mlp-week09

March 24, 2021

1 Machine Learning in Python - Workshop 9

In this week's workshop we will be returning to the NYC Parking Ticket data and exploring how to score and evaluate multiclass classification models as well as trying several addition modeling approaches for this type of data.

2 1. Setup

2.1 1.1 Packages

In the cell below we will load the core libraries we will be using for this workshop and setting some sensible defaults for our plot size and resolution.

[1]:
```
# Data libraries
import pandas as pd
import numpy as np

# Plotting libraries
import matplotlib.pyplot as plt
import seaborn as sns

# Plotting defaults
plt.rcParams['figure.figsize'] = (10,6)
plt.rcParams['figure.dpi'] = 80

# sklearn modules
import sklearn
import sklearn.linear_model
import sklearn.tree
import sklearn.ensemble
import sklearn.neighbors
import sklearn.preprocessing

from sklearn.pipeline import make_pipeline
```

2.2 1.2 Helper Functions

[2]:
```python
def plot_boundaries(bounds, x='lon', y='lat', group='precinct', n=5):
    """ Draws boundary lines for a series of groups polygons in a dataframe
    """
    sns.lineplot(x=x, y=y, hue=group, data=bounds,
                 sort=False, palette=['k']*n, legend=None)
```

[3]:
```python
def plot_pred_labels(res, pred = 'pred_label', truth = 'precinct'):
    """ Plots the predicted labels and true labels from a common data frame
    """
    plt.figure(figsize=(5,8))

    ax = sns.scatterplot(
        x='lon', y='lat', hue=pred, palette=precinct_pal, data=res
    )
    plot_boundaries(manh_bounds)

    acc = sklearn.metrics.accuracy_score(
        res[truth], res[pred]
    )

    ax.set_title("Predicted Labels (Accuracy {:.3f})".format(acc))

    plt.show()
```

2.3 1.3 Data

As described in the Week 7 workshop, these data comes from New York City's Open Data project. We have simplified the data somewhat and restricted the data to just include the five precincts (1st, 5th, 6th, 7th, and 9th) in the southern end of Manhattan. The following data files have been provided:

- `manh_tickets.csv` - Geocoded parking tickets from the 5 southern most precincts in Manhattan
- `manh_test.csv` - Points randomly sampled within the true boundaries of these precincts
- `manh_bounds.csv` - boundaries of these precincts

As before, our goal is to use these parking tickets to develop a model which correctly predicts the boundaries of the police precincts in Manhattan based only on the locations where parking tickets have been issued. We will read in all the data sets using pandas,

[4]:
```python
manh_tickets = pd.read_csv("manh_tickets.csv")
manh_test    = pd.read_csv("manh_test.csv")
manh_bounds  = pd.read_csv("manh_bounds.csv")
```

and create our basic response vector and model matrix,

```
[5]: X = manh_tickets[['lon','lat']]
y = manh_tickets.precinct
```

Finally, we create labels and a color palette which will be used for across subsequent plots.

```
[6]: precincts = ['Precinct01', 'Precinct05', 'Precinct06', 'Precinct07',␣
 ↪'Precinct09']
precincts_short = ['P01', 'P05', 'P06', 'P07', 'P09']
precincts_pred = ["pred_" + p for p in precincts_short]

# Create a color palette for precincts based on the cols25 palette from R's␣
 ↪pals package
precinct_pal = dict(
    zip(precincts,
        [(0.1215686, 0.47058824, 0.7843137), (1.0000000, 0.00000000, 0.0000000),
         (0.2000000, 0.62745098, 0.1725490), (0.4156863, 0.20000000, 0.7607843),
         (1.0000000, 0.49803922, 0.0000000) ]
    )
)
```

Using our new palette we can plot the original parking ticket data and add the precinct boundaries using `plot_boundaries` to make everything more readable.

```
[7]: # plt.figure(figsize=(5,8))

# sns.scatterplot(
#     x='lon', y='lat', hue='precinct', palette=precinct_pal, data=manh_tickets
# ).set_title("Parking Tickets")
# plot_boundaries(manh_bounds)

# plt.show()
```

---

3 1.4. Review - Multiclass logistic (multinomial) regression

At the end of the Week 7 workshop we had fit a Logistic Regression model using `multi_class=multinomial` in order to obtain a probabilistically consistent predictive model, e.g. for any given prediction the probabilities of each class sum to one. We will begin by re-constructing this same model and using it as a point of comparison.

```
[8]: m_mn = make_pipeline(
    sklearn.preprocessing.StandardScaler(),
    sklearn.linear_model.LogisticRegression(penalty='none',␣
 ↪multi_class='multinomial')
).fit(X, y)
```

This model is then used to predict both the class label for test location as well as the predicted probability for each precinct for all test locations, these results are then stored in the `res_mn` data frame.

[9]:
```python
res_mn = pd.concat(
    [ manh_test,
      pd.Series(
          data = m_mn.predict(manh_test[['lon','lat']]),
          name = "pred_label"
      ),
      pd.DataFrame(
          data = m_mn.predict_proba(manh_test[['lon','lat']]),
          columns = precincts_pred
      )
    ],
    axis = 1
)

res_mn
```

[9]:
```
        precinct        lon        lat  pred_label  pred_P01  pred_P05  \
0     Precinct01 -74.010330  40.720485  Precinct01  0.996715  0.002760
1     Precinct01 -74.005740  40.708015  Precinct01  0.878555  0.121173
2     Precinct01 -74.001993  40.707750  Precinct05  0.465160  0.523614
3     Precinct01 -74.005499  40.706179  Precinct01  0.837832  0.161541
4     Precinct01 -74.009585  40.726523  Precinct01  0.926785  0.001993
…            …          …          …          …         …         …
2495  Precinct09 -73.993883  40.725202  Precinct05  0.030168  0.439143
2496  Precinct09 -73.988460  40.729783  Precinct09  0.000017  0.002933
2497  Precinct09 -73.984370  40.725988  Precinct09  0.000001  0.002990
2498  Precinct09 -73.989221  40.723938  Precinct09  0.000641  0.140995
2499  Precinct09 -73.990723  40.728558  Precinct09  0.000381  0.021947

          pred_P06      pred_P07      pred_P09
0     5.245982e-04  2.601684e-08  6.066956e-09
1     1.312997e-07  2.714364e-04  1.333554e-09
2     2.073632e-07  1.122583e-02  4.409586e-08
3     3.234373e-08  6.272248e-04  6.308147e-10
4     7.122160e-02  7.586284e-09  3.263365e-07
…              …             …             …
2495  1.913318e-01  2.258760e-02  3.167692e-01
2496  2.501241e-02  1.313258e-03  9.707246e-01
2497  3.716451e-04  3.438030e-02  9.622564e-01
2498  7.625062e-03  1.483359e-01  7.024028e-01
2499  9.987098e-02  3.418434e-03  8.743825e-01

[2500 rows x 9 columns]
```

3.0.1 Exercise 1

Pick at least three random test locactions and using `res_mn` verify that the predicted label for each point corresponds to the class with the largest predicted probability and that the predicted probabilities are consistent (e.g. they add up to 1).

