía correcta.

1. Define Machine Learning

2. Describe 4 tipos de sistema de Machine Learning.

3. ¿Cuál es la diferencia entre Machine Learning supervisado y no supervisado?

4. Nombra las tareas no supervisadas

5. ¿Por qué son importantes el ensayo y la validación?

6. En una oración, describe lo que es Machine Learning en-línea

7. ¿Cuál es la diferencia entre Machine Learning por lote y fuera de línea?

8. ¿Cuál tipo de sistema de Machine Learning deberías usar para enseñar a un robot a caminar?

RESUMEN

En este capítulo, has aprendido muchos conceptos útiles, así que vamos a repasar algunos conceptos con los que podrías sentirte un poco perdido. Machine Learning: AA se refiere a hacer que las machinas trabajen mejor en algunas tareas, utilizando datos dados.

El Machine Learning viene en diferentes tipos tales como el supervisado, por lote, sin supervisión, y Machine Learning en-línea

Para ejecutar un proyecto de Machine Learning, debes reunir datos en un grupo de capacitación y luego alimentar un algoritmo de Machine Learning con ese grupo para obtener una salida, “predicciones”.

Si quieres obtener la salida correcta, tu sistema debe usar datos claros, los cuales son no muy pequeños y no tienen características irrelevantes.

CAPITULO 2
CLASIFICACIÓN

Instalación

Necesitarás instalar Python, Matplotlib and Scikit-larn para este capítulo. Solo ve a la sección de referencias y sigue los pasos indicados.

El MNIST

En este capítulo, irás más profundo en la clasificación de sistemas, y trabajarás con el grupo de datos MINIST. Esto es un grupo (set) de 70.000 imágenes de dígitos escritos a manos por estudiantes y empleados. Encontrarás que cada imagen tiene una etiqueta y un digito que la representa. Este proyecto es como el “Hola, mundo” (“Hello, world”) ejemplo de programación tradicional. Así que, cada principiante del Machine Learning debería comenzar por este proyecto para aprender acerca de los algoritmos de clasificación. Scikit-Learn tiene muchas funciones, incluyendo el MNIST. Echemos un vistazo al código.

```
>>> from sklearn.data sets import fetch_mldata
>>> mn= fetch_mldata('MNIST original')
>>> mn
{'COL_NAMES': ['label', 'data'],
'Description': 'mldata.org data set: mn-original',
'data': array([[0, 0, 0,..., 0, 0, 0],
[0, 0, 0,..., 0, 0, 0],
[0, 0, 0,..., 0, 0, 0],
...,
[0, 0, 0,..., 0, 0, 0],
[0, 0, 0,..., 0, 0, 0],
[0, 0, 0,..., 0, 0, 0]], dataType=uint8),
'tar': array([ 0., 0., 0.,..., 9., 9., 9.])} de
```

La descripción es una clave/llave (key) que describe el grupo de datos.

. Los datos claves aquí contienen una colección con solo una fila por ejemplo, una fila por cada característica.

Trabajemos con parte del código

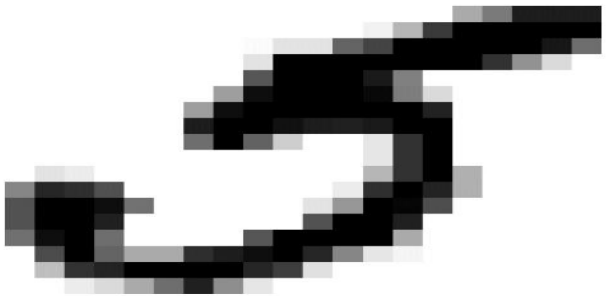
```
>>> X, y = mn["data"], mn["tar"]
>>> X.shape
(70000, 784)
>>> y.shape
(70000,)
```

70.000 aquí significa que hay 70.000 imágenes, y que cada imagen tiene más de 700 características: “784”. Porque, como puedes ver, cada imagen es de 28 x 28 pixeles, puedes imaginar que cada pixel es una característica.

Tomemos otro ejemplo del set de datos. Solo necesitarás tomar la característica de un ejemplo, luego hacer formaciones de 26 x 26, y luego mostrarlos usando la función imshow:

```
%matplotlib inline
import matplotlib
import matplotlib.pyplot as plt
yourDigit = X[36000]
Your_image = your_image.reshape(26, 26)
plt.imshow(Your_image, cmap = matplotlib.cm.binary,
interpolation="nearest")
plt.axis("off")
plt.show()
```

Como puedes ver en la siguiente imagen, se parece al número cinco, y podemos darle una etiqueta que nos diga que es el número cinco.



En la imagen siguiente, puedes ver clasificaciones más complejas de tareas del set de datos MNIST.

Además, debes crear un grupo de datos de prueba y hacerlo antes de que tus datos sean inspeccionados.

Los datos MNIST están divididos en dos sets, uno para capacitación y otro para prueba.

```
x_tr, x_tes, y_tr, y_te = x [:60000], x[60000:], y[:60000], y[60000:]
```

Vamos a jugar con tu set de capacitación como se muestra a continuación para hacer que la validación cruzada sea similar (sin ningún digito faltante)

```
Importa numpy como np

myData = np.radom.permutaion(50000)

x_tr, y_tr = x_tr[myData], y_tr[myData]
```

Ahora es hora de hacer suficientemente simple, trataremos solo de identificar un digito, por ejemplo el número 6. Este "detector de 6" será un ejemplo del clasificador binario, para distinguir entre 6 y no 6, entonces crearemos los vectores para esta tarea:

```
Y_tr_6 = (y_tr   == 6) // esto significa que será real para los 6s, y falso para otros números

Y_tes_6 = (Y_tes == 6)
```

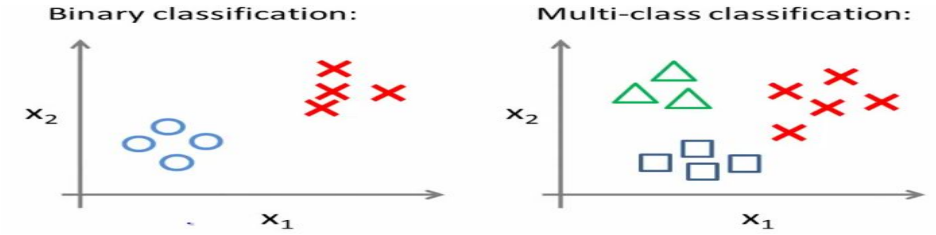
Después de esto podemos escoger un clasificador y capacitarlo. Comienza con el clasificador SGD (Stochastic Gradient Descent – Descenso de Gradiente Estocástico)

La clase de Scikit-Learn tiene la ventaja de manejar grandes sets de datos. En este Ejemplo, el SGD se encargará de los ejemplos se manera separada, como a continuación.

```
From sklearn.linear_model import SGDClassifier

mycl = SGDClassifier (random_state = 42)

mycl.fit(x_tr, y_tr_6)
```

Para usarlo para detectar 6

>>>mycl.prdict([any_digit)]



Medidas de Desempeño

Si quieres evaluar un clasificador, esto será más difícil que un regresor, entonces vamos a explicar cómo evaluar un clasificador.

En este ejemplo, vamos a usar validación cruzada para evaluar nuestro modelo.

From sklearn.model_selection import StratifiedKFold

From sklearn.base importar el clon

sf = StratifiedKFold(n=2, ran_state = 40)

para train_index, test_index in sf.split(x_tr, y_tr_6):

cl = clone(sgd_clf)

x_tr_fd = x_tr[train_index]

y_tr_fd = (y_tr_6[train_index])

x_tes_fd = x_tr[test_index]

y_tes_fd = (y_tr_6[test_index])

cl.fit(x_tr_fd, y_tr_fd)

y_p = cl.predict(x_tes_fd)

print(n_correct / len(y_p))

. Usamos la clase **StratifieldFold** para mostrar muestreo estratificado que produce pliegues que contienen una rotación por cada clase. Luego, cada iteración en el código creará un clon del clasificador para hacer predicciones en el pliegue de prueba. Y finalmente, contará el número de predicciones correctas y su proporción.

. Ahora usaremos la función cross_val_score para evaluar el clasificador SGDC por validación cruzada de pliegue k. La validación cruzada del pliegue k dividirá el set de capacitación en 3 pliegues, y luego hará predicción y evaluación de cada uno.

From sklearn.model_selection import cross_val_score

cross_val_score(sgd_clf, x_tr, y_tr_6, cv = 3, scoring = “accuracy”)

Obtendrás una proporción de precisión de “precisiones correctas” en todos los pliegues

Clasifiquemos cada clasificador en cada imagen in el no 6

From sklearn.base import BaseEstimator

class never6Classifier(BaseEstimator):

def fit(self, X, y=None):

  pass

def predict(self, x):

return np.zeros((len(X), 1), dtype=bool)

Examinemos la precisión de de este modelo con el siguiente código:

>>> never_6_cl = Never6Classifier()

>>> cross_val_score(never_6_cl, x_tr, y_tr_6, cv = 3, scoring = “accuracy”)

Output: array ([“num”, “num”, “num”])

Para la salida, obtendrás no menos de 90%: solo menos de 10% de las imágenes son 6s, así que siempre podemos imaginar que una imagen no es un 6. Estaremos en lo correcto cerca del 90% de las veces.

Ten en cuenta que la precisión no es la mejor medida de redimiento para los clasificadores, si estás trabajando con sets de datos sesgados

Matriz de Confusión

Hay un mejor método para evaluar el desempeño de tu clasificador: La Matriz de Confusión

Es fácil medir el rendimiento con la matriz de confusión, solo contando el número de veces de ejemplos de la clase X que son clasificados como clase Y, por ejemplo: para obtener el número de veces de clasificador de imagen de 6s con 2s, debes buscar en la 6ta fila y la 2da columna de la matriz de confusión

Calculemos la matriz de confusión usando la función cross_val_predict ().

From sklearn.model_selection import cross_Val_predict

y_tr_pre = cross_val_predict (sgd_cl, x_tr, y_tr_6, cv = 3)

Esta función, como la función cross_val_score(), realiza la validación cruzada del pliegue k, y además regresa predicciones en cada pliegue. Además regresa una predicción clara por cada ejemplo en tu set de capacitación.

Ahora estamos listos para obtener la matriz usando el siguiente código

from sklearn.metrics import confusion_matrix

confusion_matrix (y_tr_6, y_tr_pred)

Obtendrás una formación de 4 valores “números”.

Cada fila representa una clase en la matriz, y cada columna representa una clase predicha.

La primera fila es uno negativo: Que “contiene imágenes no 6”. Puedes aprender mucho de la matriz.

