1 Team

Team Members	CS id: d8z9a Student id: 53033149
Kaggle Team Name	onlyme

2 Solution Summary

The solution uses features cases, cases_14_100k, cases_100k, deaths to predict the death count of the next days. In this model, we fit a linear regression of the features, and the next prediction is done with the predictions of the next feature itself. Also the following model has hyper-parameters k (number of time-series to look behind), p (the polynomial basis), *log_base* (to use log base or not) and *use_prediction* (to use the predicted death value for the next prediction or the prediction from the feature itself).

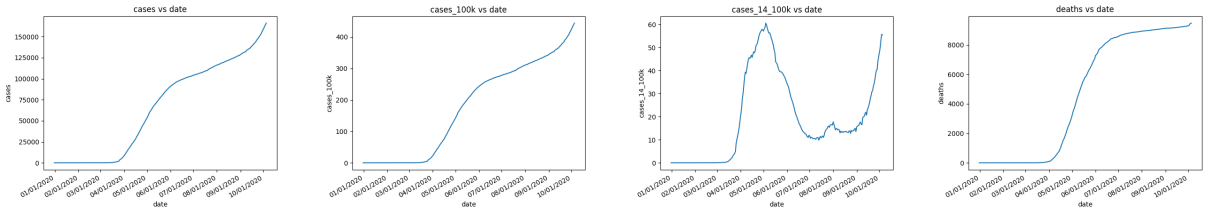
The best solution to get is the hyper-parameters that minimizes the error function. The best hyper-parameters are chosen by training the data to the date given by phase1 and predict the next days.

3 Experiments

Feature Selection

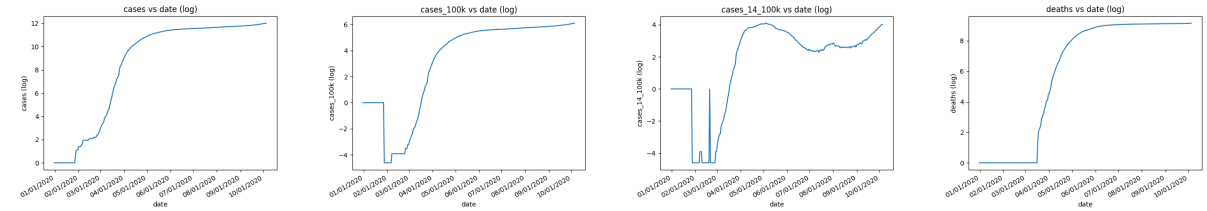
The following are chosen as features: cases, cases_100k, cases_14_100k and deaths to predict the next death cases for next day. The deaths will also be input in our feature selection because it modifies the value for the next death number cases.

Below are some plots related to our features.



(a) cases (b) cases_100k (c) cases_14_100k (d) deaths

Figure 1: Plots of cases vs date



(a) cases (log) (b) cases_100k (log) (c) cases_14_100k (log) (d) deaths (log)

Figure 2: Plots of cases_100k vs date

The log base models are trained as well since we may be dealing with big numbers that polynomial basis cannot handle well. Also since it is a prediction with the case of a virus, we know a graph more or less has to be right skewed.

Hyperparameters

Choosing k hyperparameter

The k hyperparameter can be anything that is greater than 0 and less than the size of the feature. We chose this k value by trying bunch of different values for k. In general k can be anything that best fits our model.

Choosing p hyperparameter

In general, based on the graphs defined in Figure 1, we can see that no more than $p > 5$ is needed. Also since a greater number of p greatly influences the edges of our model (which is bad for predicting the future) having a big p will lead to extreme numbers as each time we try to predict the next values. Hence our hyperparameter will be $p < 5$ in general cases.

The value of p will also be affected greatly by the k hyperparameter. A big k will look at the k past data, so the greater k is, a greater p might generate a better fit while the lower k is, a linear fit might be better.