[10]:
```
a = np.random.randint(0,2499,3)
```

Initially `a = [148, 1985, 357]`,

[11]:
```
for i in a:
    print(i, "\n" , res_mn.iloc[i, :], "\n")
```

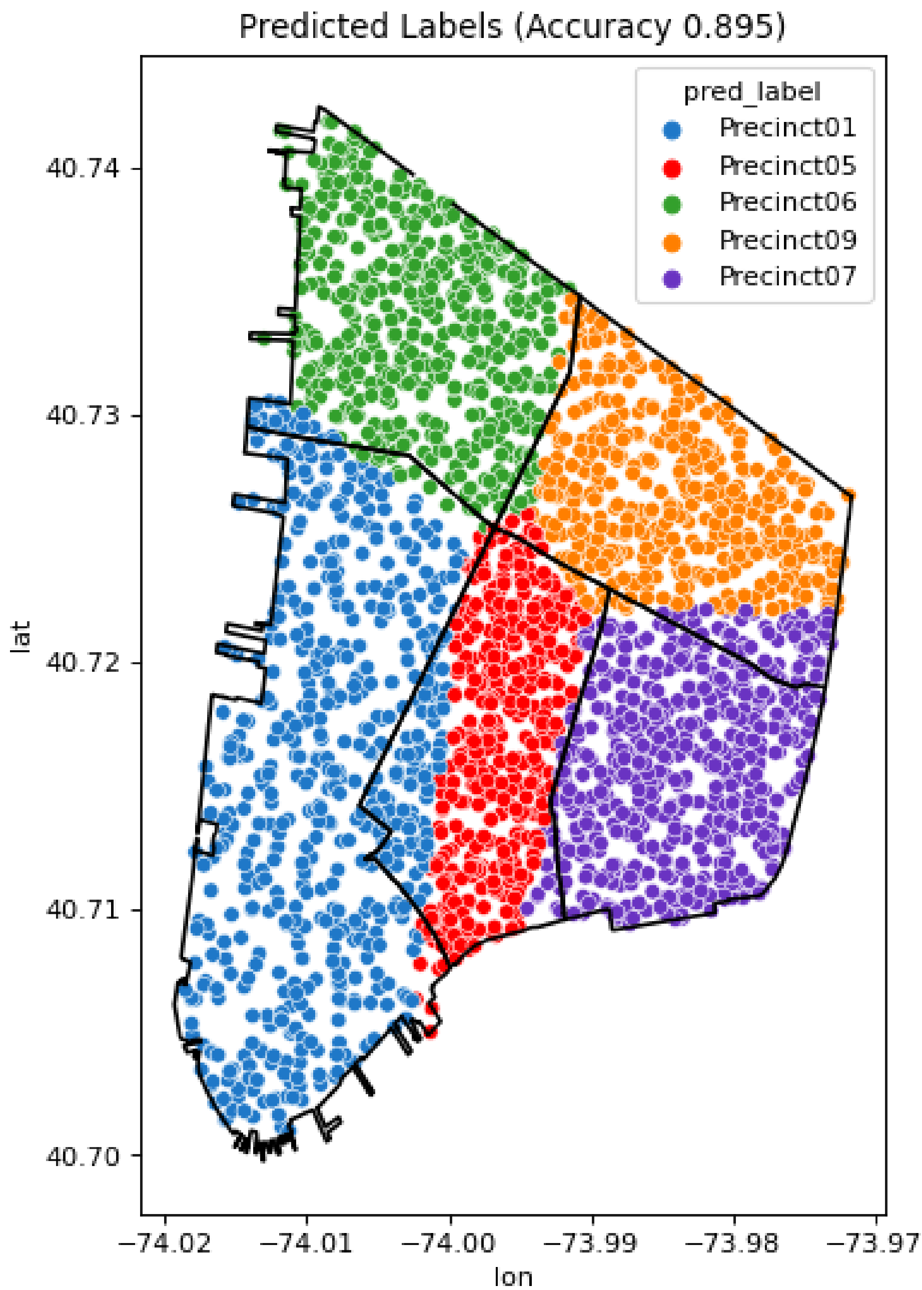
```
1380
 precinct       Precinct06
lon               -74.0024
lat                40.7337
pred_label      Precinct06
pred_P01        0.00390627
pred_P05       0.000199189
pred_P06          0.995674
pred_P07       1.02552e-08
pred_P09        0.00022095
Name: 1380, dtype: object

1412
 precinct       Precinct06
lon               -74.0037
lat                40.7285
pred_label      Precinct06
pred_P01          0.256164
pred_P05         0.0113583
pred_P06          0.732262
pred_P07       8.70555e-07
pred_P09       0.000215169
Name: 1412, dtype: object

1536
 precinct       Precinct07
lon               -73.9787
lat                40.7109
pred_label      Precinct07
pred_P01       3.67959e-10
pred_P05       0.000111303
pred_P06       6.39586e-12
pred_P07          0.999825
```

```
pred_P09       6.3998e-05
Name: 1536, dtype: object
```

We can clearly see that each point is labelled correctly corresponding to the highest probability.

We can also use the `plot_pred_labels` function, defined witht the helper functions above, to visualize the predicted labels.

[12]:
```
plot_pred_labels(res_mn)
```



4 2. Additional scoring methods / tools for multiclass models

4.1 2.1 Confusion Matrix

As we saw in week 7, perhaps the most straight forward approach to assess a model is to use the label predictions directly to construct a confusion matrix which places the true labels along the rows and the predicted labels along the columns.

[13]:
```
sklearn.metrics.confusion_matrix(res_mn.precinct, res_mn.pred_label)
```

[13]:
```
array([[473,  15,  12,   0,   0],
       [ 86, 379,   0,  27,   8],
       [ 19,   0, 476,   0,   5],
       [  0,   0,   0, 496,   4],
       [  0,   7,  10,  69, 414]])
```

4.1.1 Exercise 4

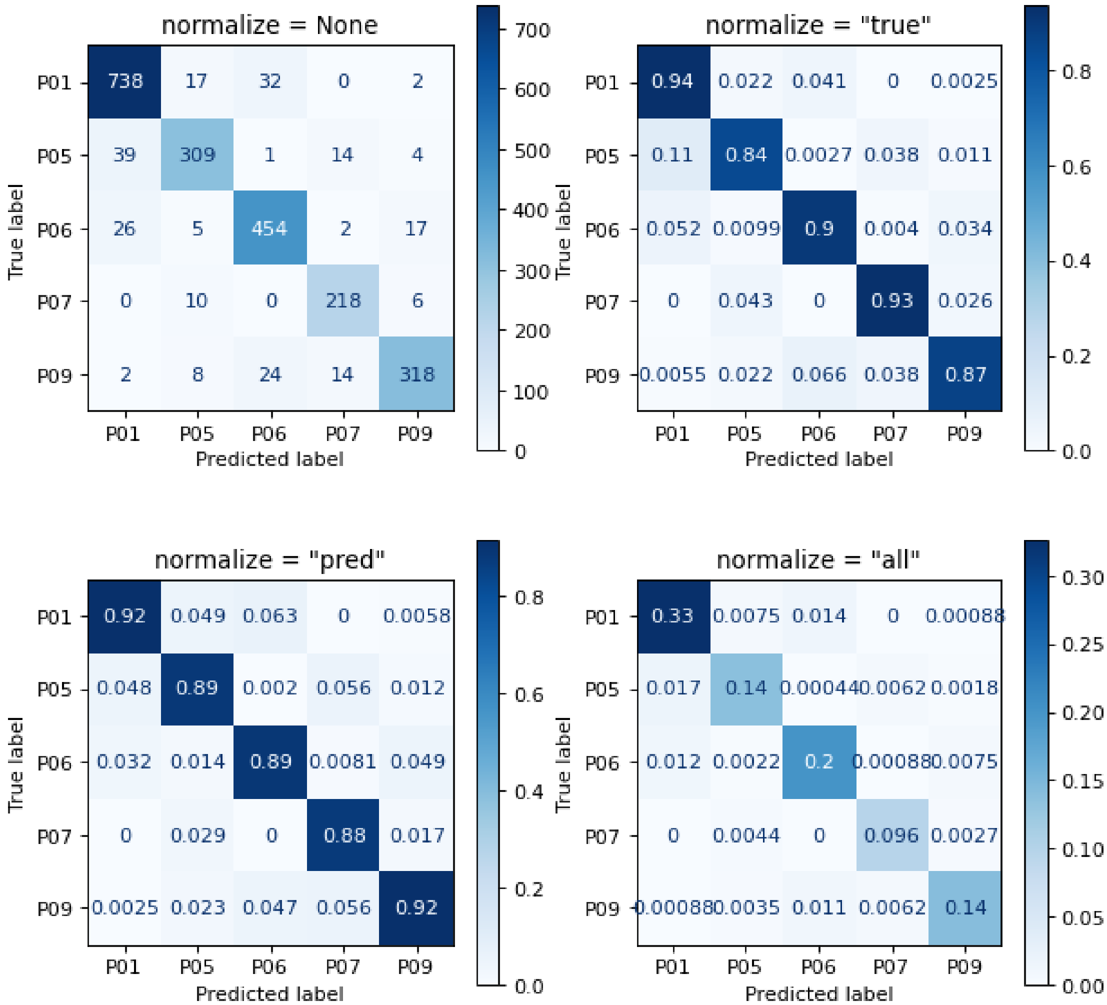
Explain what information is given by the value 27 given in the 2nd row, 5th column of this confusion matrix.

This tells us that 27 points were classified by our model as lying in precinct 5 while they actually lie in precinct 7.

To expand on this idea, and to help scale this approach to cases when there are many classes sklearn provides a convenient plotting function for visualizing the confusion matrix which provides methods for normalizing by the rows ("true"), columns ("pred"), or all values.

[14]:
```
fig = plt.figure(figsize=(9, 9))
normalize=[None, 'true', 'pred', 'all']

for i in range(len(normalize)):
    ax = fig.add_subplot(221+i)

    if (normalize[i] is None):
        ax.set_title("normalize = None")
    else:
        ax.set_title("normalize = \"" + normalize[i] + "\"")
```

```
    sklearn.metrics.plot_confusion_matrix(
        m_mn, X, y, include_values=True, ax=ax,
        normalize = normalize[i],
        display_labels = precincts_short,
        cmap = plt.cm.Blues
    )
```



4.1.2 Exercise 2

Which normalization method appears to be is most useful for determining False Positives, what about False Negatives? Explain.

A False Positive is our model predicting that the location is on the diagonal when the location lies in another precinct. Therefore with the lowest probability off the diagonal. this is given by the lighest shading off the diagonal.