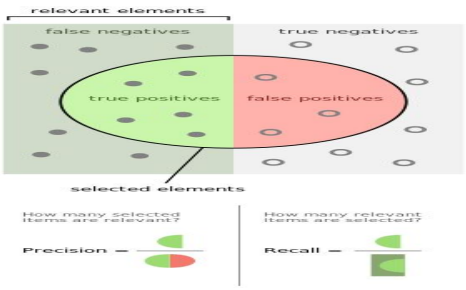
Pero además hay una con el que es interesante trabajar si te gustaría obtener la precisión de las predicciones positivas, el cual es la predicción del clasificador usando ecuación

Precision = (TP)/ (TP+FP)

TP = número de positivos verdaderos

FP = número de positivos falsos

Recall (índice de recuperación o exhaustividad) = (TP) /(TP+FN)
"sensibilidad": mide la proporción ejemplos positivos



Exhaustividad (también llamado índice de recuperación – Recall)

>>> from sklearn.metrics import precision_score, recall_score

>>> precision_score(y_tr_6, y_pre)

>>>recall_score(y_tr_6, y_tr_pre)

Es muy fácil combinar precisión y recall en un métrico, el cual es la puntuación F1.

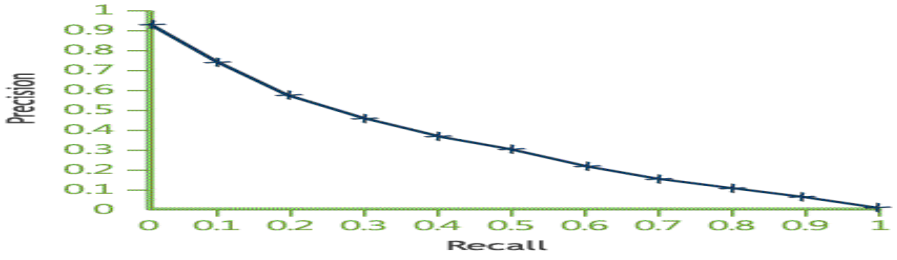
F1 es el medio de ambos, precisión y recall. Podemos calcular el puntaje de F1 con la siguiente ecuación:

F1 = 2 / ((1/precision) + (1)/recall)) = 2 * (precision * recall) / (precision + recall) = (TP) / ((TP) + (FN+FP)/2)

Para calcular la puntuación de F1, simplemente usa la siguiente función:

>>> from sklearn.metrics import f1_score

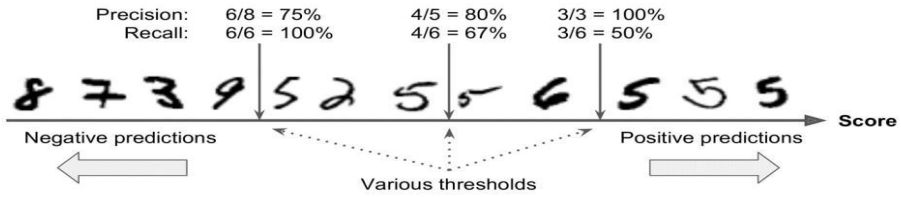
>>>f1_score (y_tr_6, y_pre)



Compensación de Exhaustividad (Recall)

Para llegar a este punto, debes echar un vistazo al clasificador SGD y cómo tomar decisiones de acuerdo a las clasificaciones. Calcula la puntuación basándose en la función de decisión, y luego compara la puntuación con el límite. Si es mayor que esta puntuación, asignará el ejemplo a la clase "positiva o negativa".

Por ejemplo, si el límite de decisión está en el centro, encontrarás 4 verdadero + al lado derecho del límite, y solo uno falso. Entonces la proporción de precisión será solo del 80%.



En Scikit-Learn, no puedes establecer un límite directamente. Necesitarás entrar a las puntuaciones de decisión, las cuales usan predicciones, y al llamar la función de decisión, ().

>>> y_sco = sgd_clf.decision_funciton([any digit])

>>> y_sco

>>> threshold = 0

>>>y_any_digit_pre = (y_sco > threshold)

En este código, el clasificador SGD contiene un límite, = 0, para regresar el mismo resultado que la función de predicción ().

>>> threshold = 20000

>>>y_any_digit_pre = (y_sco > threshold)

>>>y_any_digit_pre

Este código lo confirmará, cuando el límite se incrementa, el recall decrece.

y_sco = cross_val_predict (sgd_cl, x_tr, y_tr_6, cv =3, method="decision function)

Es hora de calcular toda precisión y recall posible para el límite al llamar la precision_recall_curve()function

from sklearn.metrics import precision_recall_curve

precisions, recalls, threshold = precision_recall_curve (y_tr_6, y_sco)

y ahora tracemos la precisión y el recall usando Matplotlib

def plot_pre_re(pre, re, thr):

plt.plot(thr, pre[:-1], “b—“, label = “precision”)

plt.plot(thr, re[:1], “g-“, label=”Recall”)

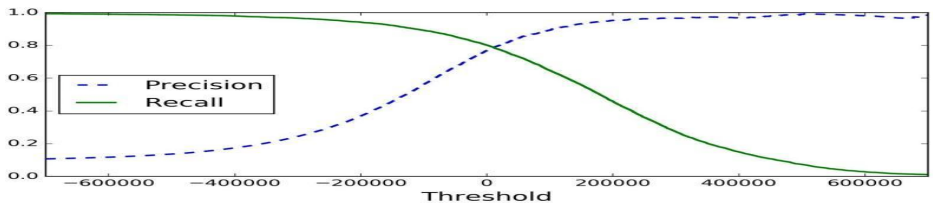
plt.xlabel(“Threshold”)

plt.legend(loc=”left”)

plt.ylim([0,1])

plot_pre_re(pre, re, thr)

plt.show



ROC

ROC son las siglas de receiver operating characteristic o característico operador receptor en español y es una herramienta usada con clasificadores binarios.

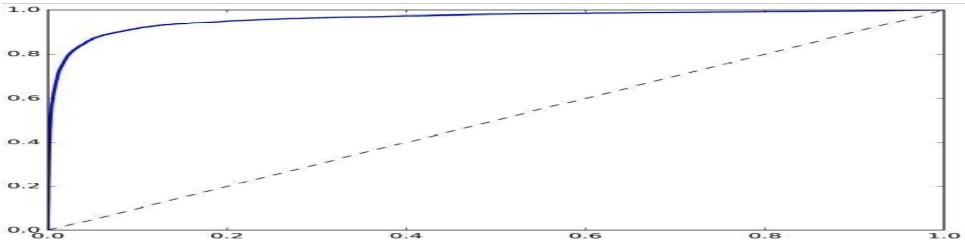
Esta herramienta es similar a la de la curva de recall (recall curve), pero no traza la precisión y el recall: traza la proporción positiva y la proporción negativa. También trabajarás con FPR, el cual es la proporción de muestras negativas. Lo puedes imaginar si es como (1—proporción negativa. Otro concepto es el TNR y es la especificidad. Recall = 1 – especificidad.

Ahora juguemos con la curva de ROC. Primero, necesitaremos calcular el TPR y el FPR, sólo con llamar la función roc-curve().

```
from sklearn.metrics import roc_curve

fp,tp, thers = roc_curve (y_tr_6, y_sco)
```

Después de eso, trazarás el FPR y el TPR con Matplotlib de acuerdo a las siguientes instrucciones.

```
 def_roc_plot (fp, tp, label=none):

plt.plot(fp, tp, linewidth=2, label = label)

plt.plot([0,1)], [0,1], “k--”)

plt.axis([0,1,0,1])

plt.xlabel(‘This is the false rate’)

plt.ylabel(‘This is the true rate’)

roc_plot (fp, tp)

plt.show
```



Clasificación de Clase Múltiple

Usamos clasificadores binarios para distinguir entre dos clases cualquieras, ¿pero qué pasa si quisieras distinguir entre más de dos clases?

Puedes hacer algo como el clasificador bosque aleatorio o clasificador Bayes, los cuales pueden comparar e entre más de dos. Pero, por otro lado, SVM (the support vector machine o máquina de vector de soporte en español) y los clasificadores lineales funcionan como clasificadores binarios.

Si quisieras desarrollar un sistema que clasifique imágenes de dígitos entre 12 clases (del 0 al 11) necesitarás capacitar 12 clasificadores binarios, y hacer uno por cada clasificador (tales como detector de 4, detector de 5, detector de 6 y así sucesivamente), y luego necesitarás obtener el DS, la "puntuación de decisión", de cada clasificador para la imagen. Luego escogerás el puntaje de clasificador más alto. A esto lo llamamos estrategia OvA: "one-versus-all" ("uno-contra-todos").

El otro método es capacitar un clasificador binario por cada par de dígitos; por ejemplo, 1 por 5s y 6s y otro por 5s y 7s. — llamamos a este método OvO, "one-versus-one" (uno contra uno") — para cantar cuantos clasificadores necesitarás, basándose en el número de clases que usan la siguiente ecuación: "N = número de clases".

N * (N-1)/2. Si quisieras usar esta técnica con el MNIST 10 * (10-1)/2, la salida será 45 clasificadores, "clasificadores binarios"

En Scikit-Learn, ejecutas OvA automáticamente cuando usas un algoritmo de clasificación binaria.

```
>>> sgd_cl.fit(x_tr, y_tr)
```

```
>>>sgd_cl.Predict([any-digit])
```

Adicionalmente, puedes llamar la decision_function () para regresar los puntajes "10 puntuaciones para una clase"

```
>>>any_digit_scores = sgd_cl.decision_function([any_digit])
```

```
>>> any_digit_scores
```

Array([“num”, “num”, “num”, “num”, “num”, “num”, “num”, “num”, “num” ,”num”]])

Capacitando un Clasificador de Bosque Aleatorio

```
>>> forest.clf.fit(x_tr, y_tr)

>>> forest.clf.predict([any-digit])

array([num])
```

Como puedes ver, capacitar un clasificador de bosque aleatorio con solo dos líneas de código es muy fácil.

El Scikit-Learn no ejecutó ningún función OvA u Ovo porque este tipo de algoritmos — “clasificadores de bosque aleatorio” — pueden automáticamente trabajar clases múltiples. Si quisieras echar un vistazo a la lista de posibilidades de clasificador, puedes llamar la función predict_oroba ()

```
>>> forest_cl.predict_proba([any_digit])

array([[0.1, 0, 0, 0.1, 0, 0.8, 0, 0, 0]])
```

El clasificador es muy preciso con su predicción, como lo puedes ver en la salida; hay 0.8 en el número índice 5.

Evaluemos el clasificador usando el función cross_val_score.

```
>>> cross_val_score(sgd_cl, x_tr, y_tr, cv=3, scoring = “accuracy”)

array([0.84463177, 0.859668, 0.8662669])
```

Obtendrás 84% más los pliegues. Cuando uses un clasificador aleatorio, obtendrás, en este caso, 10% por la puntuación de precisión. Ten en cuenta que mientras este valor sea mayor, mejor.

