time_series

December 3, 2021

1 Time Series Decision Tree Regressor and Support Vector Machine

by: Joshua Harasaki, Joanne Kim, Michael La, Tyler Chia

[3]:
```
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split

data = pd.read_csv("data_preparation.csv")
```

[4]:
```
data.rename(columns =␣
 ↪{'hospital-admissions_smoothed_covid19_from_claims_0_value':'hospital',
'chng_smoothed_outpatient_cli_1_value':'change',
'doctor-visits_smoothed_cli_2_value':'doc',
'google-symptoms_sum_anosmia_ageusia_raw_search_3_value':'google_sum',
'google-symptoms_anosmia_raw_search_4_value':'anosmia',
'indicator-combination_confirmed_incidence_num_6_value':'cases'}, inplace =␣
 ↪True)
```

[5]:
```
data
```

[5]:
```
      Unnamed: 0  geo_value  time_value   hospital    change       doc  \
0              0       6001  2020-02-20   0.110992  0.022051  0.000000
1              1       6013  2020-02-20   0.118745  0.020335  0.000000
2              4       6037  2020-02-20   0.112637  0.004175  0.134780
3              5       6059  2020-02-20   0.091819  0.005578  0.077821
4              6       6065  2020-02-20   0.090382  0.009983  0.000000
…            …          …           …          …         …         …
9421        9475       6075  2021-11-12   1.230602  0.670438  2.764137
9422        9476       6077  2021-11-12   1.706939  3.497545  2.002706
9423        9477       6081  2021-11-12   0.185958  1.626855  2.497344
9424        9478       6085  2021-11-12   0.103307  4.895631  9.675555
9425        9479       6111  2021-11-12  18.908364  1.679043  1.986365

      google_sum  anosmia  cases
0           0.12     0.06    0.0
1           0.14     0.14    0.0
```

```
2           0.09     0.06    0.0
3           0.17     0.09    0.0
4           0.10     0.06    0.0
…          …      …     …
9421        0.16     0.10    0.0
9422        0.13     0.13    0.0
9423        0.15     0.15    0.0
9424        0.15     0.09    0.0
9425        0.31     0.17    0.0

[9426 rows x 9 columns]
```

[6]:
```python
# setting dates as index
data2 = data.set_index('time_value')['2020-05-01' :'2021-11-01']
# dropping google features due to data not being comparable across geographic␣
↪regions
data2 = data2.drop(columns = ["google_sum", "anosmia","Unnamed: 0"])
```

[7]:
```python
# shifting the data to create t-1, t-2 values for each feature
data2['hosp_t1'] = data2.groupby(['geo_value'])['hospital'].shift(1)
data2['hosp_t2'] = data2.groupby(['geo_value'])['hospital'].shift(2)
data2['change_t1'] = data2.groupby(['geo_value'])['change'].shift(1)
data2['change_t2'] = data2.groupby(['geo_value'])['change'].shift(2)
data2['doc_t1'] = data2.groupby(['geo_value'])['doc'].shift(1)
data2['doc_t2'] = data2.groupby(['geo_value'])['doc'].shift(2)
data2 = data2.dropna()
```

