# This is a report for the course INF5870.

The assignment given is to use different methods of machine learning to use in the context of wind power generation. Our goal is to get accurate predictions of generated wind power from a given wind farm, based on previously recorded data from the same farm.

A dataset TrainData.csv is given, detailing data measured from a real wind generator over 22 months. The dataset contains the following data:

* TIMESTAMP: giving date and time.
* POWER: Measured output power from wind turbine. 7
* U10: Zonal (east-west) component of forecasted wind at 10m above ground level.
* V10: Meridional (north-south) component of forecasted wind, 10m above ground level.
* WS10: Wind speed at 10m above ground level.
* U100: Zonal component of forecasted wind at 100m above ground level.
* V100: Meridional component of forecasted wind, 100m above ground level.
* WS100: Wind speed at 100m above ground level.

These data will function as our training-set for this assignment.

We then have a second dataset, WeatherForecastInput.csv, that has the same information, minus POWER, for the 13 months following TrainData.csv. We use WeatherForecasting.csv as our test-set to predict the produced power in this period.

Our third dataset, Solution.csv, contains the real produced power of the wind farm in the period described in our test-set. We will try to minimize the discrepancy between our predicted output, and the actual output of the wind farm.

## Task 1

Four different machine learning models are being considered in this task:

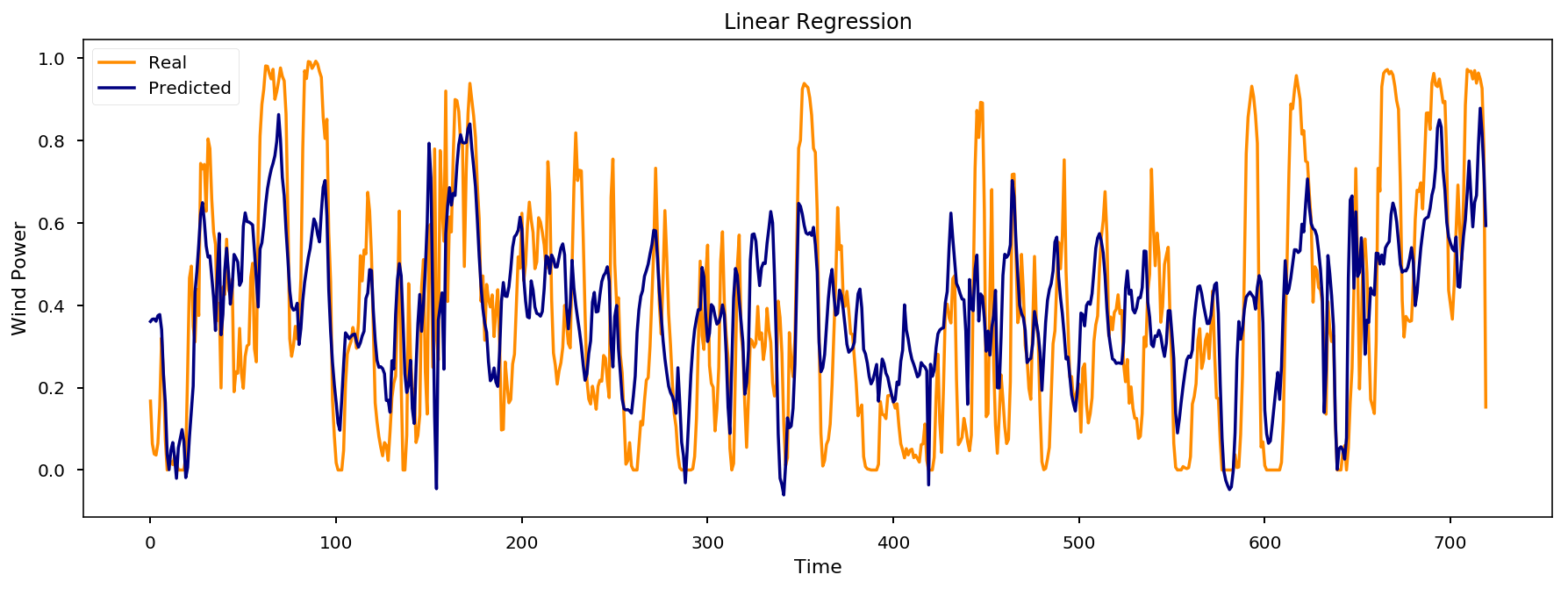
* Linear regression (LR)
* K-nearest neighbor (kNN)
* Supported Vector Regression (SVR)
* Artificial neural networks (ANN)

To train these models, we simply use windspeed (WS10) and output power (POWER). To figure out which one of these models produce the most accurate prediction, we check the predicted power output from each of the models against the measured power output using Root Mean Square Error (RMSE). The lower RMSE-value, the better our prediction is. We also use a coefficient of determination, R^2, as a double check on our test. The higher R^2 score, the better the prediction. A R^2 value of 1 shows a perfect prediction.