Análisis de Error

Primero que todo, cal desarrollar un proyecto de Machine Learning aleatorio:

1. Determina el problema;
2. Reúne tus datos;
3. Trabaja tus datos y explóralos;
4. Limpien los datos;
5. Trabaja con muchos modelos y escoge el mejor;
6. Combine tus modelos en la solución;
7. Muestra tu solución
8. Ejecuta y prueba tu sistema

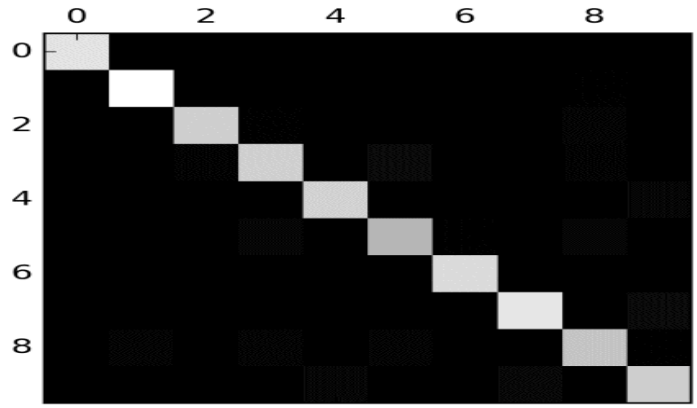
Primero, debes trabajar con la matriz de confusión y haz predicciones con la función cross-val. Luego llamarás la función de matriz de confusión:

```
>>> y_tr_pre = cross_val_prediciton(sgd_cl, x_tr_scaled, y_tr, cv=3)
>>> cn_mx  = confusion_matrix(y_tr, y_tr_pre)
>>> cn_mx
```

```
array([[5625, 2, 25, 8, 11, 44, 52, 12, 34, 6],
[ 2, 2415, 41, 22, 8, 45,  10, 10, 9],
[ 52, 43, 7443, 104, 89, 26, 87, 60, 166, 13],
[ 47, 46, 141, 5342, 1, 231, 40, 50, 141, 92],
[ 19, 29, 41, 10, 5366, 9, 56, 37, 86, 189],
[ 73, 45, 36, 193, 64, 4582, 111, 30, 193, 94],
[ 29, 34, 44, 2, 42, 85, 5627, 10, 45, 0],
[ 25, 24, 74, 32, 54, 12, 6, 5787, 15, 236],
[ 52, 161, 73, 156, 10, 163, 61, 25, 5027, 123],
[ 50, 24, 32, 81, 170, 38, 5, 433, 80, 4250]])
```

Plt.matshow(cn_mx, cmap=plt.cm.gray)

Plt.show()



Primero, debes dividir cada valor en la matriz por el número de imágenes en la clase, y luego comparas las tazas de error.

rw_sm = cn_mx.sum(axis=1, keepdims=True)

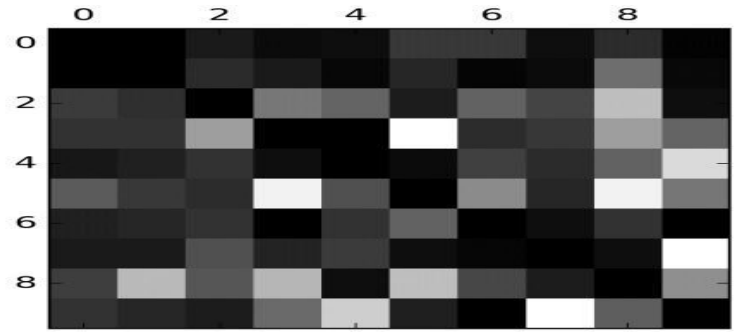
nm_cn_mx = cn_mx / rw_sum

El siguiente paso es **hacer todos los 0 en la diagonal,** y esos evitará que los errores sigan ocurriendo.

np.fill_diagonal (nm_cn_mx, 0)

plt.matshow(nm_cn_mx, cmap=plt.cm.gray)

plt.show()



Los errores son fáciles de ver en el esquema de arriba. Algo para mantener en mente es que las filas representan clases y las columnas representan valore predichos.

Clasificaciones de Etiquetas Múltiples

En el ejemplo de arriba cada clase tiene solo un ejemplo ¿Pero qué pasa si queremos asignar los ejemplos a múltiples clases? — Reconocimiento facial, por ejemplo. Supongamos que te gustaría encontrar más de una cara en la misma foto. Habrá una etiqueta por cada cara. Practiquemos con un ejemplo simple.

y_tr_big = (y_tr >= 7)

y_tr_odd = (y_tr %2 ==1)

y_multi = np.c [y_tr_big, y_tr_odd]

kng_cl = KNeighborsClassifier()

kng_cl.fit (x_tr, y_m,ulti)

en las instrucciones, hemos creado un y_mullti array que contiene dos etiquetas por cada imagen.

Y el primero contiene información si el digito es “grande” (8,9,.), y el segundo revisa si es raro o no.

Seguidamente, haremos una predicción usando el siguiente grupo de instrucciones.

>>>kng_cl.predict([any-digit])

Array([false, true], dataType=bool)

Verdadero (True) aquí significa que es raro y **Falso (false)** significa que no es grande.

Clasificación de Salidas Múltiples

A este punto, podemos cubrir el último tipo de tarea de clasificación, el cual es la clasificación de salidas múltiples.

Solamente es un caso de clasificación de etiquetas múltiples, pero cada etiqueta tendrá una clase múltiple. En otras palabras, tendrá más de un valor.

Aclarémoslo con este ejemplo, usando las imágenes de MNIST, y agregando un poco de sonido a la imagen con las funciones NumPy.

```
No = rnd.randint (0, 101, (len(x_tr), 785)))

No = rnd.randint(0, 101, (len(x_tes), 785))

x_tr_mo = x_tr + no

x_tes_mo = x_tes + no

y_tr_mo = x_tr

y_tes_mo = x_tes

kng_cl.fit(x_tr_mo, y_tr_mo)

cl_digit = kng_cl.predict(x_tes_mo[any-index]])

plot_digit(cl_digit)
```

EJERCISIOS

1. Construye un clasificador para el set de datos del MNIST. Intenta obtener más del 96% de precisión en la prueba de tu set.

2. Escribe un proyecto para cambiar una imagen del set de MNIST (derecha o izquierda) a dos pixeles.

3. Desarrolla tu propio clasificador anti-spam

- Descarga ejemplos de spam de Google
- Extrae el set de datos
- Divide los datos en capacitación para un set de prueba
- Escribe un programa para convertir cada correo electrónico en un vector de característica
- Juega con los clasificadores, e intenta construir el mejor posible, con altos valores de precisión y recall.

RESUMEN

En este capítulo, haz aprendido útiles conceptos nuevos e implementado muchos tipos de algoritmo de clasificaciones. Además haz trabajado nuevos conceptos como:

- ROC: el característico operador receptor, la herramienta con clasificadores binarios.

- Análisis de Error: Optimizando tus algoritmos.

- Como capacitar un clasificador de bosque aleatorio, usando la función de bosque en Scikit-Learn.

- Entendiendo la clasificación de salidas multiples.

- Entendiendo la clasificaciones de etiquetas múltiples

CAPITULO 3
COMO CAPACITAR UN MODELO

Después de trabajar con muchos modelos de Machine Learning y capacitar algoritmos, lo que parecen insondables cajas negras. Pudimos optimizar una regresión simple, además haz trabajado con clasificadores de imágenes. Pero desarrollamos estos sistemas sin entender lo que hay adentro y cómo funcionan, ahora necesitamos ir más profundo para que podamos entender cómo funcionan y entender los detalles de implementación.

Lograr un entendimiento profundo de estos detalles te ayudará con el modelo correcto y al escoger el mejor algoritmo de capacitación. Además, te ayudará con la depuración y análisis de error.

En este capítulo, trabajaremos con regresión polifónica, el cual es un modelo complejo que trabaja para sets de datos no lineales. Adicionalmente, trabajaremos con diferentes técnicas de regulaciones que reducen la capacitación que alienta sobreajustes.

Regresión Lineal

Como ejemplo, tomaremos l_S = θ0 + θ1 × GDP_per_cap. Esto es un modelo simple para una función lineal de la función de salida. (θ0 and θ1) son los parámetros del modelo.

En general, usarás un modelo lineal para hacer predicciones calculando un suma pesada de característica de salida y además un "tendencia" (bias) constante, como puedes ver en la siguiente ecuación.

$$\hat{y} = \theta_0 + \theta_1 x_1 + \theta_2 x_2 + \cdots + \theta_n x_n$$

. Y es el valor del predictor

. N representa las características

. X1 es el valor de las características

. Θj es el parámetro modelo de j theta

Además, podemos escribir la ecuación en una forma vectorizada, como en el siguiente ejemplo:

$$\hat{y} = h_\theta(\mathbf{x}) = \theta^T \cdot \mathbf{x}$$

. Θ es el valor que minimiza el costo

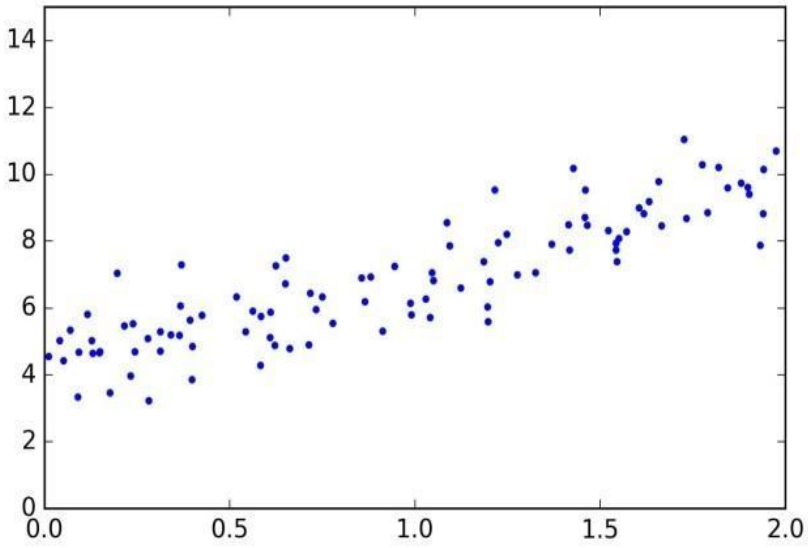
. Y contiene los valores y (1) a y (m)

Escribamos algunos códigos para practicar

Import numpy as np

V1_x = 2 * np.random.rand (100, 1)

V2_y = 4 + 3 * V1_x + np.random.randn (100, 1)



Después de eso, calcularemos el valor de Θ usando nuestra ecuación. Es tiempo de usar la función inv() desde nuestro módulo de algebra lineal de numpy (np.linalg) para calcular el inverso de cualquier matriz, y además, la función dot() para multiplicar nuestra matriz.

Value1 = np.c_[np.ones((100, 1)), V1_x]

myTheta = np.linalg.inv(Value1.T.dot(Value1)).dot(Value1.T).dot(V2_y)

>>>myTheta

Array([[num], [num]])

Esta función usa la siguiente ecuación — y = 4 + 3x + sonido “Gaussian” — para generar nuestros datos.

Ahora hagamos nuestra predicción.