Other hyperparameters

The *log_base* and *use_prediction* are just binary inputs to whether we would like to use a log based data set to generate the model or use the prediction based on the fit of all data, or just keep using the fit from the death feature column.

Training

Training is done via minimizing our error function depending on our values of hyperparameters. For different values of k, p, *log_base*, *use_prediction*, we try to get the best values that minimizes the error.

Phase 1

In general for phase 1, the hyper parameters are chosen by fitting a model based on features until October 5 and then trying to minimize the error by comparing with next 11 days (October 16).

Below is a table summarizing the trials.

log_base	*use_prediction*	k	p	$min(\sum(y_{pred} - y)^2)$
False	False	88	1	1177.149
False	True	127	1	1313.394
True	False	81	1	1678.265
True	True	137	2	1372.992

Table 1: Table summarizing k and p results for different *log_base* and *use_prediction* (phase 1)

For phase 1, the minimum error is given when $log\_base = False$, $use\_prediction = False$, $k = 88$, $p = 1$.

Phase 2

For phase 2, the values for the hyperparameters are also found using the same strategy for phase 1, now except trying to get best result from next 11 days starting October 5, the test is done with the next 20 days starting on October 5.

Below is the table summarizing the trials.

log_base	*use_prediction*	k	p	$min(\sum(y_{pred} - y)^2)$
False	False	88	1	2184.482
False	True	127	1	3198.579
True	False	81	1	2210.375
True	True	79	1	1954.314

Table 2: Table summarizing k and p results for different *log_base* and *use_prediction* (phase 2)

The only difference between Table 1 and Table 2 is the last row. Phase 1's $k = 137$ and $p = 2$ could not possibly be used for further predictions because of the polynomial extreme values going faster over each prediction. Hence we got a new value $k = 79$ and $p = 1$ which we will use it to predict the next 5 days using now all data until October 25.

4 Results

Team Name	Kaggle Phase 1 Score	Kaggle Phase 2 Score
onlyme	10.13544	N/A

5 Code

```
class AutoRegressor:
    def __init__(self, k, p=1,
                 log_base=False,
                 use_prediction=True,
                 model=LeastSquares):
        """
        k                   = the number of data sets to look behind to make next prediction
        p                   = polynomial basis
        log_base            = change data to log space
        use_prediction      = use prediction from X_k to test y_k+1, if false uses
                              time series the the deaths column itself
        model               = the linear model to use. We only use Least Squares
        In this class, everything with variable X means it is not weighted
        everything with variable Z means it has been transformed/weighted.
        """
        self.k = k
        self.p = p
        self.log_base = log_base
        self.use_prediction = use_prediction
        self.model = model
        self.y_model = model()

    def fit(self, X, y):
        N,D = X.shape
        k = self.k

        # if k is too large don't do this.
        self.size = N - k
        if (self.size <= 0): return

        # transform X to Z using transform method.
        Z = self.__transform(X)
        # get values of y for time series
        y_k = y[self.size:N]

        # set our features to predict next features.
        self.__feature_time_series(X, Z)
        # our general prediction for y_k+1
        self.y_model.fit(Z, y_k)

    def __feature_time_series(self, X, Z):
        """
        Grabs data from k to N-1 for each feature. If p > 0, also adds polynomial basis
        columns.
        Generates models to predict x_i_k+1 so that we can comput y_k+1.
        The models here are by independent, meaning they will have their own
        bias when fitting and predicting the model.
        """
        N, D = X.shape
```

```
        p = self.p

        feature_models = []
        for i in range(D):
            pos = i*p+1
            # grab the data from Z and add a bias 1 or e.
            x_i = self.__add_bias(Z[:,pos:pos+p])
            x_k = X[self.size:N,i]
            # create model and fit.
            model = self.model()
            model.fit(x_i, x_k)
            # save model to predict x_i_k+1
            feature_models.append(model)

        self.feature_models = feature_models

    def __transform(self, X):
        """
        Transforms X into Z. Adds the bias 1 if non logbase, e if logbase.
        Adds polynomial basis given p.
        """
        N,D = X.shape
        k = self.k
        p = self.p

        # new data set
        Z = np.ones(shape=(k, D*p+1))
        k_data = X[self.size-1:N-1, :]

        for i in range(D):
            pos = i*p+1
            # add polynomial basis
            Z[:,pos:pos+p] = self.__polyBasis(k_data[:,i])

        # handle log case
        if self.log_base == True:
            Z[:,0] = np.exp(1)

        return Z

    def __polyBasis(self, X):
        """
        General function to change to poly basis.
        NOTE: this does not add 1 as a bias.
        """
        N = X.size
        p = self.p

        Z = np.ones((N, p))
        for j in range(1, p + 1):
            Z[:,j-1] = X**j

        return Z

    def __add_bias(self, X):
        """
        Adds a bias of e if log_base, 1 otherwise.
        """
        N,D = X.shape
        Z = np.ones(shape=(N,D+1))
        Z[:,1:] = X

        if self.log_base == True:
            Z[:,0] = np.exp(1)