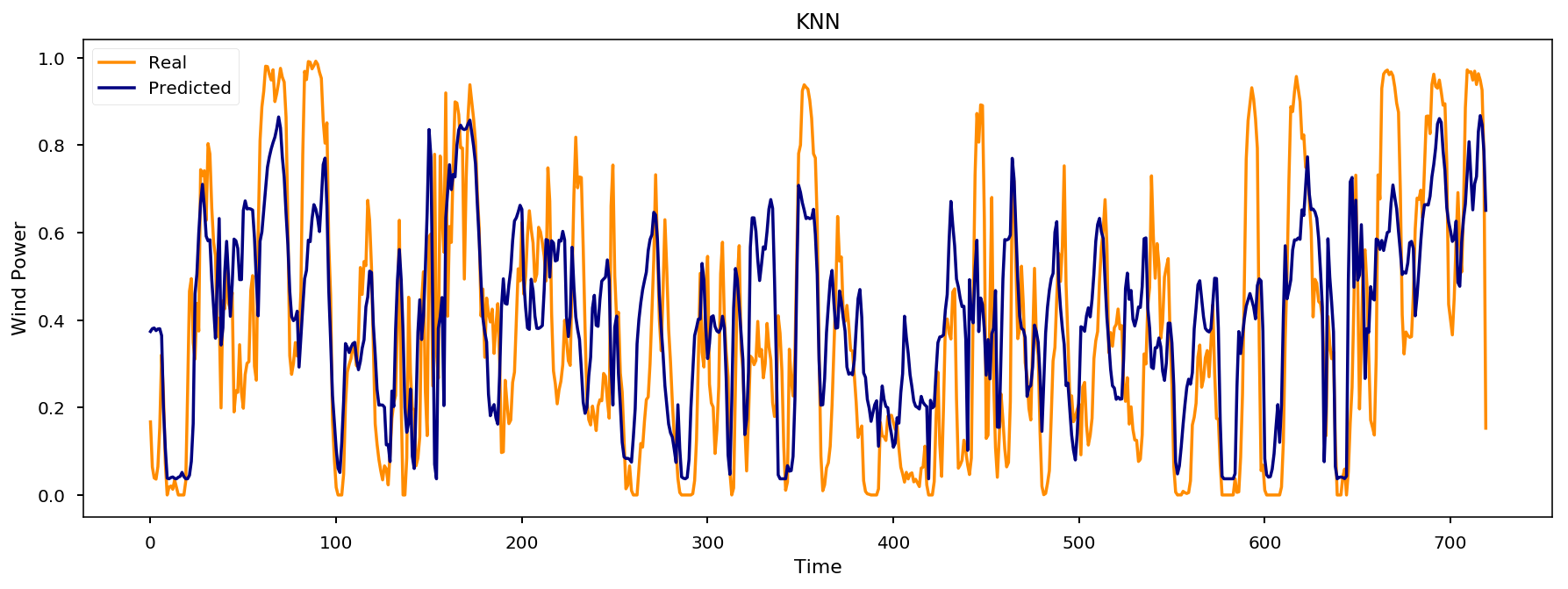
For our ANN-model, we tried out with several different numbers of layers and nodes within the layers. The architecture we decided on was 2 hidden layers with 30 nodes in the first layer and 20 nodes in the second with a dropout of 0.2 between them.

We optimized the models using cross validation grid search for some hyperparameters, except the linear regression model as it does not contain any hyperparameters to tweak. In the case of ANN we chose to do this with trial and error instead of using cross validation grid search.

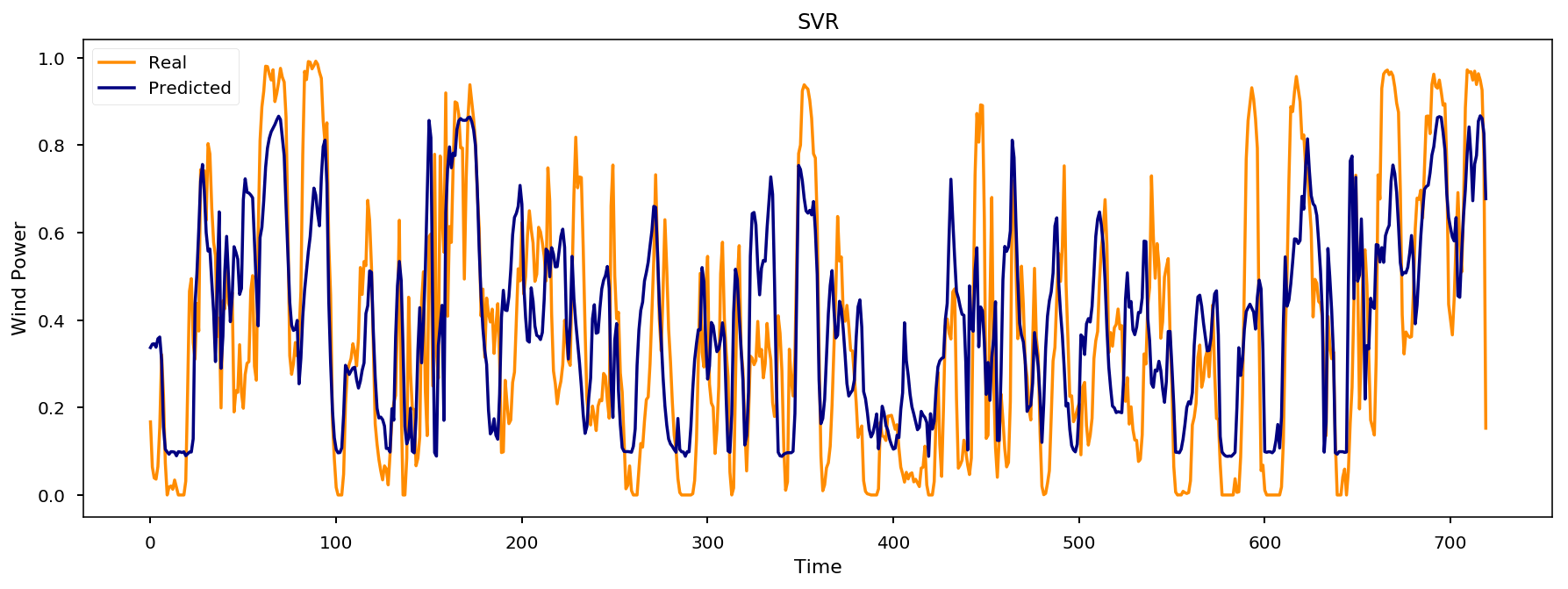
### Results:



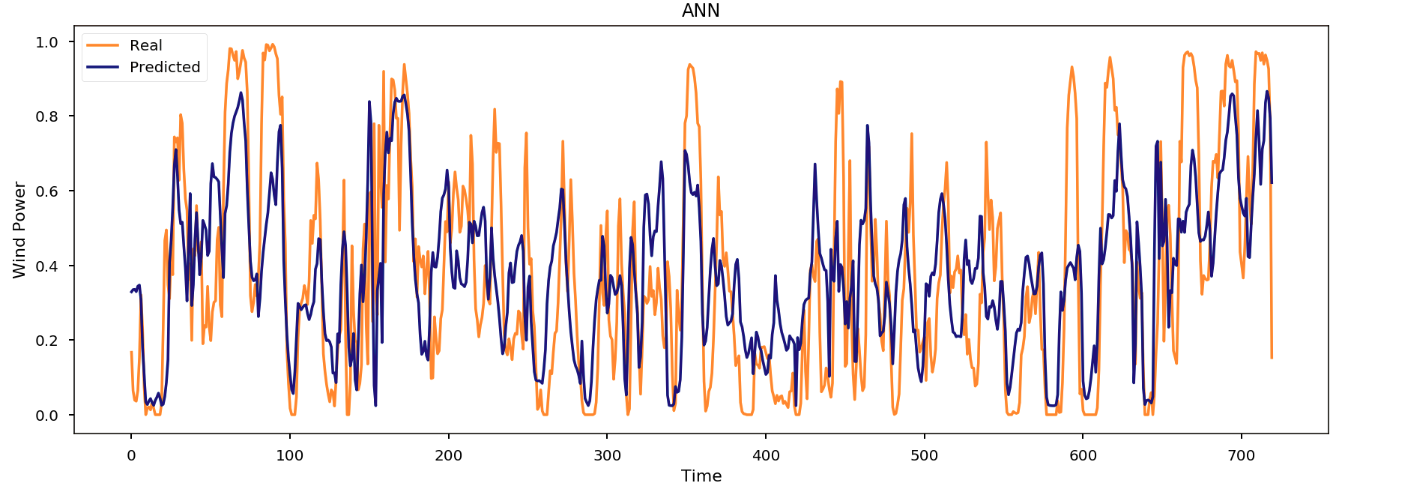
**Figure 1:** Prediction based on the linear regression model.