A False Negative occurs when the model predicts that the ticket doesn't lie on the diagonal, when in reality it does. That is, our model is predicting that the ticket lies in a different precinct than the diagonal, however it does lie in that precinct. Therefore with the …

4.2 2.2 One-vs-rest ROC curves

Previously in the case of logistic regression (binary classification) we saw how this enabled us to consider all possible decision threshold values to generate a receiver operating characteristic (ROC) curve showing the trade off between possible true positive and false positive rates.

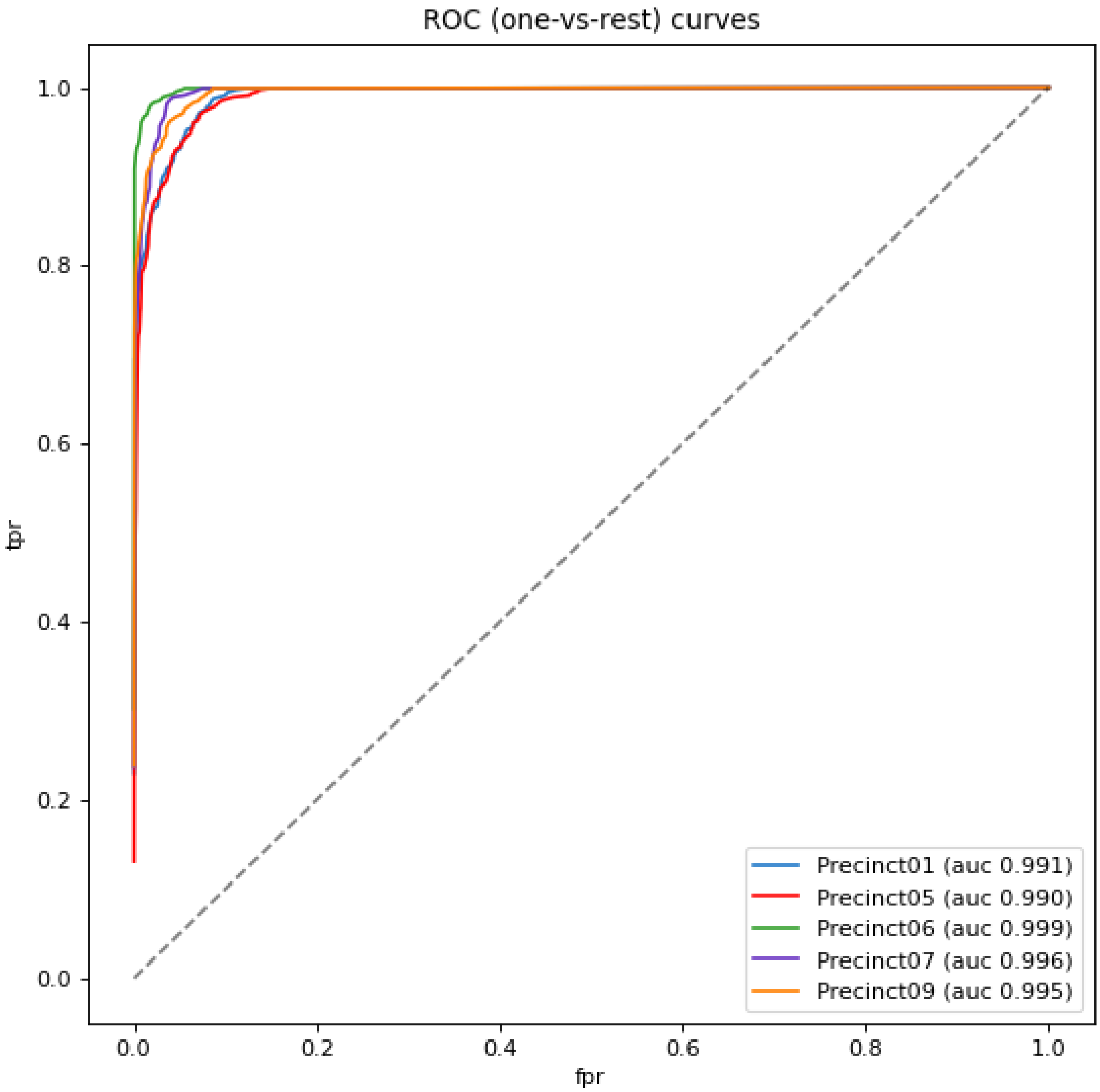
In the case of a classification model that produces probabilistic predictions we can extend this idea to the multiclass case but it is not without some significant drawbacks. Specifically, the concept of a decision threshold does not make sense in the multiclass setting since we need to make a choice among k classes. However, what we can do is consider a one-vs-rest approach where we take each class as the positive case and all other classes as the negative. In this way we can construct k different ROC curves using our original predicted probabilities. The function below implements this considering each class separately and calculating a unique ROC and AUC for each.

[15]:
```
def ovr_roc_plot(y_true, y_pred):
    """ Draw ROC curves using one-vs-rest approach
    """

    classes = y_true.unique()
    n_classes = len(y_true.unique())

    # Convert from n x 1 categorical matrix to n x k binary matrix
    y_true = pd.get_dummies(y_true).to_numpy()

    y_pred = y_pred.to_numpy()

    # Sanity Check
    if y_true.shape[1] != y_pred.shape[1]:
        raise ValueError("Truth and prediction dimensions do not match.")

    # Compute ROC curve and ROC area for each class
    rocs = dict()
    aucs = dict()
    for name, i in zip(classes, range(n_classes)):
        aucs[i] = pd.DataFrame({
            'precinct': [name],
            'auc': [sklearn.metrics.roc_auc_score(y_true[:, i], y_pred[:, i])]
        })

        rocs[i] = pd.DataFrame(
```

```
            data = np.c_[sklearn.metrics.roc_curve(y_true[:, i], y_pred[:, i])],
            columns = ('fpr', 'tpr', 'threshold')
        ).assign(
            precinct = name
        )

    # Bind rows to create a single data frame for each
    roc = pd.concat(rocs, ignore_index=True)
    auc = pd.concat(aucs, ignore_index=True)

    # Create plot
    fig = plt.figure(figsize=(8, 8))
    sns.lineplot(x='fpr', y='tpr', hue='precinct', data=roc, ci=None,␣
↪palette=precinct_pal)

    plt.plot([0,1],[0,1], 'k--', alpha=0.5) # 0-1 line
    plt.title("ROC (one-vs-rest) curves")

    L = plt.legend()
    for precinct, auc_val, i in zip(auc.precinct, auc.auc, range(n_classes)):
        L.get_texts()[i].set_text("{} (auc {:.3f})".format(precinct, auc_val))

    plt.show()

    # Return the AUCs as a Data Frame
    return(auc)
```

We can then use this function to evaluate our class probability predictions.

[16]:
```
ovr_roc_plot(res_mn.precinct, res_mn[precincts_pred])
```



[16]:

	precinct	auc
0	Precinct01	0.991378
1	Precinct05	0.989793
2	Precinct06	0.998792
3	Precinct07	0.995503
4	Precinct09	0.994983

4.2.1 Exercise 3

Does the ordering of the AUC values agree with your intuition about the models performance for the different precincts?

From the figure, we see that the model is predicting "best" the precincts (in order) 1,5,6,7,9. Which we would not agree with generally, since it appears to be predicting precincts 6 and 7 pretty well. While the model looks to be doing a bad job with precincts 1 and 5, since they have a lot of overlapping locations. This is a surprising result, also given the auc scores are so high.

4.2.2 Exercise 4

We have already established that this is reasonable for predicting most of these police precincts (accuracy of 0.895), however the AUCs reported for our model look very very good (all are >0.99). Can you explain this discrepancy?

Hint - look at the definition of TPR and FPR and then think about what happens when we use the one-vs-rest approach.

From above we have that one-vs-rest performs the following action,

"" we take each class as the positive case and all other classes as the negative. In this way we can construct different ROC curves using our original predicted probabilities. The function below implements this considering each class separately and calculating a unique ROC and AUC for each.""

and that,

$$\text{TPR} = \frac{\text{TP}}{\text{TP} + \text{FN}}$$

$$\text{FPR} = \frac{\text{FP}}{\text{FP} + \text{TN}}$$

Since the one-vs-rest approach considers only one class as positive and the rest as negative, the False Negatives are much smaller than the True Positives resulting in a high TPR. Also the True Negatives are much larger than the False Positives, resulting in a very low FPR. These two factors give a very good roc curve.

---

5 3. Other multiclass classification models

For this section we will explore a number approaches to fitting and predict multiclass classification models. For the sake of uniformity we will assess the models in the same way:

- Plot all of the predicted labels and the true labels for a visual comparison and
- Show the `classification_report` result for the predicted labels.