>>>V1_new = np.array([[0],[2]])

>>>V1_new_2 = np.c_[np.ones((2,1)), V1_new]

>>>V2_predicit = V1_new_2.dot(myTheta)

>>>V2_predict

Array([[4.219424], [9.74422282]])

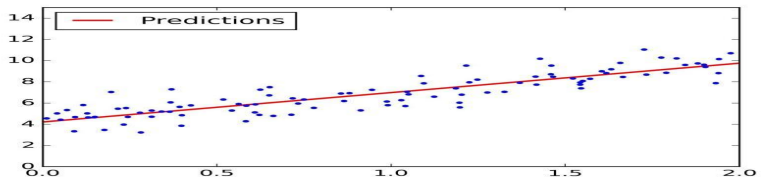
Ahora, es tiempo de trazar el modelo.

Plt.plot(V1_new, V2_predict, “r-“)

Plt.plot(V1_x, V2_y, “b.”)

Plt.axis([0,2,0,15])

Plt.show()



Complejidad Computacional

Con la fomula normal, podemos computar el inverso de M^T. M — esto es, una matriz n*n (n es el número de características). La complejidad de la inversión es algo como O(n^2.5) a O(n^3.2), el cual es basado en la implementación. Actualmente, **si haces el número de características el doble,** harás que el tiempo de la computación alcance entre 2^2.5 y 2^3.2.

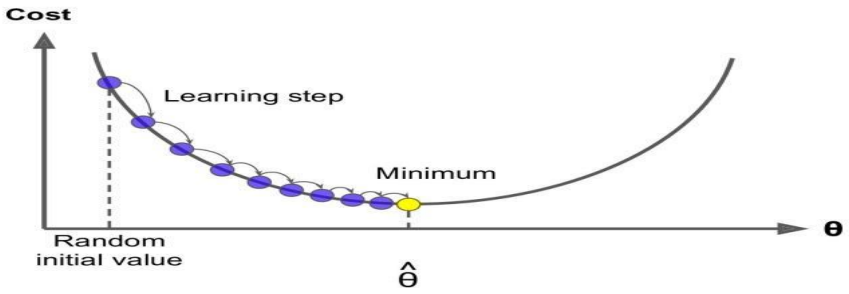
La gran noticia aquí es que la ecuación es una ecuación lineal. Esto quiere decir que fácilmente puede manejar grandes sets de capacitación y la memoria puede encajar.

Después de capacitar tus modelos, las predicciones no serán lentas, y la complejidad será simple, gracias al modelo lineal. Es hora de ir más profundo en los métodos de capacitar un modelo de regresión lineal, el cual siempre es usado cuando hay un gran número de características y ejemplos en la memoria.

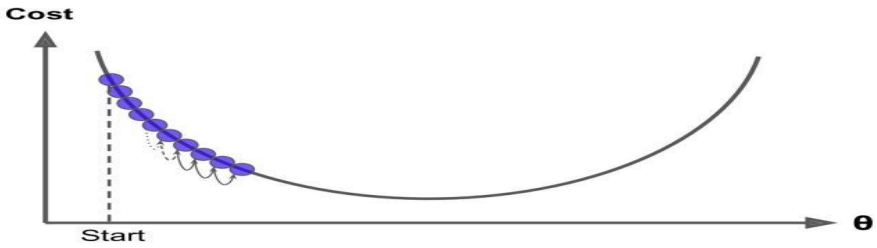
Descenso Gradiente

Este algoritmo es un algoritmo general que es usado para optimización y para proveer la solución óptima para varios problemas. La idea de este algoritmo es trabajar con los parámetros en una manera iterativa, para hacer la función de costo tan simple como sea posible.

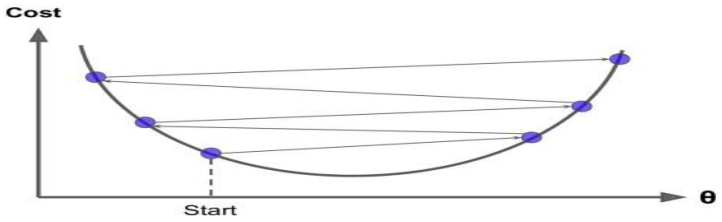
El algoritmo de descenso gradiente calculo el gradiente del error usando el parámetro theta, y trabaja con el método de descenso gradiente. Si el gradiente es igual a cero, alcanzarás el mínimo.



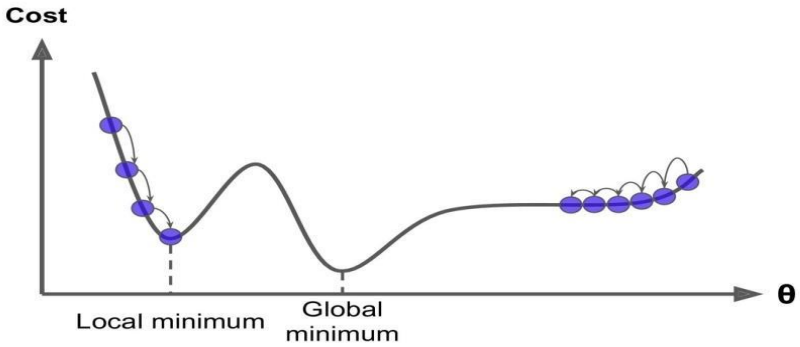
Además, debes tener en cuenta que el tamaño de los pasos es muy importante para el algoritmo, porque es muy pequeño – lo que significa que "la proporción de Machine Learning" es lenta – tomará mucho tiempo para cubrir todo lo que se necesita cubrir.



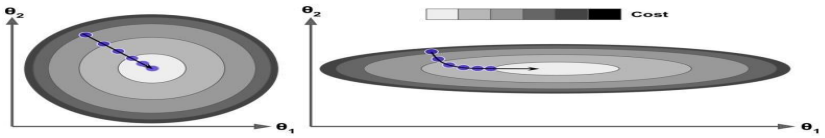
Pero cuando la taza de Machine Learning es alta, tomará poco tiempo para cubrir lo que se necesita y proveerá una solución óptima.



Al final, no siempre te conseguirás con que todos las funciones de costos son fáciles, como puedes ver, para además encontrarás funciones irregulares que dificultan conseguir una óptima solución. El problema ocurre cuando mínimo local y el mínimo global lucen de esta forma en la siguiente imagen.



Si asignas cualquiera a dos puntos cualquieras en tu curva, encontrarás que el segmento de la línea no se les unirá en la siguiente curva. Este costo de función se verá como un bol, el cual ocurrirá si las características tienen muchas escalas, como en la siguiente imagen.



Descenso Gradiente por Lote

Si quisieras implementar este algoritmo, primero debes calcular el gradiente del costo de tu función usando el parámetro theta. Si el valor del parámetro theta ha cambiado, necesitas saber la tasa de cambio de tu función de costo. Llamamos a este costo por un derivativo parcial.

Podemos calcular el derivativo parcial a través de la siguiente ecuación:

$$\frac{\partial}{\partial \theta_j} \text{MSE}(\theta) = \frac{2}{m} \sum_{i=1}^{m} (\theta^T \cdot \mathbf{x}^{(i)} - y^{(i)}) \, x_j^{(i)}$$

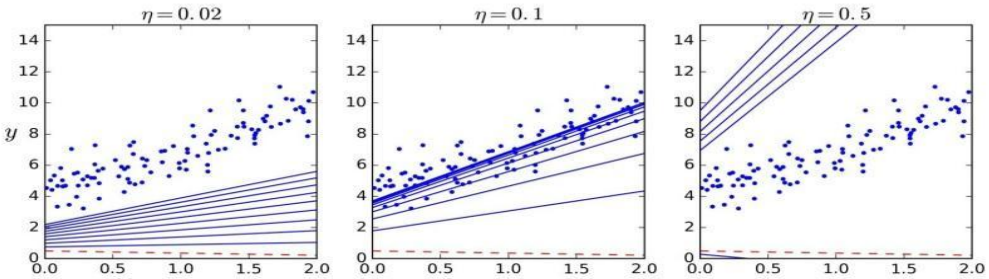
pero usaremos la siguiente ecuación para calcular el parcial derivativo y el vector gradiente juntos:

$$\nabla_\theta \, \text{MSE}(\theta) = \begin{pmatrix} \frac{\partial}{\partial \theta_0} \text{MSE}(\theta) \\ \frac{\partial}{\partial \theta_1} \text{MSE}(\theta) \\ \vdots \\ \frac{\partial}{\partial \theta_n} \text{MSE}(\theta) \end{pmatrix} = \frac{2}{m} \mathbf{X}^T \cdot (\mathbf{X} \cdot \theta - \mathbf{y})$$

Implementemos el algoritmo.

```
Lr = 1 # Lr for learning rate

Num_it = 1000 # number of iterations

L = 100

myTheta = np.random.randn (2,1)

for it in range(Num_it):

gr = 2/L * Value1.T.dot(Value1.dot(myTheta) – V2_y)

myTheta = myTheta – Lr * gr

>>> myTheta

Array([[num],[num]])
```

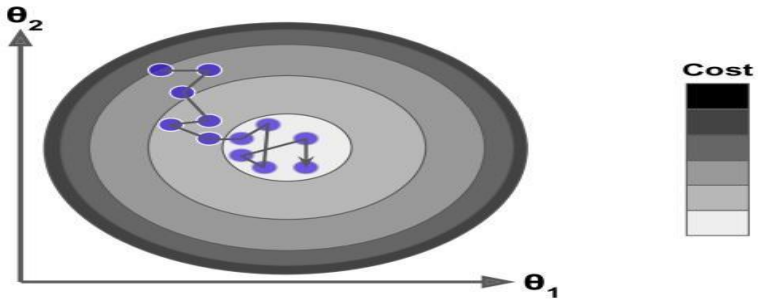
Si intentas cambiar el valor de la taza de Machine Learning, obtendrás diferentes formas, como en la siguiente imagen.


$\eta = 0.02$
$\eta = 0.1$
$\eta = 0.5$
y
0
2
4
6
8
10
12
14
0.0
0.5
1.0
1.5
2.0

Descenso Gradiente Estocástico

Encontraras un problema cuando uses el descenso gradiente por lote: necesita usar todo el set de capacitación para calcular el valor de cada paso, y que afectará la "velocidad" de desempeño.

Pero cuando uses el descenso gradiente estocástico, el algoritmo escogerá automáticamente un ejemplo para tu set de capacitación en cada paso, y luego calculará el valor. De esta forma, el algoritmo será más rápido que el descenso estocástico por lote, ya que no necesita usar todo el set para calcular el valor. Por otro lado, por lo aleatorio del proceso, será irregular en comparación con el algoritmo por lote.



Implementemos el algoritmo.