**Figure 2:** Prediction based on the k-nearest-neighbor model.



**Figure 3:** Prediction based on the supported vector regression model.



**Figure 4:** Prediction based on our artificial neural network model.

Table showing which models had the best prediction:

|  |  |  |
| --- | --- | --- |
| Name of model | RMSE | R2 |
| Linear regression | 0.2164 | 0.45 |
| k-nearest neighbors | 0.2167 | 0.45 |
| Supported vector regression | 0.2138 | 0.47 |
| Artificial neural network | 0.2110 | 0.48 |

### Conclusion:

kNN and LR are the losing models in this race and they are approximately equally accurate.

SVR produces decently accurate predictions.

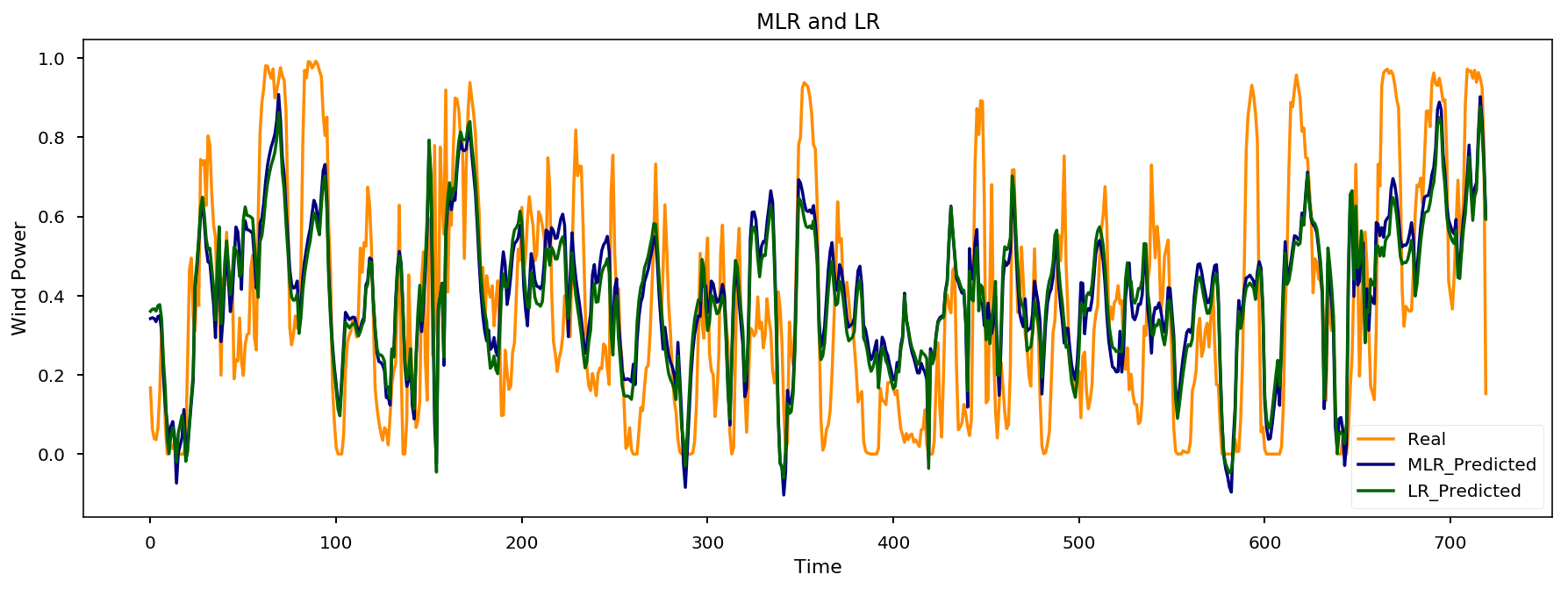
Our ANN model has a few kinks, where it produces differing quality predictions every time. For the most part it is the best model based on RMSE and R2.

## Task 2

In this task, we build a multiple linear regression model using both wind speed and wind direction (read from U10 and V10). This is to be contrasted with the linear regression model we used in task 1, where we only used wind speed.

Again we compare them to the real power generation data using RMSE and R^2.

### Results:



**Figure 5:** The figure shows our predictions based on a linear regression model and a multiple linear regression model.

Table showing how well these two models preformed:

|  |  |  |
| --- | --- | --- |
| Model name | RMSE | R2 |
| Linear regression | 0.2164 | 0.45 |
| Multiple linear regression | 0.2118 | 0.48 |

### Conclusion

The multiple linear regression model clearly makes better predictions than the linear regression model. From figure 5 it can be seen that the MLR-line stretches closer to the extreme points than the LR-line does.

## Task 3

In this task we are asked to make a prediction for power generation without the use of wind speed or direction in the training data. The last thing we got to base our predictions on at this point is TIMESTAMP, date and hour.