To avoid repeated code we will use the following helper function,

[17]:
```python
def model_test_assess(model, inc_report = True, inc_proba = False):
    # Use the model to predict test labels
    res = pd.concat(
        [ manh_test,
          pd.Series(
              data = model.predict(manh_test[['lon','lat']]),
              name = "pred_label"
          )
        ],
        axis = 1
    )

    if (inc_proba):
        res = pd.concat(
            [ res,
              pd.DataFrame(
                  data = model.predict_proba(manh_test[['lon','lat']]),
                  columns = precincts_pred
              )
            ],
            axis = 1
        )

    # Plot labels
    plot_pred_labels(res)

    if inc_report: # Print report
        print(
            sklearn.metrics.classification_report(
                res.precinct, res.pred_label,
                zero_division = 0
            )
        )
```

5.1 3.1. Support Vector Machines

5.1.1 3.1.1 SVC

Standard Support Vector Classifier models are able to handle multiclass outcome vectors by fitting all of the one-to-one SVC models for each pair of classes. This therefore involves fitting $\binom{k}{2} = k(k-1)/2$ SVC models which can be slow. The labels are predicted based on using all $\binom{k}{2}$ generate binary predictions which are then tabulated and the label with the most votes is choosen. See here for more details.

Below we define a function for fitting and assess this model using different kernel and penalty values. As SVMs can be sensitive to scale of the features we include a pipeline thats scales the latitude and logitude values before fitting the model.

```
[18]: def fit_svc(kernel, C):
          m_svc = make_pipeline(
              sklearn.preprocessing.StandardScaler(),
              sklearn.svm.SVC(C=C, kernel=kernel)
          ).fit(X,y)

          model_test_assess(m_svc)
```

5.1.2 Exercise 5

Using this function try different values of `kernel` and `C`, what seems to produce the best model? Explain why you think this model is performing better than the alternatives.

Hint - recommended kernels to try include `rbd`, `poly`, and `linear`.

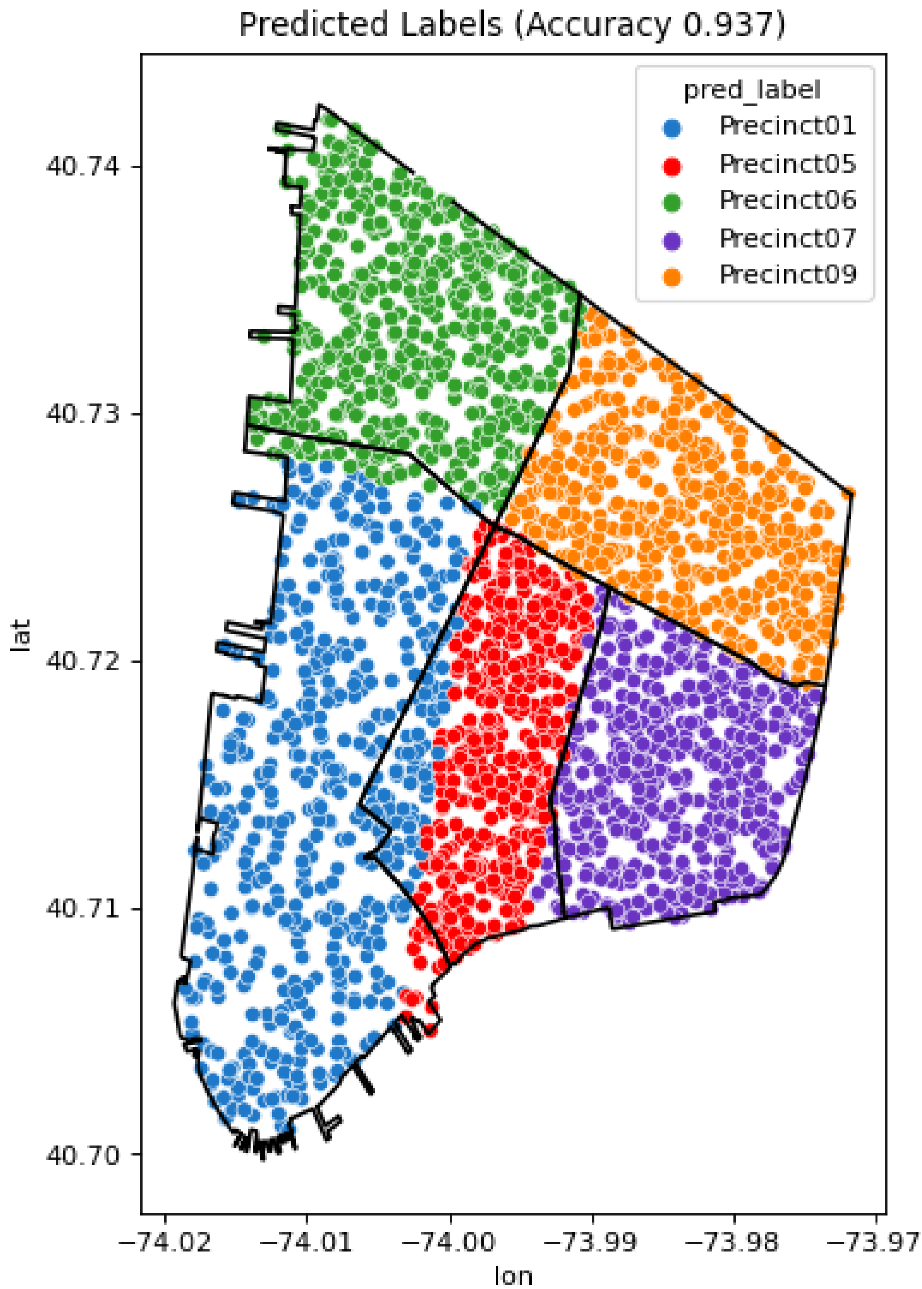
The rbf kernel: - `c=0.1` => 0.912 - `c=1` => 0.908 - `c=2` => 0.919 - `c=5` => 0.920 - `c=10` => 0.916 - `c=50` => 0.898

The linear kernel: - `c=0.1` => 0.910 - `c=1` => 0.925 - `c=2` => 0.931 - `c=5` => 0.935 - `c=10` => 0.936 - `c=50` => 0.937 - `c=100` => 0.937

The poly kernel: - `c=0.1` => 0.887 - `c=1` => 0.900 - `c=2` => 0.904 - `c=5` => 0.910 - `c=10` => 0.910 - `c=50` => 0.910 - `c=100` => 0.912 - `c=1000` => 0.914

We would prefer the linear kernel with a value of c around 50, as increases in c make no change to the accuracy above 50.

```
[19]: fit_svc(kernel = "linear", C=50)
```



	precision	recall	f1-score	support
Precinct01	0.85	0.91	0.88	500
Precinct05	0.94	0.81	0.87	500
Precinct06	0.94	0.99	0.97	500
Precinct07	0.96	0.99	0.97	500
Precinct09	0.99	0.98	0.98	500

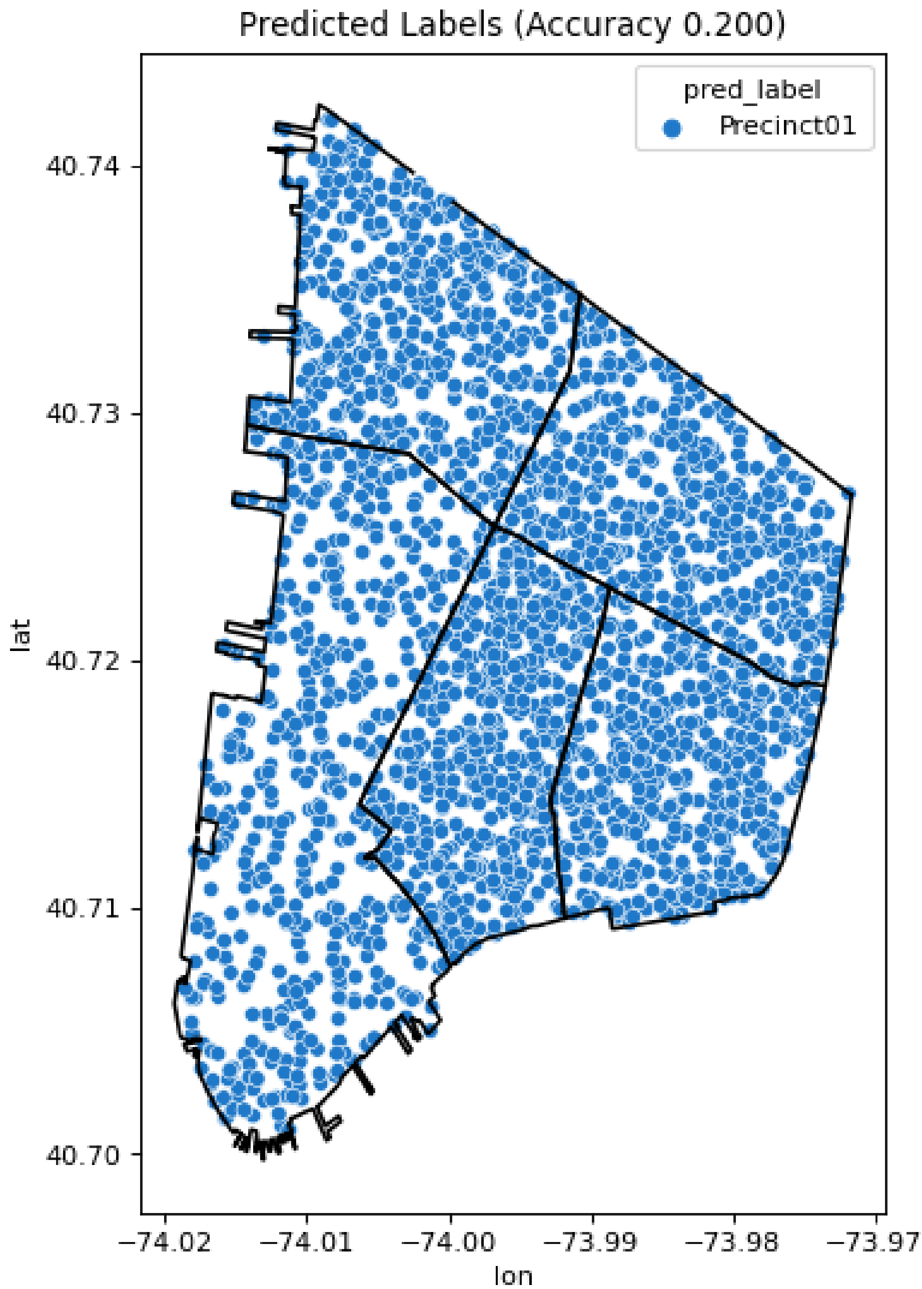
```
    accuracy                           0.94      2500
   macro avg       0.94      0.94      0.94      2500
weighted avg       0.94      0.94      0.94      2500
```