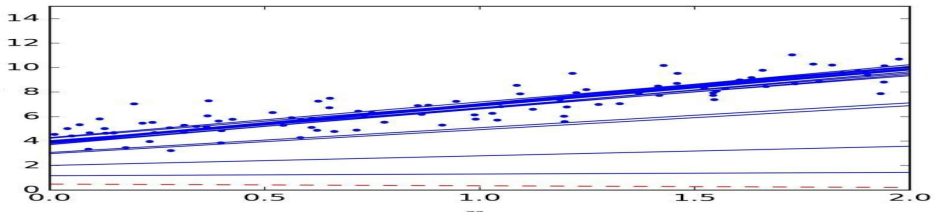
```
Nums = 50
L1, L2 = 5, 50
Def lr_sc(s):
    return L1 / (s + L2)
myTheta = np.random.randn(2,1)
for Num in range (Nums):
for l in range (f)
myIndex = np.random.randint(f)
V1_Xi = Value1[myIndex:myIndex+1]
V2_yi = V2_y[myIndex:myIndex+1]
gr = 2 * V1_xi.T.dot(V1_xi.dot(myTheta) – V2_yi)
```

Lr = lr_sc(Num * f + i)

myTheta = myTheta – Lr * gr

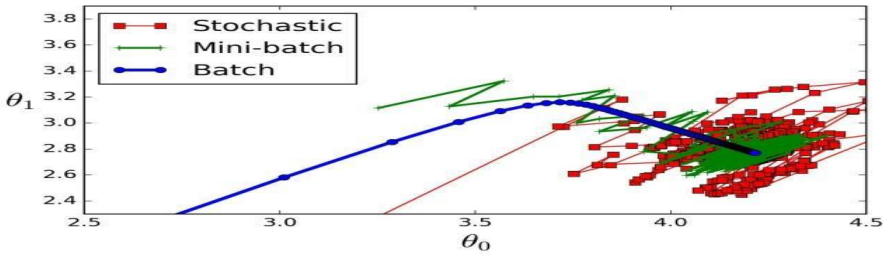
>>> myTheta

Array ([[num], [num]])



Mini Descenso Gradiente por Lote

Como ya conoces los algoritmos por lote y estocástico, este algoritmo será fácil de comprender y de trabajar. Como ya sabes, ambos algoritmos calculan el valor del gradiente, basados en todo el set de capacitación o solo un ejemplo. Sin embargo, el mini descenso gradiente por lote calculo su algoritmo basándose en sets pequeños y aleatorios, y se desempeña mejor que los otros dos algoritmos.

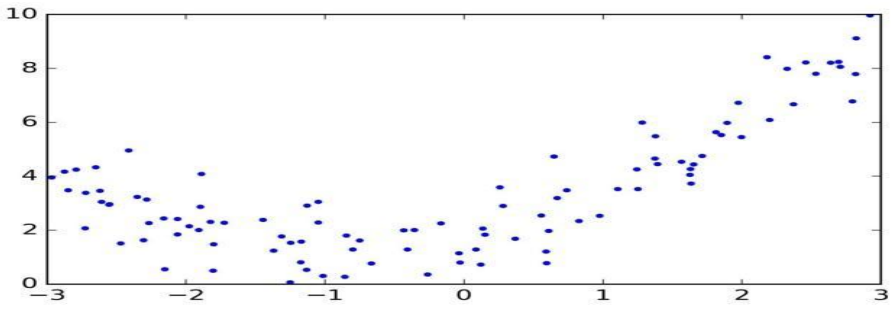


Regresión Polinomial

Usaremos esta técnica cuando trabajemos con datos más complejos, especialmente en el caso de los datos lineales y no lineales. Después de que hayamos agregado los poderes de cada característica, podemos capacitar el modelo con nuevas características. Esto es conocido como regresión polinomial.

Ahora escribamos parte del código.

```
L = 100
V1 = 6*np.random.rand(L, 1) – 3
V2 = 0.5 * V1**2 + V1 + 2 + np.random.randn(L, 1)
```

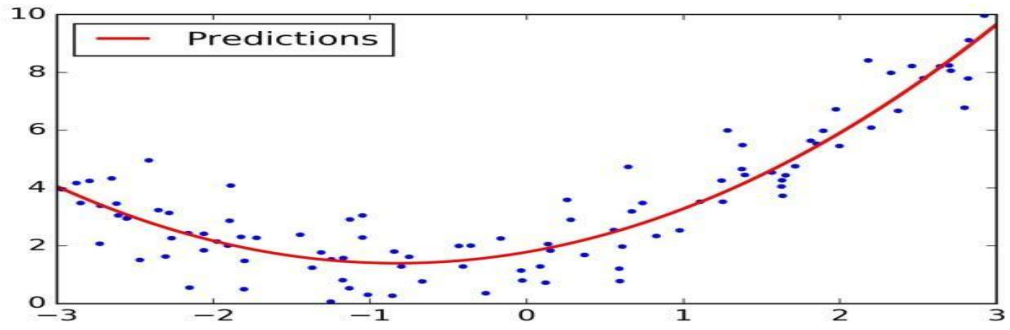


Como puedes ver, la recta jamás representara los datos en la manera más eficiente. Así que usaremos el método polinomial para trabajar en este problema.

```
>>>from sklearn.preprocession import PolynomialFeatures
>>>P_F = PolynomialFeatures(degree = 2, include_bias=False)
>>>V1_P = P_F.fit_transform(V1)
>>>V1[0]
Array([num])
>>>V1_P[0]
```

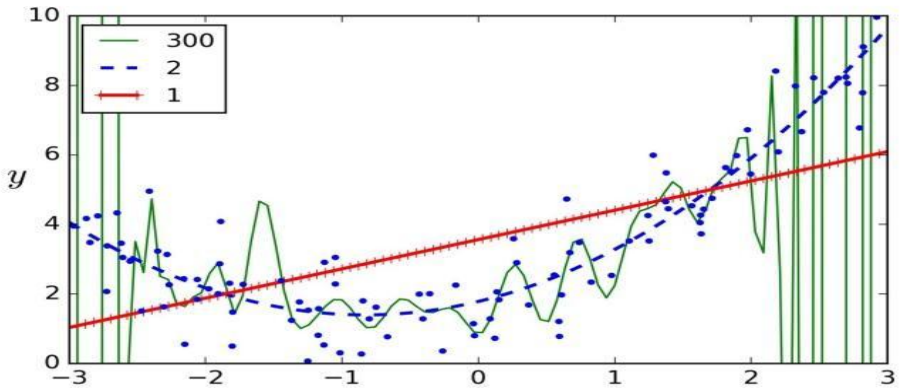
Ahora, hagamos que esto funcione apropiadamente con nuestros datos, y que cambie la línea recta.

```
>>> ln_r = LinearRegression()
>>>ln_r.fit(V1_P, V2)
>>>ln_r.intercept_, ln_r.coef
```



Curvas de Machine Learning

Asumamos que trabajarás con regresión polinomial, y quieres ajustar tus datos mejor que con la lineal. En la siguiente imagen, encontrarás un modelo de 300 grados. Además, podemos comparar el resultado final con otro tipo de regresión: “lineal normal”.



En la imagen de arriba, puedes ver el sobreajuste de los datos cuando estas usando el polinomial. Por otro lado, con el lineal, puedes ver la que data está siendo obviamente subajustada.

Modelos Lineales Regularizados

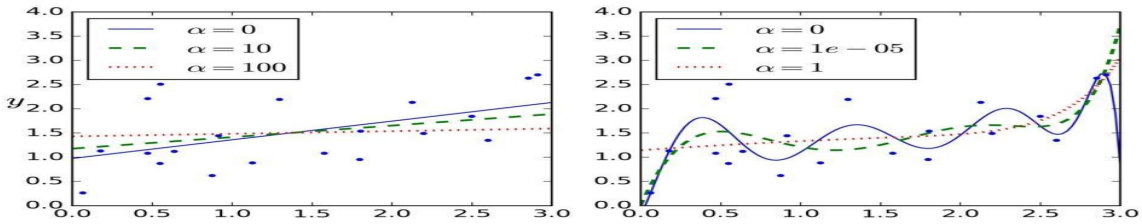
Hemos trabajo en el primero y segundo capítulo, en cómo reducir el sobreajuste regularizando un poco el modelo, por ejemplo, quisieras regularizar un modelo polinomial. En este caso, para resolver el problema, reducirás el número de grados.

Regresión Contraída

La regresión contraída es otra versión de la regresión lineal, pero, después de regularizarla y agregar peso a la función de costo, esto hace que los datos se ajusten, e incluso hace el peso del modelo tan simple como sea posible. Aquí está la función de costo de la regresión contraída:

$$J(\theta) = \text{MSE}(\theta) + \alpha \frac{1}{2} \sum_{i=1}^{n} \theta_i^2$$

Como ejemplo de la regresión contraída, solo echa un vistazo a las siguientes imágenes.



Regresión Lasso

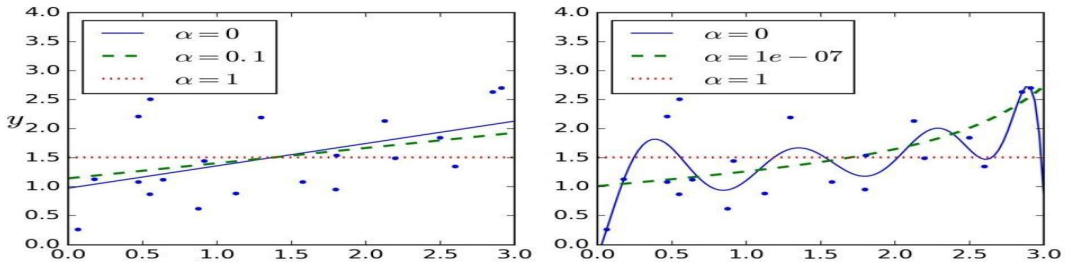
La regresión “Lasso” quiere decir regresión “Least Abosolute Shrinkage and Selection Operation” o regresión de reducción minima absoluta y operación de selección. Este es otro tipo de versión regularizada de regresión lineal.

Luce como la regresión contraída, pero con pequeñas diferencias en la ecuación, como en las siguientes figuras

La función de costo de la regresión lasso:

$$J(\theta) = \text{MSE}(\theta) + \alpha \sum_{i=1}^{n} | \theta_i |$$

Como puedes ver en la siguiente imagen, la regresión Lasso usa valores más pequeños que la contraída.



EJERCISIOS

1. Si tienes un set que contiene grandes números de características (millones de ellos), ¿Cuál algoritmo de regresión debes usar, y por que?

2. Si usas descenso gradiente por lote para trazar el error a cada periodo, y repentinamente la taza de errores de incrementa, ¿Cómo solucionaría este problema?

3. ¿Qué deberías hacer si notas que los erros de vuelven de mayor número cuando estas usando el método mini por lote? ¿Por qué?

4. De estos pares ¿Cuál método es el mejor?

¿Regresión contraída y regresión lineal?

¿Regresión Lasso y regresión contraída?

