- Dataset: vuelos de todos los estados de US durante Enero 2020 de todas las aerolíneas

- Clasificar si los vuelos llegan a tiempo o demorados (1 demorado, 0 a tiempo)

-Nos basamos en variables como origen, destino, distancia de la ruta, horario de salida programado, día de la semana y del mes

- Agregamos data sobre modelo y tipo de avión

- Sampleamos el dataset para correr las pruebas, equilibramos el desbalance de clases en la variable target y probamos 80mil datos en vez de 400mil (para no esperar tantas horas por cada modelo que entrena a modo de demo)

Limpieza, exploración y tipos de variables

```
[22]: df['ORIGIN'].value_counts()

[22]: ORD    1915
      ATL    1859
      DFW    1784
      CLT    1523
      DEN    1275
             ...
      PPG       1
      HGR       1
      LCK       1
      PSM       1
      PAH       1
      Name: ORIGIN, Length: 344, dtype: int64
```

```
[5]: df.dtypes

[5]: Unnamed: 0                 int64
     DAY_OF_MONTH               int64
     DAY_OF_WEEK                int64
     OP_UNIQUE_CARRIER         object
     OP_CARRIER_AIRLINE_ID      int64
     OP_CARRIER                object
     TAIL_NUM                  object
     OP_CARRIER_FL_NUM          int64
     ORIGIN_AIRPORT_ID          int64
     ORIGIN_AIRPORT_SEQ_ID      int64
     ORIGIN                    object
     DEST_AIRPORT_ID            int64
     DEST_AIRPORT_SEQ_ID        int64
     DEST                      object
     DEP_TIME                 float64
     DEP_DEL15                float64
     DEP_TIME_BLK              object
     ARR_TIME                 float64
     ARR_DEL15                float64
     CANCELLED                float64
     DIVERTED                 float64
     DISTANCE                 float64
     Unnamed: 21              float64
     dtype: object
```

-Definimos variables numéricas y categóricas

-Empezamos a modelar pipelines

```
[39]: #seleccionamos features numericas
      selector_numericas=FeatureSelection(selected_features=['hora_salida', 'hora_arribo', 'distancia'])

[40]: #seleccionamos features categoricas
      selector_categoricas=FeatureSelection(selected_features=['dia_semana', 'dia_mes', 'aerolinea', 'origen','destino', 'hora_salida_blk'])

[41]: pasos_numericas = [('selector',selector_numericas), ('scaler', StandardScaler())]

[42]: pasos_categoricas = [('selector',selector_categoricas), ('encoder', OneHotEncoder(handle_unknown = "ignore"))]

[43]: pipe_num = Pipeline(pasos_numericas)

[44]: pipe_cat = Pipeline(pasos_categoricas)

[45]: folds=StratifiedKFold(n_splits=5,shuffle=True, random_state=42)

[46]: pasos = [ ('feature_engineering',FeatureUnion([ ('num', pipe_num), ('cat', pipe_cat) ]) ),
               ('poly', PolynomialFeatures()),
               ('clasificador', None)]

[47]: pipe=Pipeline(pasos)
```

Probamos KNN

```
[48]: param_grid_knn= [{'clasificador__n_neighbors':range(6,20,1), 'clasificador__weights':['uniform','distance'],
                   'clasificador__metric': ['minkowski', 'manhattan', 'euclidean'],
     'feature_engineering__num__scaler': [MinMaxScaler(), StandardScaler()],
     'clasificador': [KNeighborsClassifier()]}]
```

```
[49]: grid_knn=RandomizedSearchCV(pipe, param_grid_knn, n_jobs=-1, cv=folds, scoring='accuracy')
```

```
[50]: grid_knn.fit(X_train, y_train)
```

```
[52]: grid_knn.best_score_
```

```
[52]: 0.6798333333333334
```

```
[53]: grid_knn.best_params_
```

```
[53]: {'feature_engineering__num__scaler': StandardScaler(),
       'clasificador__weights': 'distance',
       'clasificador__n_neighbors': 11,
       'clasificador__metric': 'manhattan',
       'clasificador': KNeighborsClassifier(metric='manhattan', n_neighbors=11, weights='distance')}
```