5.1.3 Exercise 6

Comment out the line of code that includes the `StandardScaler` in the pipeline below. What happens to the models predictive performance? Try adjusting `C` and or `kernel` to see if you can improve things.

In this case, the model's performance reduces dramatically to around 0.717 accuracy from above 0.9 in the example above. The polynomial kernel predicts every location as being in precinct 1.

[20]:
```
m_svc2 = make_pipeline(
    #sklearn.preprocessing.StandardScaler(),
    sklearn.svm.SVC(C=100, kernel="poly")
).fit(X,y)

model_test_assess(m_svc2, inc_report=False)
```



5.1.4 3.1.2 `LinearSVC`

An alternative to the one-vs-one behavior of `SVC` is to instead fit a one-vs-rest model, in sklearn this is only supported by the `LinearSVC` classifier model. This modeling approach needs to only fit k SVM models, and is therefore usually much faster for large k. However, due to implementation details of the underlying fitting library it does not support non-linear kernels. As with other SVM models it is important to scale our data before fitting.

[21]:
```
def fit_lsvc(C):
    m_lsvc = make_pipeline(
        sklearn.preprocessing.StandardScaler(),
        sklearn.svm.LinearSVC(C=C, max_iter=5000)
    ).fit(X,y)

    model_test_assess(m_lsvc)
```

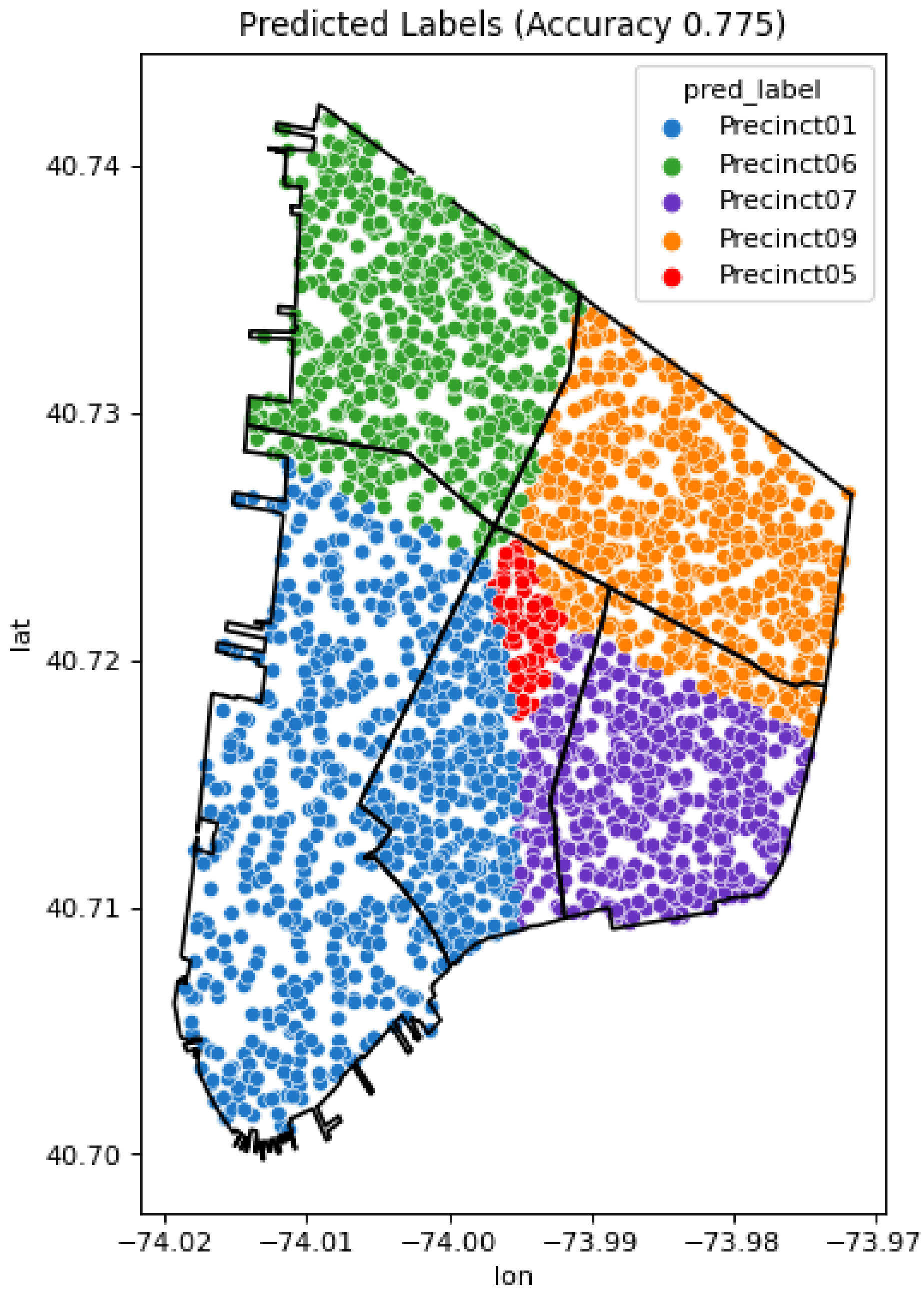
5.1.5 Exercise 7

Using this function try different values of `C` to tune the model, how does its performance compare to the `SVC` model?

We try the following values of c: - `c=0.01` => 0.775 - `c=0.1` => 0.813 - `c=1` => 0.811 - `c=2` => 0.812 - `c=5` => 0.812 - `c=10` => 0.813 - `c=50` => 0.812

Different values of C do not change the accuracy a great deal, and the accuracy of this model is much worse than for the `SVC` model.

[22]:
```
fit_lsvc(C=0.01)
```



	precision	recall	f1-score	support
Precinct01	0.60	0.91	0.73	500
Precinct05	1.00	0.14	0.25	500
Precinct06	0.89	1.00	0.94	500
Precinct07	0.82	0.84	0.83	500
Precinct09	0.82	0.98	0.89	500

```
    accuracy                           0.78      2500
   macro avg       0.83      0.78      0.73      2500
weighted avg       0.83      0.78      0.73      2500
```