5. Escribe el algoritmo de descenso gradiente por lote.

RESUMEN

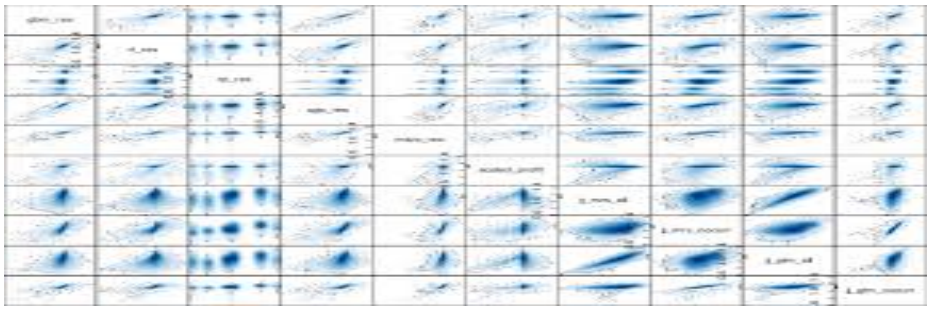
En este capítulo, has aprendido nuevos conceptos, y has aprendido como capacitar un modelo usando diferentes tipos de algoritmos. Además has aprendido cuando usar cada algoritmo, incluyendo los siguientes:

- Descenso Gradiente por Lote (Batch)
- Mini Descenso Gradiente por Lote (Mini-Batch)
- Regresión Polinomial
- Modelos Lineales Regularizados
    - Regresión Contraída
    - Regresión Lasso

Adicionalmente, ya conoces el significado de ciertos términos: Regresión Lineal, Complejidad Computacional y Descenso Gradiente.

CAPITULO 4

COMBINACIONES DE DIFERENTES MODELOS



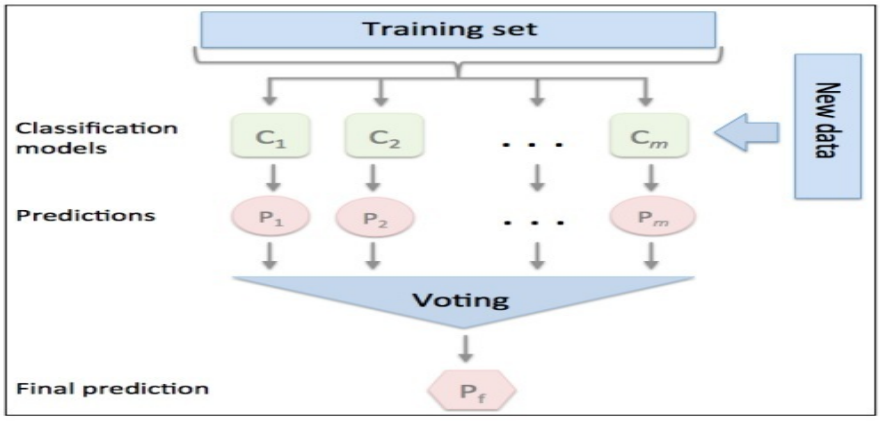
Clasificadores Arbóreos

La siguiente imagen ilustrará la definición de un objetivo general de reunir funciones lo cual es solo unir diferentes clasificadores en un clasificador único que tenga un mejor desempeño de generalización que cada clasificador individual por sí solo.

Como ejemplo, asumamos que reuniste predicciones de muchos expertos. Los métodos de conjunto nos permitirán unir estas predicciones por muchos expertos para obtener una predicción que sea más apropiada y robusta que las predicciones de un experto individual. Como podrás ver más adelante en esta parte, hay muchos métodos diferentes para crear un conjunto de clasificadores. En esta parte, introduciremos una percepción básica de cómo funcionan los conjuntos y por qué son típicamente reconocidos por producir un buen desempeño de generalización.

En esta parte, trabajaremos con el método de conjunto más popular que usa el principio de voto mayoritario. Muchos votos simplemente significan que escogemos la etiqueta que ha sido predicha por la mayoría de clasificadores; eso es, recibidos más que el 50% de los votos. Como ejemplo, el término aquí es como si los votos se refirieran solamente a ajustes de clase binaria. Sin embargo, no es difícil generar el principio de mayoría de votos a ajustes de clases múltiples, el cual es llamado voto de pluralidad. Después de eso, escogeremos la etiqueta de clase que recibió la mayor cantidad de votos. El siguiente diagrama ilustra los conceptos de voto por mayoría y voto por pluralidad para un conjunto de 10 clasificadores donde cada símbolo único (triangulo, círculo, y cuadrado) representa una etiqueta única:

Usando el set de capacitación, comenzados muchos clasificadores diferentes (C C 1, , … m). Basándose en el método, el conjunto puede ser construido de muchos algoritmos de clasificación; por ejemplo, arboles de decisión, machinas de vector de soporte, clasificadores de regresión logística, a así sucesivamente. De hecho, puedes usar la misma base de algoritmo de clasificación ajustándose a diferentes subsets del set de capacitación. Un ejemplo de este método puede ser el algoritmo de bosque aleatorio, el cual une muchas formas conjunto de decisión usando el voto mayoritario.



Para predecir una etiqueta de clase a través de una mayoría simple o voto de pluralidad, combinamos las etiquetas de clases predichas de cada clasificador individual C j y selecciona la etiqueta de clase yˆ que recibió más votos:

$$\hat{y}_m = mode\{C_1(\boldsymbol{x}), C_2(\boldsymbol{x}), \ldots, C_m(\boldsymbol{x})\}$$

Por ejemplo, en tareas de clasificación binaria donde class1 1 = − and class2 1 = +, podemos escribir la predicción de mayoría de votos.

Para ilustrar por que los métodos de conjuntos pueden trabajar mejores quelos clasificadores individuales por sí solos, apliquemos los conceptos simples de combinatoria. Para el siguiente ejemplo, asumimos que todos los clasificadores basados en n para una tarea de clasificación binaria tienen una taza de errores igual, ε. Adicionalmente, asumimos que los clasificadores, son independientes y que las tazas de errores no están correlacionadas. Como puedes ver, podemos simplemente explicar las estadísticas de errores de un conjunto de clasificadores base como una probabilidad.

Función de masa de una distribución binomial:

Aquí, n. k es el coeficiente binomial que n escoge. Como puedes ver, puedes calcular la probabilidad que la predicción del conjunto es incorrecta. Ahora, echemos un vistazo a un ejemplo más concreto de 11 clasificadores de base (n =11) con una taza de error de 0.25(ε = 0.25):

Puedes notar que la taza de error del conjunto (0.034) es más pequeño que la taza de error de la taza de cada clasificador individual (0.25) si todas las asunciones son reunidas. Nota que en esta imagen simplificada, una división del 50-50 por un número par de clasificadores n es tratado como un error, mientras estos solo es cierto la mitad de las veces. Para comparar un clasificador de conjunto tan ideal con un clasificador base sobre un rango de diferentes bases de tazas de errores, implementemos la probabilidad de función de masa en Python:

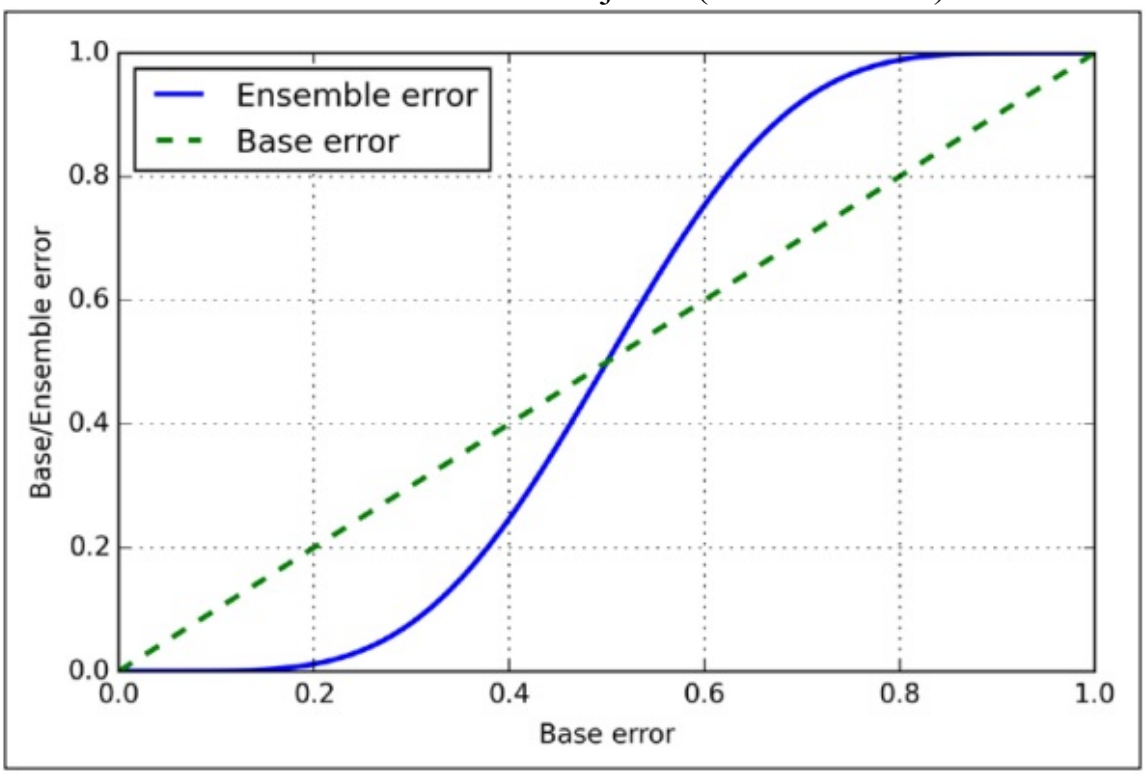
```
>>> import math
>>> def ensemble_error(n_classifier, error):
... q_start = math.ceil(n_classifier / 2.0)
... Probability = [comb(n_class, q) *
... error**q *
... (1-error)**(n_classifier - q)
... for q in range(q_start, l_classifier + 2)]
... return sum(Probability)
>>> ensemble_error(n_classifier=11, error=0.25)
0.034327507019042969
```

Escribamos algunos códigos para computar las tazas para que los diferentes errores visualicen la relación entre conjunto y errores base en una gráfica de línea:

```
>>> import numpy as np
>>> error_range = np.arange(0.0, 1.01, 0.01)
>>> en_error = [en_er(n_classifier=11, er=er)
... for er in er_range]
>>> import matplotlib.pyplot as plt
>>> plt.plot(er_range, en_error,
... label='Ensemble error',
```

```
... linewidth=2)
>>> plt.plot(er_range, er_range,
... ls='--', label='B_ er',
... linewidth=2)
>>> plt.xlabel('B_ er')
>>> plt.label('B/En_er')
>>> plt.legend(loc='upper left')
>>> plt.grid()
>>> plt.show()
```

Como podemos ver en el trazo resultante, la probabilidad de error de un conjunto siempre es mejor que el error de un clasificador base individual siempre y cuando los clasificadores base se desempeña mejor que la suposición aleatoria ($\varepsilon < 0.5$). Debes notar que el eje y represente el error base así como también el error del conjunto (línea continua):



Implementando un Clasificador por Mayoría Simple

Como vimos en la introducción al Machine Learning por unión en la última sección, trabajaremos con una capacitación de calentamiento y luego desarrollaremos un clasificador simple para mayoría de votos in programación Python. Como puedes ver, el siguiente algoritmo trabajará con ajustes de clase múltiple a través del voto pluralizado; usarás el término voto por mayoría para simplicidad como es frecuentemente hecho en literatura.

