Máster Universitario en Ingeniería Informática

Tratamiento Inteligente de Datos

PRÁCTICA 3: Clasificación

Ebay Auctions

UNIVERSIDAD DE GRANADA

Carlos Santiago Sánchez Muñoz

Grupo de prácticas 1 - Jueves

Email: carlossamu7@correo.ugr.es

26 de noviembre de 2020

Índice

1. Descripción del problema

La multinacional eBay Inc. (propietaria de la conocida web eBay.com para subasta de productos a través de internet) desea conocer mejor el comportamiento de las transacciones producidas en su web para, a la vista de los resultados, diseñar nuevos servicios que mejoren la experiencia de los usuarios vendedores para así incrementar las ventas y, por tanto, mejorar los ingresos de la compañía.

Así, se va aplicar analítica empresarial de cara a extraer conocimiento útil para la toma de decisiones a partir de algunos datos disponibles. Concretamente, el objetivo es construir un modelo que clasifique las subastas entre competitivas y no competitivas. Una subasta se considera competitiva si recibe al menos dos ofertas sobre el objeto subastado. Los datos disponibles corresponden a 1972 subastas a través de eBay.com realizadas en los últimos dos meses. Se incluyen variables que describen el objeto (categoría), el vendedor (mediante su valoración o rating) y los términos de la subasta fijados por el vendedor (duración de la subasta, precio de inicio, moneda y día de la semana en el que finaliza la subasta). Además, disponemos del precio al cual se cerró la subasta. Se pretende predecir si la subasta será o no competitiva así como comprender qué relaciones provocan dicho factor de cara a diseñar estrategias de negocio para que los clientes vendedores aumenten la tasa de subastas competitivas en sus productos.

Para ello se van a considerar varios modelos de clasificación distintos (con distintos parámetros cada uno) y se van a entrenar. Para un posterior análisis comparativo se van a emplear tablas de errores, matrices de confusión y curvas ROC. Previamente, se realizará un preprocesamiento adecuado de los datos.

Comenzamos realizando la lectura del conjunto de datos:

```
""" Lectura de datos. Devuelve el dataframe.
"""
def read_data():
    df = pd.read_excel('eBayAuctions.xls')

    if(IMPRIME_INFO):
        print("\nInformacion del dataframe:")
        print(df.info())
        print("\nDataframe:")
        print(df)
    return df
```

2. Preprocesamiento

En esta sección se va a cubrir el preprocesamiento de los datos. En primer lugar se tratarán los atributos, posteriormente se dará alguna información relativa a la distribución de las clases y finalmente y se realizará la división en conjunto de entrenamiento y test.

2.1. Tratando los atributos categóricos

El conjunto de datos contiene tres atributos categóricos que es necesario convertir a enteros que son: `currency`, `endDay` y `Category`. El tratemiento es muy similar porque lo se va a mostrar exclusivamente el de `currency`.

```
""" Convierte a int la variable 'currency'.
- df: dataframe.
"""
def int_currency(df):
    list = []
    for i in range(len(df)):
        if(df["currency"][i]=="US"):
            list.append(0)
        elif(df["currency"][i]=="EUR"):
            list.append(1)
        elif(df["currency"][i]=="GBP"):
            list.append(2)
    df = df.drop("currency", axis=1)
    df["currency"] = list
    return df
```

Tal y como se ha comentado para las otras dos variables es similar. Una función se encarga de realizar estas tres tareas de forma sencilla:

```
""" Convierte a int las variables categoricas.
- df: dataframe.
"""
def to_int_categorical(df):
    df = int_endDay(df)
    df = int_currency(df)
    df = int_Category(df)
    return df
```

2.2. Distribución de los datos

A la hora de trabajar con un conjunto de datos para clasificación es interesante observar previamente la distribución de los mismos. En Python la función `hist` nos saca un histograma de un `dataframe`. El código es el siguiente:

```
if(IMPRIME_INFO):
    for atr in df:
        print()
        print(df.groupby([atr]).size())
df.hist()
plt.gcf().canvas.set_window_title("Practica 3 - clasificacion")
plt.show()
```

Los histogramas obtenidos para cada una de las variables del problema se muestran en la Imagen 1.