[8]:
```python
data2
```

[8]:
```
            geo_value  hospital    change       doc  cases   hosp_t1  \
time_value
2020-05-03       6001  2.387752  2.193362  2.947938   44.0  2.912274
2020-05-03       6013  0.137259  2.239271  2.454575   11.0  0.138462
2020-05-03       6019  0.321797  1.316889  4.566139    0.0  0.410662
2020-05-03       6029  0.490785  1.545670  1.469381   36.0  0.490785
2020-05-03       6037  4.316431  2.488303  5.822741  768.0  3.897707
…                 …        …        …        …      …       …
2021-11-01       6075  1.635672  0.670438  2.115700    9.0  1.624891
2021-11-01       6077  3.159752  3.497545  3.134538   47.0  3.159752
2021-11-01       6081  4.988368  1.626855  5.653800   27.0  5.769160
2021-11-01       6085  1.997648  4.895631  8.086542  103.0  2.226632
2021-11-01       6111  6.162117  1.679043  2.328056   66.0  4.464349

             hosp_t2  change_t1  change_t2    doc_t1    doc_t2
time_value
2020-05-03  3.269167   2.071899   1.829670  2.722984  2.712027
2020-05-03  0.140425   1.973302   1.487273  2.038486  1.717165
```

```
2020-05-03  0.446559   1.386748   1.390057  4.399981  3.883414
2020-05-03  0.490785   1.391172   1.520922  1.628260  2.037505
2020-05-03  3.455211   2.173034   2.184908  5.397878  5.787655
…                …          …          …         …         …
2021-11-01  1.807216   0.670438   0.670438  1.945011  1.827886
2021-11-01  3.159752   3.497545   3.497545  3.383896  3.641620
2021-11-01  6.429908   1.626855   1.626855  6.707220  8.691601
2021-11-01  2.560690   4.895631   4.895631  5.330248  7.253664
2021-11-01  2.952570   1.679043   1.679043  2.132248  1.975561

[8220 rows x 11 columns]
```

2 Decision Tree Regressor

[9]:
```python
from sklearn.tree import DecisionTreeRegressor
from sklearn.metrics import r2_score
from sklearn.metrics import mean_squared_error
from sklearn.model_selection import TimeSeriesSplit
import matplotlib.pyplot as plt
from sklearn.model_selection import cross_validate
```

[10]:
```python
# creating X and y for model training
y = data2['cases']
X = data2.drop(columns = ['cases', "geo_value"])
X
```

[10]:
```
            hospital    change       doc   hosp_t1   hosp_t2  change_t1  \
time_value
2020-05-03  2.387752  2.193362  2.947938  2.912274  3.269167   2.071899
2020-05-03  0.137259  2.239271  2.454575  0.138462  0.140425   1.973302
2020-05-03  0.321797  1.316889  4.566139  0.410662  0.446559   1.386748
2020-05-03  0.490785  1.545670  1.469381  0.490785  0.490785   1.391172
2020-05-03  4.316431  2.488303  5.822741  3.897707  3.455211   2.173034
…                …         …         …         …         …          …
2021-11-01  1.635672  0.670438  2.115700  1.624891  1.807216   0.670438
2021-11-01  3.159752  3.497545  3.134538  3.159752  3.159752   3.497545
2021-11-01  4.988368  1.626855  5.653800  5.769160  6.429908   1.626855
2021-11-01  1.997648  4.895631  8.086542  2.226632  2.560690   4.895631
2021-11-01  6.162117  1.679043  2.328056  4.464349  2.952570   1.679043

            change_t2     doc_t1    doc_t2
time_value
2020-05-03   1.829670  2.722984  2.712027
2020-05-03   1.487273  2.038486  1.717165
2020-05-03   1.390057  4.399981  3.883414
2020-05-03   1.520922  1.628260  2.037505
```

```
2020-05-03   2.184908  5.397878  5.787655
…                 …        …        …
2021-11-01   0.670438  1.945011  1.827886
2021-11-01   3.497545  3.383896  3.641620
2021-11-01   1.626855  6.707220  8.691601
2021-11-01   4.895631  5.330248  7.253664
2021-11-01   1.679043  2.132248  1.975561

[8220 rows x 9 columns]
```

[11]:

```
# train test split
X_train, X_test, y_train, y_test = train_test_split(X,y,
                                                    test_size=0.2,␣
 ↪random_state=123,
                                                    shuffle=True)
```