5.2 3.2 Tree based methods

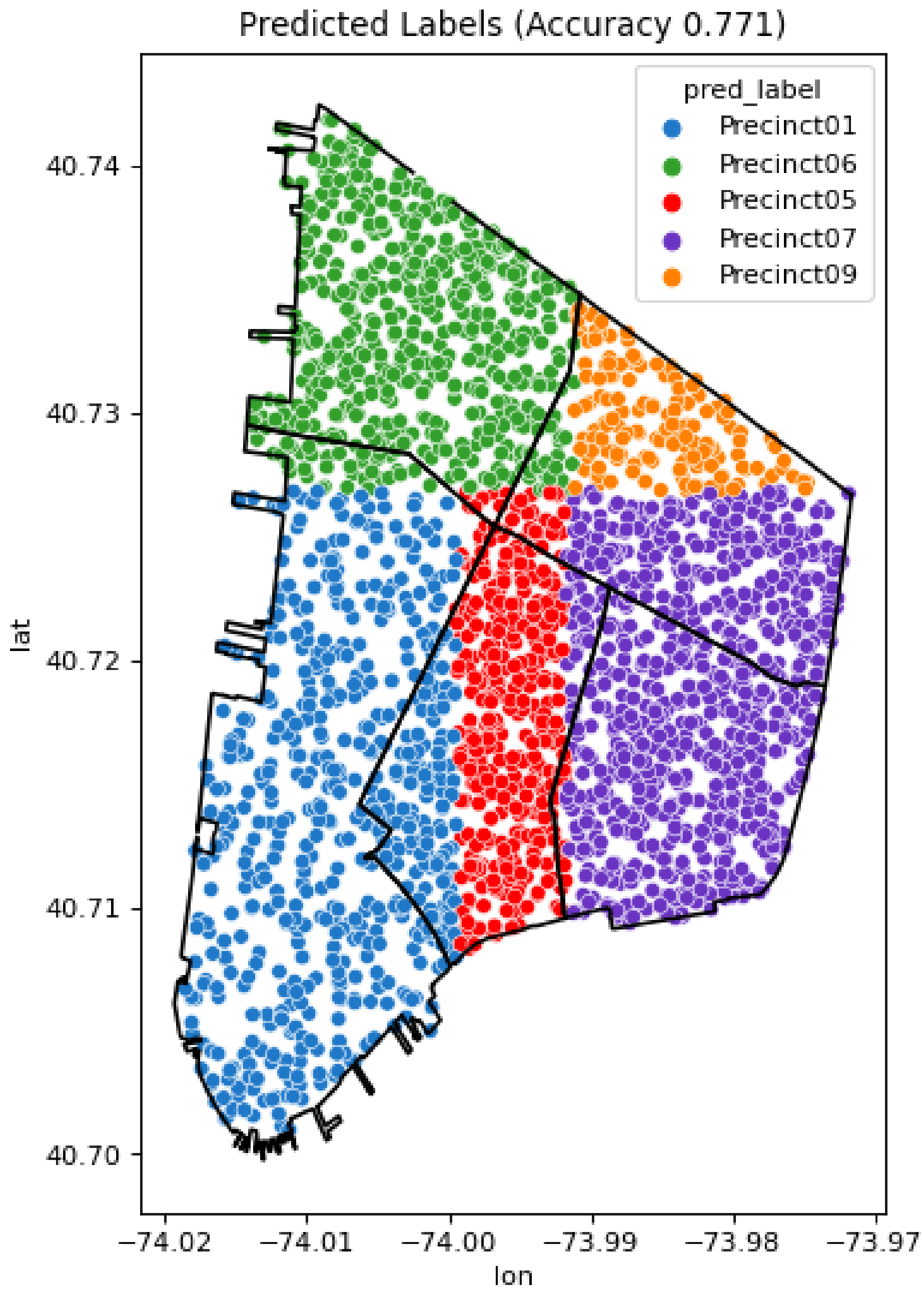
5.2.1 3.2.1 DecisionTreeClassifier

This model fits a decision tree model that attempts to classify observations by constructing a binary decision tree on the features provided. In the case of binary and multiclass classification the predicted label is based on the most common label within the terminal node. For fitting this model for these data we will solely focus on the use of the `max_depth` parameter which determines the number of branchs within the tree. Keep in mind that each additional layer added to the tree potentially increases the number of nodes by a factor of 2.

[23]:
```
def fit_dt(max_depth):
    m_dt = sklearn.tree.DecisionTreeClassifier(
        max_depth = max_depth
    ).fit(X,y)

    model_test_assess(m_dt)
```

Use the following code chunk to explore the effect of different max depths on the tree model.

[24]:
```
fit_dt(max_depth = 3)
```



	precision	recall	f1-score	support
Precinct01	0.77	0.91	0.84	500
Precinct05	0.87	0.67	0.76	500
Precinct06	0.89	0.99	0.93	500
Precinct07	0.60	0.98	0.74	500
Precinct09	0.99	0.31	0.47	500

```
    accuracy                           0.77      2500
   macro avg       0.82      0.77      0.75      2500
weighted avg       0.82      0.77      0.75      2500
```

5.2.2 Exercise 8

Using `max_depth=1` how many different classes are reflected in the predictions? Using `max_depth=2`? Based on this and given there are 5 classes we are attempting to classify, what is the minimum depth of tree should we be using?

Using `max_depth=1`, two classes are reflected in the predictions, precincts 1 and 5. Using `max_depth=2`, we have 4 classes represented. So we should be using at least `max_depth=3`, since we have 5 categories.

5.2.3 Exercise 9

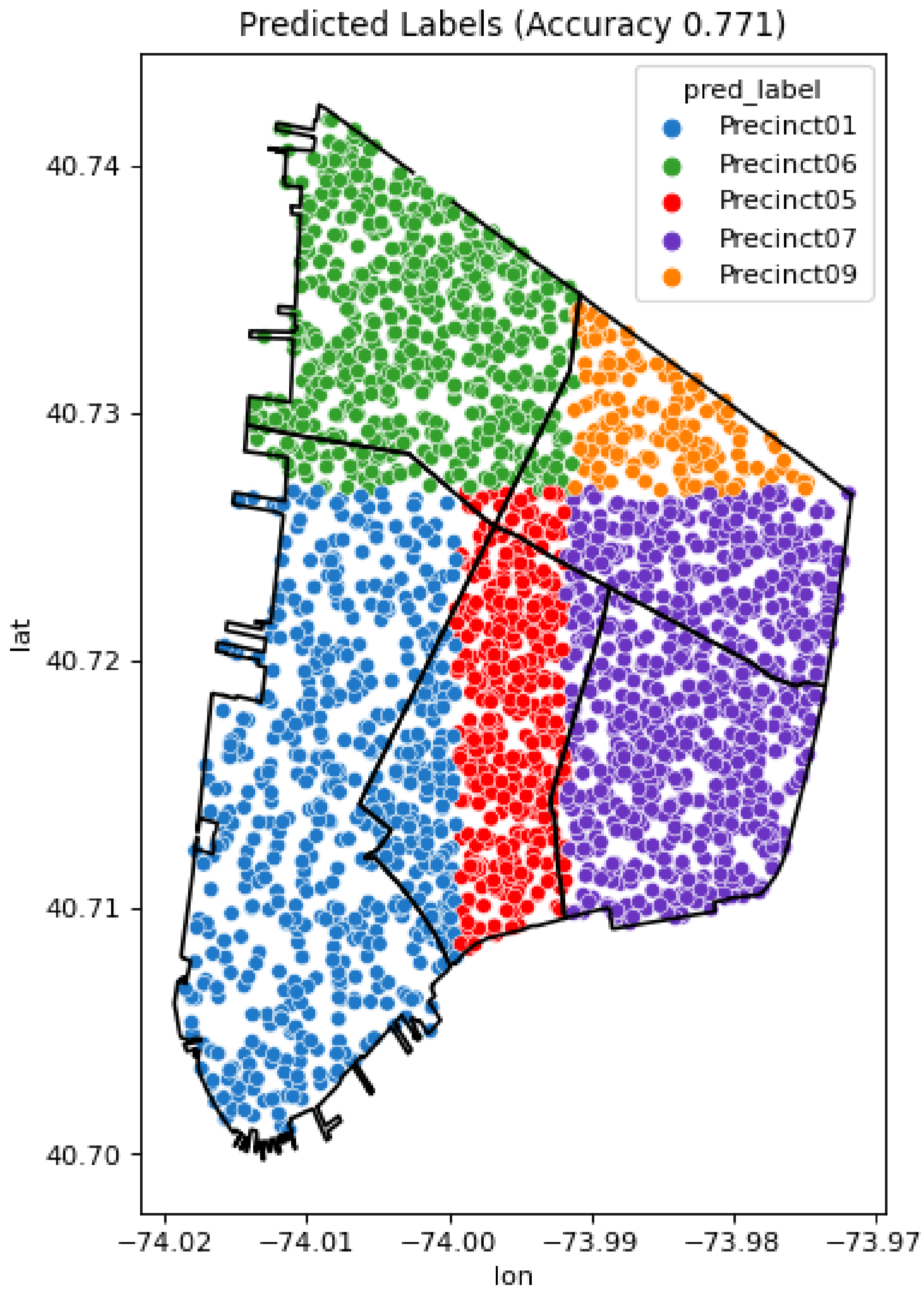
Examine the boundaries being created by the model (particularly evident with small `max_depth` values), what shape do they have? Is the potentially problematic for the task at hand?

The boundaries are always linear, and the real boundaries between precincts in the map are non-linear. So we will struggle to classify points correctly unless we use a high depth.

The issue we've just seem is similar to the issue we saw with the original single precinct logistic regression model from Week 7, and we can somewhat address it in the same way by introducing an interaction feature between `lat` and `lon` before fitting our model by adding a `PolynomialFeature` step before fitting the tree model.

[25]:
```
def fit_dt_int(max_depth):
    m_dt_int = make_pipeline(
        sklearn.preprocessing.PolynomialFeatures(
            degree=2, interaction_only=True, include_bias=False
        ),
        sklearn.tree.DecisionTreeClassifier(
            max_depth = max_depth
        )
    ).fit(X,y)

    model_test_assess(m_dt_int)
```

[26]:
```
fit_dt_int(max_depth = 3)
```



	precision	recall	f1-score	support
Precinct01	0.77	0.91	0.84	500
Precinct05	0.87	0.67	0.76	500
Precinct06	0.89	0.99	0.93	500
Precinct07	0.60	0.98	0.74	500
Precinct09	0.99	0.31	0.47	500

```
    accuracy                           0.77      2500
   macro avg       0.82      0.77      0.75      2500
weighted avg       0.82      0.77      0.75      2500
```

5.2.4 Exercise 10

What has changed about the boundaries being created by the model (particularly evident with small `max_depth` values), what shape do they have? Does this improve model performance?

