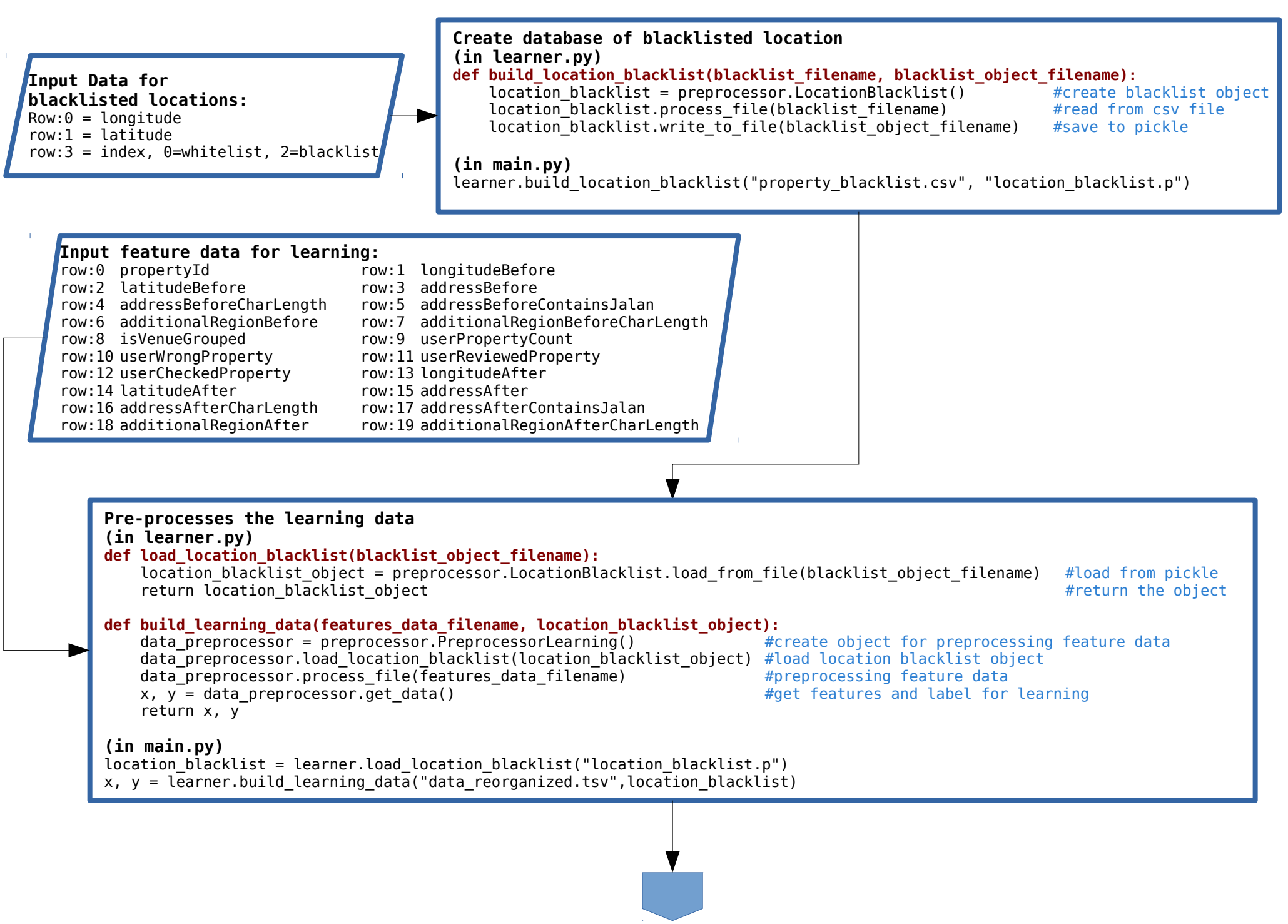
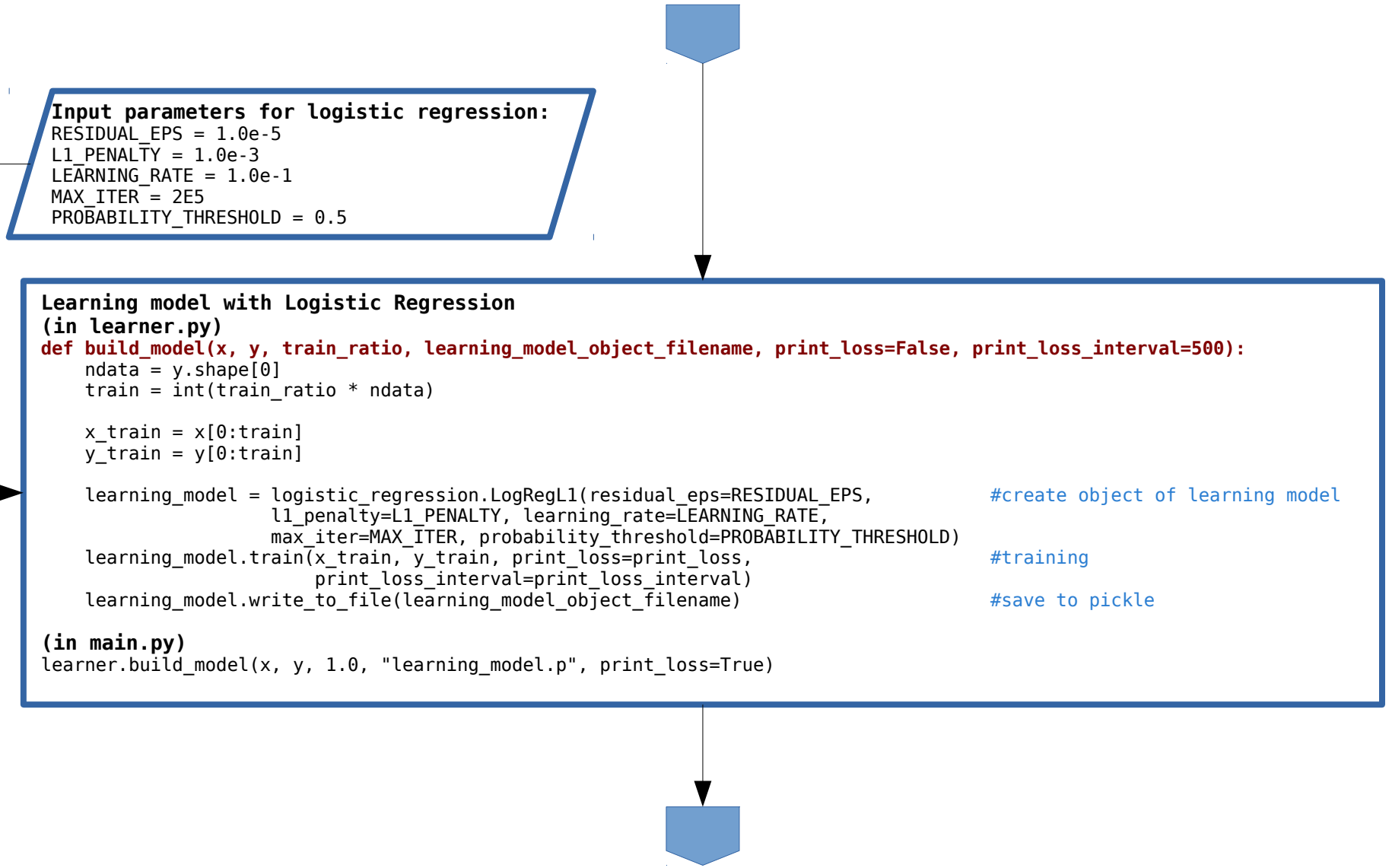
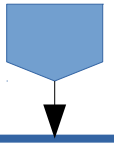