[12]:

```
# tuning the hyperparameters for decision tree
from sklearn.model_selection import cross_val_score
max_depths = range(1,11)
splitter = ['random', 'best']
training_mse = []
validation_mse = []
i_array = []

for i in max_depths:
    for j in splitter:
        dtree = DecisionTreeRegressor(max_depth = i, splitter = j, random_state␣
 ↪= 3)
        dtree.fit(X_train, y_train)
        scores = cross_validate(dtree, X_train, y_train, cv = 5, scoring =␣
 ↪'neg_mean_squared_error', return_train_score=True)
        training_mse.append(np.mean(scores["train_score"]))
        validation_mse.append(np.mean(scores['test_score']))
        i_array.append([i,j])

print("Average Training MSEs:", training_mse)
print("Average Validation MSEs:", validation_mse)
```

```
Average Training MSEs: [-1165667.4549623777, -1096917.2692433402,
-1003228.1340506433, -885227.0162738807, -847711.2758663048, -713029.687896336,
-721795.6620392221, -565703.1730339909, -594574.6801073649, -458965.15016687335,
-483795.83630079794, -381412.3557717488, -411587.9774695973, -319265.975309401,
-353021.2820235759, -259005.06582470742, -295738.6515313687,
-206642.69858058117, -253435.55549618034, -166904.3943674281]
Average Validation MSEs: [-1220978.5115009737, -1208479.2414236353,
-1132671.907161127, -1073713.8769722702, -1099248.830796444, -924677.64628476,
-962045.3586950373, -1006085.6108890154, -1033295.8706774625,
-1070449.9718836262, -989286.2128663216, -1130919.30764747, -1040053.91265161,
```

```
-1121356.7228422873, -1140278.877549642, -1192793.808905399,
-1233827.9770182383, -1260436.3988357864, -1186262.2519986262,
-1272060.8847626946]
```

[13]:
```
abs_training_mse = np.abs(training_mse)
abs_validation_mse = np.abs(validation_mse)
training_rmse = np.sqrt(abs_training_mse)
validation_rmse = np.sqrt(abs_validation_mse)
mint = min(validation_rmse)
idx = np.where(validation_rmse == mint)
idx = idx[0]
idx[0]
a = i_array[idx[0]]
a
```