```

```
        return Z

    def predict(self, X, num_times):
        """
        Predict the next sequence of num_times.
        """
        def append_time_series(Z, y_k):
            """
            Predict x_i_k+1, removes x_i_1 from Z, and appends the new value.
            If use_prediction is on, it will use the predicted value y_k+1 and its
            polynomial biases for next predict. Else it will use the values from
            its features series x_i_k+1.
            """
            N,D = Z.shape
            p = self.p
            feature_models = self.feature_models

            i = 0
            time_preds = np.ones(shape=(1,D))
            for model in feature_models:
                # predict the new features
                pos = i*p+1
                x_i = self.__add_bias(Z[:,pos:pos+p])
                pred = model.predict(x_i)[-1]
                i += 1

                time_preds[:,pos:pos+p] = self.__polyBasis(pred)

            # determine log base
            if self.log_base == True:
                time_preds[:,0] = np.exp(1)

            # remove first row, append to last row.
            Z = Z[1:]
            Z = np.append(Z, time_preds, axis=0)

            # use prediction will change the deaths column to predicted value and its
            # polynomial basis.
            if self.use_prediction == True:
                Z[:,pos:pos+p] = self.__polyBasis(y_k)

            return Z

        Z = self.__transform(X)
        y_k = self.y_model.predict(Z)[-1]

        y_pred = np.zeros(num_times)

        for i in range(num_times):
            # predict, and save results.
            Z = append_time_series(Z, y_k)
            y_k = self.y_model.predict(Z)[-1]
            y_pred[i] = y_k

        return y_pred
```

Listing 1: AutoRegressor class implementation

```
if __name__ == "__main__":
    # modify variables here
    phase = "phase2"        # the phase to test
    num_data = 5            # the number of data to predict
    log_base = True         # variable if our model is in log base or not
    use_prediction = True   # variable to use predictions in general over time series