Probamos Regresión Logística

```
[54]: param_grid_logistic= [
          {'clasificador__penalty': ['l1', 'l2'], 'clasificador__C': np.arange(0.01,100, 0.02),
           'feature_engineering__num__scaler': [MinMaxScaler(), StandardScaler()],
           'clasificador': [LogisticRegression(solver='liblinear', max_iter=3500)]}
      ]
```

```
[55]: grid_logistic=RandomizedSearchCV(pipe, param_grid_logistic, n_jobs=-1, cv=folds, scoring='accuracy')
```

```
[56]: grid_logistic.fit(X_train, y_train)
```

```
[58]: grid_logistic.best_score_
```

```
[58]: 0.7949666666666666
```

```
[59]: grid_logistic.best_params_
```

```
[59]: {'feature_engineering__num__scaler': StandardScaler(),
       'clasificador__penalty': 'l1',
       'clasificador__C': 6.249999999999999,
       'clasificador': LogisticRegression(C=6.249999999999999, max_iter=3500, penalty='l1',
                          solver='liblinear')}
```

Probamos Bayes Bernoulli

[65]:
```
param_grid_bayesB= [{ 'clasificador__alpha': np.arange(0.38, 5, 0.05),
    'feature_engineering__num__scaler': [MinMaxScaler(), StandardScaler()],
     'clasificador': [BernoulliNB()] }]
```

[66]:
```
grid_bernoulli=RandomizedSearchCV(pipe, param_grid_bayesB, n_jobs=-1, cv=folds, scoring='accuracy')
```

[67]:
```
grid_bernoulli.fit(X_train, y_train)
```

[68]:
```
grid_bernoulli.best_estimator_
```

[68]:
```
Pipeline(steps=[('feature_engineering',
                 FeatureUnion(transformer_list=[('num',
                                                 Pipeline(steps=[('selector',
                                                                  FeatureSelection(selected_features=['hora_salida',
                                                                                                      'hora_arribo',
                                                                                                      'distancia'])),
                                                                 ('scaler',
                                                                  StandardScaler())])),
                                                ('cat',
                                                 Pipeline(steps=[('selector',
                                                                  FeatureSelection(selected_features=['dia_semana',
                                                                                                      'dia_mes',
                                                                                                      'aerolinea',
                                                                                                      'origen',
                                                                                                      'destino',
                                                                                                      'hora_salida_blk'])),
                                                                 ('encoder',
                                                                  OneHotEncoder(handle_unknown='ignore'))]))])),
                ('poly', PolynomialFeatures()),
                ('clasificador', BernoulliNB(alpha=0.73))])
```

[69]:
```
grid_bernoulli.best_score_
```

[69]:
```
0.6371666666666667
```

Probamos SGDC

```
[70]: param_grid_SGDC= [
          {'clasificador__penalty':['l1', 'l2'], 'clasificador__alpha': np.arange(0.0001,0.2,0.0001),
           'clasificador__loss': ['log', 'modified_huber', 'hinge', 'perceptron'],
           'clasificador__epsilon': np.arange(0.01, 0.3, 0.1),
              'feature_engineering__num__scaler': [MinMaxScaler(), StandardScaler()],
           'clasificador': [SGDClassifier(max_iter=2000)]}
          ]
```

```
[71]: param_grid_SGDC=RandomizedSearchCV(pipe, param_grid_SGDC, n_jobs=-1, cv=folds, scoring='accuracy')
```

```
[72]: param_grid_SGDC.fit(X_train, y_train)
      param_grid_SGDC.best_estimator_
```

```
                                   loss='perceptron', max_iter=2000))])
```

```
[73]: param_grid_SGDC.best_score_

[73]: 0.7294666666666667
```