[13]:
```
[3, 'best']
```

[14]:
```
min(training_rmse)
```

[14]:
```
408.5393424964456
```

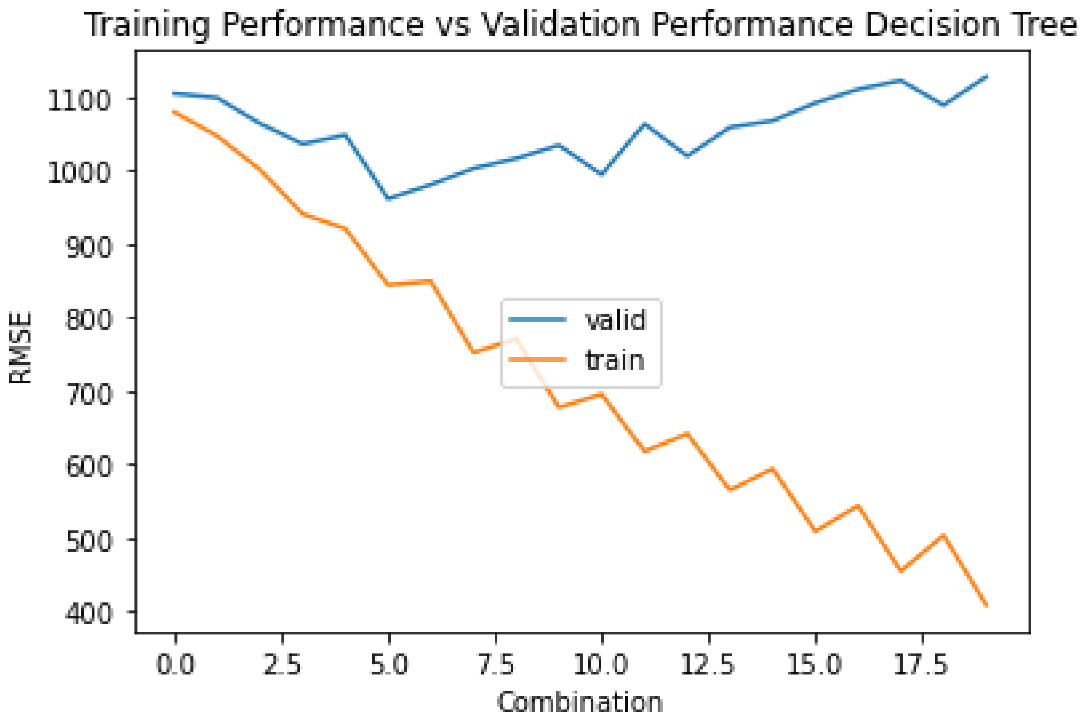
[15]:
```
min(validation_rmse)
```

[15]:
```
961.6016047640311
```

After finding the lowest average RMSE, we determined the best max_depth is 3 and the best splitter is "best."

[16]:
```
import matplotlib.pyplot as plt
# plotting training vs validation accuracies by max depth and splitter
plt.plot(range(20), validation_rmse, label = "valid")
plt.plot(range(20), training_rmse, label = "train")
plt.legend(loc="center")
plt.title("Training Performance vs Validation Performance Decision Tree")
plt.xlabel("Combination")
plt.ylabel("RMSE")
```

[16]:
```
Text(0, 0.5, 'RMSE')
```



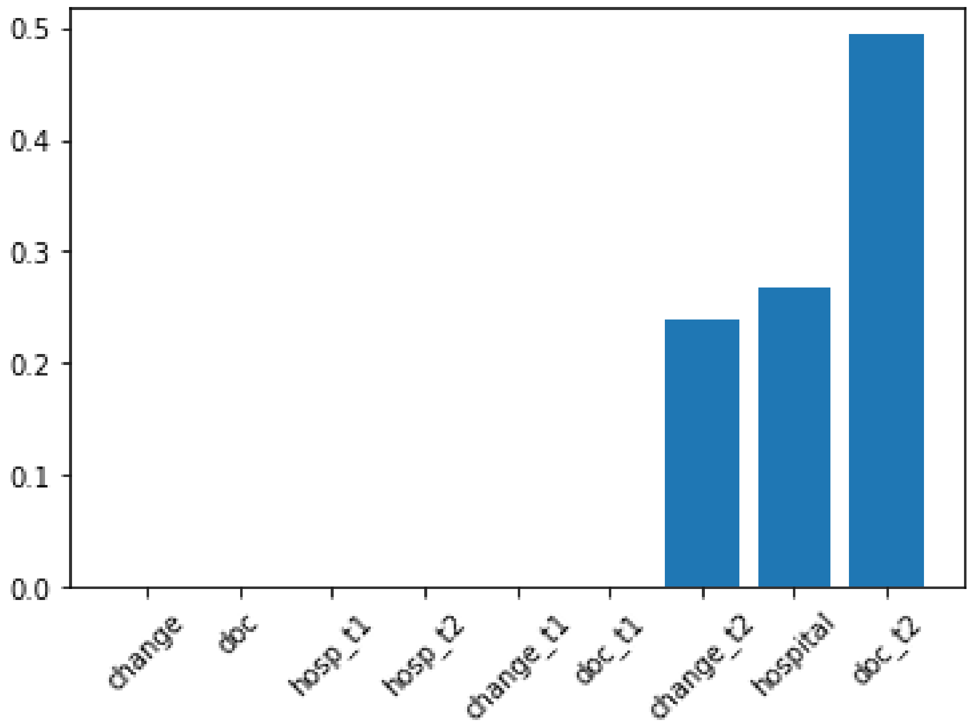
[17]:
```
# fitting final tree with tuned parameters
from sklearn.metrics import r2_score
final_tree = DecisionTreeRegressor(max_depth = 3, splitter = 'best',␣
↪random_state = 3)
final_tree.fit(X_train, y_train)
preds = final_tree.predict(X_test)
final_rmse = np.sqrt(mean_squared_error(y_test, preds))
r2 = r2_score(y_test, preds)
print("Testing RMSE for Tuned Tree is:", final_rmse)
print("R^2:", r2)
```

```
Testing RMSE for Tuned Tree is: 908.8574281943128
R^2: 0.5698735132912461
```