```

```
    # load file + set data
    df = pd.read_csv(os.path.join("..", "data", phase + "_training_data.csv"))

    X_codes = df.country_id.unique()
    X_codes = X_codes[X_codes != "CA"]

    X_ca = df.loc[df["country_id"] == "CA"]

    ca_cases = X_ca["cases"].values
    ca_100k = X_ca["cases_100k"].values
    ca_14_100k = X_ca["cases_14_100k"].values
    ca_deaths = X_ca["deaths"].values
    dates = X_ca["date"].values

    # combine column
    X = np.column_stack((ca_cases, ca_100k, ca_14_100k, ca_deaths))

    # if we are using log space
    if log_base == True:
        X = np.log(X)
        X[X == -np.inf] = 0

        ca_deaths = np.log(ca_deaths)
        ca_deaths[ca_deaths == -np.inf] = 0

    # run test only in phase 1 mode
    # this only works phase 1 data.
    # uncomment next line to get best k and p. NOTE: num_data has to be either 11 or 20
    # __find_min_poly_basis(X, ca_deaths, num_data, log_base, use_prediction)

    # code for plots
    # dates_obj = [dt.datetime.strptime(d, "%m/%d/%Y").date() for d in dates]
    # labels = ["cases vs date", "cases_100k vs date", "cases_14_100k vs date", "deaths vs
    date"]
    # ylabels= ["cases", "cases_100k", "cases_14_100k", "deaths"]
    # fnames = ["cases", "cases_100k", "cases_14_100k", "deaths"]

    # for i in range (4):
    #     if log_base == True:
    #         __plot(dates_obj, X[:,i], fnames[i]+"_log", labels[i] + " (log)", "date",
    ylabels[i] + " (log)")
    #     else:
    #         __plot(dates_obj, X[:,i], fnames[i], labels[i], "date", ylabels[i])


    # fit to autogressor model
    model = AutoRegressor(k=79, p=1, log_base=log_base, use_prediction=use_prediction)
    model.fit(X, ca_deaths)
    y_preds = model.predict(X, num_data)

    if log_base == True:
        y_preds = np.exp(y_preds)

    y_preds = y_preds.astype('int64')
    print(y_preds)

    # to csv
    output = pd.DataFrame({'deaths': y_preds,
                           'id': range(num_data)})
    out_path = os.path.join("..", "data", phase + "_out.csv")
    output.to_csv(path_or_buf=out_path,index=False)
```

Listing 2: Main function

```
def __find_min_poly_basis(X, y, num_data, log_base=False, use_prediction=True):
    # first is for predicting 11 days in the future
    # second is for predicting 20 days in the future
    # comment/uncomment as needed
    # y_test = [9504,9530,9541,9557,9585,9585,9585,9627,9654,9664,9699]
    y_test = [9504,9530,9541,9557,9585,9585,9585,9627,9654,9664,9699,
              9722,9746,9760,9778,9794,9829,9862,9888,9922]

    min_error = np.inf

    for p in range(1,5):
        for k in range(1, 280):
            model = AutoRegressor(k=k, p=p, log_base=log_base, use_prediction=use_prediction
    )
            try:
                model.fit(X, y)
            except:
                continue
            y_preds = model.predict(X, num_data)
            if log_base == True:
                y_preds = np.exp(y_preds)
            error = np.sum(np.square(y_preds-y_test))
            # print("p: %d, k: %d, error: %.3f" % (p, k, error))
            if (error < min_error):
                p_low = p
                k_low = k
                min_error = error

    print("lowest p: %d, k: %d, error: %.3f" % (p_low, k_low, min_error))
```

Listing 3: Brute force to search for k and p

```
def __plot(x, y, fname, label, xlabel, ylabel):
    plt.gca().xaxis.set_major_formatter(mdates.DateFormatter('%m/%d/%Y'))
    plt.gca().xaxis.set_major_locator(mdates.MonthLocator())
    plt.plot(x, y)
    plt.gcf().autofmt_xdate()
    plt.title(label)
    plt.xlabel(xlabel)
    plt.ylabel(ylabel)
    plt.savefig(os.path.join("..", "figs", fname))
    plt.clf()
```

Listing 4: Plotting function

6 Conclusion

In this project, I have learned how to do auto regressive models in general. I learned how to implement and interpret them. I have learned the importance of how to look at a data and transform it into a model. This was the hardest part of the project since we had to start barebone with just brief hints. Finally, I learned that when fitting a model, one should go from a simple model to complex, since trying to make something complex may not work well, which I ended up wasting a lot of time.

If I were given more time, I would have tried to get nearest neighbors of canada using KNN and also add them to the feature space. Also I would have tried doing multiple transformations into on feature space. For example, right now the model I implemented it either uses all polynomial basis of 3, or all basis of 2. I could have mixed this up for each feature, having different polynomial basis depending on the model which I would do the same case for the log space. Finally I would try using more linear transformations instead of just polynomial.

7 Additional Notes

Code style from: `https://www.overleaf.com/learn/latex/code_listing`