En el programa siguiente, desarrollaremos y además combinaremos diferentes programas de clasificación asociados a diferentes pesos para confianza. Nuestra meta es construir un meta-clasificador más fuerte que balancee las debilidades de los clasificadores individuales en un set de datos en particular. En términos matemáticos más precisos, podemos escribir la mayoría de votos pesados.

Para traducir el concepto de mayoría de votos pesados en código Python, podemos usar las funciones convenientes argmax y bincount de NumPy:

```
>>> import numpy as np
>>> np.argmax(np.bincount([0, 0, 1],
... weights=[0.2, 0.2, 0.6]))
1
```

Aquí, pij es la probabilidad predicha del clasificador jth para la etiqueta de clase i. Para continuar con nuestro ejemplo anterior, asumamos que tenemos un problema de clasificación binaria con las etiquetas de clase $i \in \{ \} 0,1$ y un conjunto de tres clasificadores $C j (j \in \{ \} 1,2,3)$. Asumamos que el clasificador $C j$ regresa las siguientes probabilidades de membrecía de clase para un muestra x en particular:

$C1 ( ) x \rightarrow [ ] 0.9,0.1 , C2 ( ) x \rightarrow [ ] 0.8,0.2 , C3 ( ) x \rightarrow [ ] 0.4,0.6$

Para implementar la mayoría de votos pesados basándose en la probabilidad de clases, podemos hacer uso de NumPy nuevamente usando numpy.average y np.argmax:

```
>>> ex = np.array([[0.9, 0.1],
... [0.8, 0.2],
... [0.4, 0.6]])
>>> p = np.average(ex, axis=0, weights=[0.2, 0.2, 0.6])
>>> p
array([ 0.58, 0.42])
>>> np.argmax(p)
0
```

Al colocar todo junto, ahora implementemos un clasificador de votos por mayoría en Python:

```
from sklearn.base import ClassifierMixin
from sklearn.pre_processing import Label_En
from sklearn.ext import six
from sklearn.ba import clone
from sklearn.pipeline import _name_estimators
import numpy as np
import operator
class MVClassifier(BaseEstimator,
ClassifierMixin):
""" Un clasificador de conjunto de voto por mayoría
Parámetros
----------
cl : array-like, shape = [n_classifiers]
Diferentes clasificadores por el voto en conjunto: str, {'cl_label', 'prob'}
Default: 'cl_label'
```

Si 'cl_label' las predicciones son basadas en el argmax de etiquetas de clase. Elif 'prob', el arg del total de lso problemas es usado para predecir la etiqueta de clase (recomendado para clasificadores calibrados).

w : arr-like, s = [n_cl]

Opcional, por default: Ninguno.

Si una lista de valores de `int` o `float`son otorgados, los clasificadores son pesados por """

def __init__(s, cl,

v='cl_label', w=None):

s.cl = cl

s.named_cl = {key: value for

key, value in

_name_estimators(cl)}

s.v = v

s.w= w

def fit_cl(s, X, y):

""" Fit_cl.

Parametros

X : {array-like, sparse matrix},

s = [n_samples, n_features]

Matrix of training samples.

y : arr_like, sh = [n_samples]

El vector de las etiquetas de clase objetivas.

Regresa

---

s : object

"""

Use LabelEncoder to ensure class labels start

with 0, which is important for np.argmax

call in s.predict

s.l_ = LabelEncoder()

s.l_.fit(y)

s.cl_ = self.lablenc_.classes_

s.cl_ = []

for cl in s.cl:

fit_cl = clone(cl).fit(X,

s.la_.transform(y))

s.cl_.append(fit_cl)

Regresa

Agregué muchos comentarios al código para entender mejor las partes individuales. Sin embargo, antes de que implementemos los métodos restantes, tomemos un pequeño descanso y discutamos algunos de los códigos que puedan parecer confusos al comienzo. Usamos las clases padres BaseEstimator y ClassifierMixin para obtener un poco de funcionalidad base sin costo, incluyendo los métodos get_params y set_params para ajustar y regresar los parámetros de clasificador así como también el método de puntuación para calcular la precisión de predicción, respectivamente. Además, puedes notar que implementamos seis para hacer el clasificador de voto mayoritario compatible con Python 2.7.

A continuación, agregaremos el método de predicción para predecir la etiqueta de clase a través de la mayoría de votos basados en las etiquetas de clases si inicializamos un nuevo objeto de clasificador de voto mayoritario con vote='classlabel'. Alternativamente, podremos inicializar el clasificador en conjunto con vote='probability' para predecir la etiqueta de clase basada en las probabilidades de membrecía de clase. Aún más, también agregaremos un método predict_proba para regresar las probabilidades

promedias, lo cual es muy útil para computar el área característica operadora receptora bajo la curva (ROC AUC).

```
def pre(s, X):
""" Etiquetas Pre Clases para X
Parámetros
----------
X : {arr-like, spar mat},
Sh = [n_samples, n_features]
Matriz de ejemplos de capacitación
Regresa
----------
ma_v : arr-like, sh = [n_samples]
Etiquetas de clase predichas
"""
if se.v == 'probability':
ma_v = np.argmax(spredict_prob(X),
axis=1)
else: # 'cl_label' v
predictions = np.asarr([cl.predict(X)
for cl in
s.cl_]).T
ma_v = np.ap_al_ax(
lambda x:
np.argmax(np.bincount(x, weights=s.w)),
axis=1,
arr=predictions)
```

ma_v = s.l_.inverse_transform(ma_v)

return ma_v

def predict_proba(self, X):

""" Prediction for X.

Parámetros

X : {arr-like, sp mat},

sh = [n_samples, n_features]

Vectores de capacitación, donde n_samples es el número de muestras y n_features número de características.

Regresa

---

av_prob : array-like,

sh = [n_samples, n_classes]

Probabilidad promedio pesada por cada muestra de clase.

"""

probs = np.asarr([cl.predict_prob(X)

for cl in s.cl_])

av_prob = np.av(probs,

axis=0, weights=s.w)

return av_prob

def get_ps(self, deep=True):

""" Obtén nombres de parámetros de clasificador para GridSearch"""

if not deep:

return super(MVC,

self).get_ps(deep=False)

```
else:
ou = s.n_cl.copy()
for n, step in\
six.iteritems(s.n_cl):
for k, value in six.iteritems(
step.get_ps(deep=True)):
ou['%s__%s' % (n, k)] = value
regresa ou
```

Combinando diferentes algoritmos para la clasificación con mayoría de votos

Ahora, es tiempo de colocar el MVC que implementamos en la sección anterior en acción. Primero debes preparar el set de datos que puedas probarlo. Ya que ya estamos familiarizados con técnicas para cargar sets de datos desde los archivos SCV, tomaremos un atajo y cargaremos el set de datos Iris desde el modelo de set de datos de Scikit-Learn.

Además, solo seleccionaremos dos características, sepal width (anchura sepal) y petal length (largura petal), para hacer la tarea de clasificación más desafiante. Aunque nuestro clasificador de voto mayoritario, o MVC, generaliza los problemas de clase múltiple, solamente clasificaremos ejemplos flower de dos clases. Ir-Versicolor y Ir-Virginica, para computar el ROC AUC. El cocido es el siguiente:

```
>>> import sklearn as sk

>>> import sklearn.cross_validation as cv

>>> ir = datasets.load_ir()

>>> X, y = ir.data[50:, [1, 2]], ir.target[50:]

>>> le = LabelEncoder()

>>> y = le.fit_transform(y)
```

Seguidamente, dividimos las muestras Iris en 50 de capacitación y 50% de datos de prueba:

```
>>> X_train, X_test, y_train, y_test =\

... train_test_split(X, y,

... test_size=0.5,

... random_state=1)
```

Usando los datos de prueba de capacitación, ahora capacitaremos tres clasificadores diferentes – una clasificación de regresión logística, un clasificador de árbol de decisión y un clasificador de vecinos más cercanos k (k-nearest neighbors o KNN) – y echaremos un vistazo a su desempeño

individual a través de una validación cruzada 10 en el set de datos de capacitación antes de que los unamos en un conjunto:

import the following

sklearn.cross_validation

sklearn.linear_model

sklearn.tree

sklearn.pipeline

Pipeline

numpy as np

```
>>> clf1 = LogisticRegression(penalty='l2',
... C=0.001,
... random_state=0)
>>> clf2 = DTCl(max_depth=1,
... criterion='entropy',
... random_state=0)


>>> cl = KNC(n_nb=1,
... p=2,
... met='minsk')
>>> pipe1 = Pipeline([['sc', StandardScaler()],
... ['clf', clf1]])
>>> pipe3 = Pipeline([['sc', StandardScaler()],
... ['clf', clf3]])
>>> clf_labels = ['Logistic Regression', 'Decision Tree', 'KNN']
>>> print('10-fold cross validation:\n')
>>> for clf, label in zip([pipe1, clf2, pipe3], clf_labels):
```

```
... sc = crossVSc(estimator=clf,
>>> X=X_train,
>>> y=y_train,
>>> cv=10,
>>> scoring='roc_auc')
>>> print("ROC AUC: %0.2f (+/- %0.2f) [%s]"
... % (scores.mean(), scores.std(), label))
```

Las salidas que fueron recibidas, como se muestra en la siguiente retazo, muestra que los desempeños predictivos de los clasificadores individuales son casi iguales:

10-fold cross validation (validación de 10 pliegues cruzados):

ROC AUC: 0.92 (+/- 0.20) [Logistic Regression]

ROC AUC: 0.92 (+/- 0.15) [Decision Tree]

ROC AUC: 0.93 (+/- 0.10) [KNN]

Podrías preguntarte por qué capacitamos la regresión logística y el clasificador KNN como parte de la tubería. La causa aquí es que, como dijimos, la regresión logística y los algoritmos KNN (usando la distancia métrica euclidea) no son invariables en escala en contraste con el árbol de decisiones. Sin embargo, las ventajas de Ir son todas medidas en la misma escala; es un buen habito trabajar con características estandarizadas.

Ahora prosigamos a una parte más emocionante y combinemos los clasificadores individuales por regla de mayoría de voto en nuestro M_V_C:

```
>>> mv_cl = M_V_C(
... cl=[pipe1, clf2, pipe3])
>>> cl_labels += ['Majority Voting']
>>> all_cl = [pipe1, clf2, pipe3, mv_clf]
>>> for cl, label in zip(all_clf, clf_labels):
```

```
... sc = cross_val_score(est=cl,
... X=X_train,
... y=y_train,
... cv=10,
... scoring='roc_auc')
... % (scores.mean(), scores.std(), label))
R_AUC: 0.92 (+/- 0.20) [Logistic Regression]
R_AUC: 0.92 (+/- 0.15) [D_T]
R_AUC: 0.93 (+/- 0.10) [KNN]
R_AUC: 0.97 (+/- 0.10) [Majority Voting]
```

Adicionalmente, la salida del MVC ha sustancialmente mejorado sobre el clasificador individual en la evaluación de la validación cruzada de 10 pliegues.