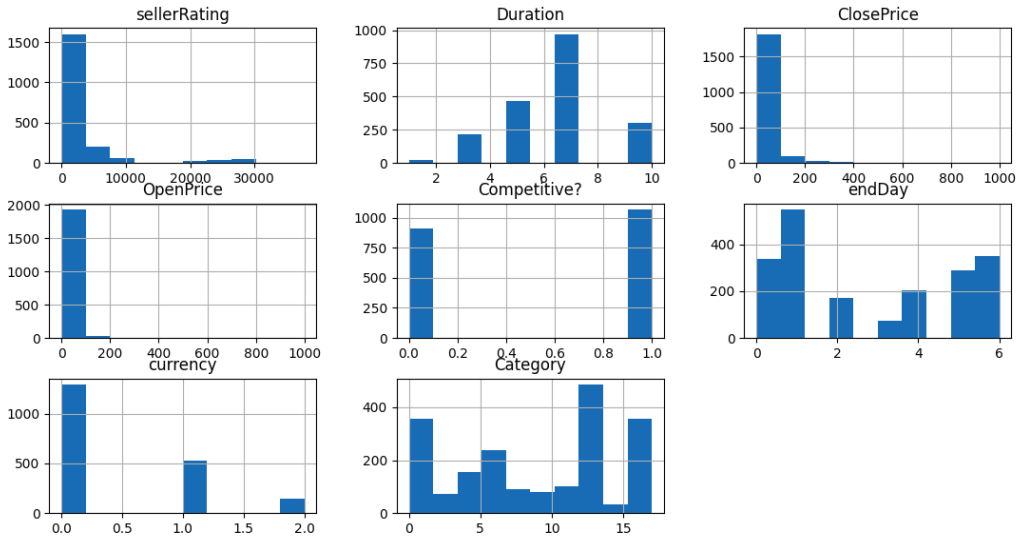

Imagen 1: Histograma

Algunos atributos como `sellerRating` y `ClosePrice` están un poco desbalanceados ya que algunas clases tienen mucha más frecuencia que otras.

2.3. Dividiendo en entrenamiento y test

Por último, otro de los pasos básicos antes de realizar la clasificación: dividir los datos en conjunto de entrenamiento y test. Para ello, el primer paso es seleccionar el atributo `Competitive?` como etiquetas y el resto como el conjunto de entrada. Posteriormente usando `train_test_split` de Python se puede dividir el dataset.

No obstante posteriormente para cada modelo se va a usar una validación cruzada a través de la función `cross_val_score`, a la cual se le pasará el modelo, los datos y en cuantos subconjuntos se divide el conjunto de datos para realizar la validación cruzada. En mi caso voy he elegido 5 para ese parámetro, es decir, entreno con 4/5 de los datos y testeo con el último quinto.

```
""" Divide los datos quitando la etiqueta a predecir. Devuelve X e y.
- df: dataframe.
"""
def split_data(df):
    return df.drop(columns="Competitive?", axis=1), df["Competitive?"]

# Dividiendo datos en input/output
X, y = split_data(df)
# Dividiendo datos en train y test
X_train, X_test, y_train, y_test = train_test_split(X, y, random_state=0)
```

3. K vecinos más cercanos

El primer modelo de clasificación que se va a usar es k vecinos más cercanos. Antes de pasar a él se va a comentar la siguiente función que imprime resultados de la clasificación de cualquier modelo. Imprime por pantalla la matriz de confusión y también se llama a la función `classification_repot` del paquete `sklearn.metrics` disponible en Python.

```
""" Muestra matriz de confusion y un reportaje de clasificacion.
- y_test: etiquetas reales.
- y_pred: etiquetas predichas.
"""
def print_plot_sol(y_test, y_pred):
    print("Matriz de confusion:")
    print(confusion_matrix(y_test, y_pred))
    print(classification_report(y_test, y_pred))
```