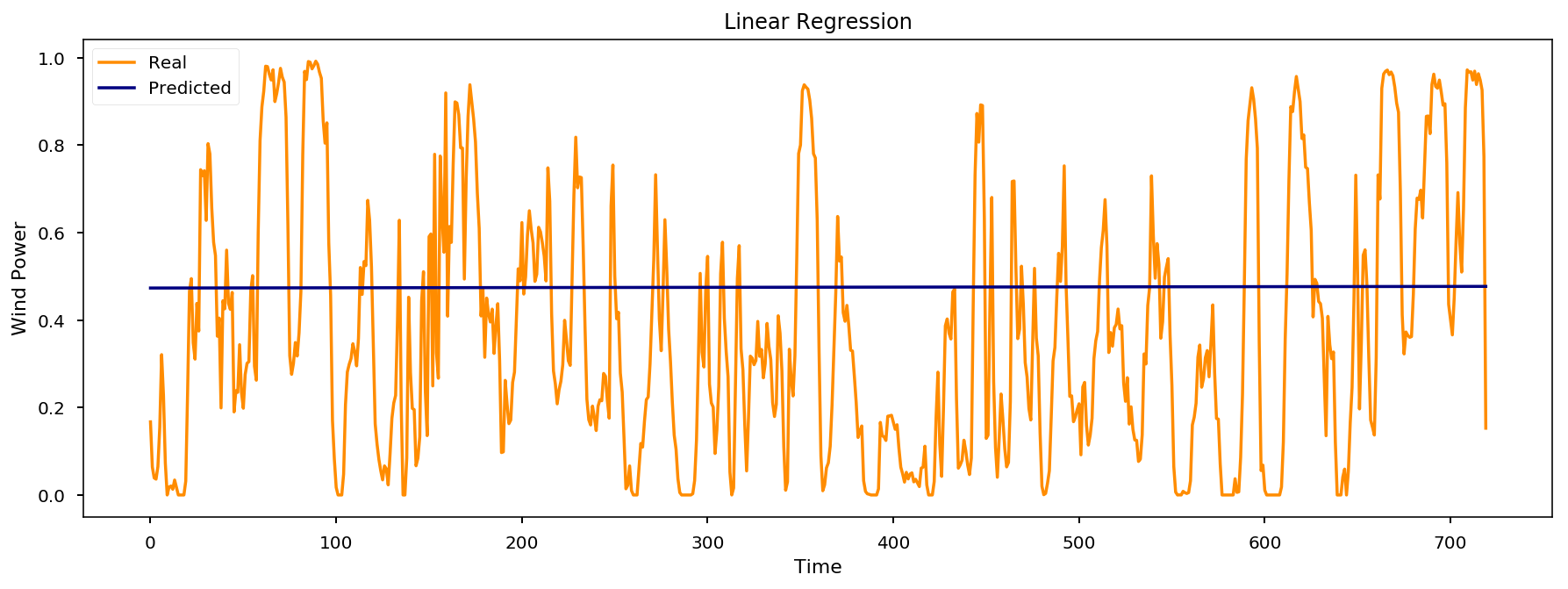
We will use a LR model and a recursive neural net (RNN) to solve this task.

To make the training set for our RNN, we restructured the dataset so that each row contained one hour of production and the following hour of production. Our goal was to make a model where the next hour of production is predicted using the current hour’s production.

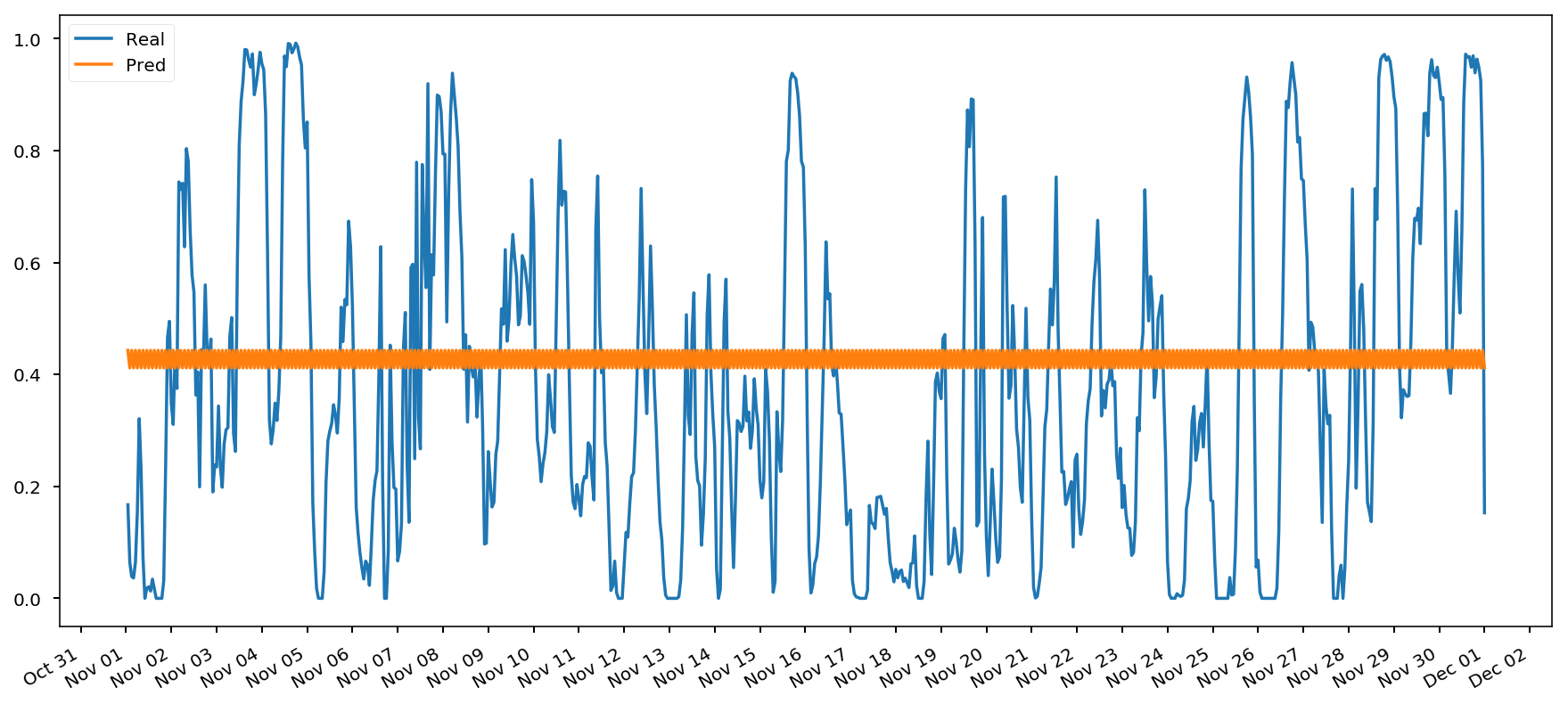
To make the linear regression model, we simply removed everything from the dataset except POWER, and used index as our timestamp, since it is equivalent with one hour increments.

We used RMSE and R^2 to check which method produced the most accurate prediction.