[19]:
```
final_tree.feature_importances_
```

[19]:
```
array([0.26762272, 0.        , 0.        , 0.        , 0.        ,
       0.        , 0.23847841, 0.        , 0.49389887])
```

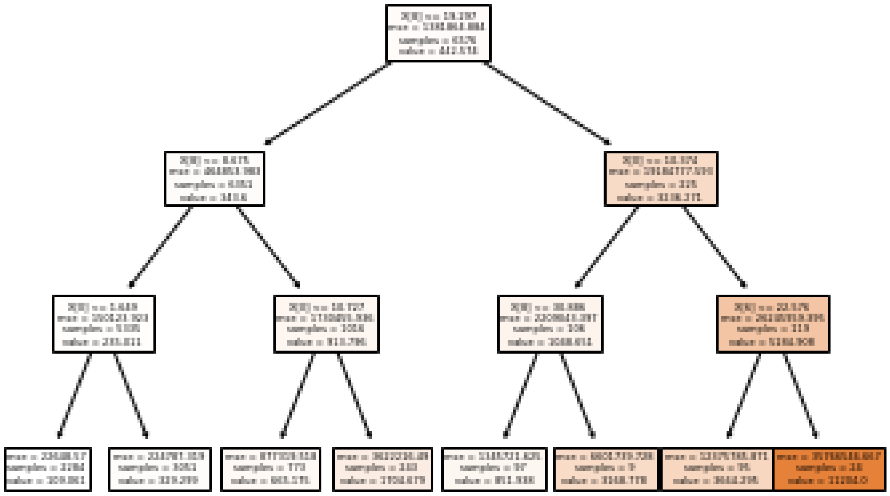
[20]:
```
idx = final_tree.feature_importances_.argsort()
plt.bar(X_train.columns[idx], final_tree.feature_importances_[idx])
plt.xticks(rotation = 45);
```



The feature importances represent the total reduction of the * criterion *

The most important feature is the estimated percentage of outpatient doctor visits primarily about COVID-related symptoms at time (t - 2).

[21]:
```
# plotting our final tree
from sklearn.tree import plot_tree
plt.figure()
plot_tree(final_tree, filled=True)
plt.show()
```