The boundaries created by the model are now slightly curved which improves model performance as better approximating the boundaries between precincts. However, we would still prefer more non-linear boundaries.

5.2.5 Exercise 11

For either model, for what values of `max_depth` do you start seeing clear evidence of overfitting?

At `max_depth=5`, we see overfitting as some locations inside precinct9 are predicted as belonging to precint 7 and there are many points in precinct 7 being predicted as in precinct 9.

5.2.6 3.2.2 `RandomForestClassifier`

As discussed in lecture, this class of model is an extension of the decision tree frame work whereby a number decision tree models are fit to random sub-sample (i.e. bootstrap sample) of the features. When making predictions each tree predicts a label and the most common label across all of the trees is used as the final prediction. For now we will just examine the effect of `n_estimators` which corresponds to the number of trees and `max_depth` which is the maximum depth of the trees.

[27]:
```python
def fit_rf(n_estimators, max_depth):
    m_rf = sklearn.ensemble.RandomForestClassifier(
        n_estimators=n_estimators, max_depth=max_depth
    ).fit(X,y)

    model_test_assess(m_rf)

def fit_rf2(n_estimators, max_depth):
    m_rf = sklearn.ensemble.RandomForestClassifier(
        n_estimators=n_estimators, max_depth=max_depth
    ).fit(X,y)
```

```
    res = pd.concat(
        [ manh_test,
          pd.Series(
              data = m_rf.predict(manh_test[['lon','lat']]),
              name = "pred_label"
          )
        ],
        axis = 1
    )

    acc = sklearn.metrics.accuracy_score(
        res['precinct'], res['pred_label']
    )

    return acc
    #ax.set_title("Predicted Labels (Accuracy {:.3f})".format(acc))

    #model_test_assess(m_rf)
```

Use the following code chunk to explore the effect of different values of `max_depth` and `n_estimators` on the model's performance.

[28]:
```
perform = np.array([])

for i in range(1,15):
    for j in range(1,15):
        #print("n_estimators = " +  str(i), "max_depth = " + str(j),␣
 ↪fit_rf2(n_estimators = i, max_depth = j))
        perform = np.append(perform, fit_rf2(n_estimators = i, max_depth = j))
#fit_rf(n_estimators = 5, max_depth = 4)
print(np.sort(perform))
```

```
[0.3836 0.3836 0.3836 0.3844 0.3856 0.3864 0.394  0.3968 0.3968 0.4136
 0.4144 0.4168 0.5076 0.5336 0.5348 0.5628 0.5684 0.6196 0.6264 0.63
 0.6324 0.6332 0.6368 0.6512 0.7056 0.7284 0.7416 0.7708 0.782  0.7856
 0.792  0.7996 0.8016 0.804  0.8092 0.814  0.8204 0.8244 0.8256 0.8288
 0.8292 0.83   0.8332 0.8356 0.8368 0.8368 0.8396 0.84   0.8436 0.8444
 0.8456 0.8456 0.8468 0.8512 0.852  0.8544 0.8544 0.8548 0.8556 0.8564
 0.8576 0.8576 0.8584 0.8592 0.8596 0.86   0.8604 0.8616 0.862  0.864
 0.8652 0.8656 0.8656 0.866  0.8664 0.8668 0.8672 0.8684 0.8684 0.8684
 0.8688 0.8688 0.8696 0.8696 0.87   0.87   0.87   0.8704 0.8708 0.8724
 0.8728 0.8728 0.8732 0.8748 0.8748 0.8752 0.876  0.8768 0.8768 0.8768
 0.8768 0.878  0.878  0.8792 0.8792 0.8796 0.8804 0.8808 0.8812 0.8816
 0.8816 0.882  0.8824 0.8828 0.8828 0.8828 0.884  0.8844 0.8872 0.8876
 0.8876 0.8876 0.888  0.8884 0.8888 0.89   0.8904 0.8904 0.8916 0.8916
 0.8924 0.8928 0.8928 0.8936 0.894  0.894  0.8952 0.8952 0.8956 0.896
 0.8968 0.8972 0.898  0.8996 0.9    0.9004 0.9016 0.9048 0.9068 0.9072
 0.908  0.9084 0.9084 0.9112 0.9124 0.9136 0.914  0.914  0.9144 0.9148
```

```
0.9152 0.9164 0.9164 0.9176 0.9176 0.918  0.9184 0.9212 0.9212 0.9212
0.9216 0.922  0.9224 0.9224 0.9232 0.924  0.9252 0.9252 0.9264 0.9268
0.9272 0.9276 0.9276 0.9276 0.9284 0.93   0.9304 0.9316 0.9316 0.932
0.9352 0.936  0.9376 0.9376 0.9388 0.9408]
```

[29]:
```
#for i in range(1,15):
#    for j in range(1,15):
#        print("n_estimators = " +  str(i), "max_depth = " + str(j),
 ↪fit_rf2(n_estimators = i, max_depth = j))
```

5.2.7 Exercise 12

Which combination of `n_estimators` and `max_depth` seems to produce the best model performance? How does it compare to the other models considered.

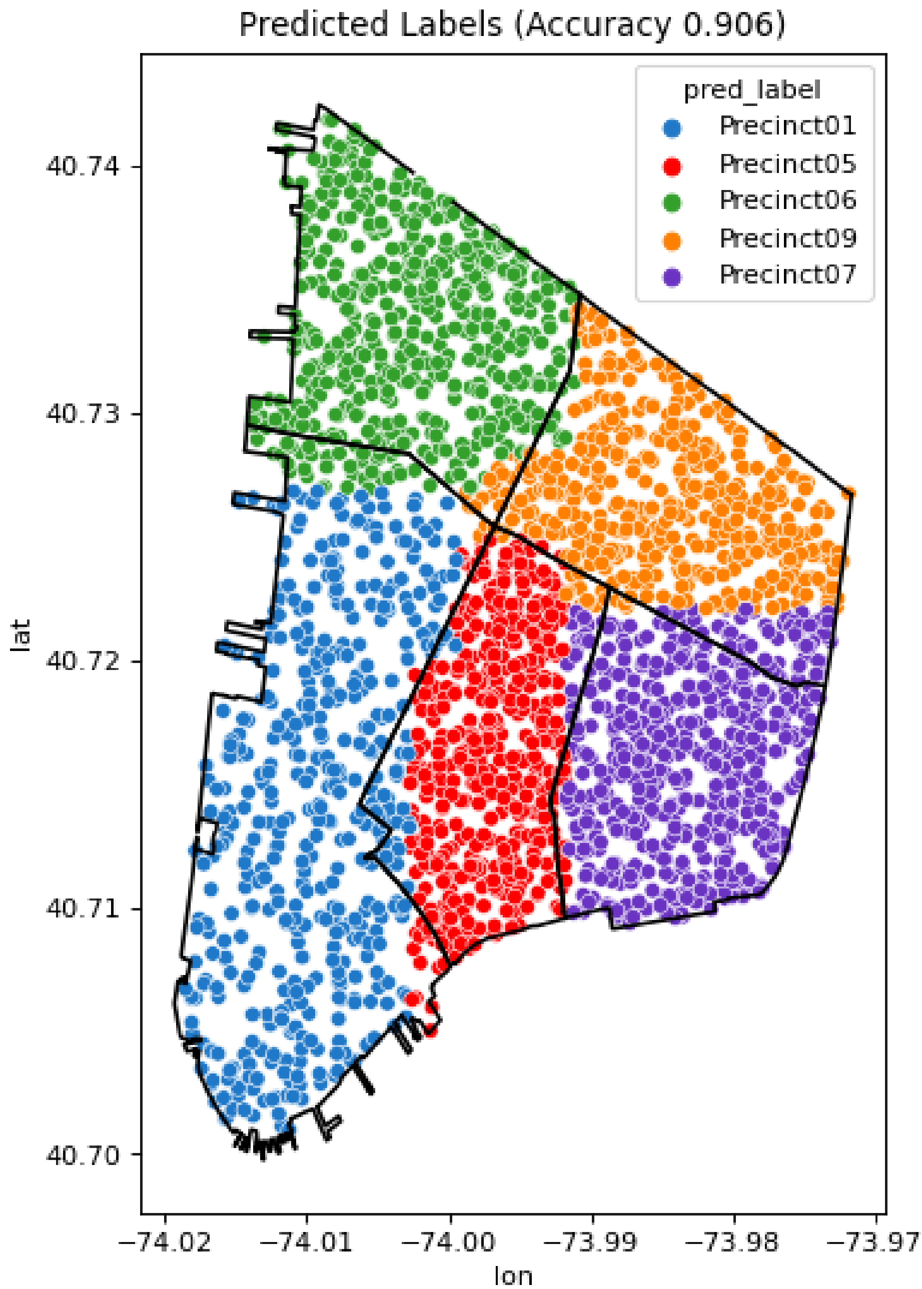
The best model performance is `n_estimators = 10` and `max_depth = 6` from the cell above. However the best values change with each version of the random forrest and it is not easy to pick exact best values. The model performance is good, around 0.94 accuracy, that is comparible to the linear SVC model with a `C=50`. We would prefer the linear svc model for its interpretability.