### Results:



**Figure 6:** Prediction based on a linear regression model using only time data. One unit on the x-axis is one hour.



**Figure 7:** Prediction based on a recursive neural network model using only TIMESTAMP.

Table showing how well the models preformed.

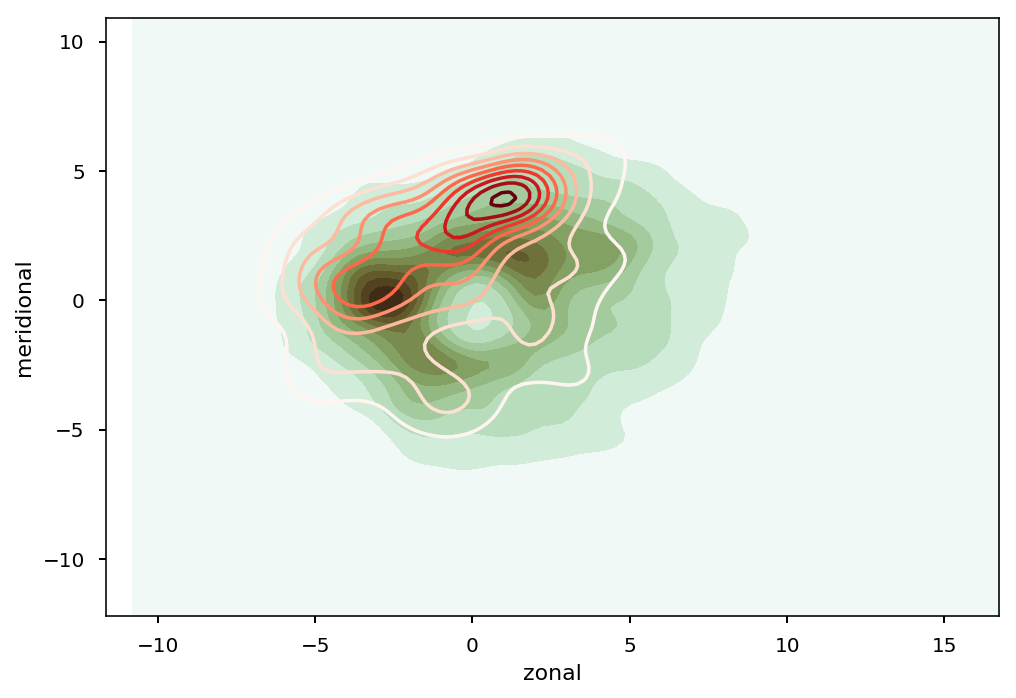
|  |  |  |
| --- | --- | --- |
| Model name | RMSE | R2 |
| Linear regression | 0.3087 | -0.11 |
| Recursive neural network | 0.2976 | -0.0323 |

### Conclusion:

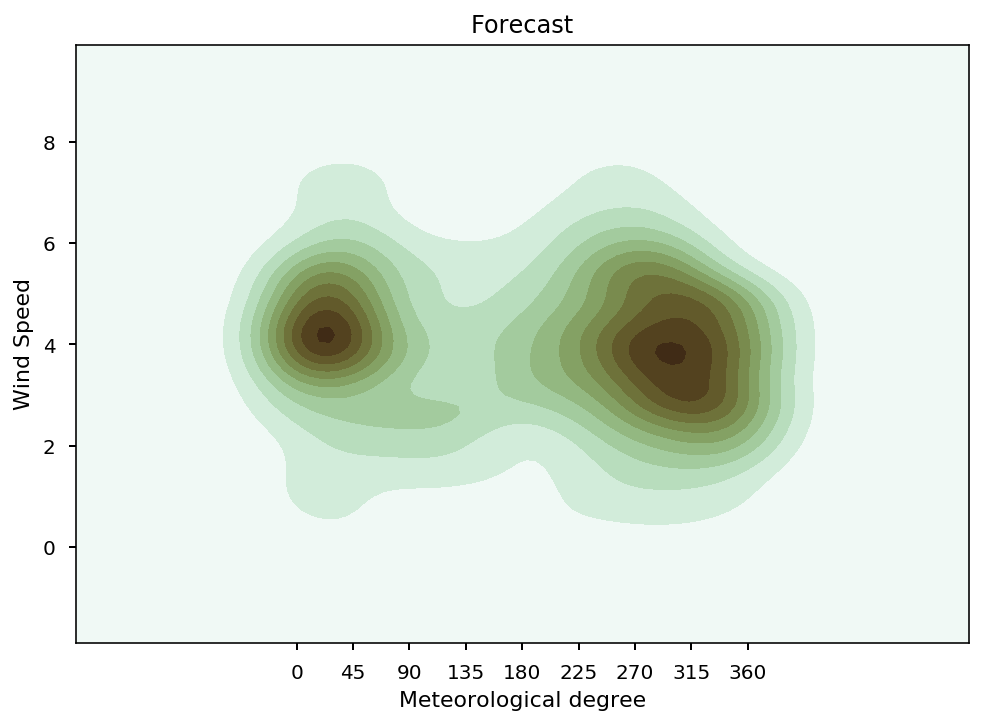
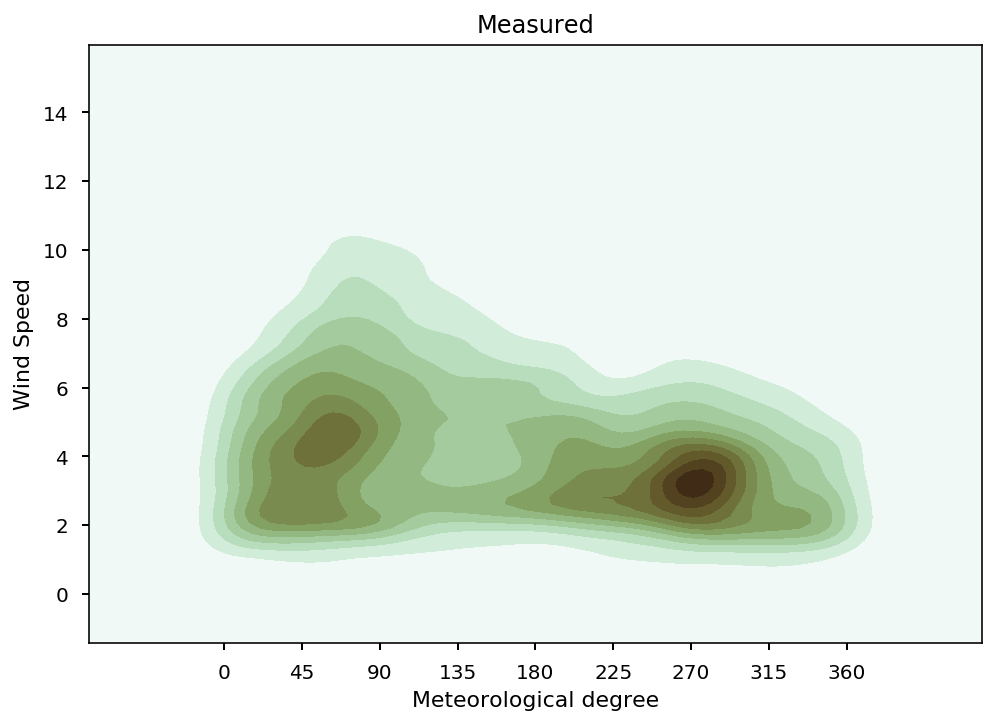
None of the models predict the generated power well at all. A negative R2 suggests that just taking the average of all the data would provide a better prediction than using one of our models.