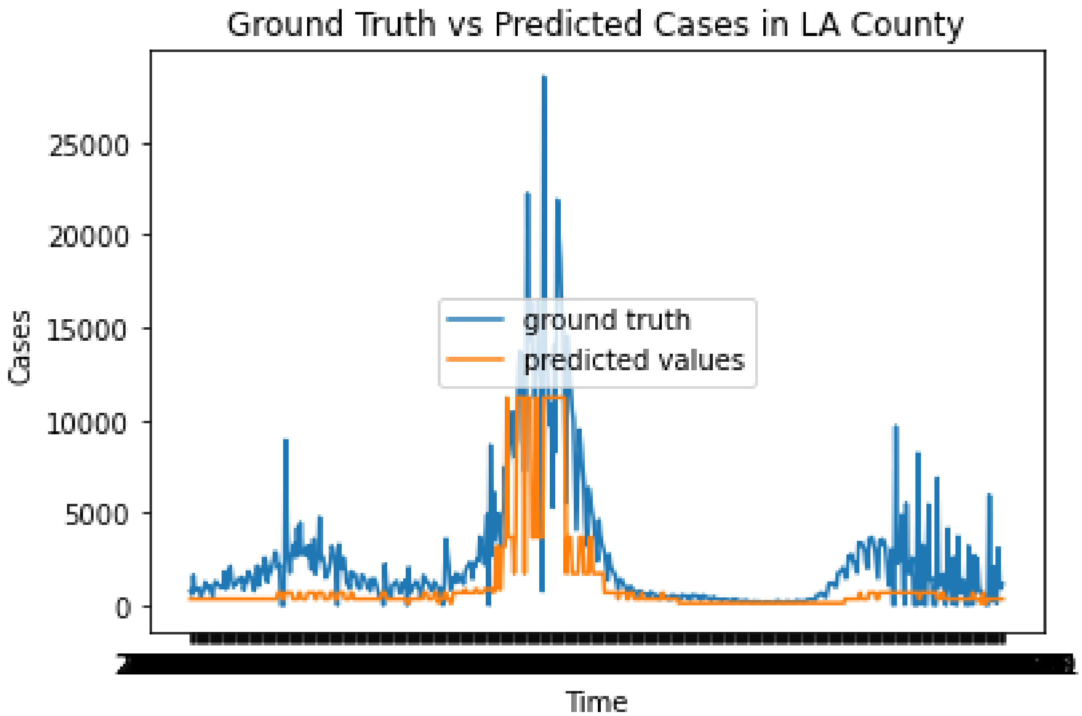


[22]:
```
# filtering to ground truth cases in LA county
county_cases = data2[data2['geo_value'] == 6037]['cases']
```

[23]:
```
# predicted number of cases in LA county
county = data2[data2['geo_value'] == 6037]
county = county.drop(columns = ['geo_value', 'cases'])
county_predicted = final_tree.predict(county)
```

[24]:
```
# plot of ground truth vs predicted cases in LA county
plt.plot(county_cases, label = "ground truth")
plt.plot(county_predicted, label = "predicted values")
plt.legend(loc="center")
plt.title("Ground Truth vs Predicted Cases in LA County")
plt.xlabel("Time")
plt.ylabel("Cases")
```

[24]:
```
Text(0, 0.5, 'Cases')
```



3 SVM

[25]:
```
# normalizing the data for preparation for SVM model
from sklearn import preprocessing
d = preprocessing.normalize(X)
scaled_df = pd.DataFrame(d, columns=["hospital", "change", "doc", "hosp_t1",␣
↪"hosp_t2", "change_t1", "change_t2", "doc_t1", "doc_t2"])
scaled_df.head()
```

[25]:

	hospital	change	doc	hosp_t1	hosp_t2	change_t1	change_t2	\
0	0.306269	0.281336	0.378123	0.373548	0.419326	0.265756	0.234686	
1	0.027840	0.454194	0.497864	0.028084	0.028483	0.400247	0.301665	
2	0.041084	0.168127	0.582957	0.052429	0.057012	0.177046	0.177468	
3	0.121480	0.382587	0.363704	0.121480	0.121480	0.344345	0.376461	
4	0.343410	0.197966	0.463250	0.310097	0.274892	0.172884	0.173829	

	doc_t1	doc_t2
0	0.349268	0.347863
1	0.413468	0.348294
2	0.561744	0.495794
3	0.403030	0.504327
4	0.429448	0.460458

[26]:
```
X_train, X_test, y_train, y_test = train_test_split(scaled_df,y,
                                                     test_size=0.2,␣
↪random_state=123,
                                                     shuffle=True)
```

[27]:
```
# attempting to reduce the data using PCA
from sklearn.decomposition import PCA
from sklearn.svm import SVR
# reduce data using pca
pca = PCA(n_components=1)
principalComponents = pca.fit_transform(X_train)
clf = SVR()
# fit reduced data to SVM
clf.fit(principalComponents, y_train)
```

[27]: SVR()