5.2.8 Exercise 13

For a given `max_depth` (e.g. 4) compare the performance of a single decision tree model to random forrest models with different values of `n_estimators`, how do they compare?

We can see the performance of the single decision tree model with `max_depth=4`,

[30]:
```
fit_dt(max_depth = 4)
```



	precision	recall	f1-score	support
Precinct01	0.93	0.88	0.90	500
Precinct05	0.92	0.86	0.89	500
Precinct06	0.92	0.97	0.94	500
Precinct07	0.85	0.97	0.91	500
Precinct09	0.92	0.84	0.88	500

```
    accuracy                           0.91      2500
   macro avg       0.91      0.91      0.91      2500
weighted avg       0.91      0.91      0.91      2500
```

Now, we alter the `fit_rf` function to have a fixed `max_depth=4` and return the accuracy,

[31]:
```python
def fit_rf_fixed_depth(n_estimators):
    m_rf = sklearn.ensemble.RandomForestClassifier(
        n_estimators=n_estimators, max_depth=4
    ).fit(X,y)

    res = pd.concat(
        [ manh_test,
          pd.Series(
              data = m_rf.predict(manh_test[['lon','lat']]),
              name = "pred_label"
          )
        ],
        axis = 1
    )

    acc = sklearn.metrics.accuracy_score(
        res['precinct'], res['pred_label']
    )

    return acc
```

Now, we try all values of `n_estimators` from 1 to 100,

[32]:
```python
best = np.array([])

for i in range(1,100):
    best = np.append(best, fit_rf_fixed_depth(i))
print(np.amax(best))
```

```
0.8912
```

we can see that for values of `n_estimators` up to 100, we cannot get an accuracy above 0.9, that is achieved by the single decision tree model.

5.2.9 Exercise 14

Do you think the inclusion of an interaction feature would help or hurt this model's performance? Explain.

In theory, the inclusion of interaction features should help the model performance since it allows for non-linear decision boundaries that will be able to better fit the non-linear shapes of our real life precinct boundaries.

[33]:
```
def fit_rf_pipeline(n_estimators, max_depth):
    m_rf = make_pipeline(
        sklearn.preprocessing.PolynomialFeatures(
            degree=2, interaction_only=True, include_bias=False
        ),
        sklearn.ensemble.RandomForestClassifier(
            n_estimators=n_estimators, max_depth=max_depth
        )
    ).fit(X,y)

    res = pd.concat(
        [ manh_test,
          pd.Series(
              data = m_rf.predict(manh_test[['lon','lat']]),
              name = "pred_label"
          )
        ],
        axis = 1
    )

    acc = sklearn.metrics.accuracy_score(
        res['precinct'], res['pred_label'])

    return acc
```

[34]:
```
perform = np.array([])

for i in range(1,15):
    for j in range(1,15):
        #print("n_estimators = " +  str(i), "max_depth = " + str(j),␣
↪fit_rf2(n_estimators = i, max_depth = j))
        perform = np.append(perform, fit_rf_pipeline(n_estimators = i,␣
↪max_depth = j))
#fit_rf(n_estimators = 5, max_depth = 4)
print(np.sort(perform))
```

```
[0.3608 0.3612 0.3888 0.3892 0.396  0.3968 0.3988 0.4084 0.4132 0.4172
 0.4492 0.4584 0.4632 0.5112 0.5452 0.558  0.5808 0.6192 0.6236 0.6328
 0.636  0.6736 0.6748 0.6788 0.6824 0.696  0.7088 0.7336 0.7344 0.764
 0.7716 0.776  0.778  0.7876 0.7892 0.7932 0.8008 0.808  0.812  0.818
 0.8248 0.8284 0.83   0.83   0.8332 0.8336 0.8352 0.8372 0.8376 0.8392
 0.84   0.8448 0.848  0.8516 0.8528 0.854  0.8576 0.8584 0.8588 0.8596
 0.8596 0.8604 0.8612 0.8616 0.8644 0.8692 0.8708 0.8708 0.8712 0.8732
 0.8744 0.8744 0.8768 0.878  0.8788 0.8788 0.8792 0.88   0.8804 0.8808
 0.8816 0.8816 0.8816 0.8816 0.884  0.8848 0.8868 0.8872 0.8876 0.888
 0.8884 0.8904 0.8908 0.892  0.8924 0.8932 0.8936 0.8936 0.894  0.8944
```

```
 0.8948 0.8956 0.8964 0.8964 0.8972 0.8976 0.8976 0.898  0.9004 0.9008
 0.9032 0.9044 0.9048 0.9052 0.906  0.906  0.9068 0.9068 0.9068 0.9072
 0.9092 0.9096 0.91   0.9104 0.9104 0.9104 0.9108 0.9112 0.9116 0.9116
 0.9116 0.912  0.912  0.9136 0.9136 0.9148 0.9152 0.9152 0.9156 0.9156
 0.9164 0.9164 0.9164 0.9172 0.9176 0.9184 0.9184 0.9188 0.9192 0.9192
 0.9196 0.9196 0.92   0.92   0.9212 0.9216 0.922  0.9224 0.9228 0.9232
 0.924  0.924  0.9244 0.9244 0.9248 0.9252 0.9252 0.926  0.926  0.926
 0.926  0.9264 0.9264 0.9268 0.9276 0.9276 0.9276 0.9292 0.9292 0.93
 0.9308 0.9312 0.9324 0.9328 0.9332 0.9332 0.934  0.9344 0.9348 0.9348
 0.9352 0.9352 0.9352 0.9356 0.9368 0.9404]
```

However, in practise, it appears that this change doesn't have a dramatic influence on the final accuracy of the model. We can note here, however, that more of the trees have an accuracy above 0.9 than in the model without interaction terms, so we could argue this is an improvement.

5.3 3.3 Comparisons

5.3.1 Exercise 15

You have now been given the opportunity to experiment with a number of different multiclass classification models. Based on your experience with this data set which model do you think is best for this particular classification task? Justify your answer.

You answer should contain some discussion of the potential for overfitting.

From week 7, we firstly used the multiclass logistic regression model using one-versus-rest with `multi_class = 'ovr'`. This model gave fairly unsatsfactory decision boundaries that do not approximate the precincts well.

We then fitted a mulitnomial model that has much better decision boundaries than the first model,

This multiclass logistic (multinomial) regression has an accuracy of 0.895, slightly lower than the models we used this week.

The best model with support vector machines was the linear model with a large C value (greater than 50) as shown here, with an accuracy of 0.937.

We note that the linearSVC models used this week are very poor and lead to a lot of overfitting, hence we would not prefer this to other methods. The random forrest method also provided an accuracy of around 0.94, however, we would not prefer this method since it only performs slightly better than our linear support vector model and is much more complex.

Therefore, the preffered model for this task would be the linear support vector model due to its relative simplicity and high accuracy of 0.937.

5.4 4. Competing the worksheet

At this point you have hopefully been able to complete all the preceeding exercises. Now is a good time to check the reproducibility of this document by restarting the notebook's kernel and rerunning all cells in order.

Once that is done and you are happy with everything, you can then run the following cell to generate your PDF and turn it in on gradescope under the `mlp-week09` assignment.

[35]:
```
!jupyter nbconvert --to pdf mlp-week09.ipynb
```

```
[NbConvertApp] Converting notebook mlp-week09.ipynb to pdf
[NbConvertApp] Support files will be in mlp-week09_files/
[NbConvertApp] Making directory ./mlp-week09_files
[NbConvertApp] Making directory ./mlp-week09_files
[NbConvertApp] Making directory ./mlp-week09_files
[NbConvertApp] Making directory ./mlp-week09_files
[NbConvertApp] Making directory ./mlp-week09_files
[NbConvertApp] Making directory ./mlp-week09_files
[NbConvertApp] Making directory ./mlp-week09_files
[NbConvertApp] Making directory ./mlp-week09_files
[NbConvertApp] Making directory ./mlp-week09_files
[NbConvertApp] Writing 94152 bytes to ./notebook.tex
[NbConvertApp] Building PDF
[NbConvertApp] Running xelatex 3 times: ['xelatex', './notebook.tex', '-quiet']
[NbConvertApp] Running bibtex 1 time: ['bibtex', './notebook']
[NbConvertApp] WARNING | bibtex had problems, most likely because there were no
citations
[NbConvertApp] PDF successfully created
[NbConvertApp] Writing 1308746 bytes to mlp-week09.pdf
```

Created in Deepnote