Input Data for
blacklisted locations:
Row:0 = longitude
row:1 = latitude
row:3 = index, 0=whitelist, 2=blacklist
Create database of blacklisted location
(in learner.py)
def build_location_blacklist(blacklist_filename, blacklist_object_filename):
location_blacklist = preprocessor.LocationBlacklist() #create blacklist object
location_blacklist.process_file(blacklist_filename) #read from csv file
location_blacklist.write_to_file(blacklist_object_filename) #save to pickle
(in main.py)
learner.build_location_blacklist("property_blacklist.csv", "location_blacklist.p")
Input feature data for learning:
row:0 propertyId row:1 longitudeBefore
row:2 latitudeBefore row:3 addressBefore
row:4 addressBeforeCharLength row:5 addressBeforeContainsJalan
row:6 additionalRegionBefore row:7 additionalRegionBeforeCharLength
row:8 isVenueGrouped row:9 userPropertyCount
row:10 userWrongProperty row:11 userReviewedProperty
row:12 userCheckedProperty row:13 longitudeAfter
row:14 latitudeAfter row:15 addressAfter
row:16 addressAfterCharLength row:17 addressAfterContainsJalan
row:18 additionalRegionAfter row:19 additionalRegionAfterCharLength
Pre-processes the learning data
(in learner.py)
def load_location_blacklist(blacklist_object_filename):
location_blacklist_object = preprocessor.LocationBlacklist.load_from_file(blacklist_object_filename) #load from pickle
return location_blacklist_object #return the object
def build_learning_data(features_data_filename, location_blacklist_object):
data_preprocessor = preprocessor.PreprocessorLearning() #create object for preprocessing feature data
data_preprocessor.load_location_blacklist(location_blacklist_object) #load location blacklist object
data_preprocessor.process_file(features_data_filename) #preprocessing feature data
x, y = data_preprocessor.get_data() #get features and label for learning
return x, y
(in main.py)
location_blacklist = learner.load_location_blacklist("location_blacklist.p")
x, y = learner.build_learning_data("data_reorganized.tsv",location_blacklist)