Para el algoritmo en sí se va a usar el paquete `sklearn.neighbors` que contiene `KNeighborsClassifier`, la función deseada. A parte de eso se imprime alguna información relacionada al accuracy, tanto en el conjunto de entrenamiento como posteriormente en la validación cruzada. Veamos esto con más detalle:

```
""" Entrenando con k vecinos mas cercanos.
- X_train: datos de entrenamiento.
- X_test: datos de test.
- y_train: etiquetas del entrenamiento.
- y_test: etiquetas de test.
"""
def knn(X_train, X_test, y_train, y_test):
    print("\n------- KNN -------")
    knn = KNeighborsClassifier(5)
    print("Entrenando KNN")
    knn.fit(X_train, y_train)
    print('Accuracy de K-NN en el conjunto de entrenamiento: {:.2f}'
        .format(knn.score(X_train, y_train)))
    print('Accuracy de K-NN en el conjunto de test: {:.2f}'
        .format(knn.score(X_test, y_test)))
    print("Accuracys de la cross-validation (5 particiones)")
    score = cross_val_score(knn, X_train.append(X_test),
                            y_train.append(y_test), cv=5)
    print(score)
    print("Accuracy medio de la cross-validation: {:.4f}"
        .format(stats.mean(score)))
    print("Prediciendo etiquetas")
    return knn.predict(X_test), knn.predict_proba(X_test)
```

El accuracy obtenido en entrenamiento es $0,85$, el cual es bastante bueno. El porcentaje de acierto, en media sobre la validación cruzada es $0,7738$ que está bien y no se distancia demasiado del de entrenamiento. No parece que exista un excesivo *overfitting* o sobreajuste a los datos. Por último en la matriz de confusión se observa el número de instancias de cada clase que se clasificó bien y también las que se clasificaron erróneamente.

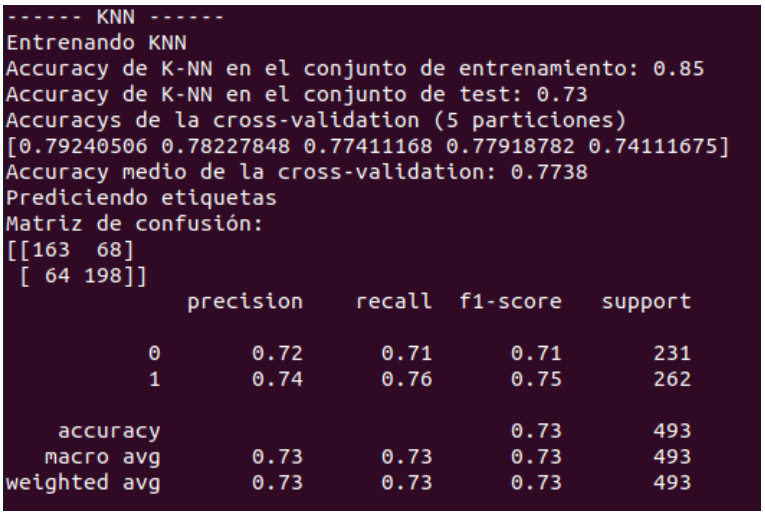

Imagen 2: K vecinos más cercanos

4. Árbol de decisión

El siguiente algoritmo que se va a usar para clasificar es un árbol de decisión. En Python está disponible la función `DecisionTreeClassifier` en el paquete `sklearn.tree`. Al igual que antes se imprime información relativa al acierto sobre el conjunto de entrenamiento y en segundo lugar sobre toda la validación cruzada, dando la media de dicha precisión.

```
""" Entrenando con arbol de decision.
- X_train: datos de entrenamiento.
- X_test: datos de test.
- y_train: etiquetas del entrenamiento.
- y_test: etiquetas de test.
"""
def decision_tree(X_train, X_test, y_train, y_test):
    print("\n------- DecisionTreeClassifier -------")
    tree = DecisionTreeClassifier(max_depth=10)
    print("Entrenando el árbol de ódecisin")
    tree = tree.fit(X_train, y_train)
    print('Accuracy de DecisionTreeClassifier en el conjunto de entrenamiento: {:.2f}'
        .format(tree.score(X_train, y_train)))
    print('Accuracy de DecisionTreeClassifier en el conjunto de test: {:.2f}'
        .format(tree.score(X_test, y_test)))
    print("Accuracys de la cross-validation (5 particiones)")
    score = cross_val_score(tree, X_train.append(X_test),
                            y_train.append(y_test), cv=5)
    print(score)
    print("Accuracy medio de la cross-validation: {:.4f}"
        .format(stats.mean(score)))
    print("Prediciendo etiquetas")
    return tree.predict(X_test), tree.predict_proba(X_test)
```