[28]:
```
# reduce test data
test_pca = pca.fit_transform(X_test)
# run model on reduced test data
preds = clf.predict(test_pca)
currAccuracy = mean_squared_error(y_test, preds)
currAccuracy = np.sqrt(currAccuracy)
print("The PCA SVM RMSE is:", currAccuracy)
```

```
The PCA SVM RMSE is: 1411.5100867101175
```

[29]:
```
# attempting to reduce the data using LDA
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis
model = LinearDiscriminantAnalysis(n_components=1)
model.fit(X_train, y_train)
# reduce data through LDA
lda_transformed = model.transform(X_train)
clf_svr = SVR()
# fit reduced data on SVM
clf_svr.fit(lda_transformed, y_train)
```

[29]: SVR()

[30]:
```
# reduce test data
test_lda = model.transform(X_test)
# run model on reduced test data
preds = clf_svr.predict(test_lda)
currAccuracy = mean_squared_error(y_test, preds)
currAccuracy = np.sqrt(currAccuracy)
print("The LDA SVM RMSE is:", currAccuracy)
```

```
The LDA SVM RMSE is: 1409.2917198391272
```

[31]:
```
model_svm = SVR()
model_svm.fit(X_train, y_train)
pred = model_svm.predict(X_test)
currAccuracy = mean_squared_error(y_test, pred)
currAccuracy = np.sqrt(currAccuracy)
print("The No Reduction SVM RMSE is:", currAccuracy)
```

```
The No Reduction SVM RMSE is: 1409.2897840279618
```

Saw no changes in RMSE when reducing the data through PCA and LDA. Therefore we will stick to an SVM model with no reduction methods.

[32]:
```
# tuning the hyperparameter for SVM model
from sklearn.model_selection import cross_val_score
from sklearn.svm import SVR
kernel = ['linear', 'poly', "rbf", 'sigmoid']
training_mse_svm = []
validation_mse_svm = []
i_array_svm = []

for j in kernel:
    svm = SVR(kernel = j)
    svm.fit(X_train, y_train)
    scores = cross_validate(svm, X_train, y_train, cv = 5, scoring =␣
↪'neg_mean_squared_error', return_train_score=True)
    training_mse_svm.append(np.mean(scores["train_score"]))
    validation_mse_svm.append(np.mean(scores['test_score']))
    i_array_svm.append([j])

print("Average Training MSEs:", training_mse_svm)
print("Average Validation MSEs:", validation_mse_svm)
```

```
Average Training MSEs: [-1463963.5679282534, -1419846.9080660832,
-1437436.0610767617, -1478195.520902185]
Average Validation MSEs: [-1464017.7139960553, -1420295.5622715442,
-1437581.9039549076, -1478226.238298564]
```