Input parameters for logistic regression:
RESIDUAL_EPS = 1.0e-5
L1_PENALTY = 1.0e-3
LEARNING_RATE = 1.0e-1
MAX_ITER = 2E5
PROBABILITY_THRESHOLD = 0.5
Learning model with Logistic Regression
(in learner.py)
def build_model(x, y, train_ratio, learning_model_object_filename, print_loss=False, print_loss_interval=500):
ndata = y.shape[0]
train = int(train_ratio * ndata)
x_train = x[0:train]
y_train = y[0:train]
learning_model = logistic_regression.LogRegL1(residual_eps=RESIDUAL_EPS, #create object of learning model
l1_penalty=L1_PENALTY, learning_rate=LEARNING_RATE,
max_iter=MAX_ITER, probability_threshold=PROBABILITY_THRESHOLD)
learning_model.train(x_train, y_train, print_loss=print_loss, #training
print_loss_interval=print_loss_interval)
learning_model.write_to_file(learning_model_object_filename) #save to pickle
(in main.py)
learner.build_model(x, y, 1.0, "learning_model.p", print_loss=True)



Predict

(in learner.py)

```
def load_model(learning_model_object_filename):
    learning_model = logistic_regression.LogRegL1.\
       load_from_file(learning_model_object_filename)#load object of learning model
    return learning_model

def predict(learning_model, input_row):
    label_i, _ = learning_model.predict(input_row)    #predict (get the label in value 0/1)
    return label_i
```

(in map_util.py)

```
class MapUtil(object):
    oversize_threshold = MAP_BBOX_OVERSIZE_THRESHOLD

    @staticmethod
    def bbox_oversize_predictor(NE_bounding_box, SW_bounding_box):
        latitude1, longitude1 = NE_bounding_box
        latitude2, longitude2 = SW_bounding_box

        if abs(latitude1 - latitude2) > MapUtil.oversize_threshold and \
           abs(longitude1 - longitude2) > MapUtil.oversize_threshold:
            return True
        else:
            return False
```

(in main.py)

```
LABEL_OUT = ["Correct", "Incorrect"]                              #label 0 = correct, label 1 = incorrect

learning_model = learner.load_model("learning_model.p")          #load learning model
location_blacklist = load_location_blacklist(blacklist_object_filename)  #load location blacklist object

preprocessor_row = preprocessor.PreprocessorRow()                #create object for pre-processing
preprocessor_row.load_location_blacklist(location_blacklist)     #load location blacklist object into preproccesor

with open("data/short_data_reorganized_for_testing.tsv") as csvfile:
reader = csv.reader(csvfile, dialect='tsv_dialect')
       next(reader, None) # skip the headers
       for row in reader:
          map_bbox_oversize = map_util.MapUtil.\
                    bbox_oversize_predictor(row[13], row[14])          #check if the map bounding box is oversize

          if map_bbox_oversize:                                        #if the map bounding box is oversize
              print "property id :", property_id, " :: prediction:", LABEL_OUT[1]
          else:                                                        #if the map bounding box is NOT oversize
              property_id, xt = preprocessor_row.process_row(row)
              label_i = learner.predict(learning_model, xt)
              print "property id :", property_id, "prediction:", LABEL_OUT[label_i]
```

Input data for predict:

```
row:0  propertyId
row:1  longitudeBefore
row:2  latitudeBefore
row:3  addressBefore
row:4  addressBeforeCharLength
row:5  addressBeforeContainsJalan
row:6  additionalRegionBefore
row:7  additionalRegionBeforeCharLength
row:8  isVenueGrouped
row:9  userPropertyCount
row:10 userWrongProperty
row:11 userReviewedProperty
row:12 userCheckedProperty
Row:13 NE_boundingboxBefore
row:14 SW_boundingboxBefore
```