Clasificador

En esta parte, vas a computar la curva R_C con el set de prueba para revisar si el clasificador_MV generaliza bien los datos no visibles. Debemos recordar que el set de prueba no serpa utilizado para la selección de modelo; la única meta es reportar un estimado de precisión de un sistema clasificador. Echemos un vistazo a los métricos de importación.

```
 import roc_curve from sklearn.metrics import auc

cls = ['black', 'orange', 'blue', 'green']

ls = [':', '--', '-.', '-']

for cl, label, cl, l \

... in zip(all_cl, cl_labels, cls, ls):

... y_pred = clf.fit(X_train,

... y_train).predict_proba(X_test)[:, 1]

... fpr, tpr, thresholds = rc_curve(y_t=y_tes,

... y_sc=y_pr)

... rc_auc = ac(x=fpr, y=tpr)

... plt.plot(fpr, tpr,

... color=clr,

... linestyle=ls,

... la='%s (ac = %0.2f)' % (la, rc_ac))

>>> plt.lg(lc='lower right')

>>> plt.plot([0, 1], [0, 1],

... linestyle='--',

... color='gray',

... linewidth=2)

>>> plt.xlim([-0.1, 1.1])

>>> plt.ylim([-0.1, 1.1])
```

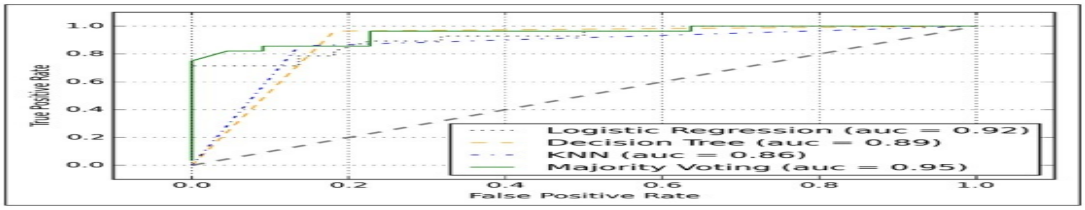
>>> plt.grid()

>>> plt.xlb('False Positive Rate')

>>> plt.ylb('True Positive Rate')

>>> plt.show()

Como podemos ver en el ROC resultante, el clasificador en conjunto además se desempeña bien en el set de prueba (ROC AUC = 0.95), mientras que el clasificador KNN parece estar sobreajustando los datos de capacitación (capacitación o training ROC AUC = 0.93, prueba o test ROC AUC = 0.86):



Solo escoges las características de las tareas de clasificación. Será interesante mostrar cómo la región de decisión del clasificador en conjunto realmente luce. Aunque no es necesario estandarizar las características de capacitación antes de que encaje el modelo ya que nuestra tubería de regresión logística y KNN automáticamente se encargaran d esto, harás el set de capacitación para que las regiones del árbol de decisión estará en la misma escala por propósitos de visualización.

Echemos un vistazo:

>>> sc = SS()

X_tra_std = sc.fit_transform(X_train)

From itertools import product

x_mi= X_tra_std[:, 0].mi() - 1

x_ma = X_tra_std[:, 0].ma() + 1

y_mi = X_tra_std[:, 1].mi() - 1

y_ma = X_tra_std[:, 1].ma() + 1

xx, yy = np.meshgrid(np.arange(x_min, x_max, 0.1),

```
... np.arange(y_mi, y_ma, 0.1))
 f, axarr = plt.subplots(nrows=2, ncols=2,
sharex='col',
sharey='row',
figze=(7, 5))
for ix, cl, tt in zip(product([0, 1], [0, 1]),
all_cl, cl_lb):
... cl.fit(X_tra_std, y_tra)
... Z = clf.predict(np.c_[xx.ravel(), yy.ravel()])
... Z = Z.resh(xx.shape)
... axarr[idx[0], idx[1]].contou(_xx, _yy, Z, alph=0.3)
... axarr[idx[0], idx[1]].scatter(X_tra_std[y_tra==0, 0],
... X_tra_std[y_tra==0, 1],
... c='blue',
... mark='^',
... s=50)
... axarr[idx[0], idx[1]].scatt(X_tra_std[y_tra==1, 0],
... X_tra_std[y_tra==1, 1],
... c='red',
... marker='o',
... s=50)
... axarr[idx[0], idx[1]].set_title(tt)
>>> plt.text(-3.5, -4.5,
... z='Sl wid [standardized]',
... ha='center', va='center', ftsize=12)
>>> plt.text(-10.5, 4.5,
```

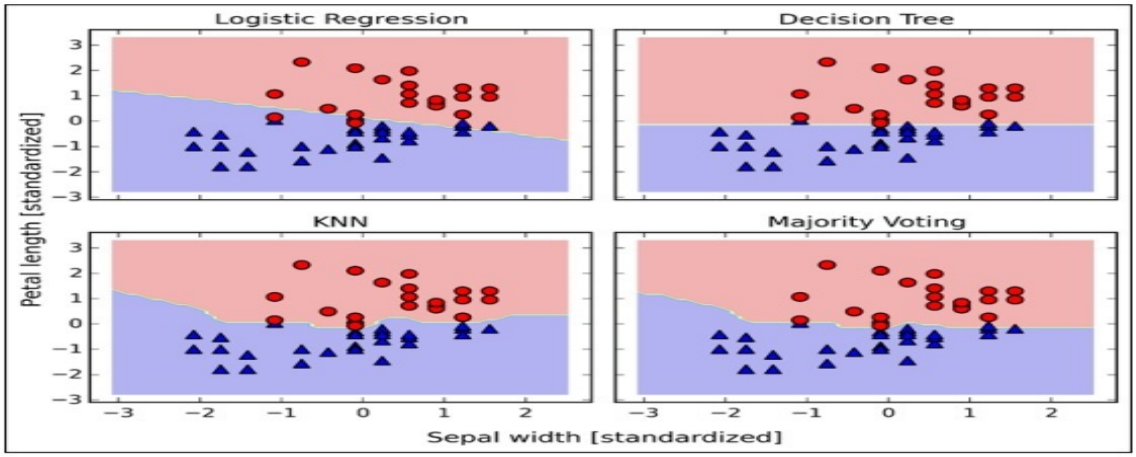
... z='P_length [standardized]',

... ha='center', va='center',

... f_size=13, rotation=90)

>>> plt.show()

Interesantemente, pero además como es esperado, las regiones de decisión del clasificador en conjunto parece ser un hibrido de las regiones de decisiones de los clasificadores individuales. A primera, los límites de la decisión de mayoría de voto se parecen mucho al límite de decisión del clasificador KNN. Sin embargo, podemos ver que es ortogonal al eje y para sepal width ≥1, así como el muñón del árbol de decisión:



Antes de que aprendas a afinar los parámetros del clasificador individual, llamemos el método get_ps para encontrar una idea esencial de cómo podemos entrar a los parámetros individuales dentro de un objeto GridSearch:

>>> mv_clf.get_params()

{'decisiontreeclassifier': DecisionTreeClassifier(class_weight=None,

criterion='entropy', max_depth=1,

max_features=None, max_leaf_nodes=None, min_samples_

leaf=1,

min_samples_split=2, min_weight_fraction_leaf=0.0,

random_state=0, splitter='best'),
'decisiontreeclassifier__class_weight': None,
'decisiontreeclassifier__criterion': 'entropy',
[...]
'decisiontreeclassifier__random_state': 0,
'decisiontreeclassifier__splitter': 'best',
'pipeline-1': Pipeline(steps=[('sc', StandardScaler(copy=True, with_
mean=True, with_std=True)), ('clf', LogisticRegression(C=0.001, class_
weight=None, dual=False, fit_intercept=True,
intercept_scaling=1, max_iter=100, multi_class='ovr',
penalty='l2', random_state=0, solver='liblinear',
tol=0.0001,
verbose=0))]),
'pipeline-1__clf': LogisticRegression(C=0.001, class_weight=None,
dual=False, fit_intercept=True,
intercept_scaling=1, max_iter=100, multi_class='ovr',
penalty='l2', random_state=0, solver='liblinear',
tol=0.0001,
verbose=0),
'pipeline-1__clf__C': 0.001,
'pipeline-1__clf__class_weight': None,
'pipeline-1__clf__dual': False,
[...]
'pipeline-1__sc__with_std': True,
'pipeline-2': Pipeline(steps=[('sc', StandardScaler(copy=True, with_
mean=True, with_std=True)), ('clf', KNeighborsClassifier(algorithm='au

```
to', leaf_size=30, metric='minkowski',
metric_params=None, n_neighbors=1, p=2,
w='uniform'))]),
'p-2__cl”: KNC(algorithm='auto', leaf_
size=30, met='miski',
met_ps=None, n_neighbors=1, p=2,
w='uniform'),
'p-2__cl__algorithm': 'auto',
[...]
'p-2__sc__with_std': T}
```

Dependiendo de los valores regresados por el método get_ps, tu sabrás como entrar a los atributos de los clasificadores individuales. Trabajemos con los parámetros de regularización inversa C del clasificador de regresión logística y de profundidad de árbol de decisión a través del grid search por propósitos de demostración. Echemos in vistazo:

```
>>> from sklearn.grid_search import GdSearchCV
>>> params = {'dtreecl__max_depth': [0.1, .02],
... 'p-1__clf__C': [0.001, 0.1, 100.0]}
>>> gd = GdSearchCV(estimator=mv_cl,
... param_grid=params,
... cv=10,
... scoring='roc_auc')
>>> gd.fit(X_tra, y_tra)
```

Después de que el grid search se haya completado, podemos imprimir las diferentes combinaciones de valores de híper parámetros y la puntuación promedio de R_C AC computados a través de la validación cruzada de 10 pliegues. El código es el siguiente:

```
>>> for params, mean_sc, scores in grid.grid_sc_:
```

```
... print("%0.3f+/-%0.2f %r"
... % (mean_sc, sc.std() / 2, params))
0.967+/-0.05 {'p-1__cl__C': 0.001, 'dtreeclassifier__
ma_depth': 1}
0.967+/-0.05 {'p-1__cl__C': 0.1, 'dtreeclassifier__ma_
depth': 1}
1.000+/-0.00 {'p-1__cl__C': 100.0, 'dtreeclassifier__
ma_depth': 1}
0.967+/-0.05 {'p-1__cl__C': 0.001, 'dtreeclassifier__
ma_depth': 2}
0.967+/-0.05 {'p-1__cl__C': 0.1, 'dtreeclassifier__ma_
depth': 2}
1.000+/-0.00 {'p-1__cl__C': 100.0, 'dtreeclassifier__
ma_depth': 2}
>>> print('Best parameters: %s' % gd.best_ps_)
Best parameters: {'p1__cl__C': 100.0,
'dtreeclassifier__ma_depth': 1}
>>> print('Accuracy: %.2f' % gd.best_sc_)
Accuracy: 1.00
```

Preguntas

1. Explica cómo combinar diferentes modelos en detalle.

2. Cuales son las metas y los beneficios de combinar modelos?