## Other stuff:

Using wind directions, we created some gradient maps to visualize where winds blow from.



**Figure 8:** Red gradient shows predicted wind direction distribution. Green gradient shows measured wind direction distribution. Darker gradient means a higher rate of winds comes from the given direction.



**Figure 9:** Measured wind direction and speed. The x-axis shows angular coordinates.

**Figure 10:** Forecasted wind direction and speed. The x-axis shows angular coordinates.

# Appendix: Code

*# Import standard***import** numpy **as** np  
**import** pandas **as** pd  
**import** matplotlib  
**import** matplotlib.pyplot **as** plt  
**import** tensorflow **as** tf  
**import** seaborn **as** sns  
**import** missingno  
**from** matplotlib.dates **import** DayLocator, DateFormatter  
  
*# Import keras***import** keras  
**from** keras.models **import** Sequential  
**from** keras.layers **import** InputLayer, Input, SimpleRNN  
**from** keras.layers **import** Dense, Dropout  
**from** keras.optimizers **import** RMSprop  
  
*# Import Scikit-Learn***from** sklearn.linear\_model **import** LinearRegression  
**from** sklearn.svm **import** SVR  
**from** sklearn.neighbors **import** KNeighborsRegressor  
  
**from** sklearn.preprocessing **import** StandardScaler  
**from** sklearn.preprocessing **import** MinMaxScaler  
**from** sklearn.model\_selection **import** train\_test\_split  
**from** sklearn.pipeline **import** Pipeline  
**from** sklearn.model\_selection **import** cross\_val\_score  
**from** sklearn.model\_selection **import** GridSearchCV  
**from** sklearn.metrics **import** mean\_squared\_error, r2\_score  
  
*# Import data*filename1 = **'TrainData.csv'**filename2 = **'Solution.csv'**filename3 = **'WeatherForecastInput.csv'**data = pd.read\_csv(filename1, parse\_dates=[0])  
solution = pd.read\_csv(filename2, parse\_dates=[0])  
weather\_forecast = pd.read\_csv(filename3, parse\_dates=[0])  
  
  
*# setting index*data.set\_index([**'TIMESTAMP'**], inplace=True)  
solution.set\_index([**'TIMESTAMP'**], inplace=True)  
weather\_forecast.set\_index([**'TIMESTAMP'**], inplace=True)  
  
*# Setting understandable feature names*data[**'windspeed'**] = data[**'WS10'**]  
data[**'zonal'**] = data[**'U10'**]  
data[**'meridional'**] = data[**'V10'**]  
data.drop(columns=[**'U10'**,**'V10'**,**'WS10'**,**'U100'**,**'V100'**,**'WS100'**], inplace=True)  
  
weather\_forecast[**'windspeed'**] = weather\_forecast[**'WS10'**]  
weather\_forecast[**'zonal'**] = weather\_forecast[**'U10'**]  
weather\_forecast[**'meridional'**] = weather\_forecast[**'V10'**]  
  
drop = [**'U10'**,**'V10'**,**'WS10'**,**'U100'**,**'V100'**,**'WS100'**]  
**for** i **in** drop:   
 **if** i **in** weather\_forecast.columns:  
 weather\_forecast.drop(columns=i, inplace=True)  
  
*# Wind data*cardinal\_degree = {  
 **'N'** : [348.75 , 11.25],  
 **'NNE'** : [11.25 , 33.75],  
 **'NE'** : [33.75 , 56.25],  
 **'ENE'** : [56.25 , 78.75],  
 **'E'** : [78.75 , 101.25],  
 **'ESE'** : [101.25 , 123.75],  
 **'SE'** : [123.75 , 146.25],  
 **'SSE'** : [146.25 , 168.75],  
 **'S'** : [168.75 , 191.25],  
 **'SSW'** : [191.25 , 213.75],  
 **'SW'** : [213.75 , 236.25],  
 **'WSW'** : [236.25 , 258.75],  
 **'W'** : [258.75 , 281.25],  
 **'WNW'** : [281.25 , 303.75],  
 **'NW'** : [303.75 , 326.25],  
 **'NNW'** : [326.25 , 348.75]  
}  
  
**def** wind\_dir(data):  
 *'''  
 Dataset with zonal and meridional coordinates. Outputs the cardinal- and   
 the degree direction, respectively.  
 '''* car=[]  
 deg=[]  
 u,v = data[**'zonal'**], data[**'meridional'**]  
 wind\_degree = 180/np.pi\*np.arctan2(-u,-v)+180  
 wind\_dir = pd.DataFrame(columns=[[**'car\_dir'**,**'deg\_dir'**]],index=data.index)  
  
 **for** ind, bear **in** enumerate(wind\_degree):  
 **for** direction, interval **in** cardinal\_degree.items():  
 low, high = interval   
 **if** bear >= low **and** bear < high:  
 car.append(direction)  
 deg.append(bear)  
 **if** ind != len(car)-1:  
 car.append(**'N'**)  
 deg.append(bear)   
 **return** car, deg  
  
  
  
  
data[**'car\_dir'**], data[**'deg\_dir'**] = wind\_dir(data,**'Measured'**)  
weather\_forecast[**'car\_dir'**], weather\_forecast[**'deg\_dir'**] =\  
 wind\_dir(weather\_forecast,**'Forecast'**)  
  