Testeamos la performance de Logistic Regression

[74]:
```
#Veamos cuánto da el clasificador nulo
y_test.value_counts(normalize=True)
y_test.mean()
1.0 - y_test.mean()
```

[74]: 0.5115000000000001

[75]:
```
grid_logistic.best_score_
```

[75]: 0.7949666666666666

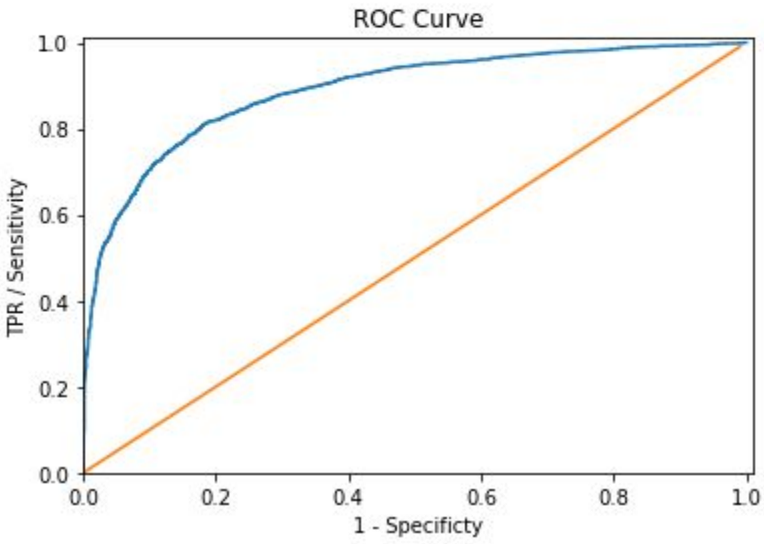
[76]:
```
y_pred_logistic = grid_logistic.predict(x_val)
confusion_logistic = confusion_matrix(y_val, y_pred_logistic)
confusion_logistic
```

[76]:
```
array([[2429,  532],
       [ 591, 2448]], dtype=int64)
```

[77]:
```
print('Accuracy=', accuracy_score(y_val, y_pred_logistic))
```

```
Accuracy= 0.8128333333333333
```

[78]:
```
print(recall_score(y_val, y_pred_logistic))
```

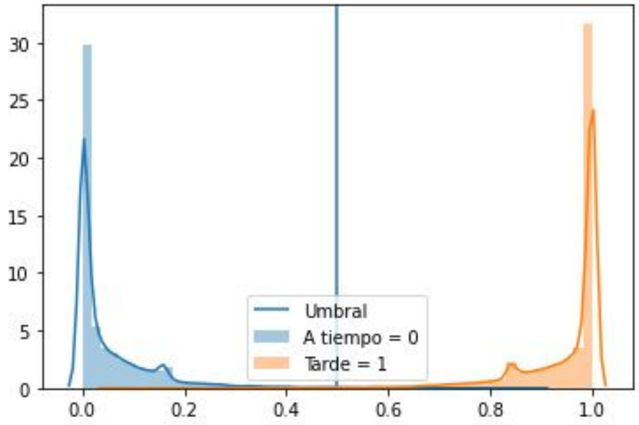



```
[82]: print('AUC=', auc(fpr_lg, tpr_lg))
```

AUC= 0.8888879998497523

```
[84]: sns.distplot(grid_logistic.predict_proba(X_train[y_train==0])[:,1])
      sns.distplot(grid_logistic.predict_proba(X_train[y_train==1])[:,1])
      ylim = plt.ylim()
      plt.vlines(0.5, ylim[0], ylim[1])
      plt.ylim(ylim)
      plt.legend(['Umbral', 'A tiempo = 0', 'Tarde = 1'])
```