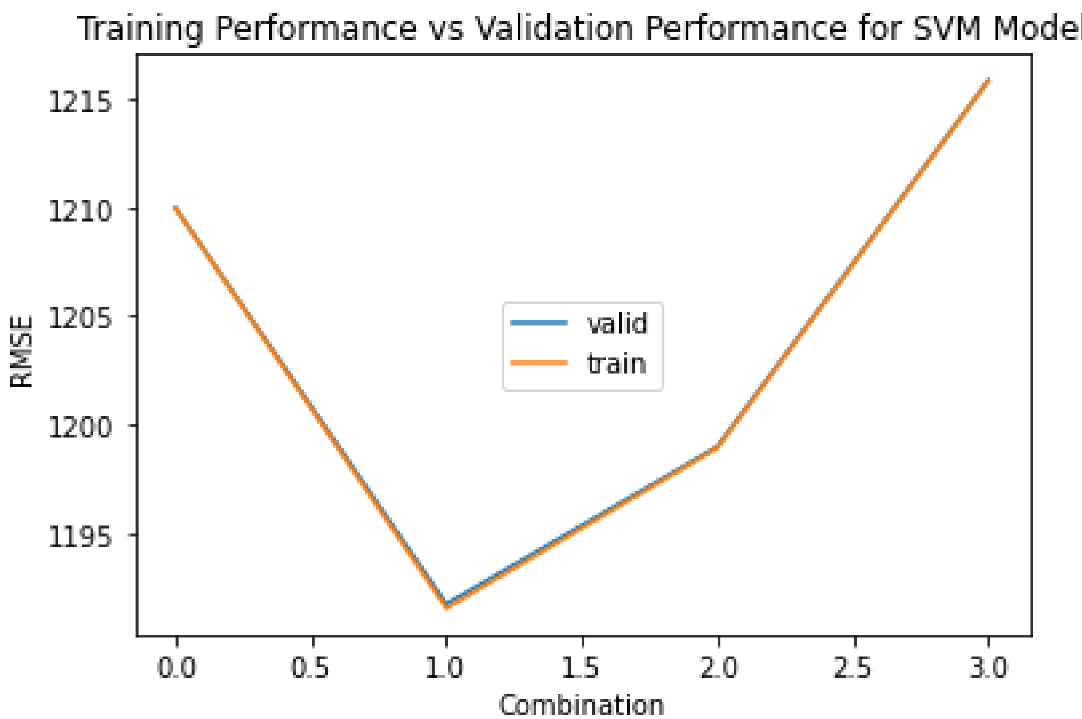
[33]:
```
abs_training_mse = np.abs(training_mse_svm)
abs_validation_mse = np.abs(validation_mse_svm)
training_rmse = np.sqrt(abs_training_mse)
validation_rmse = np.sqrt(abs_validation_mse)
mint = min(validation_rmse)
best_kernel_idx = np.where(validation_rmse == mint)
best_kernel_idx = best_kernel_idx[0][0]
best_kernel = kernel[best_kernel_idx]

print("The Best Kernel is:", best_kernel)
print("The Validation RMSE for the Best Kernel is:", mint)
```

```
The Best Kernel is: poly
The Validation RMSE for the Best Kernel is: 1191.7615375030125
```

[34]:
```
import matplotlib.pyplot as plt
# plotting training vs validation accuracies by kernel
plt.plot(range(4), validation_rmse, label = "valid")
plt.plot(range(4), training_rmse, label = "train")
plt.legend(loc="center")
plt.title("Training Performance vs Validation Performance for SVM Model")
plt.xlabel("Combination")
plt.ylabel("RMSE")
```

[34]:
```
Text(0, 0.5, 'RMSE')
```



[35]:
```
# fitting final model with tuned parameters
final_svm = SVR(kernel="poly")
final_svm.fit(X_train, y_train)
preds = final_svm.predict(X_test)
currAccuracy = mean_squared_error(y_test, preds)
currAccuracy = np.sqrt(currAccuracy)
r2 = r2_score(y_test, preds)
print("The Final SVM RMSE is:", currAccuracy)
print("R^2:", r2)
```

```
The Final SVM RMSE is: 1403.1987097003807
R^2: -0.02528125628862421
```

[36]:
```
# filtering to ground truth cases in LA county
county_cases = data2[data2['geo_value'] == 6037]['cases']
```

[37]:
```
# predicting number of cases in LA county using SVM model
county = data2[data2['geo_value'] == 6037]
county = county.drop(columns = ['geo_value', 'cases'])
county = preprocessing.normalize(county)
county_predicted = final_svm.predict(county)
```

[38]:
```
# plotting ground truth vs predicted cases in LA county
plt.plot(county_cases, label = "ground truth")
plt.plot(county_predicted, label = "predicted values")
plt.legend(loc="center")
plt.title("Ground Truth vs Predicted Cases in LA County")
plt.xlabel("Time")
plt.ylabel("Cases")
```

[38]:
```
Text(0, 0.5, 'Cases')
```