Observamos los resultados en la Imagen 3.

```
------ DecisionTreeClassifier ------
Entrenando el árbol de decisión
Accuracy de DecisionTreeClassifier en el conjunto de entrenamiento: 0.97
Accuracy de DecisionTreeClassifier en el conjunto de test: 0.87
Accuracys de la cross-validation (5 particiones)
[0.89113924 0.89367089 0.90862944 0.88324873 0.88324873]
Accuracy medio de la cross-validation: 0.8920
Prediciendo etiquetas
Matriz de confusión:
[[202  29]
 [ 35 227]]
              precision    recall  f1-score   support

           0       0.85      0.87      0.86       231
           1       0.89      0.87      0.88       262

    accuracy                           0.87       493
   macro avg       0.87      0.87      0.87       493
weighted avg       0.87      0.87      0.87       493
```

Imagen 3: Árbol de decisión

Sobre el conjunto de entrenamiento el porcentaje de acierto obtenido es $0,97$ y sobre el de test 087. La validación cruzada arroja 5 resultados del trabajo realizado por el algoritmo, cuya media es un accuracy de $0,892$ lo cual está bastante bien y además supera considerablemente el obtenido por k vecinos más cercanos.

5. Máquinas de vectores de soporte

El tercer algoritmo que se ha decidido usar es el de Máquinas de Vectores de Soporte (*Support Vector Machine*, SVM), en su enfoque para problemas de clasificación. De igual modo que antes se imprime información relativa a los resultados de la clasificación y también de la validación cruzada.

En Python está disponible el algoritmo `svm` del paquete `sklearn`. Obsérvese el código:

```
""" Entrenando con SVC.
- X_train: datos de entrenamiento.
- X_test: datos de test.
- y_train: etiquetas del entrenamiento.
- y_test: etiquetas de test.
"""
def SVC(X_train, X_test, y_train, y_test):
    print("\n------ SVC ------")
    svc = svm.SVC(kernel='linear', probability=True, max_iter=500000)
    print("Entrenando SVC")
    svc.fit(X_train, y_train)
    print('Accuracy de SVM en el conjunto de entrenamiento: {:.2f}'
        .format(svc.score(X_train, y_train)))
    print('Accuracy de SVM en el conjunto de test: {:.2f}'
        .format(svc.score(X_test, y_test)))
    print("Accuracys de la cross-validation (5 particiones)")
    score = cross_val_score(svc, X_train.append(X_test),
                            y_train.append(y_test), cv=5)
    print(score)
    print("Accuracy medio de la cross-validation: {:.4f}"
        .format(stats.mean(score)))
    print("Prediciendo etiquetas")
    return svc.predict(X_test), svc.predict_proba(X_test)
```

Los resultados están disponibles en la Imagen 4.

```
------ SVC ------
Entrenando SVC
Accuracy de SVM en el conjunto de entrenamiento: 0.80
Accuracy de SVM en el conjunto de test: 0.82
Accuracys de la cross-validation (5 particiones)
[0.46835443 0.45316456 0.5786802  0.55837563 0.54568528]
Accuracy medio de la cross-validation: 0.5209
Prediciendo etiquetas
Matriz de confusión:
[[175  56]
 [226  36]]
              precision    recall  f1-score   support

           0       0.44      0.76      0.55       231
           1       0.39      0.14      0.20       262

    accuracy                           0.43       493
   macro avg       0.41      0.45      0.38       493
weighted avg       0.41      0.43      0.37       493
```

Imagen 4: Máquinas de vectores de soporte

Para el entrenamiento no se ha usado límite máximo de iteraciones para el algoritmo obteniéndose una tasa de acierto de $0,82$ para el conjunto de test. Sin embargo para la validación cruzada, debido a la complejidad computacional, sí que se ha activado el parámetro `max_iter` y tal como se observa no llega a converger.