[84]: <matplotlib.legend.Legend at 0x1f496fc55c8>



Testeamos la performance de SGDC

```
[90]: param_grid_SGDC.best_score_
```

```
[90]: 0.7294666666666667
```

```
[91]: y_pred_SGDC = param_grid_SGDC.predict(x_val)
      confusion_SGDC = confusion_matrix(y_val, y_pred_SGDC)
      confusion_SGDC
```

```
[91]: array([[1885, 1076],
             [ 550, 2489]], dtype=int64)
```

```
[92]: print('Accuracy=', accuracy_score(y_val, y_pred_SGDC))
```

```
Accuracy= 0.729
```

Testear performance de KNN

[94]:
```
grid_knn.best_score_
```

[94]: 0.6798333333333334

[95]:
```
y_pred_knn = grid_knn.predict(x_val)
confusion_knn = confusion_matrix(y_val, y_pred_knn)
confusion_knn
```

[95]:
```
array([[2021,  940],
       [1014, 2025]], dtype=int64)
```

[96]:
```
print(recall_score(y_val, y_pred_knn))
```

```
0.6663376110562685
```

[97]:
```
print(precision_score(y_val, y_pred_knn))
```

```
0.6829679595278246
```

[102]:
```
print('Recall umbral 0.5=', recall_score(y_val, y_pred_knn))
print('Recall umbral 0.4=', recall_score(y_val, y_pred_knn_new))
```

```
Recall umbral 0.5= 0.6663376110562685
Recall umbral 0.4= 0.794998354721948
```

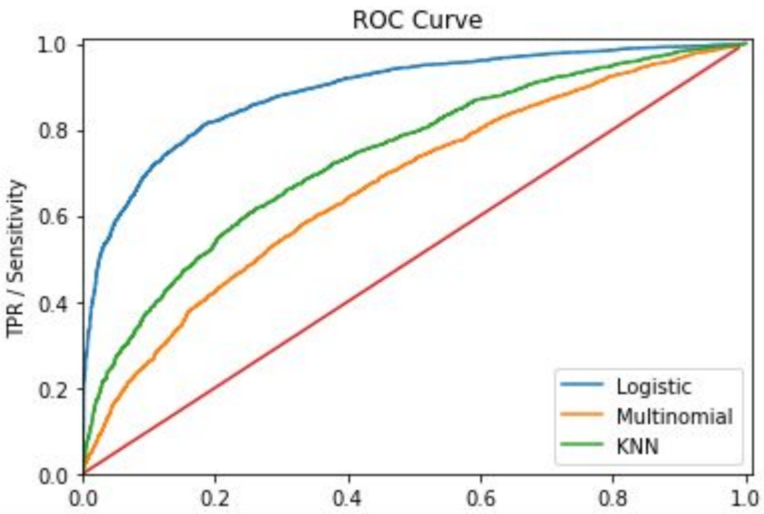
[103]:
```
print(precision_score(y_val, y_pred_knn))
print(precision_score(y_val, y_pred_knn_new))
```

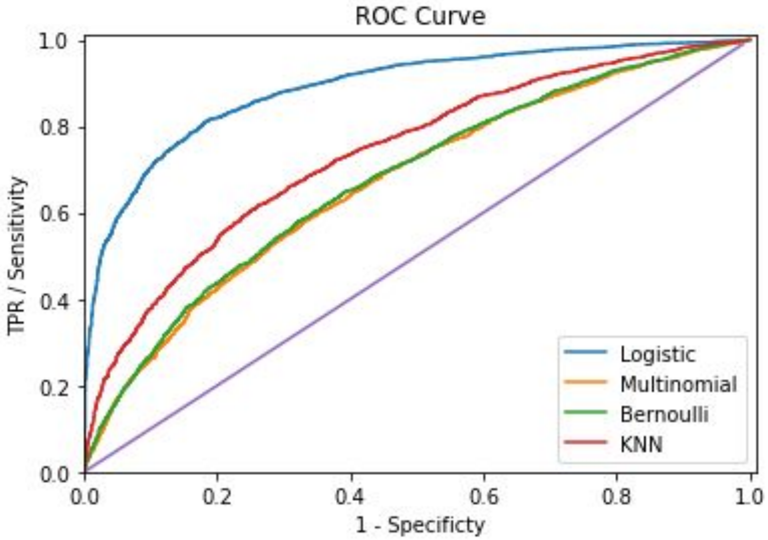
```
0.6829679595278246
0.6186939820742637
```