  
*# Defining the regression models***def** linear\_regression(X\_train, y\_train, X\_test, y\_test, plot=True):  
 *'''  
 Linear regressor function that trains the model, predicts, prints   
 errors and plots the results.   
 '''* regr = LinearRegression()  
 regr.fit(X\_train, y\_train)  
  
 *# Make predictions using the testing set* y\_pred = regr.predict(X\_test)  
  
 *# Error* rms\_r2\_score(y\_test, y\_pred)  
   
 **if** plot:  
 *# Plot outputs  
 #plot\_task1(X\_test, y\_test, y\_pred, 'Linear Regression')* plot\_powergeneration(y\_test, y\_pred, **'Linear Regression'**)  
 **return** y\_pred   
  
**def** knn\_crossval(X,y,n\_folds=10):  
 *'''  
 Defining hyperparameters for grid search, and performs   
 cross validation.  
 '''* num\_neighbours = [1, 5, 20, 50, 100, 500, 800, 1000]  
 leaf\_size = [10, 30, 50, 100, 200, 500]  
 param\_grid = [{**'n\_neighbours'**: num\_neighbours,  
 **'weights'**:[**'uniform'**],  
 **'leaf\_size'**:leaf\_size},  
 {**'n\_neighbours'**: num\_neighbours,  
 **'weights'**:[**'distance'**],  
 **'leaf\_size'**:leaf\_size}]  
 grid\_search = GridSearchCV(KNeighborsRegressor(),  
 param\_grid,  
 cv=n\_folds,  
 n\_jobs=-1)  
 grid\_search.fit(X, y)  
 grid\_search.best\_params\_  
 **return** grid\_search.best\_params\_   
   
**def** k\_nearest\_neighbors(X\_train, y\_train, X\_test, y\_test, plot=True):  
 *'''  
 K-nearest neighbors function that trains the model, predicts, prints   
 errors and plots the results. Cross validation and grid search for   
 hyperparameters are used to get the best model.  
 '''* best\_params = knn\_crossval(X\_train, y\_train)   
 neigh = KNeighborsRegressor().set\_params(\*\*best\_params)  
   
 neigh.fit(X\_train\_selected, y\_train)   
 y\_pred = neigh.predict(X\_test)  
  
 *# The Root mean squared error* **print**(**"Root Mean squared error: %.4f"** % np.sqrt(mean\_squared\_error(y\_test, y\_pred)))  
 *# Explained variance score: 1 is perfect prediction* **print**(**'Variance score: %.2f'** % r2\_score(y\_test, y\_pred))  
   
 **if** plot:  
 *# Plot outputs  
 #plot\_task1(X\_test, y\_test, y\_pred, 'KNN')* plot\_powergeneration(y\_test, y\_pred, **'KNN'**)  
  
  
**def** svr\_crossval(X, y, n\_folds=10):  
 *'''  
 Defining hyperparameters for grid search, and performs   
 cross validation.  
 '''* Cs = [0.001, 0.01, 0.1, 1, 10, 100]  
 gammas = [0.001, 0.01, 0.1, 1, 10]  
 param\_grid = {**'C'**: Cs, **'gamma'** : gammas}  
 grid\_search = GridSearchCV(SVR(kernel=**'rbf'**),  
 param\_grid,  
 cv=nfolds,  
 n\_jobs=-1)  
 grid\_search.fit(X, y)  
 grid\_search.best\_params\_  
 **return** grid\_search.best\_params\_   
   
**def** support\_vector\_regression(X\_train, y\_train, X\_test, y\_test, plot=True):  
 *'''  
 SVR function that trains the model, predicts, prints   
 errors and plots the results. Cross validation and grid search for   
 hyperparameters are used to get the best model.  
 '''* best\_params = svr\_crossval(X\_train, y\_train)  
 svr\_rbf = SVR().set\_params(best\_params)  
   
 y\_pred = svr\_rbf.fit(X\_train\_selected, y\_train).predict(X\_test\_selected)  
  
 *# The Root mean squared error* **print**(**"Root Mean squared error: %.4f"** % np.sqrt(mean\_squared\_error(y\_test, y\_pred)))  
 *# Explained variance score: 1 is perfect prediction* **print**(**'Variance score: %.2f'** % r2\_score(y\_test, y\_pred))  
   
 **if** plot:  
 *#plot\_task1(X\_test, y\_test, y\_pred, 'SVR')* plot\_powergeneration(y\_test, y\_pred, **'SVR'**)  
  
  
**def** ann\_model(X\_train, y\_train, X\_test, y\_test, plot=True):  
 *'''  
 Trains an artificial neural network. n-input channels and one   
 output(the predicted power).  
 '''* input\_shape = X\_train.shape[1]  
 output\_shape = 1  
 model = Sequential()  
  
 model.add(InputLayer(input\_shape=(input\_shape,)))  
 model.add(Dense(30, kernel\_initializer=**'lecun\_normal'**,  
 bias\_initializer=**'ones'**,activation=**'selu'**))  
 model.add(Dropout(0.2))  
 model.add(Dense(20, kernel\_initializer=**'lecun\_normal'**,  
 bias\_initializer=**'ones'**,activation=**'softmax'**))  
 model.add(Dense(output\_shape))  
  
 model.compile(optimizer=**'rmsprop'**,  
 loss=**'mean\_squared\_error'**)   
  
 model.fit(X\_train, y\_train, epochs=10, verbose=0)  
  
 y\_pred = model.predict(X\_test)  
   
 *# Error* rms\_r2\_score(y\_test, y\_pred)  
  
 **if** plot:  
 plot\_powergeneration(y\_test, y\_pred, model=**'ANN'**)  
  
  
*# Task 1  
  
#Select training dataset*train = data.drop(**'POWER'**, 1)  
train\_y = data[**'POWER'**]*#.reshape(-1,1)  
  
#X\_train, X\_test, y\_train, y\_test =\  
# train\_test\_split(train,train\_y,test\_size=0.3,random\_state=1)*sel=-1   
X\_train = train[:sel]  
y\_train = train\_y[:sel]  
  
X\_test = weather\_forecast  
y\_test = solution  
   
X\_train\_selected = X\_train[**'windspeed'**].values.reshape(-1 ,1)  
X\_test\_selected = X\_test[**'windspeed'**].values.reshape(-1 ,1)  
   