6. Regresión Logística

Como último algoritmo de clasificación se propone usar Regresión Logística. En Python está disponible en el paquete `sklearn.linear_model` la función `LogisticRegression`. De modo análogo a los algoritmos anteriores se imprimen resultados relativos al algoritmo.

```
""" Entrenando con LogisticRegression.
- X_train: datos de entrenamiento.
- X_test: datos de test.
- y_train: etiquetas del entrenamiento.
- y_test: etiquetas de test.
"""
def LR(X_train, X_test, y_train, y_test):
    print("\n------ LogisticRegression ------")
    lr = LogisticRegression()
    print("Entrenando óregresin ílogstica")
    lr.fit(X_train, y_train)
    print('Accuracy de LogisticRegression en el conjunto de entrenamiento: {:.2f}'
        .format(lr.score(X_train, y_train)))
    print('Accuracy de LogisticRegression en el conjunto de test: {:.2f}'
        .format(lr.score(X_test, y_test)))
    print("Accuracys de la cross-validation (5 particiones)")
    score = cross_val_score(lr, X_train.append(X_test), y_train.append(y_test), cv=5)
    print(score)
    print("Accuracy medio de la cross-validation: {:.4f}"
        .format(stats.mean(score)))
    print("Prediciendo etiquetas")
    return lr.predict(X_test), lr.predict_proba(X_test)
```

Los resultados están en la Imagen 5.

```
------ LogisticRegression ------
Entrenando regresión logística
Accuracy de LogisticRegression en el conjunto de entrenamiento: 0.79
Accuracy de LogisticRegression en el conjunto de test: 0.82
Accuracys de la cross-validation (5 particiones)
```

```
[0.78987342 0.78987342 0.78680203 0.78172589 0.81218274]
Accuracy medio de la cross-validation: 0.7921
Prediciendo etiquetas
Matriz de confusión:
[[215  16]
 [ 72 190]]
              precision    recall  f1-score   support

           0       0.75      0.93      0.83       231
           1       0.92      0.73      0.81       262

    accuracy                           0.82       493
   macro avg       0.84      0.83      0.82       493
weighted avg       0.84      0.82      0.82       493
```

Imagen 5: Regresión logística

La tasa de acierto sobre el conjunto de entrenamiento es $0,79$ y sobre el de test $0,82$ (por lo que no parece haber *overfitting*). La validación cruzada da un accuracy medio de $0,7921$, la cual es bastante buena.

7. Curvas ROC

Para comparar los algoritmos ya se han expuesto algunas medidas como las matrices de confusión y las tablas de errores. En este apartado se va a exponer una forma más de realizar la comparación que es mediante el uso de curvas ROC.

En Python se pueden calculas las curvas ROC usando `roc_curve` del paquete `sklearn.metrics`. También se va a usar `roc_auc_score` para sacar la puntuación. Observemos el código:

```
def plot_roc_curve(y_test, probs_knn, probs_tree, probs_svc, probs_lr):
    # keep probabilities for the positive outcome only
    probs_knn = probs_knn[:, 1]
    probs_tree = probs_tree[:, 1]
    probs_svc = probs_svc[:, 1]
    probs_lr = probs_lr[:, 1]

    # Calculamos scores
    auc_knn = roc_auc_score(y_test, probs_knn)
    auc_tree = roc_auc_score(y_test, probs_tree)
    auc_svc = roc_auc_score(y_test, probs_svc)
    auc_lr = roc_auc_score(y_test, probs_lr)

    # Los imprimimos scores
    print("KNN: ROC AUC={:.3f}".format(auc_knn))
    print("Tree: ROC AUC={:.3f}".format(auc_tree))
    print("SVC: ROC AUC={:.3f}".format(auc_svc))
    print("Logistic: ROC AUC={:.3f}".format(auc_lr))

    # Calculamos las curvas ROC
    fpr_knn, tpr_knn, _ = roc_curve(y_test, probs_knn)
    fpr_tree, tpr_tree, _ = roc_curve(y_test, probs_tree)
    fpr_svc, tpr_svc, _ = roc_curve(y_test, probs_svc)
    fpr_lr, tpr_lr, _ = roc_curve(y_test, probs_lr)

    # Dibujamos las curvas
    plt.plot(fpr_knn, tpr_knn, linestyle='—', label='KNN')
    plt.plot(fpr_tree, tpr_tree, linestyle=':', label='Tree')
    plt.plot(fpr_svc, tpr_svc, linestyle='–.', label='SVC')
    plt.plot(fpr_lr, tpr_lr, marker='.', label='Logistic')
    # Etiqueta los ejes
    plt.xlabel('False Positive Rate')
    plt.ylabel('True Positive Rate')
    plt.legend()
    plt.title("Curva ROC")
    plt.gcf().canvas.set_window_title("Practica 3 – clasificacion")
    plt.show()
```