[104]:
```
print(accuracy_score(y_val, y_pred_knn))
print(accuracy_score(y_val, y_pred_knn_new))
```

```
0.6743333333333333
0.648
```

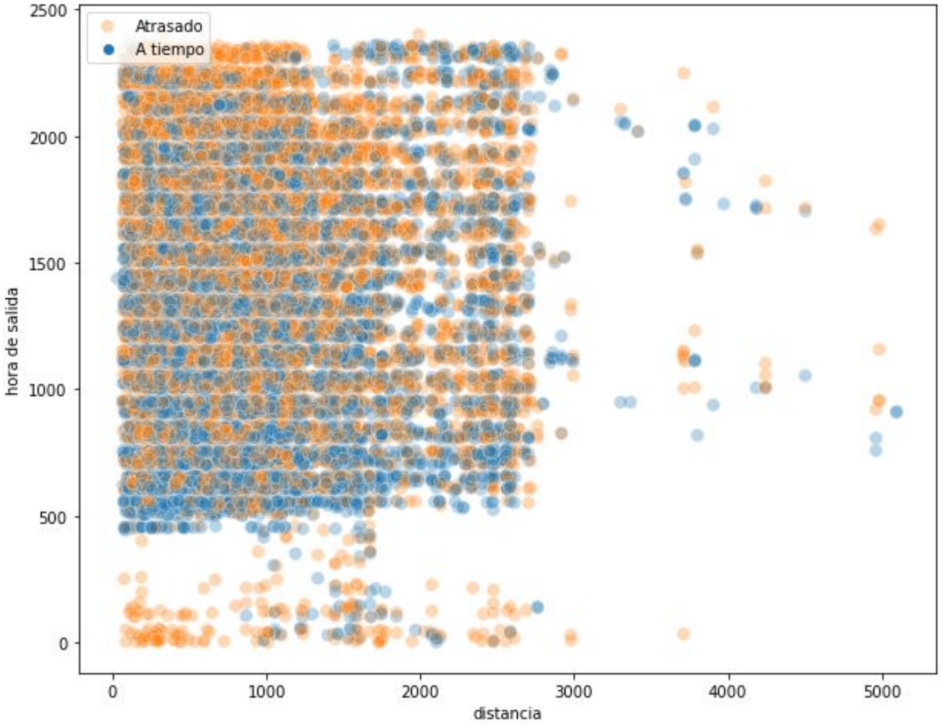
```python
df_mu = pd.DataFrame(dict(fpr=fpr_mu, tpr=tpr_mu, thr = thr_mu))

plt.axis([0, 1.01, 0, 1.01])
plt.xlabel('1 - Specificty')
plt.ylabel('TPR / Sensitivity')
plt.title('ROC Curve')
plt.plot(fpr_lg,tpr_lg)
plt.plot(fpr_mu,tpr_mu)
plt.plot(fpr_knn,tpr_knn)
#plt.plot(fpr_sg,tpr_sg)
plt.legend(['Logistic', 'Multinomial', 'KNN'])
plt.plot(np.arange(0,1, step =0.01), np.arange(0,1, step =0.01))
plt.show()
```





Comparación datos originales vs predicción

Predicción



Usamos el modelo que mejor funciona para hacer las predicciones

```
[125]: #comparamos el score del mejor modelo (Regresión Logística)
       y_pred_logistic_test = grid_logistic.predict(X_test)
       print('Accuracy=', accuracy_score(y_test, y_pred_logistic_test))

       Accuracy= 0.80525

[126]: #score de validación
       print('Accuracy=', accuracy_score(y_val, y_pred_logistic))

       Accuracy= 0.8128333333333333
```