**print**(X\_train\_selected.shape,X\_test\_selected.shape, y\_train.shape, y\_test.shape)  
  
  
*#Training and predicting*train\_test = X\_train\_selected, y\_train, X\_test\_selected, y\_test  
  
np.random.seed(1)  
linear\_regression(\*train\_test)  
k\_nearest\_neighbors(\*train\_test)  
support\_vector\_regression(\*train\_test)  
ann\_model(\*train\_test)  
  
  
*# TASK 2  
# Define dataset*train = data.drop(**'POWER'**, 1)  
train\_y = data[**'POWER'**]*#.reshape(-1,1)  
  
#X\_train, X\_test, y\_train, y\_test =\  
# train\_test\_split(train,train\_y,test\_size=0.3,random\_state=1)*X\_train = train  
y\_train = train\_y  
  
X\_test = weather\_forecast  
y\_test = solution  
  
X\_train\_one = X\_train[**'windspeed'**].values.reshape(-1 ,1)  
X\_test\_one = X\_test[**'windspeed'**].values.reshape(-1 ,1)  
  
X\_train\_two = X\_train[[**'windspeed'**,**'deg\_dir'**]]  
X\_test\_two = X\_test[[**'windspeed'**,**'deg\_dir'**]]  
  
*#Training and prediction*train\_test\_two = X\_train\_two, y\_train, X\_test\_two, y\_test  
train\_test\_one = X\_train\_one, y\_train, X\_test\_one, y\_test  
  
lr\_pred=linear\_regression(\*train\_test\_one,plot=False)  
mlr\_pred=linear\_regression(\*train\_test\_two,plot=False)  
  
*# Plotting MLR, LR and real values*plt.figure(figsize=(15,5))  
  
plt.plot(y\_test.values, color=**'darkorange'**, label=**'Real'**)  
  
plt.plot(mlr\_pred, color=**'navy'**, label=**'MLR\_Predicted'**)  
plt.plot(lr\_pred, color=**'darkgreen'**, label=**'LR\_Predicted'**)  
  
plt.xlabel(**'Time'**)  
plt.ylabel(**'Wind Power'**)  
plt.title(**'MLR and LR'**)  
plt.legend()  
*#plt.ylim(-0.1,y\_test.max().all()+0.1)*plt.show()  
  
*#TASK 3  
  
#Linear Regression*train\_y = data[**'POWER'**]  
train = np.arange(len(train\_y)).reshape(-1,1)  
y\_test = solution  
test = np.arange(train[-1],len(y\_test)+train[-1]).reshape(-1, 1)  
  
train\_test = train, train\_y, test, y\_test  
  
\_=linear\_regression(\*train\_test)  
  
*#RNN  
  
# frame a sequence as a supervised learning problem***def** timeseries\_to\_supervised(data, lag=1):  
 *'''  
 Transformes the timeseries data into supervised dataset  
   
 [a1, a2]  
 [a2, a3]  
 [a1,a2,a3,a4,a5,a6] ---> [a3, a4]  
 [a4, a5]  
 [a5, a6]  
 [a6, 0]  
 '''* df = pd.DataFrame(data)  
 columns = [df.shift(i) **for** i **in** range(1, lag+1)]  
 columns.append(df)  
 df = pd.concat(columns, axis=1)  
 df.fillna(0, inplace=True)  
 **return** df  
   
*# fit an rnn network to training data***def** fit\_rnn(train, batch\_size, nb\_epoch, neurons):  
 *'''  
 Defines the RNN model and fits to the training data.  
 '''* X, y = train[:, 0:-1], train[:, -1]  
 X = X.reshape(X.shape[0], 1, X.shape[1])  
 model = Sequential()  
 model.add(SimpleRNN(neurons, batch\_input\_shape=(batch\_size, X.shape[1], X.shape[2]), stateful=True))  
 model.add(Dense(1))  
 model.compile(loss=**'mean\_squared\_error'**, optimizer=RMSprop(lr=0.001))  
 **for** i **in** range(nb\_epoch):  
 model.fit(X, y, epochs=1, batch\_size=batch\_size, verbose=0, shuffle=False)  
 model.reset\_states()  
 **if** i%10==0:  
 **print**(**'Finnished with %s epochs'** %(i+1))  
 **return** model  
   
*# make a one-step forecast***def** forecast\_rnn(model, batch\_size, X):  
 *'''  
 Uses a pre-trained model and predicts the next value.  
 '''* X = X.reshape(len(X),1, 1 )  
 yhat = model.predict(X, batch\_size=batch\_size)  
 **return** yhat[0,0]  
  
*# transform data to be stationary*series = train\_y  
raw\_values = series.values[80:]*#[-3000:]  
  
# transform data to be supervised learning*supervised = timeseries\_to\_supervised(raw\_values, 1)  
  
supervised\_values = supervised.values  
   
batch\_size = 100  
epochs = 100  
neurons = 4  
**print**(**'\*\*----Start train model----\*\*'**)  
*# fit the model*rnn\_model = fit\_rnn(supervised\_values, batch\_size, epochs, neurons)  
**print**(**'\*\*----Finnished training----\*\*'**)  
  
*# forecast the entire training dataset to build up state for forecasting*train\_reshaped = supervised\_values[:, 0].reshape(len(supervised\_values), 1, 1)  
rnn\_model.predict(train\_reshaped, batch\_size=batch\_size)  
  
*# walk-forward validation on the test data*predictions = list()  
history = supervised\_values[-batch\_size:,-2]  
**for** i **in** range(len(y\_test)):  
 X=np.array(history[-batch\_size:])  
 yhat = forecast\_rnn(rnn\_model, batch\_size, X)  
 *# store forecast* predictions.append(yhat)  
 history= np.concatenate((history,np.array([yhat])), axis=0)  
   
*# report performance*r2 = r2\_score(y\_test.values, predictions)  
**print**(**'R^2: %.4f'** % r2)  
rmse = np.sqrt(mean\_squared\_error(y\_test.values, predictions))  
**print**(**'RMSE: %.4f'** % rmse)  
  
*# line plot of observed vs predicted  
#plot data*fig, ax = plt.subplots(figsize=(15,7))  
*#data.plot(ax=ax)*ax.plot(y\_test.index,y\_test.values, label=**'Real'**)  
*#set ticks every week*ax.xaxis.set\_major\_locator(DayLocator())  
*#set major ticks format*ax.xaxis.set\_major\_formatter(DateFormatter(**'%b %d'**))  
ax.plot(pd.DataFrame(predictions,index=y\_test.index),label=**'Pred'**)  
plt.legend()  
plt.gcf().autofmt\_xdate()  
plt.show()