A continuación se muestra la siguiente gráfica obtenida en donde se han pintado todas las curvas ROC de los algoritmos, cada una de un color diferente. El crecimiento de la tasa de verdaderos positivos es muy bueno para todos. Cabe destacar la excelencia del árbol de decisión, además de que es sencillo.

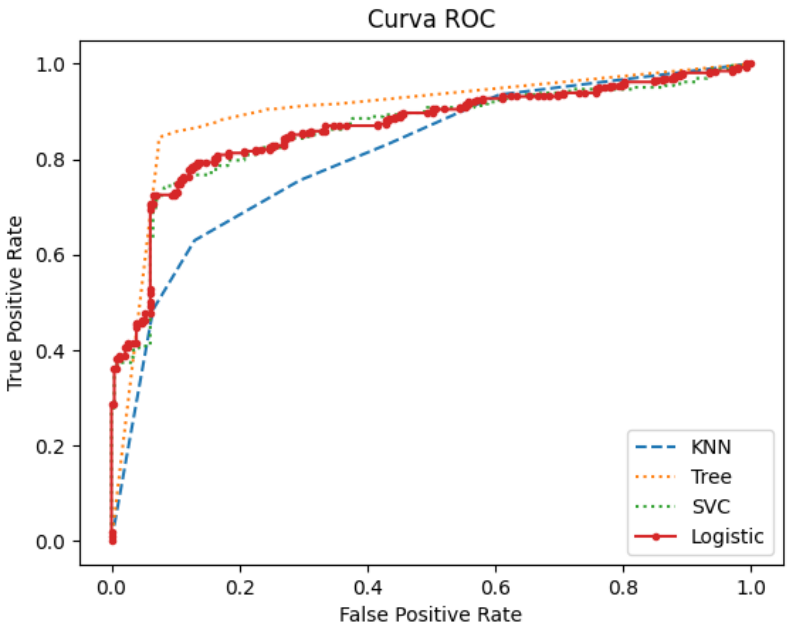

Imagen 6: Curvas ROC

Los *scores* obtenidos en las curvas ROC son:

```
KNN: ROC AUC=0.810
Tree: ROC AUC=0.899
SVC: ROC AUC=0.446
Logistic: ROC AUC=0.865
```

Imagen 7: Puntuaciones obtenidas en la curva ROC

Se observa claramente que SVC no llega a converger (cuando se limita el número de iteraciones) y su tasa es inferior a la del resto de algoritmos.

Si se quisiera recomendar a un vendedor para hacer que sus subastas tengan más probabilidad de ser competitivas en función de los resultados obtenidos podríamos pintar el árbol de decisión (que además tiene muy buen acierto) y a partir de él establecer alguna conclusión. Veamos la Imagen 8.

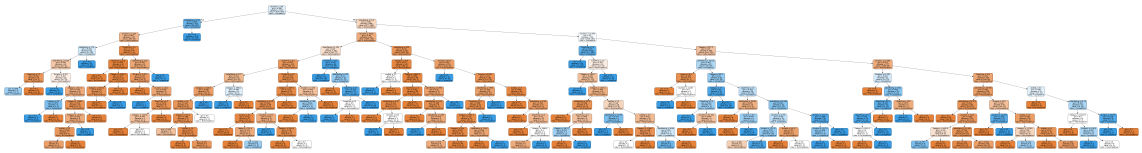

Imagen 8: Árbol obtenido

Para que sus subastas sean más competitivas se podría proponer, por ejemplo, que la duración sea mayor que 3 días y el `sellerRating` sea inferior a 10 ya que en esa zona del

árbol hay una gran presencia de clasificaciones positivas.